SAFURE

D6.7
Final Specifications of the SAFURE Framework and Methodology

Project number:	644080
Project acronym:	**SAFURE**
Project title:	SAFety and secURity by dEsign for interconnected mixed-critical cyber-physical systems
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-644080-D6.7/ 1.00
Work Package:	WP 6
Due Date:	May 2018 - M40
Actual Submission Date:	7$^{\text{th}}$ June 2018
Responsible Organisation:	SYM
Editor:	Björn Gebhardt
Dissemination Level:	PU
Revision:	1.00
Abstract:	This report describes the framework for development and designing safe and secure embedded systems and discusses lessons learned from application in demonstrators.
Keywords:	Algorithms, Mixed-Criticality, Temperature, Data integrity, Timing integrity, Resource sharing integrity

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644039.

This work is supported (also) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0059. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

Editor

Björn Gebhardt (SYM)

Contributors

André Osterhues, Alexander Ptok (ESCR)
Björn Gebhardt (SYM)
Sylvain Girbal (TRT)
Robin Hofmann (TUBS)
Jaume Abella (BSC)
Marco Di Natale (SSSA)
Rehan Ahmed (ETHZ)
Stefania Botta (MAG)
Don Kuzhiyelil (SYSG)
Edin Arnautovic (TTT)

Disclaimer

Executive Summary

Embedded systems are much more complex nowadays than they were a few years ago. As a result, it is becoming increasingly necessary to develop appropriate strategies during the design phase that make such systems uncritical with regard to data integrity, timing or temperature. Not only in the design phase, but also in the verification phase, appropriate tools are necessary to measure and check such aspects in order to optimize them afterwards.
In this document, the various SAFURE partners will present strategies, methodologies and tools for each phase to make embedded systems safe in this respect. The methodologies and tools developed were tested using so-called demonstrators, which have already been made available in Deliverables D6.3 to D6.6. Adequate conclusions or, so to say, ”lessons learned” from the demonstrators should be used as a guideline for later, similar projects from research and industry and provide a framework for such developments.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The complexity of today's embedded systems is growing rapidly. But just like their complexity, the error proneness also increases, which requires an intuitive and agile framework for a stable design of such systems. To increase the safety, frameworks have to cover questions regarding methodologies for design regarding thermal influences, data integrity and timing.

To reach this goal, the SAFURE partners have proposed design guidelines for developing strategies and introduce some tools to guide architects and developers of such systems. This document shows what the individual partners in the SAFURE project have considered, developed and implemented with regard to various development processes and methodologies for implementing such systems, which tools can be used and what the partners have learned from the results of the demonstrators.

SAFURE uses the fundamental approaches already described in Deliverable D3.2 and the methodologies taken up in Deliverable D3.3. The knowledge gained will enable developers to design mixed-critical systems for improved safety and security and serve as a helpful basis for leanings on the project. Figure 1.1 shows the SAFURE framework developed to ensure that such an embedded system can be developed in terms of safety and security.

In addition, the structure of the document is based entirely on the structure shown in the figure 1.1. Each line of the illustration corresponds to a chapter and each box to a section or subsection. These in turn represent the design and execution phases within a development process, which is explained as an example in the next section.

