D1.3

SAFURE Framework Specifications

Project number:	644080
Project acronym:	SAFURE
Project title:	SAFURE: SAFety and secURity by dEsign for interconnected mixed-critical cyber-physical systems
Start date of the project:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1

Deliverable type:	Report
Deliverable reference number:	ICT-644080 / D1.3/ FINAL | 1.0
Work package	WP 1
Due date:	31.10.2015
Actual submission date:	30.10.2015

Responsible organisation:	TCS
Editor:	Dominique Ragot
Dissemination level:	Public
Revision:	FINAL | 1.0

Abstract:	This document defines the initial specifications of the SAFURE Framework, which will be input to the development WPs. A final public version of these specifications will be released in M36 following the Integration activities in WP6
Keywords:	Framework, Modelling, Methods, Architectural concepts, building blocks

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

This work was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0025. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

This document has gone through the consortium's internal review process and is still subject to the review of the European Commission. Updates to the content may be made at a later stage.

Editors

Dominique Ragot, Emmanuel Gureghian, Matthieu Walle (TCS)

Contributors/Reviewer (ordered according to beneficiary numbers)

André Osterhues, Lena Steden (ESCR)

Stefania Botta (MAG)

Carolina Reyes (TTT)

Mikalai Krasikau (SYSG)

Jonas Diemer (SYM)

Daniel Thiele (TUBS)

Jaume Abella (BSC)

Marco di Natale (SSSA)

Rehan Ahmed, Lothar Thiele (ETHZ)

NOTE: This document defines the initial specifications of the SAFURE Framework, which will be input to the development WPs. A final public version of these specifications will be released in M36 following the Integration activities in WP6

Contents

List of Figures

List of Tables

Chapter 1 Definition and introduction

1.1 Definition

The SAFURE Framework is a collection of architectural concepts, functional building blocks, modeling approaches, analysis methods and tools, as well as assurance and certification methods aligned to develop and validate mixed-critical systems.

1.2 Introduction to the SAFURE framework

The SAFURE framework is at the heart of Common Criteria (ISO 15-508), legacy system engineering and model driven methodologies. Thus, the according methodology must encompass two important topics:

- Describing appropriately the mixed criticality set of properties on the complex systems envisioned in the SAFURE project, either by exhibiting the said properties or by setting them as an environment hypothesis
- Preserving mixed criticality properties consistency along the envisioned systems (thus describing some important properties shared between those systems) and internally inside its system (for example achieving both safety and security policies enforcement)

1.2.1 System engineering aspects

In the following parts we will adopt the unified presentation of mixed criticality properties as Jean-Claude Laprie described in *Dependability of Computer Systems: from Concepts to Limits*.

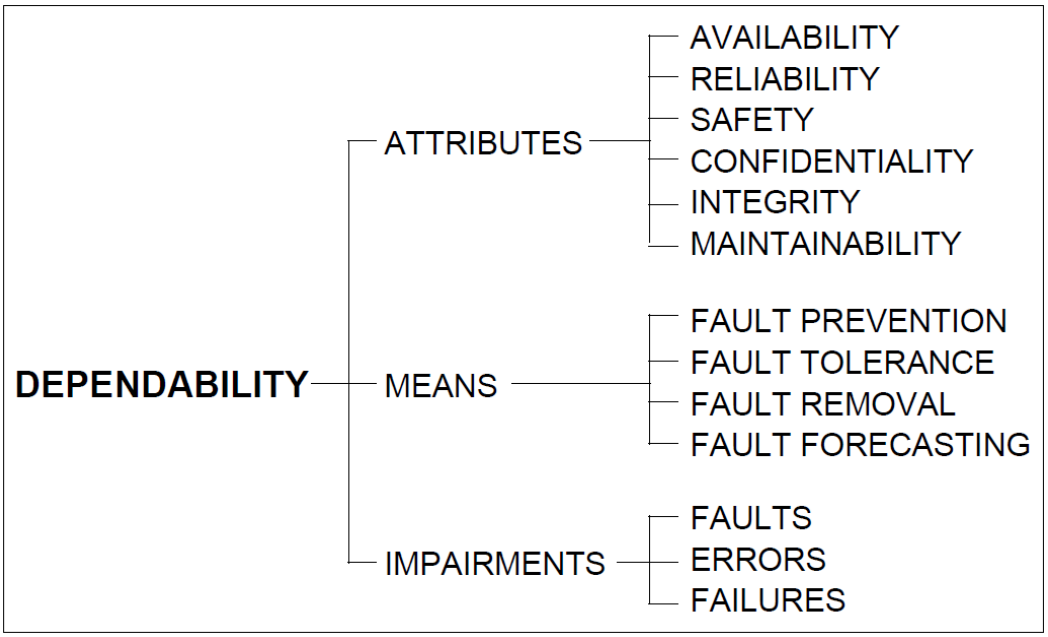

Figure 1: dependability capability tree

It is not the purpose of this chapter to describe exactly these attributes, means and impairments.

The SAFURE objectives heavily relies on the satisfaction of all attributes also described as set of security properties in the Common Criteria as two sub-sets of attributes can be refined:

- Safety attributes: maintainability, reliability and safety
- Security (as of IT security) attributes : availability, confidentiality and integrity

One could note not that availability could be part of both subsets but usually in IT security terms, availability is envisioned as a qualitative property while in pure dependability while in safety attributes it is a quantitative concept part of the reliability attributes.

The SAFURE framework will heavily rely on pre-existing environments which are very different from one partner to another. Thus, all suitable works performed in SAFURE shall exhibit the following properties:

- Correct from a mixed-criticality point of view
- Consistent inside and between each system environment

But it cannot be extensively defined, meaning that a complete system definition is not under the scope of this methodology, and that only an inductive partial definition shall be required, notably through the definition of properties over parts of the systems which are outside of the technical scope of the SAFURE project. For example, the availability of an internet connection shall be explicitly either required or either tolerated through the definition of the appropriate architecture.

1.3 Interoperability

The nice thing about standards is that you have so many to choose from

Andrew S. Tanenbaum

1.3.1 Status at start of project

Interoperability within and between critical systems is a crucial objective at the heart of the SAFURE project. Anyway, since the SAFURE projects gets its root from critical systems requirements such as automotive, avionics or medical systems, it is important to note that interoperability in its common acceptation is deemed less critical than the safety and security properties which must exhibited by the said system. This is at the opposite of usual COTS products strategy, which are ready to sacrifice critical properties interoperability in order to impact positively services and products diffusion. Furthermore, in the SAFURE framework two interoperability aspects must be considered:

- The usual functional interoperability whom management is crucial for the product versatility and therefore for its commercial success. This also includes the choice of interoperable components during the design easing product and system development and integration. This interoperability is at the heart of a successful time-to-market strategy.
- But critical properties interoperability is also crucial since it is also subdivided in two main concerns:

- Critical properties interoperability which are at the heart of the SAFURE project. Managing consistently all the safety and security properties is an important concern since sometimes, satisfying all these properties can be deemed impossible at equipment level. Thus, only at system level, one can decide which property is more important than another.
- Products or equipment interoperability at integration level which raise the case of preserving products critical properties at system level by the mean of appropriate architectures. This aspect is outside of SAFURE framework.

Finally, hardware issues are not considered here since they are outside of the SAFURE framework project.

1.3.1.1 What is needed and required

Requirements of the SAFURE projects can be studied at 3 levels: equipment level, system level and system of systems level. The projections of the impact of the main building blocks are described in the following table:

	Equipment	System	System of Systems
Hardware Security Module	X		
Separation Kernel	X		
Secure Software Update	X	X	
Secure Boot	X		
Secure Communication	X	X	X
Secure GUI	X		
Secure Element	X	X	
Multi stakes owners trusted foundations	X	X	X

Table 1: impact of main building blocks

This separation includes both building blocks technologies studied in Chapter 3 but also some underlying technologies which encompass those three layers *e.g.* all the Internet Protocol related technologies defined in the informal or formal interest groups such as RFC, IEEE, IETF.

This table also shows that the most prevalent building blocks around the SAFURE framework are secure communication and multi-stakes trusted foundations whose impact is seen on the three layers: equipment, system and system of systems.

Multi stakes owners trusted foundations is important not only as a functional building block in itself but because it is a key trigger of trust between organizations and therefore to the establishment of communications between heterogeneous systems of systems. And as a way to support this statement, IP (Internet Protocol) communication technologies are at the heart of the SAFURE framework since they are a principal of the secure communication sub-framework.

This three layers visions is consistent with the actual trends of actual IT security concerns and three couples appears as an obvious association between the equipment level and internet of things (IoT), system level and SCADA security and finally between the systems of systems layer and service oriented architectures. And we are seeing a convergence between Information Technology (IT) and Operational Technologies which are often long-lasting legacy technologies.

Thus two kinds of candidates for adoption are envisioned:

- Equipment : IoT standards such as AllJoyn, OIC, Google's Brillo or even Apple's HomeKit
- Systems or Systems of systems which are in fact complex systems. SCADA and OT standards are prevalent here and we will for example study the OPC Unified Architecture impacts.

Nevertheless, it is important to note that the later will be examined at the impact level on the equipment as a set of requirements since delivering a secure and safe equipment is the main focus of the SAFURE project. Thus, since the use of IP communication between the OT and equipments cannot be deemed as reliable. Real-time safety concerns and their impact will be analyzed only at the equipment level while security concerns shall be addressed at all levels.

The other relevant standards will be analyzed during the SAFURE project.

1.3.2 Status at end of project

While section 1.3.1 describes the current status and how we target to improve within the SAFURE project, this section will be updated by the end of the project. It will create the relation how the SAFURE outcomes influenced the critical interoperability and how the set of standards could be refined.

Chapter 2 Architectural Concepts

The SAFURE project adapts the architectural concept developed in the EURO-MILS project [9] to cover the safety and security aspects.

The typical MILS system is composed of

1. A set of partitions holding the functionalities (e.g. applications and middleware)
2. A platform, composed of
    a. A core including: a separation kernel, HW and its configuration (e.g. CPU, MMU, IOMMU, memory bus), and optional critical HW devices along with their own software.
    b. Other blocks such as device virtualization software, security and audit generation, chain of trust.

While in a MILS system there is a strong focus on independence and security (although the S could also mean safety, as showed in the IMA section of the MILS white paper), in SAFURE the focus is on integration of safety and security properties, as depicted in Chapter 1 of the present document. In this respect, the SAFURE architecture could be named as Modular Integration Layout for Dependable Systems (MILDS). Explanation:

1. The primary target is modularity, not multiplicity, since compositional construction and proof are mandatory to achieve complex/evolutive systems validation. MILS can be used in complement to bring multiplicity through independence.

2. Integration is also mandatory since in many systems security and safety properties are neither strictly separable nor independent, and thus have to be handled coherently and consistently. This does not imply however that some degree of separation cannot be pursued for the sake of validation efficiency but that joint-dependencies of security and safety properties have also to be addressed in the design process.

3. Layout is proposed as an extension of the terms Levels, as a way to focus on components, with their properties and their interactions.

4. Dependable Systems is a reference to the dependability explained in Chapter 1 that encompasses security and safety properties.

The SAFURE architecture includes:

1. Building blocks (BB) able to be used in a cross-domain approach and implementing dependability (security and/or safety) properties.

2. A communication infrastructure able to provide interactions between building blocks in a dependable way, e.g. by filtering or limiting the access to some shared resources or between some functions.

2.1 Timing

A number of on-chip hardware shared resources exist in COTS multicores. Among those we identify on-chip interconnects, shared cache memories and memory controllers. Uncontrolled sharing of those resources allows tasks running in some cores clogging the system by accessing those shared resources frequently, thus causing starvation in other tasks running in other cores, which may have higher criticality. This fact is particularly challenging when performance decreases drastically for real-time tasks.

The other place where timing plays a role is timing covered channels. A separation kernel allows separate partitions by limitation of direct access to the hardware resources where the target software is running but there could be some indirect ways to exchange the data between partitions. For example a cache is protected from direct attacks against the confidentiality of content with the help of Memory Management Unit but observing the timing behavior can reveal the information about how other applications have used the cache. This timing behavior can be leveraged to form a covert information flow channel between the partitions. An information flow channel is called a timing covert channel when the basis of information transfer is not by direct copying of data but rather by modulating and observing the availability or behavior of a physical or logical resource.

2.2 Temperature and Energy

Due to rapid and increasing rise in power densities of modern processing platforms, power and thermal constraints are becoming increasing critical. Furthermore, for battery powered mobile computing platforms, energy consumption is also a critical consideration. These (relatively) new mediums have several safety and security implications. In the remaining section, we will explore these aspects:

2.2.1 Denial of Service Attacks

Modern processing platform employ Dynamic Thermal Management (DTM) to keep system temperature below a safe operating threshold. These DTM approaches reduce system temperature by throttling down frequency/voltage; in turn reducing the system power consumption and allowing the platform to cool. However, reducing frequency also causes system performance to degrade. This side-effect of DTM makes denial of service attacks possible [5].

One can imagine an application over exercising the platform and causing DTM to kick-in. In-turn this application will cause other applications to get reduced service due to degraded system performance. This effect is even more adverse in a mixed-criticality scenario. An application of low criticality may raise processor temperature; causing other, potentially higher criticality applications, to get reduced service.

In battery-powered mobile platforms, a denial of service attack based on energy can also be conceived [6]. An application may excessively drain the battery by spawning operations that have high power consumption and/or not allowing the platform to stay in sleep state. Such an over utilization of battery may cause other, potentially higher criticality applications, to get reduced/no service due to lack sufficient energy.

2.2.2 Thermal/power covert channels:

Thermal sensors are widely present in modern computing platforms to enable DTM. However, they represent a security breach in privilege-separated or sandboxed systems. Thermal sensors can be used to implement a covert channel that allows applications to leak data [3]. For instance, consider the dual-core system depicted in Figure 2. A source (src) app runs on core 0 and has access to sensitive data that is only stored locally, but it does not have network access. A sink (snk) app runs on core 1 and can freely communicate over the

network, but has no rights to access the sensitive data. In theory, privilege-separation should disallow communication between the two applications and keep the sensitive data secured. However, if the sink app can read the on-chip temperature sensors, communication is possible. Source application can modulate system temperature by its activity and transfer sensitive data through the thermal covert channel, regardless of privilege separation. This poses as a security risk. Covert channels can also be conceived in the power domain. Applications may communicate by altering the system power consumption and reading power sensors.

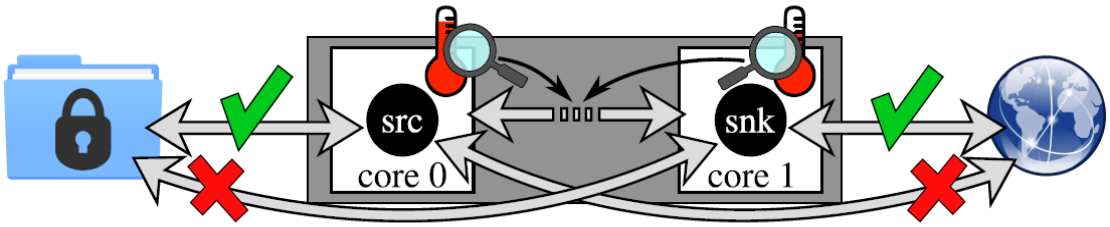

Figure 2: Thermal covert channel example[1].

2.3 Security Architecture

As depicted in the Figure 3, we defined 4 architectures suitable for mixed-criticality applications, all of them are using a separation kernel to compartmentalize critical (or sensitive) and non-critical (or non-sensitive) environment.

At the lowest (but definitely acceptable for most applications) security level, applications and possibly user operating system are running on top of the separation kernel which assigns resources to partitions and implement communication between partitions. The security is stateful, meaning that all data and configurations are maintained in the device. Confidentiality and integrity of this information shall be ensured by an appropriate local protection implementing a set of countermeasures, in particular in case of physical access by an attacker. In this level, all code and data are processed by the main microprocessor and the root key is stored in clear (potentially hidden or obfuscated) in the memory. No secure element is used.

To improve the security level of mobile devices running mixed critical applications, there is two, non-exclusive, solutions. The first one consists in using a secure element to protect data or even to execute part of the critical environment. The second uses virtualization and centralization of the critical environment in a data center, we call it stateless security since data and configurations are not maintained in the device.

When combining stateless security, secure element, and possibly other security functions, we obtain a high-grade security device. This security level, along with its constraints, is devoted to governmental application, for example to deal confidential EU information. Therefore it is not in the scope of the SAFURE project.

[1] The source app (src) has access to restricted data but no network access; the sink app (snk) has no access to the restricted data but has network access. A compromised source app can leak sensitive data to the sink app through the thermal covert channel

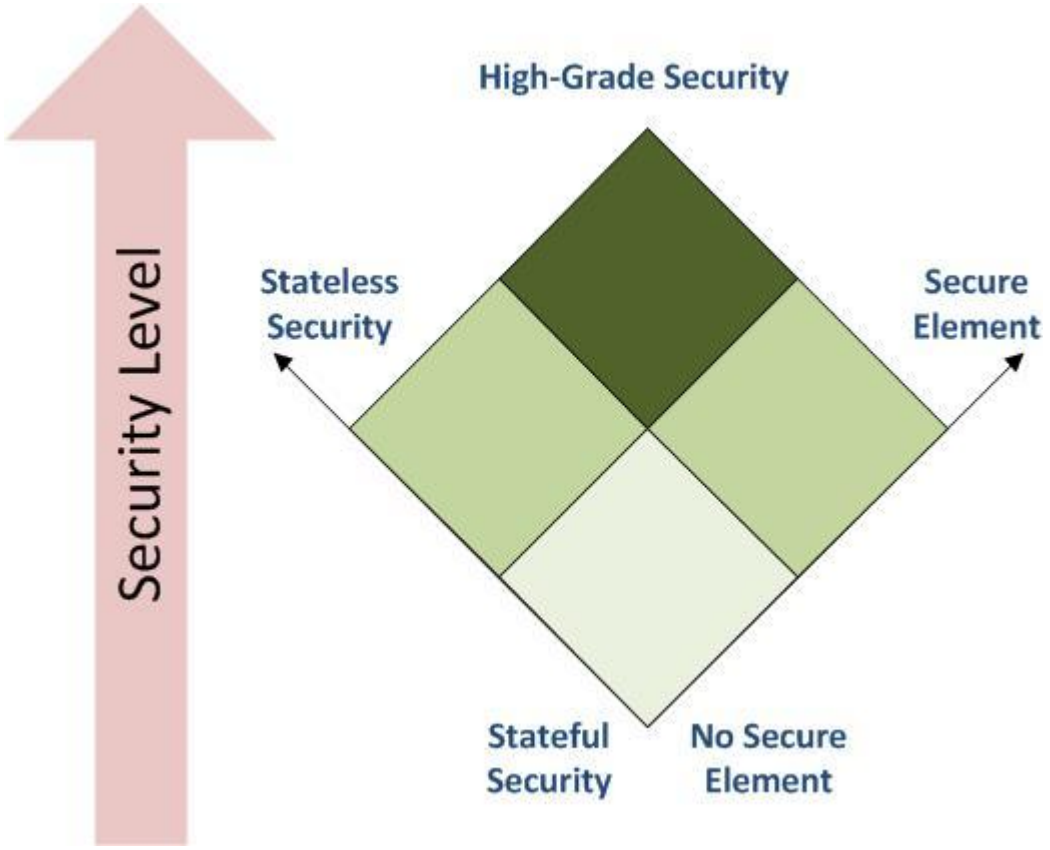

Figure 3: Security Architecture using a Separation kernel

2.3.1 Stateless Security

Instead of running critical applications in a dedicated environment of the device, it might be advantageous to move the entire environment to a data center. Then, the user device will simply access services securely in the data center, just from a virtualization client or even just a browser. As discussed in [10], by having corporate applications and data moved to the data center, the separation of business and personal services is achieve

While this solution is suitable for Professional vs. Personal application (Use case 2), it is not for appropriate for a device controlling an IMD for obvious availability reasons.

- A client is installed on a dedicated partition and all data are stored and processed in a server architecture that requires a connection from the access terminal to the infrastructure (data center).
  - Hosted shared Desktop
  - Hosted virtual desktop
  - Central hosted desktop
  - Local Virtual Desktop

Chapter 3 Functional Building Blocks

This section describes the main functional building blocks that have been identified in contributing to fulfillment of mixed criticality requirements

Note: If relevant, this list will be updated during the SAFURE project.

3.1 Hardware Security Modules (HSMs)

Note: although, the following building blocks (or BB in this document) are defined as hardware security modules BB in this chapter, one has to keep in mind that the following HSM can also be emulated or implemented partially or completely in software. For example, software only versions of the TPM BB which is described in the next chapter can be implemented and bound to a similar hardware service.

HSM usage is prevalent in all security fields as a way to usually offer at least one of the following kinds of services:

- Cryptographic operations offering integrity (which encompass both fidelity and authenticity properties) and confidentiality (secret-key and public-key ciphering)
- Secure storage of highly critical information through cryptographic safe services

Since the scope of this study is closely related to the embedded systems, we do not encompass large HSMs such as those designed by Thales for banks, EMC/Visa Cards services, or large web services providers. Their use can be made mandatory in order to design the infrastructure and notably the internet infrastructure dedicated to the devices developed in the context of the SAFURE framework. Nevertheless, this kind of HSM does not represent a new kind of technical or scientific challenge and are not studied here.

3.1.1 Trusted Platform Module (TPM)

The TPM is a small versatile solution whom specifications are standardized by the TCG (Trusted Computing Group) which defines both the hardware interfaces and mechanisms of the chip itself and the associated software stack (the Trusted Secure Stack TSS) and the business oriented solutions defined in the corresponding workgroups hosted by the TCG such as TNC (Trusted Network Connect) for network related uses.

Note: Thales, member of the SAFURE project, is also a member of the TCG and follows some of the different workgroups.

Although the TPM was initially designed at the beginning of this century with DRM issues in mind, its goals are now quite different and are aiming at platform integrity (which is by the way necessary in order to offer resilient DRMs). This unfortunately seriously slowed the adoption of the TSS and therefore made the presence of the TPM almost useless. Today, thanks to the cloud computing and to privacy concerns amongst PC users, the TPM adoptions gains traction.

In order to offer a pervasive solution worldwide, the TPM only offers essentially integrity services as follows:

- Platform integrity is implemented through the use of hashing functions over integrity seeds using a Merkle tree as data structure

- Cryptographic secrets storage is bound to the integrity references and only reveal cryptographic secrets if platform integrity is verified.

It is important to see that the TPM does not perform confidentiality operations such as ciphering due to two issues:

- the TPM is usually located on a low bandwidth bus which prevents it from performing efficient data ciphering
- Such behavior would have prevented US companies from exporting their products worldwide

TPM are supported by Windows (it is a key aspect of Microsoft's BitLocker software ciphering solution), by Linux (the TrouSerS library) and by PikeOS.

3.1.2 Secure Element

Secure elements (SEs) are usually found in mobility solutions performing full computations and key storage and hosting security-related applications. They are very similar to credit cards and usually belong to one of the following sets: Universal Integrated Circuit Card (UICC): typically the SIM card used in GSM (also called USIM for UMTS) mobile devices.

- embedded SE: they are usually soldered on the board and are either controlled by the manufacturer, the distributor or the end-user organisation.
- microSD: security devices integrated inserted into smartphones and only relevant to organisation owning the device.

Each set is designed for specific use. UICCs usually are the trusted link between the mobile device and the telephony operator provider. In case of litigation, between a mobile user they host the relevant technical proof. Embedded SEs are used on embedded dedicated devices. Either the embedded SE is designed to be totally managed by the organization which owns or manage the assets on the device, either the embedded SE acts as a trusted party in a multi-owners assets management. Embedded SE can be solutions can be the result of specialized IP present on the main SOC of the device and applications separated by the trusted zone offered by ARM processors. TPMs can be seen as a subset of SE Elements. MicroSDs are usually exclusively managed by the end-user organization which allows a separated and thus, easier control of the security of the device. They can be used in order to store large amounts of data, performs cryptographic operations such as VPM access, etc. Their additional cost is usually considered as a good trade-off between versatility and efficiency.

Although the difference between these three sets can be somehow artificial in terms of technologies, only UICC and microSD are removable and offer both versatility and adaptability through the use of hosted applications and multi-assets seclusion.

3.1.3 Secure Hardware Extension (SHE)

The Secure Hardware Extension (SHE) is a specification for an on-chip extension to any given microcontroller. It is intended to move the control over cryptographic keys from the software domain into the hardware domain and therefore protect those keys from software attacks. However, it is not meant to replace highly secure solutions like TPM chips or smart cards, i.e. no tamper resistance is required by the specification.

The main goals for the design at hand are:

- Protect cryptographic keys from software attacks.
- Provide an authentic software environment.
- Let the security only depend on the strength of the underlying algorithm and the confidentiality of the keys.
- Allow for distributed key ownerships.
- Keep the flexibility high and the costs low.

Basically SHE consists of three building blocks, a storage area to keep the cryptographic keys and additional corresponding information, a implementation of a block cipher (AES) and a control logic connecting the parts to the CPU of the microcontroller, see Figure 1 for a simplified block diagram.

SHE can be implemented in several ways, e.g. a finite state machine or a small, dedicated CPU core.

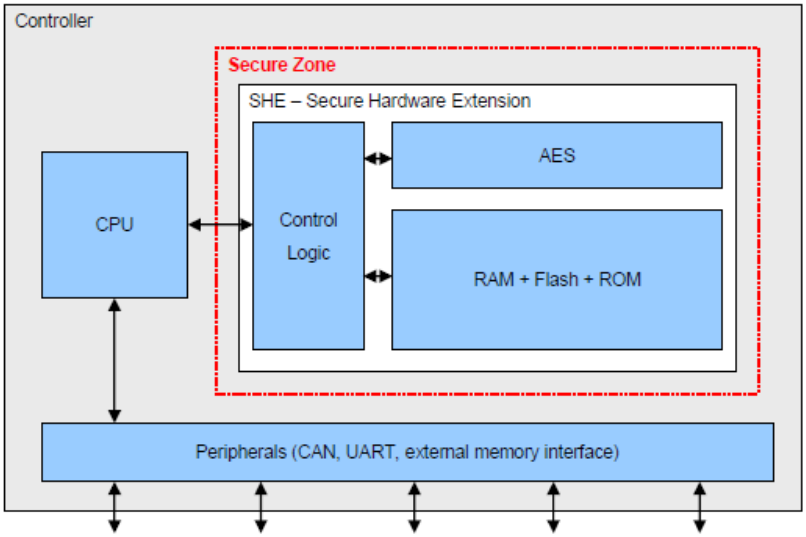

Figure 4: Simplified logical structure of SHE

HSM and CSE are an implementation of SHE device, and they are present in both the candidate microcontroller families selected for SAFURE Automotive Scenario.