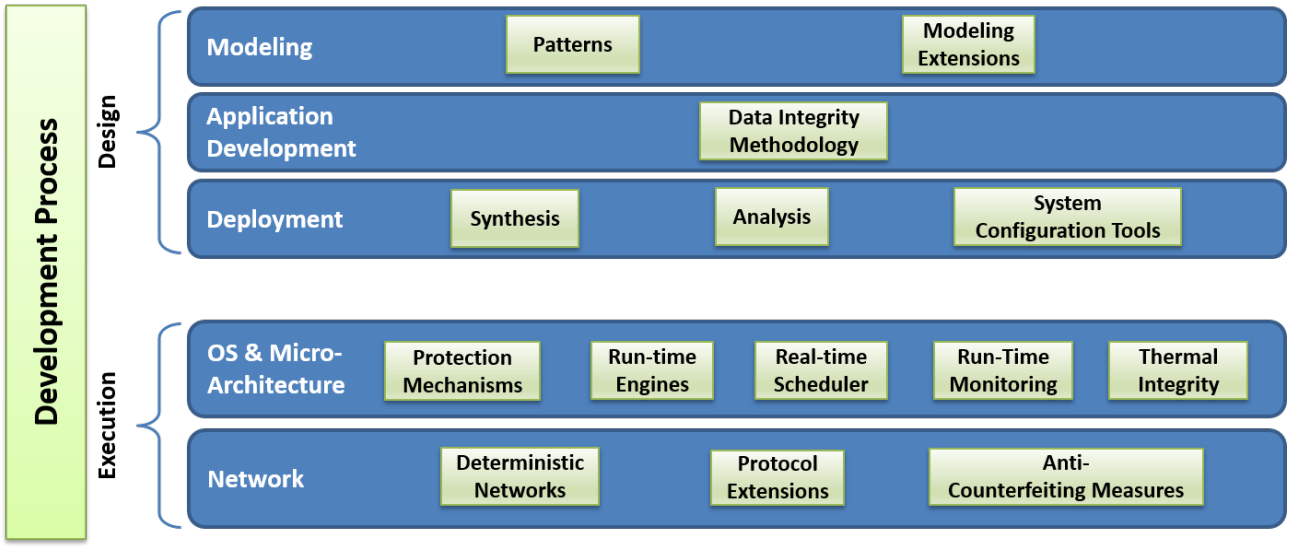


Figure 1.1: Overview of the SAFURE Framework

1.1 Development Process

This document aims at software/systems developers who design and implement a mixed-critical product. Therefore, a well-structured development process is mandatory. There are several different de-

velopment processes and we chose an example from the automotive domain in the following section. However, this example can also be applied to other industry domains. The key idea is that all parts of the SAFURE Framework, presented in Figure 1.1, are covered in the process and the mixed criticalities are considered in the design phase. While some criticalities can be considered individually and without much interference, other criticalities like safety and security have to be considered in a holistic approach and throughout the whole development process.

1.1.1 Process description

Considering that there is a contribution of security to the (continued) safety of a system, it is apparent that the domain of security needs to receive careful attention while developing applications and systems. A development process for software and hardware should reflect this in its tasks and outputs. When conforming to ISO 26262 [20], a development process can be modelled as a ”V” (V-model) depicted in Figure 1.2. Where the upper ends of the ”V” correspond to an abstract, all-encompassing view, and the tip of the ”V” represents the more concrete and detailed perspective on singular units of the system. In the automotive domain, this approach has been refined to better meet the requirements of automotive products.