Automotive SHE Implementation: HSM

The Hardware Security Module (HSM) is an optional peripheral module used in some powertrain microcontrollers, and its main applications are:

- Secure boot
- Tuning protection (e.g. integrity of calibration data)
- Secure sensor communication (authentication and integrity of sensor data)
- Authentication
- Secure flash load
- Immobilizer (theft protection)
- Secure log and
- Secure debug authentication

The HSM contains all necessary elements to allow software to implement the Secure Hardware Extension (SHE) version 1.1.

Automotive SHE Implementation: CSE

The Cryptographic Services Engine (CSE) is a peripheral module that implements the security functions described in the Secure Hardware Extension (SHE) Functional Specification Version 1.1.

The CSE design includes a host interface with a set of memory mapped registers that are used by the CPU to issue commands and a system bus interface that allows the CSE to directly access system memory.

Two dedicated blocks of system Flash memory are used by the CSE for secure key storage.

The CSE has the following features:

- Secure storage for cryptographic keys
- AES-128 encryption and decryption
- AES-128 CMAC authentication
- True random number generation
- Secure boot mode
- System bus master interface

Unlike HSM, CSE doesn't have a programmable microcontroller.

3.2 Separation Kernel

The Separation Kernel is a component that enforces the separation between partitions and information flow between partitions based on the security policy. The allocation and management of system resources are also implemented by the Separation Kernel. It is also one of the components of the MILS Trusted OS[2].

3.2.1 Functionality

The Separation Kernel guarantees separation and controlled information flow by enforcing the following security policies:

- *Resource allocation policy*: This policy defines how the system resources such as CPU time and main memory are allocated to the partitions. If some resources are shared between partitions, this policy defines how the resource sharing shall be done such that the separation between partitions is enforced. For example the Separation Kernel enforces spatial separation by allocating disjoint memory areas to the partitions and by controlling the memory accesses from the partitions. Similarly temporal separation between partitions is enforced by executing partitions in separate, non-overlapping time windows.
- *Access control policy*: An access control policy specifies the access rights of objects under the control of the Separation Kernel. The implicit information flow between partitions is defined by this policy. For example an access control policy might assign

[2] The MILS architecture template defined in Section 3.1 of MILS Architecture (EURO-MILS Report) is adopted to create the concrete MILS architecture of Trusted OS.

communication object C as writable to partition A and readable to partition B, thereby creating an implicit information flow from A to B.

- *Information flow policy*: The information flow policy defines how the partitions shall exchange information. This includes rules based on the sender/receiver of a message and the message content. The Separation Kernel enforces the information flow policy explicitly using the communication rights of partitions and implicitly by the access rights of objects.

3.2.2 Modules/Subcomponents

In a typical microkernel based separation kernel, the security functionality is split between a kernel and a user space component. The kernel component provides the mechanisms and the user space component implements the security policy making use of those mechanisms.

- *Microkernel*: The microkernel abstracts the CPU time, addresses space, exceptions and external interrupts. It provides the low level communication primitives (IPC) for the partitions to communicate with each other. The microkernel also provides the synchronization primitives.
- *User space System Software*: The user space system software makes use of the services/mechanisms provided by the microkernel to implement the security policy. It retrieves the security policy from the configuration provided by the system integrator and establishes the system configuration. This module is responsible for loading, starting, stopping and resource allocation of user partitions. It also provides sophisticated communication channels with controlled information flow built using the low level IPC and mapping primitives provided by the microkernel.

3.2.3 Provided Interfaces

- *Synchronization API*: For the user space synchronization, Trusted OS provides services like one time initializers, mutexes, condition variables, reader-writer locks, thread synchronization barriers, semaphores and spinlocks.
- *Communication API*: Trusted OS provides communication mechanisms to allow threads to exchange data. It also implements communication rights for the partitions to control the information flow between threads belonging to different partitions.
- *User space interrupt handling API*: Using this service, the handling of interrupts can be entirely managed from user space. A task's ability to handle an interrupt is also controlled by Trusted OS.
- *Task and Thread APIs*: Trusted OS provides abstraction of address spaces in the form of tasks and provides a set of schedulable entities bounded to a task called threads. The task APIs allow operations such as task creation, starting and terminating the task, retrieving and changing the attributes of a task. Thread APIs allow thread operations like creation and deletion of threads, retrieving and changing the state, CPU affinity and priority of threads.
- *Exception Handling API*: The Exception handling APIs allows certain exceptions like illegal memory access or floating point exceptions to be handled in user space.
- *Memory Management API*: The trusted OS provides services to modify the virtual address mapping of tasks.

3.2.4 Dependency

For implementing the separation between partitions, the Separation Kernel depends on the Separation Supporting Hardware which is a MILS core component. The following hardware components are used by the Separation Kernel:

- CPU with different privilege levels
- MMU
- IOMMU
- Hardware Timer
- Boot loader and initialization routine

The Separation Supporting Hardware shall control the interactions between the hardware components. For example the memory access from CPU is guarded by MMU and similarly the memory access from a DMA capable device is guarded by IOMMU. For implementing temporal separation between partitions that share the same hardware resource such as the CPU, the Separation Kernel depends on hardware timer. The bootloader is responsible for loading the Separation Kernel and initializing the hardware to create an environment where the Separation Kernel can start execution.

In a typical operating system, to make the Separation Kernel architecture independent, the core hardware components are abstracted using a software layer called the Board Support Package (BSP) or Platform Support Package (PSP). The Separation Kernel accesses the hardware by making use of the interfaces provided by the BSP.

The Separation Kernel also depends on the interface provided by the Audit Module for logging security events such as violation of communication rights or memory access permissions by the user partitions.

3.3 Secure Software Update

In this section, we will explain a process for secure software updates. Software updates are necessary to fix security and safety risks or flaws in the functionality of a device[3] after delivery. Adversaries might be motivated to manipulate a software update or issue their own update of the software. Therefore, a secure process to update devices in field needs to be defined. Although an over-the-air (OTA) scenario is presented, distribution via different communication paths, e.g. via a diagnostic interface, are possible. The essential steps to secure the integrity and authenticity of the software update remain the same.

The main idea is that the update is wrapped in an update container (cf. Figure 2) that contains the software update itself, a signature, and required parameters, e.g. applied algorithm, key length, ID of the signing key, etc. With all this information and a public verification key the vehicle can validate the correctness of the signature and therefore verify the authenticity of the software update. It is important that the public verification key is associated with the private signing key and that the public key is known by the device and stored in tamper resistance storage. The signature also ensures that no manipulated software can be installed on the device. That means even if the attacker obtains a software update container he cannot manipulate the software in a way that the device accepts the update.

[3] In the SAFURE context, a device can either be the ECU of a vehicle, a tablet or a smartphone as described in the use cases and scenarios in deliverable D1.1.

If, for example due to timing or hardware constraints, symmetric cryptography is preferred, a similar approach is possible. A MAC is calculated over the software update and the update itself, the MAC and necessary parameters are wrapped in an update container. The symmetric key which has been used to generate the MAC needs to be stored on the device in a secure way, i.e. in a way that ensures integrity, authenticity and confidentiality of the key material. Prior to installation, the device calculates the MAC of the received update and compares it to the MAC that has been sent in the update container.

For secure update processes based on asymmetric cryptography, RSA signatures or ECDSA are recommended. In a key size and parameters report published by ENISA in 2014 [1], recommendations for near term use (expected to be secure for at least the next ten years) and long term use (expected to be secure for thirty to fifty years) are given. For RSA signatures, key length of at least 3072 bit or 15360 bit for long term use shall be chosen. ECDSA can achieve a comparable level of security with shorter keys, i.e. 256 bits for near term use and 512 bit elliptic curves for long term use. For the symmetric approach, CMAC or HMAC based on AES-256 shall be chosen. For all algorithms, it is important to monitor recent publications of well-known standardization organization, e.g. NIST, ENISA [1], ECRYPT [2] or BSI, and keep the long-term security of a device during its expected lifetime in mind when choosing key length.

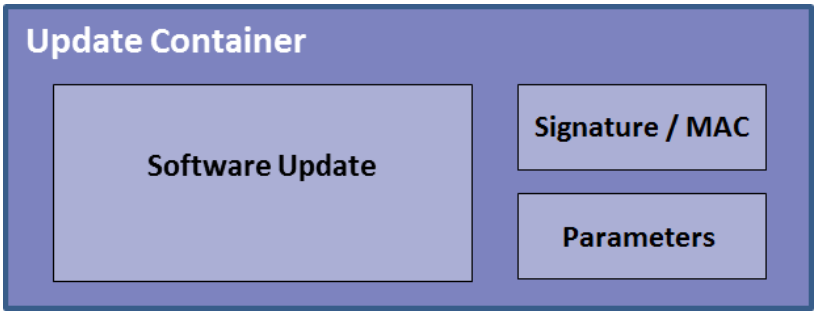

Figure 5: Structure of the Update Container

An exemplary update process is depicted in Figure 6. In the first step, the software developer generates the software update and an authorized employee sends a signature request to backend. In order to do this, the employee has to authenticate himself against the backend and the backend verifies the authorization of the request. Afterwards the hash value of the software update is transferred to the backend and a digital signature over the hash value is calculated at the backend using the private signature key. The signature is then sent back to the software development PC. In the next step, the software developer generates the update container consisting of the update itself, the signature, and other relevant parameters. This container is then sent to the OTA distribution system. When a device now initiates one of its scheduled secure connects to check for updates, the new update container can be downloaded. After successful verification of the signature, the update will be installed on the device.

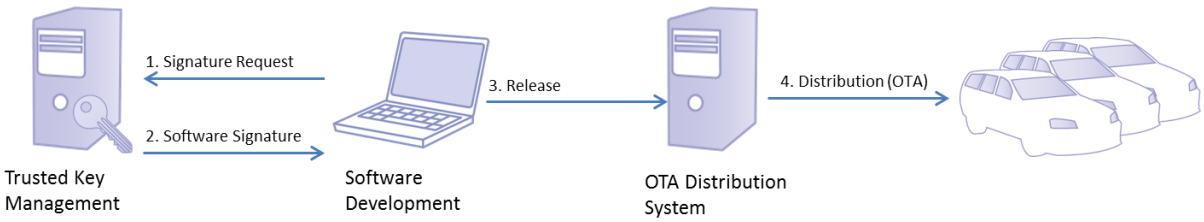

Figure 6: Signature generation and distribution process

3.4 Secure Boot

To detect manipulations of software on the device prior to execution, a secure boot mechanism is designed and implemented within the SAFURE framework.
It is especially important to protect the software which is loaded directly after start-up, i.e. the bootloader and the operating system. Otherwise, the security of the runtime environment cannot be guaranteed and integrated security mechanisms cannot be trusted.

The secure boot process implements a chain of trust. It needs to start with a trusted hardware component, such as a HSM[4]. The trusted hardware component is necessary to prevent manipulations of the security anchor of the chain of trust. If the integrity of this initial value cannot be guaranteed, the entire chain becomes unreliable. The software components that are required during the subsequent step of the boot process can be authenticated before execution. In the first step the integrity of the bootloader is verified by comparing digital signatures, e.g. RSA signatures. The verification process uses a public key that is pre-stored in a part of the internal memory which is protected against modification, e.g. in One-Time-Programmable (OTP) memory or in eFUSEs. If the verification of the signature fails, the event is logged and the boot process is aborted, cf. **Figure 7**. If this verification is successful, the operating system shall be verified using signatures. If one of these two steps is not successful, the boot process will be aborted. In the next step, the integrity and the authenticity of all critical data and executables on the microcontroller will be verified by the operating system. If in one of these critical files a manipulation is detected, an error messages is logged and the executables are shut down, otherwise the processor continuous with the boot process.

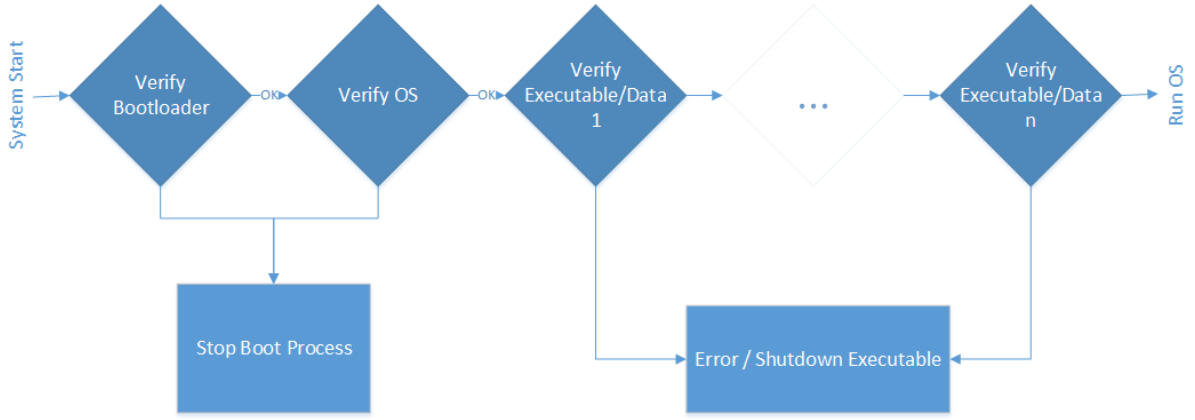

Figure 7: Secure Boot Process

The verification of the various data types requires different carefully designed concepts. During production, the hash of the bootloader is calculated by using a cryptographic secure hash function, e.g. SHA-256. Afterwards, the hash is signed, e.g. using RSA-4096, and a private signing key which is only used to sign the bootloader. The bootloader, signature of the hash value, and the associated public key will be stored inside the protected area of the processor memory. During the verification, the hash of the actual bootloader is calculated and the signature is verified using the public key. The result of the verification step is the hash value of the original bootloader. If the original and the calculated hash are identical, the bootloader has not been manipulated.
Also, the operating system is signed by a trusted party during production. The public signing key shall be included inside the bootloader and the bootloader calls the verification function.

[4] Please refer to Section 3.1 for an overview of currently available technologies.

To verify the integrity and authenticity of critical data and executables, i.e. software and libraries, it is important that file manipulations can be detected but also missing or additional files should be identified.

3.5 Secure Communication

Secure communication is the capability of having communication channels that cannot be intercepted by an entity external to the communication endpoints. Usually secure communications rely on cryptography to prevent interception. However for co-located, internal communications some lighter and more efficient means can also be envisioned.

3.5.1 Internal communication

Securing internal communication is needed when two or more security domains are used in the same equipment. Each domain has to be able to manage its own local communication channels in a way that ensures they are not redirected elsewhere and that no rogue application in another domain may have access to it. Moreover communication channels between security domains have to be handled as well. In the SAFURE framework we propose to handle security domains with Virtual Machines and to have the underlying Separation Kernel manage VM interconnection. Furthermore we propose to add explicit labelling to every inter-VM communication channel in order to be able to audit anytime that no communication channel is altered. These features can be added without impacting the Separation kernel so as to maintain its isolation properties.

3.5.2 External Communication (e.g. IPSEC)

Internet Protocol Security (IPSec) is an extension of the Internet Protocol (IP) aimed to set up a secure connection like host-to-host (transport mode), network-to-host (tunnel mode), or network-to-network (tunnel mode). IPSec uses several cryptography algorithms and provides data origin authentication, data integrity, data confidentiality, and replay protection. It modifies a real IP packet to hide or protect payload data as well as destination of the IP packet. It is a good candidate for integration as a service with the separation kernel. It allows to establish a safe and secure connection between virtual machines and external clients.

This section explains the process of enhancing safety and security in a deterministic network, such as TTEthernet, when cryptographic algorithms are used. For safety-critical systems SAE AS6802 specifies a fault-tolerant Multi-Master synchronization strategy, in which each component is configured either as Synchronization Master (SM), Synchronization Client (SC) or as Compression Master (CM), as shown in Figure 8.

Typically, the end systems would be configured as SM, while the central role of the CM suggests its realization in the switch in the computer network, though this is not mandatory.

All other components in the network are configured as SCs and only react passively to the synchronization strategy. The synchronization information is exchanged in Protocol Control Frames (PCFs).

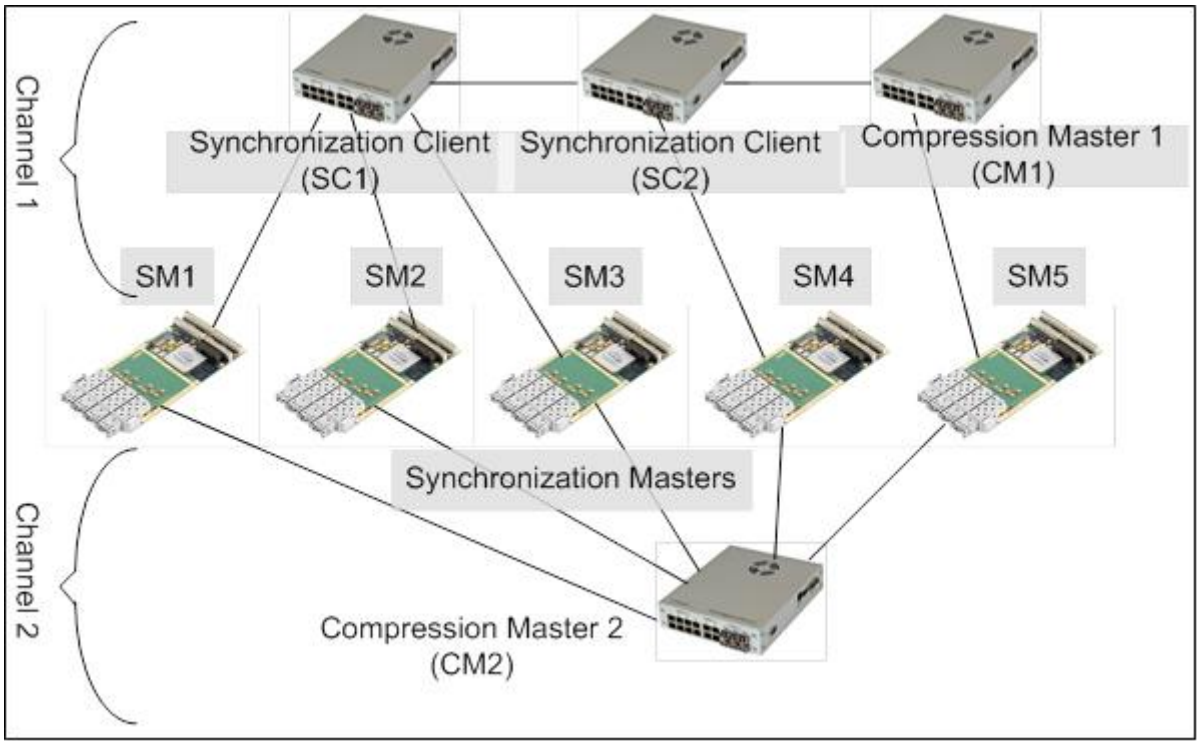

Figure 8: TTEthernet Network

In general, entities use a network to exchange the current values of their local clocks. In order to allow synchronization at all, the network must provide a time-preserving transmission service with known timing error. For example if the local clock values are exchanged by using a message-based transmission service, the transmission latency and transmission jitter need to be predictable and therefore deterministic. The quality of the transmission latency and jitter of the service typically also directly influence the quality of the synchronization, i.e., the smaller the latency and jitter the better the local clocks can be synchronized to each other.

TTEthernet can converge real-time controls traffic with regular best effort traffic on one Ethernet network. So far, the safety aspects have been covered by a time-scheduled traffic that is untangled from any other network traffic and is thus immune to disturbance. This means that in a Deterministic Ethernet network, latency of critical scheduled communication is guaranteed.

An existing approach for covering network security aspects has been the development of cryptographic algorithms. Different encryption algorithms are available for end-to-end security across TTEthernet networks, for instance, the Metadat Scrambler (MDS).

The Metadat Scrambler alters a binary string with finite length according to rules set by the user. It outputs a unique binary string in-situ, which is usually longer than the original sequence, and codes individually the Internet packers on the OSI Layer 2. This transformation is not invertible and guarantees that the decryption happens only at the End System/ Switches that has the key. MDS is a Hardware device and it works in a transparent fashion within an Ethernet/TTEthernet network. Below, Figure 9 shows a TTEthernet combining encrypted and unencrypted data transmission.

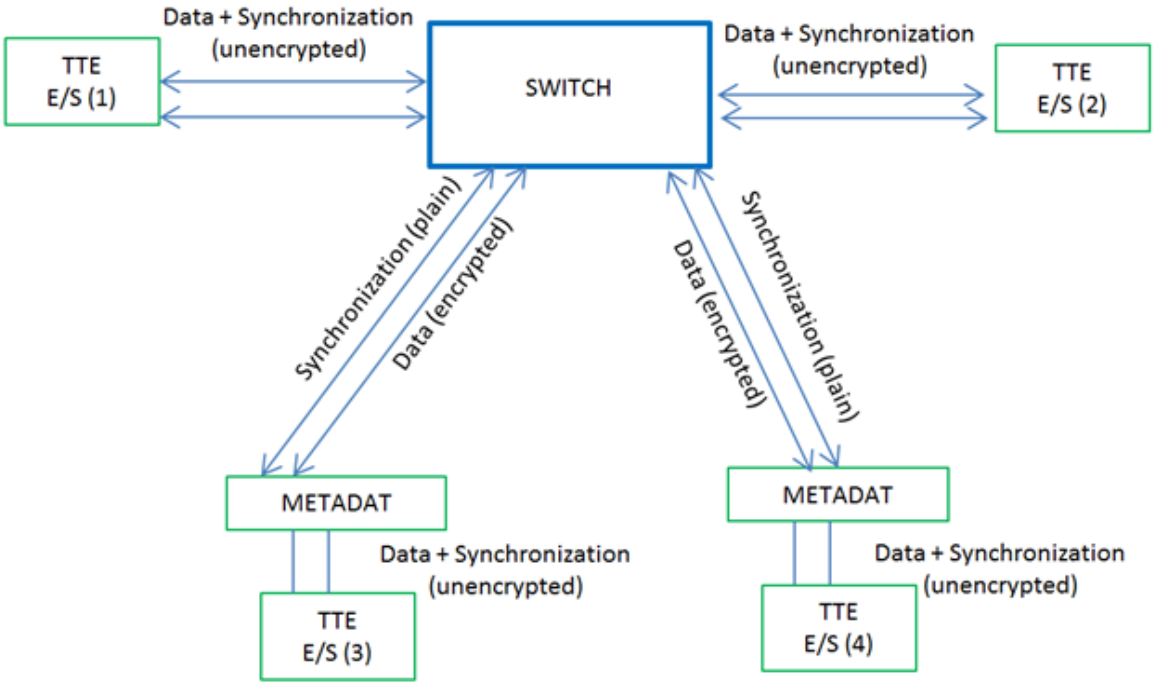

Figure 9: TTEthernet Network using METADAT Stream Cypher (MDSC).

3.6 Secure GUI

Secure GUI (telecom use-case) with following uses-cases:

- Multiple GUI seclusion
- Security dedicated GUI

Complex Graphical User Interface (GUI) often designed to ease communication among application without security protocol are becoming more and more subject to security threat. This is clearly manifest for mixed-criticality application when a single Man-Machine Interface (MMI) is used by two environment of heterogeneous criticality. Most notably a malicious application executed in one environment could be spying on other critical applications through unsecure GUI implementation that would unwittingly disclose information due to side-channel leak. For example password or other secret information during user authentication phase.

In particular, architecture where applications directly access hardware such as a display controller or keyboard controller shall be avoided. Instead, a compartmentalized GUI server providing a secure communication channel between the user and critical application can be placed between hardware controllers and applications or virtualized environments. However, the challenge is to be able to preserve performances and ergonomics while ensuring isolation between applications

Chapter 4 Modelling Approaches

This section is dedicated to modelling approaches for safety, security and time and their combination in the context of mixed criticality systems. The identification of the modeling approach requires the definition of three different levels of requirements.

4.1 Conceptual modeling

The first level pertains to the identification of the ***conceptual needs***, that is the concepts and elements that need to be expressed by the model. The modeling features should in general allow the definition of constraints and metrics that apply to the *functional model* of the system and refer to safety, security and time.

However, the modeling concepts for constraints and metrics need to be complemented with the modeling features that allow to describe the structure of the application functions for what pertains to the safety and security features.

The definition of the modeling for the system functionality is not enough. Modeling recommendations should be complemented with two additional layers.

One layer pertains to the definition of the execution platform, including the hardware components that may support or enable the management of safety and security features. Hardware implemented modules, such as HSM and CSM for security in automotive systems, but also basic fundamental components like watchdogs belong to this category.

Also, the platform modeling features should be able to identify all the issues that impact the time, safety and security properties, such as the placement and access of hardware (shared) resources, computation nodes and communication devices, as well as the need for expressing (replica) placements.

Finally, a fundamental part of the platform and functional modeling features is tha availability of predefined architecture patterns that can be reused (after adaptation) in system design while carrying some of their properties.

Examples of architecture-level patterns can be found within the platform, at the physical level, such as the HSM and CSM, but also at the software level, with the hypervisors, and at the functional level with primary-backup and multiple copies plus voter configurations.

The conceptual model is a needed starting point. It results in metamodeling definitions, similar to what has been done in several other EU projects, including SAFE and EVITA. For its construction, in WP2, we make use of the metamodeling features of the Eclipse environment (Ecore).

4.2 Modeling Languages and Standards

The conceptual definition, however, is often not readily usable, because of the limited support and availability of tools for a set of custom definitions in Ecore. Also, a requirement for the modeling approach is that it is compliant with the main industrial modeling languages and standards.

Among the languages and standards to be considered in SAFURE, of particular importance are:

- Eclipse and Ecore
- UML and SysML
- AUTOSAR and EAST-AADL
- ISO26262 (standard only)

The conceptual modeling features identified in the previous step need to be recast or expressed in a way that is compliant, to the largest possible extent, with the languages and the standard recommendations of the previous set.

This is not always possible, and the size and time availability of the SAFURE project prevents an exhaustive analysis.

This is why the mapping of the conceptual model onto an actual modeling language is restricted to UML and AUTOSAR.

In addition, UML is easily extensible, at least for its structural part, by means of profiles and stereotypes. The semantics will be necessarily specified informally or as part of the textual description.

AUTOSAR is in general not open for extensions, and any formal addition or modification should go through the standardization groups, which makes it impractical for the timeframe of the project.

However, in SAFURE, we make use of an AUTOSAR modeling tool that inherits from UML and SysML and therefore allows all the extension mechanisms of UML.

4.3 State of the Art

The state of the art for the modeling of features related to time, safety and security is potentially huge, since it includes not only work on the modeling itself, but also on the analysis methodologies and the definition of features and constraints.

As part of the state of the art, we need to consider the academic and professional literature on the subject, drawing from multiple domains and conferences.

Examples of contribution coming from conferences and journals are: for modeling, the MODELS conference and the transactions on SW Engineering, for timing, the Real-Time System Symposium and the Real-Time and Embedded Application Symposium conferences, the ACM transactions on Embedded Systems Journal and the Real-Time Systems Journal, for Safety, the International Conference on Computer Safety, Reliability and Security(SAFECOMP), the European Dependable Computing Conference (EDCC), the IEEE High Assurance Systems Engineering Symposium (HASE), and the IEEE Transactions on Dependable and Secure Computing and the International Journal of Critical Computer-based Systems. Finally, for Security,

Next, we need to consider the contribution of international standards. Among those, the AUTOSAR automotive standard is considered with its standard modeling features, but especially for the extensions that apply to time modeling (AUTOSAR Timing Extensions and AUTOSAR_TR_TimingAnalysis), to security (AUTOSAR SWS SecureOnboardCommunication) and safety (AUTOSAR_TPS_SafetyExtensions).