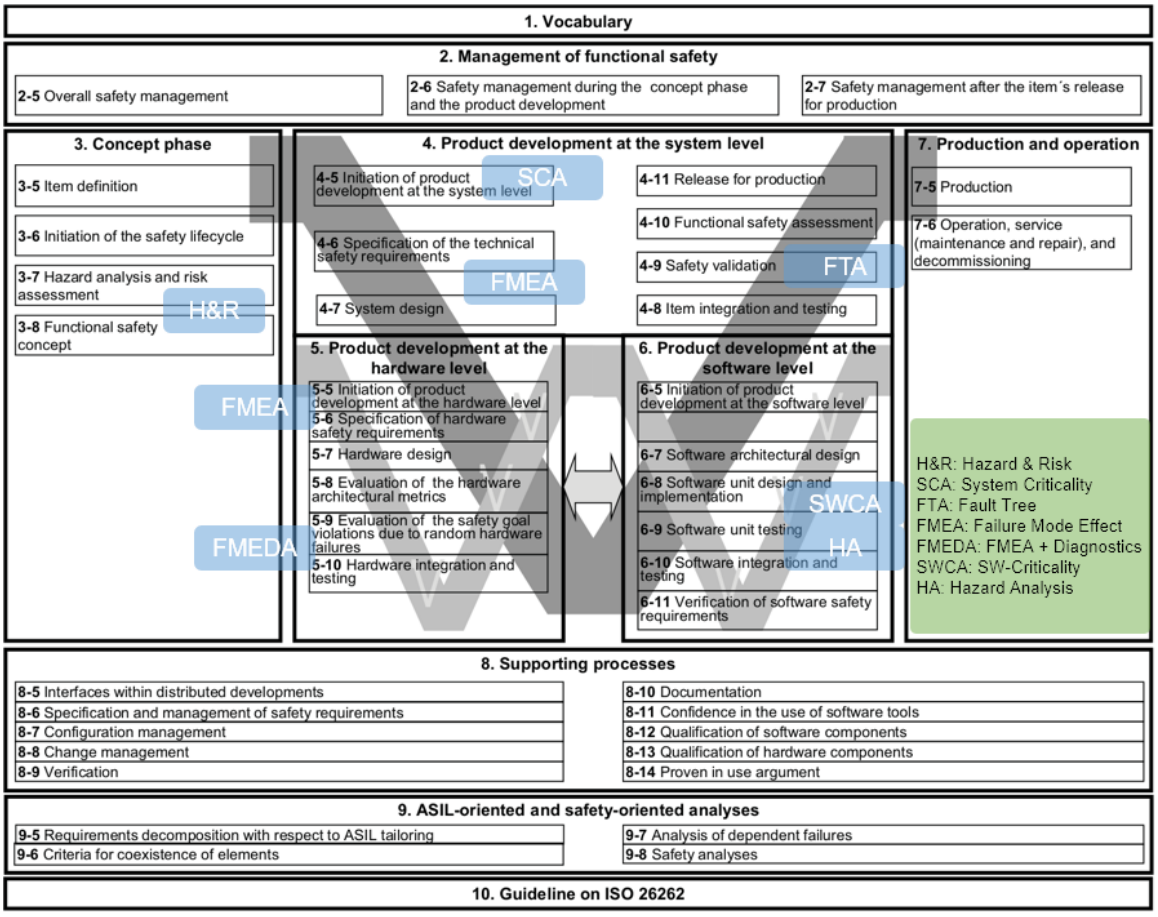


Figure 1.2: V-model applied at system, hardware, and software levels in product development (according to ISO26262 [20]), including safety-related analyses

Figure 1.3 depicts the process laid out by Automotive SPICE® [26], a derivative of SPICE® (superseded by the ISO 33001 family of standards [21]). It also addresses requirements design, which helps in establishing a level of confidence in the safety of the system. As a form of expectations towards the behaviour or shape of the system, requirements serve as a common understanding and communication between stakeholders in the process. They drive decisions during the development

process and serve as a set of criteria against which the system can be validated. Requirements can stem from external sources, like customers and jurisdiction. Otherwise, they derive from the use-cases for the product, business strategy, or other internal factors.

The following list of steps closely follow the V-model. The first three items are ordered in the increasing degree of detail, whereas the last three steps are given in the order of increasing abstraction. The steps address security engineering methods to complement the safety efforts. Thus, the requirements mentioned below are requirements pertaining to security as opposed to safety.