Finally, the last contribution to the state of the art comes from other European, international and national project - among those, the EVITA and SAFE projects.

4.4 Toolset

We have identified three toolsets to capture the modeling aspects related to time, safety and security and express them in the context of mixed criticality behavior on system and components level. They are:

- IBM Rhapsody, for the AUTOSAR, UML, and SysML modeling of the automotive applications in SAFURE.
- Arcadia, for the modeling of Communications systems
- Enterprise Architect for the UML modeling

4.4.1 IBM Rhapsody

IBM Rational Rhapsody is a modeling tool supporting several modeling languages such as UML, SysML and enabling C, C++, Java, Ada and C# code generation.
Rhapsody supports the AUTOSAR modeling as a specialization of UML [3], meaning that every AUTOSAR modeling element has been defined as a UML stereotype. Profiles, stereotypes and tags are extensibility mechanisms that extend the UML metamodeling elements with specific properties and they are often used to describe objects in particular domains [4].
Example automotive systems modeled in AUTOSAR are also available from the partners in Rhapsody. These will be used as proof-of-concept for the use of the recommended additional modeling features developed in WP2.
The possibility to define new user defined stereotypes is available in Rhapsody and will be leveraged in SAFURE to provide stereotypes and architecture templates/patterns. These will formally extend the set of UML modeling elements, but since AUTOSAR meta-classes are defined on UML, they will also be used to verify the inclusion of the recommended modeling features in AUTOSAR designs.

The AUTOSAR models developed in Rhapsody will be available in a number of formats. Besides the typical ARXML output, the models are available in the OMG standard XMI format (the modeling extensions will be made available as XMI exports).

4.4.2 Arcadia (e.g. Capella)

ARCADIA is a system & software architecture engineering method, based on model-driven engineering activities. It targets systems whose architecture is largely constrained by issues such as performance, safety and security [10].

In particular this method adopts a multi-viewpoint approach. A viewpoint defined how to represent the whole system from a perspective of related set of concerns. The major engineering concerns of projects targeted by the SAFURE framework are Security and Safety in a context of mixed-criticality applications.

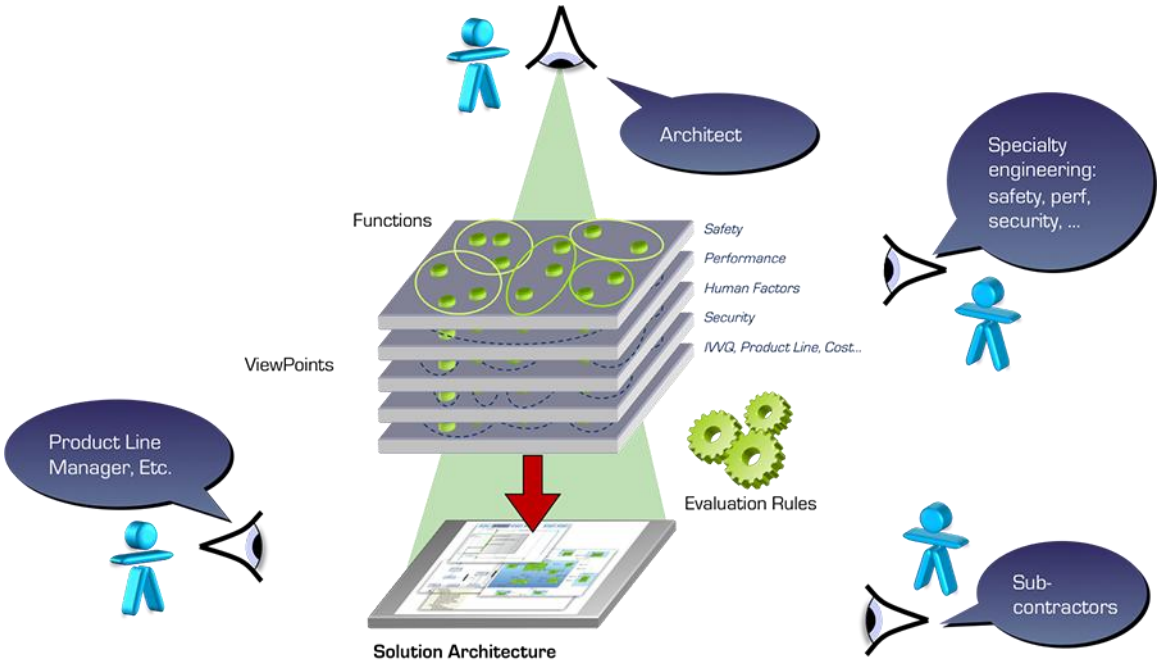

Figure 10: Using view point with Capella

The Capella workbench is Eclipse application providing a domain-specific language implementing Arcadia, both as a simplification and a semantic enrichment of the UML and SysML [11].

During the design phase of the telecom use case prototype, this tool can be used to model safety and security constraints using viewpoints.

Modeling tools, technique and even pattern are part of the SAFURE Framework. For example, a generic model for multi-level endpoint in a multi-level system (personal vs. medical container/environment in the devices and personal vs. medial server/services in the Internet/cloud)…

Nevertheless, there is one main difference between the mobile platforms identified in this project and other equipment present in the consolidated IT systems (pharmacy IT system, medical device, hospital, private network, etc.). The mobile platform is the only one which is not dedicated to one IT system and must address the whole set of IT system in a secure yet interoperable way. Therefore the platform must exhibit the usual properties of multi-level systems. This is clearly a challenge since the platform hardware and legacy software are not designed to allow such use.

4.4.3 Enterprise Architect

Enterprise Architect is a visual modelling and design tool based on UML. It supports all types of UML diagrams and can thus be used to model a complete hardware/software system, including all sub-components, interfaces, activities, and sequences.

Enterprise Architect does not provide modules supporting the modelling approach chosen in SAFURE; however it provides all means to implement those.

Chapter 5 Analysis Methods and Tools

In this section we describe the analysis methods and tools used as an integral part of the SAFURE framework.

5.1 Time

Time is an important factor, both for safety and security analysis. In safety-critical applications, timely responses of the system are often a non-functional requirement and part of the safety of the system. Regarding security, timing of a system can be affected by e.g. denial-of-service attacks (e.g. flooding) and also used as a side-channel for other attacks.

In this section, model-based timing analysis is described, as offered by the tool SymTA/S. With this approach, timing of a system can be understood, verified and optimized. The term “system” includes both single- and multi-core processor systems or ECUs, and also networks (e.g. CAN, FlexRay, Ethernet) and whole distributed systems (including ECUs and networks).

Analogously, the approach to timing analysis for on-chip shared resources in multicores (e.g., on-chip interconnects, shared caches and shared memory controllers) is also described in this section.

5.1.1 Timing Model for Model-Based Timing Analysis

The analysis works on an abstract timing model. The model consists mainly of the following entities (using the terminology of the SymTA/S tool, which is mostly aligned with automotive terms):

- Hardware Structure
  - ECUs, Buses, Switch Ports
  - Processing Units (Cores and Memories)
  - Topology
- Software Structure
  - Software Components
  - Tasks
  - Functions (Runnables)
  - Shared Variables
- Communication Structure
  - PDUs
  - Frames

Each component of the system model has key temporal properties/configuration (e.g. execution time of tasks and runnables, length of frames, activation patterns of tasks and frames, scheduling or arbitration policy of cores and buses, etc.). This information can be obtained from configuration (e.g. task or frame periods), measurement (e.g. by tracing the

actual implementation), by analysis (e.g. WCET[5] analysis of tasks) or by estimations (especially in early phases of development).

Specifically, timing model specifies the topology and architecture of the system, e.g. the available cores, tasks, runnables and their respective mapping (runnable-to-task and task-to-core) and configuration (e.g. task priorities, runnable repetition factors); for networks the available busses, switches, links and frames (with their respective mapping). In terms of timing, cores (or busses) provide services (i.e. processing/communication time) that are consumed by tasks executing on the cores (or frames transmitted over the buses).

Activation models describe, how often these are ready for execution/transmission, e.g. periodically every 10ms. This information can be derived from configuration (for periodic tasks), from a behavioural description (for external interrupts) or by the analysis of "triggering" tasks (for task chaining, modelled using triggers).

Furthermore, tasks and runnables specify how much time they require for each execution (min. and max. execution time). Also, for each task and/or runnable, access to shared variables are specified, i.e. which variables are accessed, how often they are accessed, and how (e.g. read/write, but also directly or via the RTE). Likewise, minimum/maximum size of frames and signals is specified.

Note that the input model can be manually created, imported from other models (e.g. AUTOSAR), or generated from a scheduling trace of the system (containing task activations, terminations etc.).

5.1.2 Model-Based Timing Analysis

The timing analysis works on the described model to derive the timing of the system considering the specific protocols that resolve concurrent resource access. For an ECU system, this is the scheduling policy of the operating system.

In SymTA/S, the scheduling analysis can be performed in two ways: system distribution analysis and formal worst-case analysis.

- The **system distribution analysis** basically performs a Monte-Carlo timing simulation: The system (e.g. the OS scheduler) is simulated, and during the simulation, system properties that are not specified to exact values are randomized, such as the execution time of tasks or size of frames (between min. and max. specified in the model), or the "offset" or phase between unsynchronized events (such as periodic timer tick and external interrupt). This is done many times with different random seeds, resulting in a distribution of the system behaviour. Results of the system distribution analysis include average core/bus loads, histograms of task activation distances, task response times and frame latencies.
- The **worst-case analysis** uses a mathematical theory [14] (compositional performance analysis) to compute lower and upper bounds on the system's timing behaviour, esp. task/frame response times. For this, worst-case (or "critical instant") scenarios are derived for individual tasks/frames by deriving a worst-case occurrence and alignment of events that maximizes the interference on a task/frame (e.g. blocking and/or pre-emption of higher-priority tasks/frames). This is done locally for each task/frame. From this analysis, event models of dependent tasks/frames (e.g. due to event triggering) can be derived iteratively to compute the response time bounds for the complete system. For each scheduling technology (e.g. static priority task scheduling, CAN scheduling, Ethernet AVB scheduling), a mathematical formulation of the scheduler is required for the worst-case analysis.

[5] WCET = Worst-Case Execution Time

To consider multi-core systems, the architectural effects of shared resources (e.g. main memory) must be considered during both analysis types. On the abstraction level used for SymTA/S, these effects are considered as access time overheads for accessing shared memories. During simulation, these additional delays can be considered directly, but the worst-case analysis requires more effort. Here, the max. number of accesses to shared resources require a dedicated analysis, which is interdependent with the response time analysis, because resource access depends on the actual scheduling of the tasks [15].

To consider new communication technologies, also both analysis types need to be extended. The system-distribution needs to be extended to cover the arbitration protocol of the new technology during the simulation. The worst-case analysis requires new formalism to find an upper bound on the timing [16]. A key focus of SAFURE will be to research and extend these algorithms and make them applicable to real-world scenarios.

With the timing analysis component of the SAFURE framework, also timing effects on security can be researched, such as the potential of a denial-of-service attack on shared resources.

5.1.3 Timing Analysis using Microbenchmarks

The analysis of on-chip inter-task interferences in multicores will be performed by means of architectural-dependent microbenchmarks that will be developed and suited specifically for the hardware architecture under analysis. Those microbenchmarks will be either programmed in C and compiled in a controlled manner to ensure they behave as expected or programmed in assembly.

The purpose of those microbenchmarks is triggering scenarios with high contention (including the worst contention) in on-chip shared resources such as shared interconnects and shared memory controllers. By running smartly those microbenchmarks on bare metal we will be able to quantify the impact in execution time of contention in shared hardware resources, as well as the impact that different applications can suffer when running in the particular multicore due to contention in shared resources due to inter-task interferences. Later, those microbenchmarks are intended to be ported on top of the real-time operating system (RTOS) and/or corresponding hypervisor so that they can be used by end users on top of the final platform once it is up and running.

So far somehow similar microbenchmarks have been developed for other platforms [17]. However, benchmarks triggering the specific timing behaviour pursued on top of the specific multicore architecture targeted have not been developed yet. Thus, they need to be developed from scratch.

5.2 Energy and Temperature

Temperature is an important factor since it is an important design constraint in modern processing platforms. Energy is also of vital importance for battery powered processing platforms. As indicated in Chapter 2, both of these mediums have critical safety and security implications.

5.2.1 Temperature Model

There are several ways of accurately modelling system temperature. Each of these approaches has separate set of tradeoffs. On one end of the spectrum we have so called lumped models which approximate the entire processor as a point source of heat. The high level of abstraction makes such models computationally fast on one hand, and erroneous on the other. The other end of the spectrum consists of fine-grained numerical simulators, such as Hotspot. These simulators require lot of details to be known about the processor (e.g.,

exact floor-plan, power trace for each micro-architectural unit inside the processor, cooling model to name a few) and are considered to be accurate. However, numerical simulators are computationally intensive, and computing each new temperature data point requires solving large number of system equations. A third set includes calibration based thermal models which run a series of calibration experiments on a target platform of interest [8]. These models have been shown to be accurate and computationally non-intensive. However, a given model is specific to a target platform and a set of target applications.

5.2.2 Temperature analysis

In multi-core systems, executing a task on one core has effects on the temperature of the entire platform (including other cores). These inter-core thermal interactions, and high temperature conditions in general, pose safety and security concerns. To counter these risks, analysis methods will be developed that quantify the thermal interactions of running activities with different safety and security requirements on multi-core platforms. The methods will take into account information about the mapping of tasks to resources, the used scheduling policies on processing elements and shared resources, the thermal properties of the tasks, and the thermal properties of the platform. To aid the analysis methods, a thermal model for multi-core platforms will be developed. The analysis methods will help to better understand the effects of temperature and temperature controlling mechanisms (e.g. DVS) on the timing properties of tasks in safety and security-aware systems.

5.3 Taint analysis

Taint analysis is a technique to increase security and some aspects of safety of system designs. The technique is focused on how a system input (that is often called source and often comes from untrusted user) is propagated through system to a specific destination (that is often called sink).

The main idea behind this analysis that input can be provided by an untrusted user and processing this input is a security and safety risk. In software the input is represented as input variables and how these input variables affect, i.e. *taint* internal variables, control flow, and system state. The typical example where this technique is used is to detect and prevent SQL injection when accessing databases.

Taint analysis can be also applied to any software system, which interacts with a potentially malicious user. The canonical use-case is an operating system with a user being an application executed on top of the considered operating system.

The taint analysis can be done manually or with a tool support. The manual analysis is too error prone and too expensive, thus, we do not consider it here. For efficient, trustworthy and repeatable results a tool support is required. While this topic is not new, there exist just few tools with strong limitations on the representation of analyzed object, i.e. description language. Another critical topic is user-guidance of the tool, i.e. how much annotation or tool specific information a user has to create.

For the sake of simplicity, every time we say taint analysis, we mean a tool supported taint analysis. The object representation for analysis can be split into two categories: abstract model and programming code.

The checking of taint analysis on abstract model has an advantage that the tool can be very efficient and needed tool guidance can be added directly to the modeling blocks as parameters. The disadvantage is that an abstract information flow is checked that can have some side effects and additional interferences when a real implementation (i.e. a refined model) is considered. This is a typical problem because security and safety properties do not

hold under generic refinement. Thus, while one wins effectiveness, one has losses in precision with respect to the real model.

A taint analysis carried out directly on the source code has an obvious advantage of working directly on the implementation and the ability to consider CPU architecture and/or hardware specifics. The disadvantage is that the complexity of language semantics, e.g. C language, and hardware related code (e.g. assembly) make analysis very hard and sometimes hardly feasible. Thus, some sort of pre-processing and code annotations is needed.

Taint analysis can cover the following security aspects: processing of untrusted user input, data and control split between for user data, dependencies of system state on user data, internal information flows due to the provided user data.

Taint analysis can cover the following safety aspects: robustness of system calls, dependencies on input for computational paths and worst case time execution.

The taint analysis can be used to create assurance arguments for requirements from ISO 26262, DO-178C, and Common Criteria (ADV_ARC).

5.4 Security Analysis

A security analysis is a structured approach to analyze the security of a system and derive security requirements that need to be implemented in order to prevent attacks.

In the first step, the system model is described, including all entities, components, objects and interfaces. This can be done using UML component diagrams (e.g. with Enterprise Architect, cf. Section 4.4.3).

Then, all relevant use cases are described. The description shall include actors, pre- and post-conditions, dependencies, default actions, and alternative actions (e.g. in case of an error). This can also be modeled using UML sequence diagrams.

In the next step, the security objectives are derived. For every use case, a high-level description of the security assets (e.g., data, functions, and services), potential attackers and security threats is given. If available, Common Criteria Protection Profiles can be applied in this step.

Then, the threat analysis is conducted, which includes the security environment (facts, measures, and assumptions), the attacker model, and an identification of feasible threats. The result is an attack tree for every security objective with every leaf corresponding to a threat. Figure 11 shows an example attack tree for an attack on the confidentiality of gateway software. It is the attacker's aim to extract software from the gateway of a vehicle. This aim is presented in the root of the tree. On the next level of node, three potential ways to achieve this aim are shown: The adversary could extract the software via a physical or logical interface or obtain the source code from an insider. For the first two options, sub-steps are listed in the leaves of the attack tree. The OR node indicated that either of the child node can be executed to achieve the aim stated in the parent node. It is also possible to use AND node in an attack tree if several sub-steps are required to achieve the parent node.

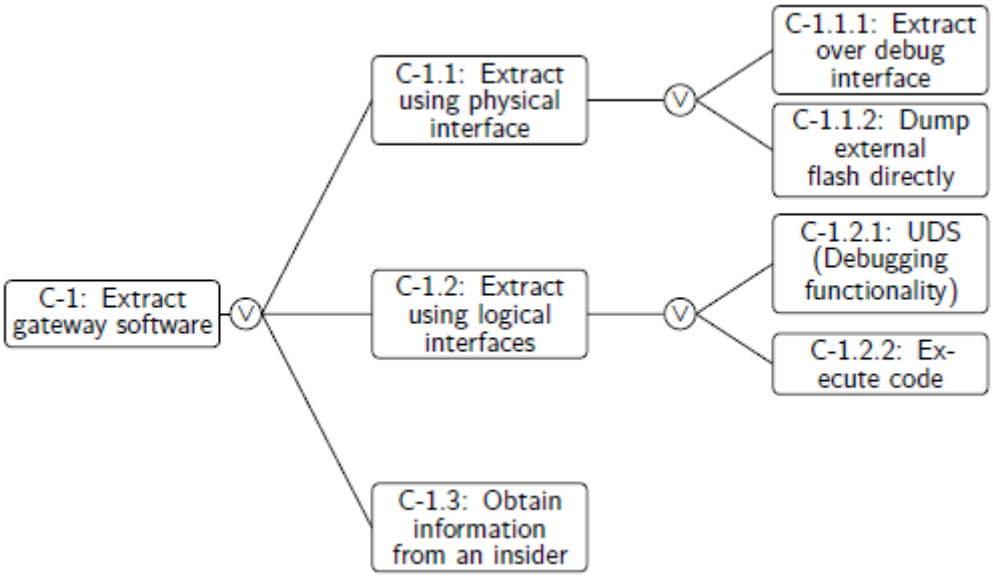

Figure 11: Example Attack Tree

In the next step, a risk analysis is executed in order to assess the risk for each threat. Therefore, the attack potential (AP) and the damage potential (DP) are evaluated. The attack potential is an estimation of resources required to successfully mount an attack in terms of elapsed time, specialist expertise, target knowledge, access perimeters, and technical equipment. A low attack potential means that the attack is easy to conduct and thus has a high probability to be chosen by an attacker. Therefore, the probability of an attack is reciprocal to the attack potential. Analogously, the damage potential is an estimation of the safety, financial and operational damage caused by the attack. The risk can then be calculated as follows:

Risk = Probability of attack x Expected Damage

The risk assessment matrix shown in Figure 12 helps to classify risks and to identify attacks with inacceptable or undesirable risks which need to be considered with a high priority in a security concept.

AP↓	**Risk assessment** = Adapted 6x4 risk matrix from railway safety engineering [EN50126]			
Basic	Undesirable	Inacceptable	Inacceptable	Inacceptable
Enhanced Basic	Tolerable	Undesirable	Inacceptable	Inacceptable
Moderate	Tolerable	Undesirable	Inacceptable	Inacceptable
High	Negligible	Tolerable	Undesirable	Inacceptable
Beyond High	Negligible	Negligible	Tolerable	Inacceptable
	Negligible	Negligible	Negligible	Undesirable
DP →	Insignificant	Medium	Critical	Catastrophic

Figure 12: Risk Assessment Matrix based on EN50126

Finally, security requirements can be derived from the attack paths with the highest risk. For each of these, countermeasures are defined in order to prevent the attack. This includes functional security requirement, security properties, security policies, and organizational security requirements. The result is a set of technical and organizational security requirements that can prevent threats with high risks.

5.5 Safety analysis in automotive systems

5.5.1 Overview

In automotive systems, Safety Analysis is based on the ISO 26262 standard. The standard consists of 9 normative parts and a guideline for the ISO 26262 as the 10th part. Here some extracts from ISO 26262 are provided, that describe different Safety Analysis area covered by ISO-26262, for details please refer to the full ISO 26262 documentation.

ISO 26262 is intended to be applied to safety-related systems that include one or more E/E systems and that are installed in series production passenger cars with a max gross weight up to 3,5 t. ISO 26262 does not address unique E/E systems in special purpose vehicles such as vehicles designed for drivers with disabilities. Systems developed prior to the publication date of ISO 26262 are exempted from the scope.

ISO 26262 addresses possible hazards caused by malfunctioning behavior of E/E safety-related systems including interaction of these systems. It does not address hazards as electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, release of energy, and similar hazards unless directly caused by malfunctioning behavior of E/E safety-related systems.

ISO 26262 does not address the nominal performance of E/E systems, even if dedicated functional performance standards exist for these systems (for example active and passive safety systems, brake systems, ACC).

Shortly, ISO 26262:

- Provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and supports tailoring the necessary activities during these lifecycle phases;
- Provides an automotive specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs);
- Uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk; and provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

ISO 26262 gives requirements and guidelines for Safety Analysis at:

- Product Development: System Level
- Hardware Level
- Software Level

5.5.2 ASIL Levels

ASIL is one of four levels used to specify the item's or element's necessary requirements of ISO 26262 and safety measures for avoiding an unreasonable residual risk with **D** representing the most stringent and **A** the least stringent level.

5.5.3 Freedom from interference by software partitioning

Iso-26262 provides design guidelines to accomplish at SW level “Freedom of Interference”, thus allowing to avoid that safety-relevant components and data could be corrupted by non-ASIL components.

Automotive Scenario related to SAFURE Project will implement protection mechanisms aligned to these ISO guidelines.

The following text is taken from ISO-26262 and is about **Freedom of Interference**, **Appendix D of ISO-26262 norm:**

Objectives

The objective is to prevent propagation of a failure in one software partition to another software partition.

NOTE: Errors in the state of the executing software can occur due to systematic software faults or due to random as well as systematic hardware faults. Such errors in one partition could disturb the operation of other software partitions either due to shared resources or due to error propagation.

General

D.2.1 *Software partitioning allows the co-existence of software partitions that use the same resources. It allows*

a) software components to be free from interference from other software components; and
NOTE: Different software partitions can be assigned different values of ASIL or a value of QM (see ISO 26262-9:—, Clause 5).
b) changes to be made to one software partition without the need to re-verify the unmodified software partitions.

Impact on system and software design
Depending on the system architecture, two approaches can be used:
c) several software partitions within a single microcontroller (see Figure D.1) with shared resources such as CPU time, memory, I/O-devices; and

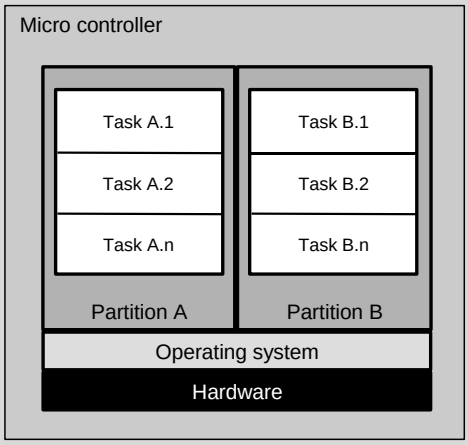

Figure 13: Several software partitions within a single microcontroller

d) several software partitions within the scope of a micro controller network (see Figures D.2 and D.3) with shared resources such as I/O-devices, especially internal and external data buses.

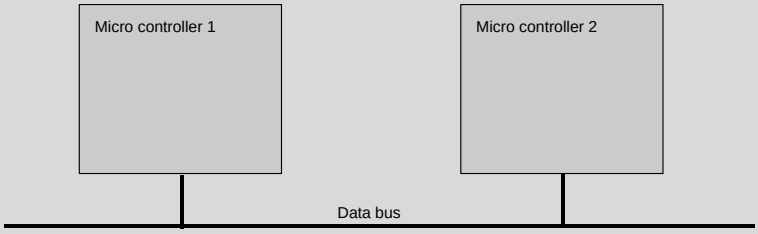

Figure 14: Several partitions within the scope of a micro controller network

NOTE: The micro controller network can consist of several processors in a single electronic control unit communicating via an internal data bus (intra processor communication). This is illustrated in figure D.3.

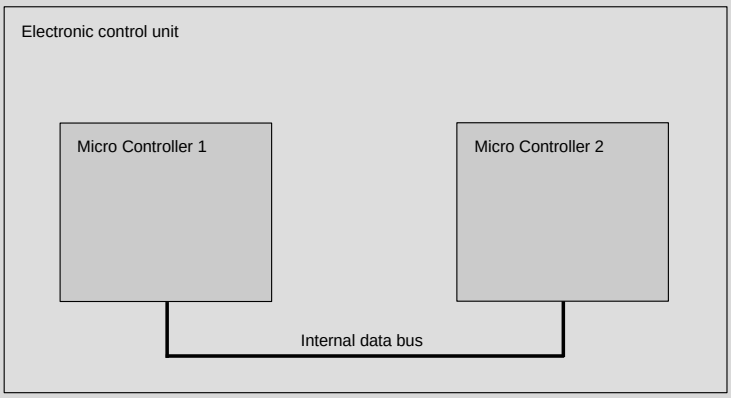

Figure 15: Several partitions within the scope of a multi-processor electronic control unit

Software components are executed within their respective software partition on their respective microcontroller as illustrated in Figures D.2 and D.3.

Impact on shared resources

Software partitioning requires adequate support by system resources.

In order to isolate multiple software partitions in a shared resource environment, the hardware has to provide the operating system with the ability to restrict access to shared resources for each software partition.