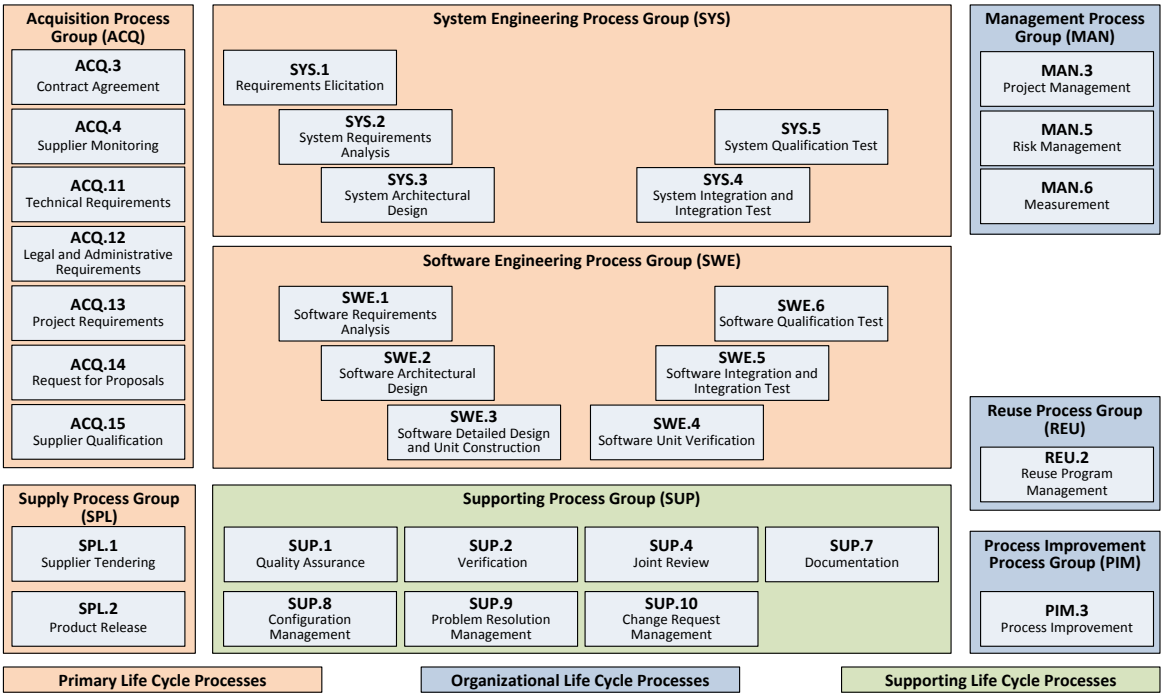


Figure 1.3: V-model according to Automotive SPICE® [26]

- **STEP 1: ASSESSMENT** In order to determine data that need to be protected for integrity, the communication flows of the target need to be scrutinised and, based on their potential or expected effects, their need for protection identified. It is important to make the distinction of whether the data are transmitted over a network (e.g., sensor data, commands to actuators, or update packages) or are stored on a device (e.g., cryptographic material, firmware, its configuration, or data gathered for analysis and logging). Regarding the assessment of the impact of a potential manipulation of said data, the well-established methodology of a security threat and risk analysis offers itself. It establishes a link from security goals (which designate security objectives, such as confidentiality, integrity, and availability, to assets of the system) to the risks attributed to consequences stemming from those security goals being violated. The result of such an analysis thereby demonstrates the need for measures to protect assets. For instance, in case of the asset being data, the integrity often is of great relevance and formulated into a security need of specific data needing integrity protection. Based on the risk assessment and security goals, one can derive security requirements for the system. These are an important stepping stone for further conceptualisation and mix with functional, safety, and external requirements (s.a., legal or customer requirements). They also serve as a basis for establishing traceability, as discussed in Section 1.1.2. An exemplary threat and risk analysis has been given in Deliverable D5.3.

- **STEP 2: CONCEPT PHASE** The needs of relevant data artefacts for integrity as determined in step one are addressed by iteratively identifying adequate measures. The iterative approach

is motivated by the impact that those measures have on performance, data throughput, and timing behaviour of components of the system. Since being interleaved with the design of those other components, their requirements are influenced by the security measures added and might result in another iteration of the design step. A security concept documents the measures which need to be applied to the system. It is beneficial to have reviewers for the concept that give their input on the effectiveness and viability of the security measures being added. Measures often include cryptographic means to protect the confidentiality and integrity of data as well as the availability of functions and services. The principle of minimality, together with security testing, encapsulates an additional approach to minimise the attack surface.

- **STEP 3: IMPLEMENTATION** After the design phase of the system, the implementation of the concrete measures is to happen. Some principles apply for this phase to assure the high quality of components delivering protection to the system. As a principle, well-known cryptographic libraries or hardware providing cryptographic algorithms should be preferred over home-made cryptography. A relevant standard for the evaluation of cryptographic modules is FIPS publication 140-2 [23].

  For the software part of the system, the development should incorporate standards and guidelines pertaining to secure coding for the programming language of choice. Chief among them is the validation of input (from the user or via interfaces). Assurance methodologies include the review and testing of code. While testing is addressed in the next step, code review can help finding and addressing exploitable weaknesses in the code, that might lead to compromise or disclosure of data. Besides independent reviewers, automated frameworks for detecting flaws in the source code exist (e.g. static code analyzers like QA-C [1]).