CPU time

To ensure freedom from interference of software partitions within a single microcontroller, the fault effects

- *blocking of partitions due to communication deadlocks; and*
- *wrong allocation of processor execution time*

are to be prevented by:

e) time triggered scheduling;
NOTE 1: Software partitions are considered coequally in allocating processor execution time and the same priority is assigned to all of them.
NOTE 2: Regarding the allocation of processor time, spare time is allocated in each processing cycle because of incoming interrupts.
NOTE 3: Time triggered scheduling is considered to have effectiveness "high" against protection against the fault effect "wrong processor execution time".

f) cycling execution scheduling policy;
NOTE 4: The time triggered scheduling method specifies a scheduling algorithm based on a predetermined fixed schedule, repetitive with a fixed periodicity.
NOTE 5: Using the time triggered scheduling method the allocation of processor execution time takes place through a static allocation table. Thus, for each task, a fixed point in time is predetermined for activating the task. Usage of time triggered scheduling method precludes priority-based scheduling.

g) fixed priority based scheduling;

h) monitoring of processor execution time of software partitions in accordance with the allocation;
NOTE 6: Monitoring of each partition by software checks if all partitions are executed in conformance with the predefined static allocation table.

i) program sequence monitoring;
NOTE 7: Program sequence monitoring is based on a hardware device (see ISO 26262-5:—, Table D.10).

j) arrival rate monitoring.
NOTE 8: Monitoring of processor execution is an additive method of Program sequence monitoring. If both methods are combined the effectiveness is "high" protection against the fault effect "wrong processor execution time".

Memory

To ensure freedom from interference of software partitions within a single microcontroller, the fault effect memory corruption due to unintended writing to memory of another partition

is to be prevented by:

k) memory protection mechanisms;
NOTE 1: The memory protection mechanisms refer to processors with Memory Management Unit or Memory Protection Unit.
NOTE 2: A Memory Management Unit enables the concept of virtual address space. This prevents a task of one partition corrupting the memory space of another task by unintended writing into that memory space, since every partition has its own address space.
NOTE 3: Usage of a Memory Management Unit requires support of the operating system.
NOTE 4: Provisions are made that the Memory Management Unit cannot be ignored, i.e. tasks are executed in a so-called user mode and the real addressing mode is not to be used.

l) verification of safety-related data;
NOTE 5: RAM locations containing safety-related data are verified by additional methods. This can be accomplished for example by using parity bits, Error Correcting/Correction Code (ECC), Cyclic Redundancy Checksum (CRC) or redundant storage.
NOTE 6: The effectiveness of these methods depends very heavily on the verification quality.
NOTE 7: Verification of safety-related data is done at run time.

m) offline analysis of code and data of other partitions;

n) restricted access to memory;

o) static analysis; and
NOTE 8: Static analysis methods defined in Table 10 can be used for reviewing pieces of code that access memory locations containing safety-related data.

p) static allocation.
NOTE 9: Static allocation means that resources are allocated statically during initialisation.

I/O-devices (communication)

To ensure freedom from interference of software partitions in communication microcontrollers, the fault effects

- *loss of peer to peer communication;*
- *unintended message repetition due to the same message being unintentionally sent again;*
- *message loss during transmission;*
- *insertion of messages due to receiver unintentionally receiving an additional message, which is interpreted to have correct source and destination addresses;*
- *re-sequencing due to the order of the data being changed during transmission, i.e. the data is not received in the same order as in which it was been sent;*
- *message corruption due to one or more data bits in the message being changed during transmission;*
- *message delay due to the message being received correctly, but not in time;*

- *blocking access to data bus due to a faulty node not adhering to the expected patterns of use and making excessive demands for service, thereby reducing its availability to other nodes, e.g. while wrongly waiting for non existing data; and*
- *constant transmission of messages by a faulty node, thereby compromising the operation of the entire bus.*

are to be prevented by:

q) identifier for communication objects;

r) keep alive messages;
NOTE 1: Keep alive messages is considered "high" effectiveness for detection of "Failure of communication peer".

s) alive counter;
NOTE 2: Alive counter is considered "high" protection against "Unintended message repetition" and "medium" protection against "Message loss", "Insertion of messages" and "Constant transmission of messages".

t) sequence number;
NOTE 3: Sequence number is considered "high" protection against "Unintended message repetition", "Message loss", "Insertion of messages", "Re-sequencing" and "Medium" protection against "Constant transmission of messages".

u) error detection codes;
NOTE 4: Cyclic Redundancy Checks are used as error detection codes if the residual error rate of the CRC implemented in the bus system is considered not to be sufficient. In this case an additional CRC at the application level is recommended.
NOTE 5: Alive Counter and CRC are transmitted (embedded in the frame for instance) and checked by the receiver.

v) error correction code;

w) message repetition;
NOTE 6: Message repetition is considered "high" protection against "Message loss", "Medium" protection against "Re-sequencing", and "Message corruption".

x) loop back;

y) acknowledge;
NOTE 7: Acknowledge is considered "high" effectiveness protection against the fault effect "Wrong communication peer".

z) separated point-to-point unidirectional communication objects;
NOTE 8: Exactly two uni-directional communication objects are used between two partitions respectively for data exchange.
NOTE 9: Method j) is considered "Medium" effectiveness protection against the fault effect "Wrong communication peer".

aa) unambiguous bidirectional communication object;
NOTE 10: Unambiguous bidirectional communication object uses unique numbers for identifying communication peers and/or acknowledges receipt of messages by the communication peer.
NOTE 11: Method k) is considered "medium" effectiveness protection against the fault effect "Wrong communication peer".

bb) asynchronous data communication; and
NOTE 12: In using asynchronous data communication there is no waiting state completed by the communication itself.
NOTE 13: Method l) is considered "high" effectiveness protection against the fault effect "Blocking of partitions".

cc) synchronous data communication.
NOTE 14: Access is synchronised between both software partitions using shared memory for communication. This can be done e.g. using semaphores.

NOTE 15: Communication objects in a) and j) between software partitions are e.g. pipes, message queues, shared memory. These communication objects cannot to be used for synchronizing partitions.
NOTE 16: Blocking read or write access is to be avoided by design when using message queues.

For bus allocation within a microcontroller network the following methods have to be considered:

dd) time-triggered data bus;
NOTE 1: Time-triggered data bus is considered "high" protection against "Failure of communication peer", "Insertion of messages", "Message delay", and "Medium" protection against "Blocking access to data bus".

ee) event-triggered data bus;

ff) event-triggered data bus with time-triggered access;

gg) mini-slotting;
NOTE 2: Mini slotting is considered "high" protection against "Constant transmission of messages" and "Medium" protection against "Message delay".
NOTE 3: Mini-slotting (see [ARINC 629]) requires each micro controller connected to the bus to wait a certain period before it is permitted to access the bus again.

hh) bus arbitration by priority;

ii) bus guardian.
NOTE 4: Bus guardian is considered "high" protection against "Blocking access to data bus", "Constant transmission of messages" and "Medium" protection against "Message corruption".

Chapter 6 Assurance and Certification methods

The SAFURE framework is set of recommendations and practices regarding safety and security matters to help system designer for the development of embedded system for mixed criticality application in the domain of automotive, avionic, telecommunication and so on.

The SAFURE framework is composed of a set of security requirement described in the SAFURE Protection Profile (SFPP) as well as a consistent design practices for designing mixed criticality system such as modeling defined in the SAFURE Methodology. The SFPP is generic form of a Security Target providing an implementation independent specification of information assurance security requirements.

6.1 CC aspects

6.1.1 CC Methodology

Common Criteria or CC normalized as ISO 15508, is a unified set of security criteria defining:

1. a standardized set of assurance requirements to evaluate security also known as SAR (Security Assurance Requirements),

2. a set of normalized security mechanisms also named known as SFR (security Functional Requirements) notably suitable to be normalized inputs of the former SARs.

In order to perform an evaluation, Common Criteria do not recommend or specify a special development or design methodology, they only provide indirectly through the Common Evaluation Methodology a support for the evaluation and certification bodies involved in the security process.

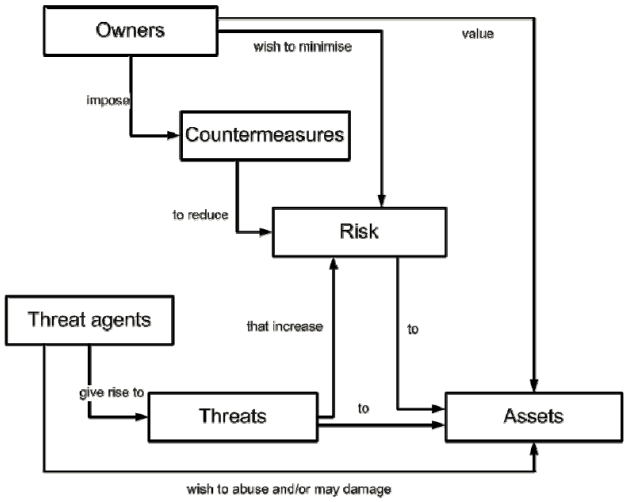

Figure 16 : Common Criteria Evaluation Methodology

All the artefacts described in this figure are usually extensively described in the Security Target (ST) or in the Protection Profiles (PP).

In the SAFURE methodology, only assets, threats and owners artefacts will be considered while the adopted countermeasures will be essentially technical. Although CC takes into account both organisational and technical countermeasures, they are usually very specific to one product and are therefore not really relevant to the SAFURE project. Technical countermeasures are associated with SFRs and their composition is described in the STs or PPs.

On the other side, SARs are much more precisely defined and are poised to raise the assurance in order to satisfy the edges properties defined in the previous figure. The SARs are defined as assurance families as follows:

- **Development** (notably ADV_ARC)
  - Guidance documents
  - Life-cycle support
- **Security Target evaluation** (through the following PP)
  - Tests
- **Vulnerability assessment** (AVA_VAN considerations)

Since the SAFURE project does not involve a CC certification, we will leverage on the **bold** elements of the former list and notably insist on elements related to security-efficiency.

As much as possible, the SAFURE methodology shall prepare the evaluation of products hosting SAFURE's resulting technologies. Thus the SAFURE methodology will enforce high granularity modelling suitable to the forecasted evaluation levels as defined below.

In the same way since the SAFURE projects proposes to communalize some technologies across the different uses cases, the following families of SFRs will be considered :

1. FIA: IDENTIFICATION AND AUTHENTICATION
2. FDP: USER DATA PROTECTION
3. FCO: COMMUNICATION
4. FAU: SECURITY AUDIT
5. FPR: PRIVACY
6. FTP: TRUSTED PATH/CHANNELS
7. FTA: TOE ACCESS

6.1.2 CC assurance levels

The CC provisions 7 progressive levels of assurance ranked from EAL-1 to EAL-7. The necessary assurance levels are determined from the level of threats and the importance of the corresponding assets.

The CC methodology is directly bounded to the chosen CC assurance levels.

Within the SAFURE projects, the associated uses cases (medical devices, automobile, etc.) associates roughly to an EAL level between 3 and 5. Some special cases could require an EAL-6 level but they shall not be managed inside an R&D project since methodologies for the upper levels are in fact dependant of the national certification bodies (e.g. ANSSI for France, BSI for Germany). As a reminder the following levels are defined in the CC as follow:

EAL	Objectives
3	*EAL3 permits a conscientious developer to gain maximum assurance from positive security engineering at the design stage without substantial alteration of existing sound development practises* *EAL3 is applicable in those circumstances where developers or users require a moderate level of independently assured security, and require a thorough investigation of the TOE and its development without substantial reengineering.*
4	*EAL4 permits a developer to gain maximum assurance from positive security engineering based on good commercial development practises which, though rigorous, do not require substantial specialist knowledge, skills, and other resources. EAL4 is the highest level at which it is likely to be economically feasible to retrofit to an existing product line.* *EAL4 is therefore applicable in those circumstances where developers or users require a moderate to high level of independently assured security in conventional commodity TOEs and are prepared to incur additional security specific engineering costs.*
5	*EAL5 permits a developer to gain maximum assurance from security engineering based upon rigorous commercial development practices supported by moderate application of specialist security engineering techniques. Such a TOE will probably be designed and developed with the intent of achieving EAL5 assurance. It is likely that the additional costs attributable to the EAL5 requirements, relative to rigorous development without the application of specialised techniques, will not be large.* *EAL5 is therefore applicable in those circumstances where developers or users require a high level of independently assured security in a planned development and require a rigorous development approach without incurring unreasonable costs attributable to specialist security engineering techniques.*
6	*EAL6 permits developers to gain high assurance from application of security engineering techniques to a rigorous development environment in order to produce a premium TOE for protecting high value assets against significant risks.* *EAL6 is therefore applicable to the development of security TOEs for application in high risk situations where the value of the protected assets justifies the additional costs.*

6.1.3 Evaluation through composition

Different options for compositional certification exist. One option is the use of the "ACO" classes of the Common Criteria itself, however the drawback is that it only gives medium assurance (up to EAL4) and it has rarely been used. Community-based alternatives do exist: One of the most successful Common Criteria communities is the smart card industry. In this domain, the base platform is the smart card, and an application is running on top of it. As applications are easier to change than hardware, a typical use case is that a new product consists of a new application with a pre-existing hardware; however there is no limitation to

that particular use case. The community developed guidance when "the requirements on the application, imposed by the underlying platform, are fulfilled in the composite product". [18].

In [19, Section 3] use cases for composite systems are identified.

In the EURO-MILS project, a MILS architectural template is used [9] for two demonstrators based on separation kernels (one in automotive, one in avionics) to identify interfaces for compositional certification.