- **STEP 4: UNIT TESTING** Defining a unit in a system enables the testing of a specific software or hardware component disregarding the complexity of the entire system. A test at the unit level should cover all of the intended functionality of the unit. This includes interfaces, both for hardware and software. All functions that can be called should be exposed to positive and negative tests. Positive tests are those that confirm that the functionality requested is delivered without fault when all necessary preconditions are met. Negative tests are those that confirm that missing preconditions are detected reliably. Functions that take parameters should be covered by tests that address these parameters. For cryptographic algorithms, there exist test vectors which can be used to evaluate implementations. They are often included in the algorithm's specification or come as a stand-alone publication.

- **STEP 5: INTEGRATION** On a larger scale, units come together to form more complex parts, or components, of the system. The integration phase establishes a transition from a fully-detailed view of the unit to a more abstract view of the functionality delivered.

  For integrating units to components, mocking of other components is relevant. To test whether units combined can deliver the relevant functionality, their consumer (or user) needs to be emulated. Also, the part of the system under test might require other components to produce the expected results. Security-wise, penetration tests can start at this stage. As a form of negative testing, penetration tests attempt to circumvent security measures like access control or gaining control over the target to steer its behaviour by using vulnerabilities that the test target bears. Automated vulnerability detection for single components of the system can assist and complement manual testing. It is beneficial to include testing for known vulnerabilities of implementations for the protocols and algorithms employed.

- **STEP 6: ACCEPTANCE** At the end of the development process, the entire system is exposed to testing. Instead of considering singular components (or units), their interactions are

[1] http://www.prqa.com/static-analysis-software/qac-qacpp-static-analyzers/

tested. Security-wise, penetration tests are again an effective means to close the attack surface. The methodology—besides accommodating the growing complexity of matching interdependencies—does not differ substantially from that in the previous phase. It now seeks to defeat complex interactions to obtain control over parts of, or, the entire system. The rule of thumb holds that the more complex a system is, the more attack paths are available. As a consequence, penetration tests at this stage aim to reduce the number of attack paths resulting from the interactions of components and sub-systems.

In this section, a software development process capable to consider mixed-criticalities has been presented. In the next section, we show how this process can be augmented with requirements traceability, which enables to increase the confidentiality that all steps defined in the development process are carried out correctly.

1.1.2 Requirements Traceability

According to [15], Requirements Traceability is ”the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)”.

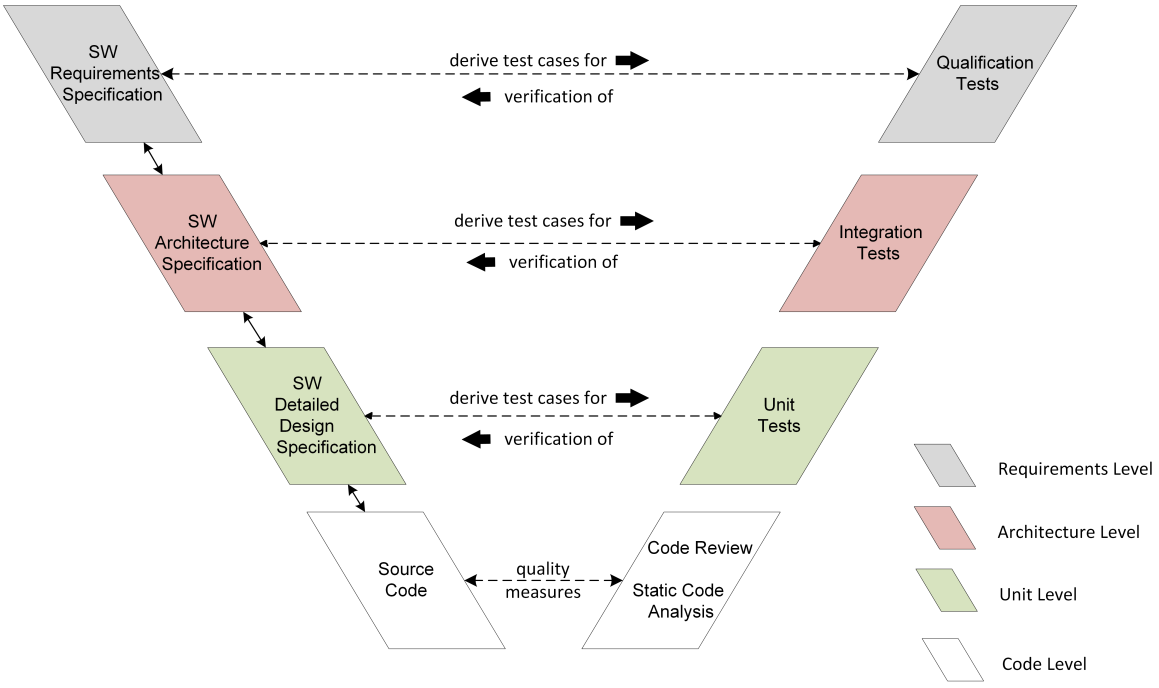


Figure 1.4: Tracing according to the V-Model

Specifically, the following items should be covered:

- the identified requirements from the assessment in step 1 (see gray boxes in figure 1.4),
- the architecture and design derived from the concept in step 2 (see red boxes), and
- all implemented code units from step 3 (see green and white boxes, here a distinction between SW Detailed Design and the Source Code is made).

Traceability also increases the confidence in the product, because it shows that

- all requirements can be traced to the implementation (functional completeness, see arrows from SW Requirements Specification down to Source Code in figure 1.4),

- the source code can be traced back to the requirements (backwards requirement coverage, see arrows from Source Code up to SW Requirements Specification),
- there are tests for all requirements (test derivation, see arrows from left side of ”V” to the right side), and
- the tests cover the specification, the architecture, and the implementation (full test coverage, see arrows from the right side of the ”V” to the left side). Specifically,
  - the software requirements are verified by qualification tests,
  - the software architecture is verified by integration tests,
  - the software detailed design is verified by unit tests, and
  - the source code is verified using code reviews and static code analysis.

To summarize, figure 1.4 shows how the requirements traceability can be realized for software developed using the V-model.

In SAFURE, the requirements were collected and categorized in deliverable D1.2. For the demonstrators, the requirements coverage is shown in chapter 4.3 of deliverable D6.4 and chapter 5 of deliverable D6.5. However, the SAFURE requirements are defined in a rather high-level fashion, because SAFURE does not aim at one particular product.

In general, and especially for product development, the requirements should be as precise and specific as possible in order to facilitate a thorough development process that achieves a full requirements traceability.

1.2 Deliverable Outline

As already mentioned, the structure of the document is based on that of figure 1.1. Each chapter stands on its own and cannot always be integrated into an overall context of the following chapters. They each offer a collection of findings from partners from the most diverse areas and are therefore sometimes too different to create transitions.

The chapters 2, 3 and 4 deal with the basic design of embedded systems. They point out suitable strategies and approaches which are necessary for a successful design of such systems with regard to safety and security.

Chapters 5 and 6 describe mechanisms to ensure various aspects, such as temperature or timing, within already designed embedded systems. These mechanisms make it possible for the system engineer to adapt his system accordingly.

Chapter 2 Design: Modeling

In order to perform the assessment of all the system communications and computations that are sensitive with respect to security and safety threats and to adequately represent the design and implementation provisions that must be put in place in order to address them, it is important that a suitable abstraction of the system is available as a model. A model of the system and its components shall be made available to address the needs of the assessment and concept phases and to provide support for the implementation and analysis phases.

In the context of the project framework outlined in Figure 1.1, the definition of the modeling extensions and the patterns fit within the general scope of the support to the design stages (as opposed to the execution) by providing adequate abstractions for the representation of the safety and security properties of mixed-critical systems.

In the context of the development process, outlined in Section 1.1, the modeling extensions and the modeling patterns are meant to be used in the first two stages (assessment and concept) to possibly drive the analysis and implementation.

In SAFURE, the analysis and development of models for safety and security sensitive systems has been performed by identifying two different sets of activities. First, there is clearly the need to extend existing commercial and standard modeling languages and frameworks to allow the definition of the attributes and constraints that characterize mixed-critical systems.