A MILS system consists of components interacting with each other and three main components in a MILS system are identified:

- MILS core
- MILS platform
- Partition

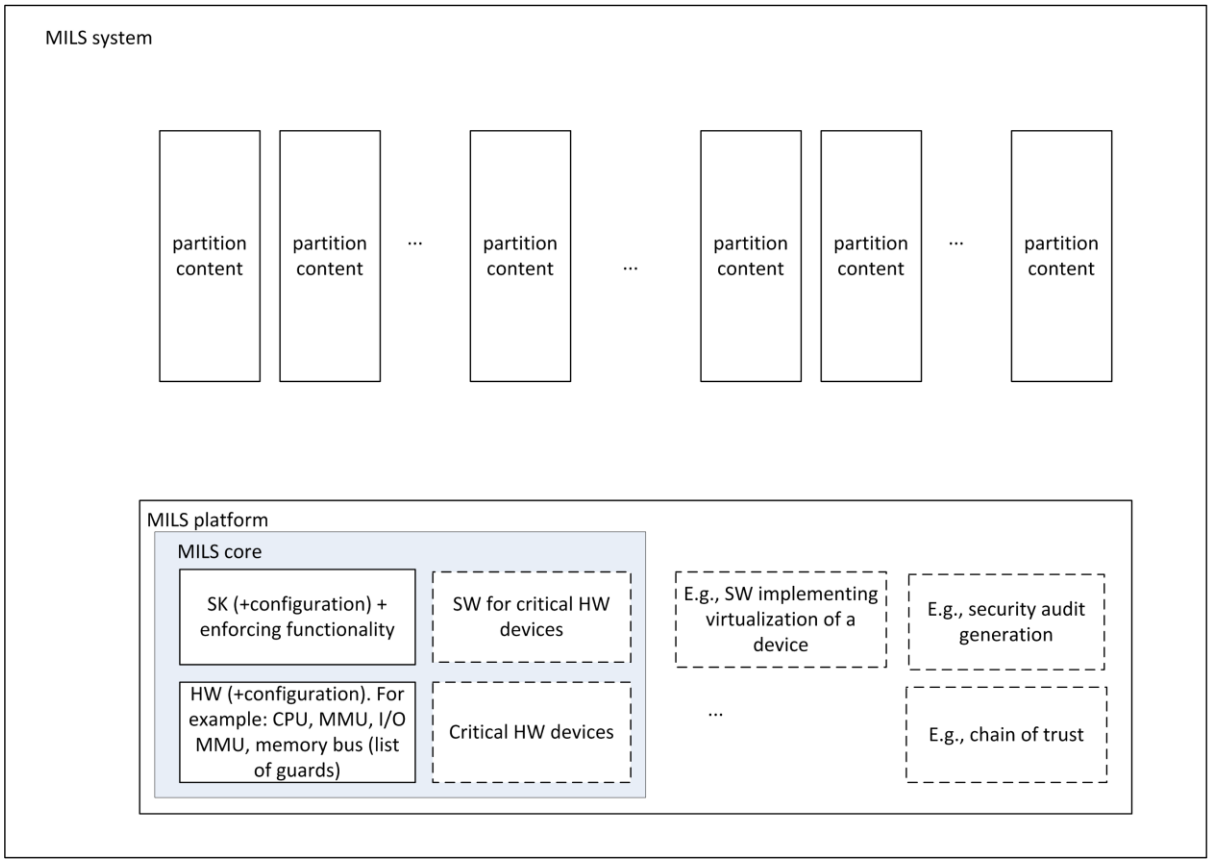

Figure 17: MILS architecture template (components in dashed lines are optional).

In [20], it is pointed out that the use of a MILS platform allows it to first certify the MILS platform itself and then certify the system based on the platform in a second step. The implementation of one layer may rely on the implementation of another layer. It is established a priori that the components of the base layer guarantee non-interference.

6.1.4 SAFURE Framework Protection Profile for Mixed Criticality Applications (SFPP)

The SAFURE Framework Protection Profile for mixed-criticality Applications is based on the existing Common Criteria Protection Profile “Protection Profile for Mobile Devices Fundamentals” developed by the Mobility Technical Community (MDFPP). This Protection Profile is a security standard for mobile devices in an enterprise environment and consists in about 80 security requirements in area such as key management, and storage, security policy enforcement, data encryption and so on. In the context of mixed criticality application, the security objectives defined in the MDFPP are not enough. Notably, it is primordial to enforce sealing between environments of heterogeneous level, i.e. medical or professional vs. personal application, to ensure the confidentiality, integrity and availability of critical functionalities.

The SAFURE Framework is an extended version of the MDFPP oriented toward mixed criticality support of mobile devices. It describes essential security services that should be provided by mobile device in a context of mixed criticality application (e.g. medical or profession vs. personal).

An instance of such mobile devices will then be developed in WP6 “telecommunication use case” in accordance to the previously established Protection profile.

6.2 Safety aspects

The DO-178 is the most common standard for certification of avionics software systems. The DO-178 applies a V-model, focusing on process assurance for the stages planning, software requirements, design, implementation, verification (including testing), software configuration and liaison with certification authorities[6]. For instance, planning documents include a software quality assurance plan that outlines all the risks. The DO-178 is graded from low assurance ("E") to high assurance ("A"), and the DO-178 assurance level determines a set of process assurance objectives to be achieved for each stage (for each stage, higher assurance levels require a higher set of objectives to be fulfilled). For instance, testing is required to contain structural coverage, that is analysis that all code statements are covered by tests from level C and higher, and the structural coverage requirement is strengthened to decision coverage for level B and modified condition/decision coverage for level A.

DO-178 supports compositional certification by partitioning of functionality. If such partitioning is used, then the technical base of the partitioning (the partitioning system / separation kernel) undergoes a partitioning analysis. A partitioning analysis can be done by identifying potential threats to partitioning and then explain how these threads are mitigated.

6.3 Timing Aspects

Timing is an important issue for safety-critical (and also mixed-critical) systems. First, time is a key non-functional requirement in many embedded and cyber-physical applications. Timing is often impacted by interference between functions (which is resolved using scheduling and arbitration mechanisms). This interference must be considered by providing isolation or analysis of the interference.

[6] See either the DO-178C document itself, provided by RTCA, or [21].

Thus, part of the SAFURE framework is a timing analysis solution that can be used to analyse temporal effects on electronic control units (ECUs) and interconnecting networks (e.g. CAN, FlexRay, Ethernet). This method is described in Section 5.1.1.

Software Defined Networking (SDN) offers (logically) centralized network control to implement safety-related features in Ethernet. These features include admission control (e.g. to avoid congestion in the network or to provide sufficient independence/isolation between traffic streams of different criticality) but also reconfiguration (e.g. to mitigate local network faults) and monitoring (e.g. to detect and defeat attacks). SDN relies on a network controller to manage the network. Management frames from this controller are sent via in-band communication, i.e. over Ethernet links, which are shared with all other traffic streams. In order to prevent or react to congestion, faults, and attacks, the management frames must be delivered in a timely manner, e.g. to contain and isolate faults and attacks on the network. Hence, the communication between the SDN controller and the network infrastructure is time-critical. A focus of SAFURE will be to provide a formal timing analysis for SDN-controlled networks and to investigate whether this concept is suitable to address the safety requirements of future networks.

6.4 Mixed criticality patterns

SAFURE's innovation relies on the mixed-criticality management framework it offers. It is no stranger to the reader that security and safety methodologies belong to different processes. In Figure 18: Mixed-criticality requirements, as long as developers choose the bottom squares, no problems outside normal works should arise.

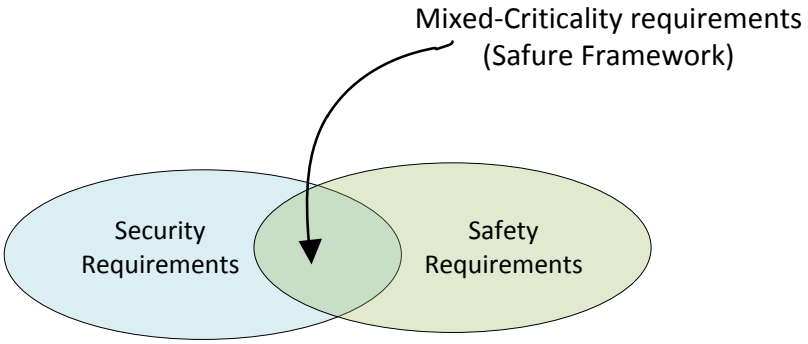

Figure 18: Mixed-criticality requirements

Nevertheless, if one wants to follow a mixed-critical development; the appropriated methodology shall be followed.

1. Critical services and assets shall be merged in one consistent set of critical assets.
2. An association between CC Threats and the safety faults shall be enforced and merged in a global mixed critical faults set which shall in turn be refined as 3 non-exclusive subsets[7] :
    a. fault handled through fault prevention
    b. fault handled through fault tolerance

[7] Fault forecasting and fault removal present in the JC Laprie model is not part of the SAFURE's goals.

3. The mixed critical faults sets shall be ordered two by two with a criticality order. This step essentially is a way to categorize some faults as more critical than others. Three main consideration shall occur thereafter:
    a. If the set if fully ordered, then the criticality order is a strong order and the according development shall follow exactly such order. For example, if in a medical device such as an insulin pump, the failure of the drug delivery is considered as more important than the presence of a malicious user, the system shall act accordingly by delivering insulin and raising an alert.
    b. If the resulting set is a lattice, then the identification of the missing association between faults shall be performed and two sub-processes shall be engaged for each association :
        i. If both faults belongs to fault prevention, an explicit association shall be established
        ii. If the both fault belongs to fault tolerance, a redesign of the architecture shall be performed in order to exhibit a deterministic property. In the security field, such architecture shall exhibit defence in depth properties. In the safety field fault removal shall be handled through serial invocation of mechanisms per faults. For example through redundancy.
        iii. If one fault belongs to fault prevention and the other to fault tolerance an explicit association shall be established.
    c. Finally if none of the above applies, two sub-processes shall be applied :
        i. A consistency process: the whole consistency of the fault sets shall be verified.
        ii. If and only if the former process is conclusive, a deterministic process similar to the lattice process shall be engaged :
            1. If both faults belongs to fault prevention, an explicit association shall be established
            2. If the both fault belongs to fault tolerance, a redesign of the architecture shall be performed in order to exhibit a deterministic property. In the security field, such architecture shall exhibit defence in depth properties. In the safety field fault removal shall be handled through serial invocation of mechanisms per faults. For example through redundancy.
            3. If one fault belongs to fault prevention and the other to fault tolerance an explicit association shall be established.

Chapter 7 References

[1] European Union Agency for Network and Information Security (ENISA), "Algorithms, key size and parameters report," 2014.

[2] European Network of Excellence in Cryptology II (ECRYPT II), „ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012),“ 2012.

[3] Richard F. Boldt, \Modeling AUTOSAR systems with a UML/SysML profile" (2009)

[4] Object Management Group, \UML Superstructure Specification", v2.0 (2005)

[5] Hasan, Jahangir, et al. "Heat stroke: power-density-based denial of service in SMT." High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. IEEE, 2005.

[6] Martin, Thomas, et al. "Denial-of-service attacks on battery-powered mobile computers." Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on. IEEE, 2004.

[7] R. J. Masti, D. Rai, A. Ranganathan, C. M¨uller, L. Thiele, and S. Capkun. “Thermal covert channels on multi-core platforms”. pages 865–880, 2015

[8] Rai, Devendra, and Lothar Thiele. "A calibration based thermal modeling technique for complex multicore systems." Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium, 2015.

[9] EURO-MILS MILS Architecture Whitepaper, 2014 http://www.euromils.eu/downloads/2014-EURO-MILS-MILS-Architecture-white-paper.pdf

[10] White paper from Fujitsu: BOYD: IT’s About Infrastructure and Policies.

[11] Method & Tools to secure and support collaborative architecting of constrained systems. JL Voirin.

[12] https://www.polarsys.org/proposals/capella

[13] Aprille, Roudier : Towards the Model-Driven Engineering of Secure yet Safe Embedded System. GRAMSEC 2014

[14] Richter, K. (2005). “Compositional Scheduling Analysis Using Standard Event Models“. Dissertation, Technische Universität Braunschweig.

[15] Schliecker, S. (2011). “Performance Analysis of Multiprocessor Real-Time Systems with Shared Resources”. Dissertation, Technische Universität Braunschweig.

[16] Jonas Diemer, Daniel Thiele und Rolf Ernst, "Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching" in 7th IEEE International Symposium on Industrial Embedded Systems (SIES12), June 2012, Invited Paper.

[17] Mikel Fernández, Roberto Gioiosa, Eduardo Quiñones, Luca Fossati, Marco Zulianello, Francisco J. Cazorla “Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain”, in proceedings of the 12nd International Conference on Embedded Software (EMSOFT), 2012.

[18] Composite product evaluation for Smart Cards and similar devices, no. CCDB-2012-04-001, 2012, Common Criteria Supporting Document: Mandatory Technical Document, https://www.commoncriteriaportal.org/files/supdocs/CCDB-2012-04-001.pdf.

[19] Andreas Daniel Sinnhofer, Wolfgang Raschke, Christian Steger, Christian Kreiner, Evaluation paradigm selection according to Common Criteria for an incremental product development, International Workshop on MILS: Architecture and Assurance for Secure Systems (HiPEAC 2015), 2015, https://m.euromils.eu/downloads/hipeac_literature/08-mils15_submission_9.pdf.

[20] EURO-MILS Non-interfering composed evaluation whitepaper, 2015,

http://www.euromils.eu/downloads/white_paper_non.pdf

[21] Leanna Rierson, Developing Safety-Critical Software: A Practical Guide for Aviation Software, CRC Press 2013