These extensions, however, are not sufficient. The definition of a language and a set of modeling elements shall be complemented by the identification of a set of patterns that describe common design solutions and features and illustrate how the modeling features should be used for the definition of subsystems of interest.

2.1 Modeling Extensions

The modeling extensions of SAFURE are defined with several objectives in mind. They should:

- Satisfy the needs of the requirements from the case studies and the analysis performed in WP1.
- Be produced according to the results of a gap analysis that considers multiple sources: past (EU) projects, scientific literature, analysis methods and techniques, existing standards.
- Be implementable as extensions on commercial or standard modeling languages and on commercial or open source tools so that they can be actually tested and be used.
- Be validated by application to a selected number of design, analysis and synthesis cases, including the case studies of WP6.

The modeling extensions can be used in the development process to drive not only the design, but also the analysis and synthesis stages. As outlined in Figure 2.1, the modeling extensions can be applied to a modeling tool (Rhapsody [17] has been used among other tools in the context of SAFURE) and the annotated designs can be exported from the tool in any standard format (or through its API) to be then processes for the purpose of analysis and synthesis.

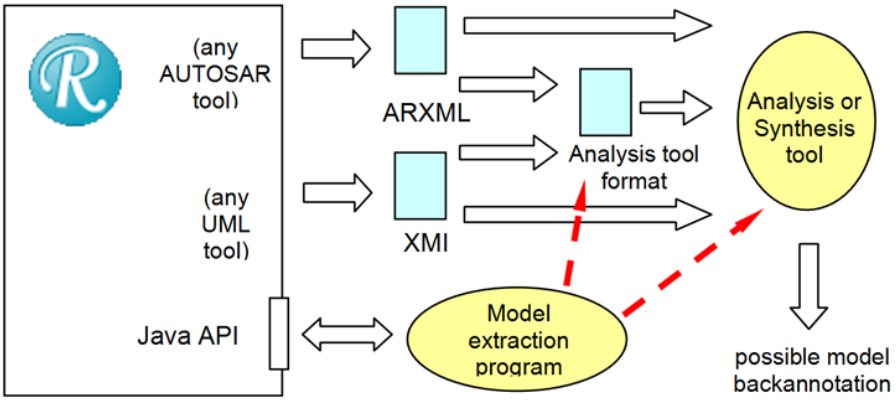


Figure 2.1: Tool flow from modeling to analysis and synthesis

2.1.1 Modelling Extensions for the Specification and Analysis of Safety and Security Properties

A precondition for the integration of analysis techniques for the evaluation of safety (timing) and security properties is the availability of modeling features at the right level of abstraction to provide the expressive means for the definition of the application requirements and the properties of the mechanisms and policies that can be employed to satisfy the requirements and validate them.

In SAFURE, the need for modeling features that can support the analysis and validation of components has been addressed in WP2 with a study that provided the definition of the required abstract features, followed by a concrete mapping and implementation is standard or commercial languages and tools.

The associated results have been presented in D2.2 and are used as a starting framework for the application of analysis and synthesis techniques and tools. As a general concept, the metamodels defined for the abstract concepts have been implemneted as far as possible using tools and languages that allow a validation using the methods and tools in the SAFURE framework.

In detail, we provided modeling features for the representation of application-level security requirements. These have been described as UML stereotypes and applied to AUTOSAR examples. These models have been used to validate the synthesis of security mechanisms, as described in the project Deliverables and several research papers.

The modeling concepts for the description of timing constraints, including constraints for robust systems with occasional misses, as in the m-k model, have been used as input for analysis techniques described and implemented in WP3 and published in the IEEE EMSOFT conference [25].

Also, the modeling features for the description of criticality levels are used for the automatic generation of code using the timing protection features of the AUTOSAR [7] automotive standard. This process has been described in WP4 and implemented as part of the case study analysis in WP6.

Figure 2.2 shows the process connections of the modeling extensions with the analysis and synthesis activities performed in the context of the project. Of course, in principle, the modeling extensions could be used throughout the project for several other analysis activities.

2.1.2 Modelling Extensions for the Compositional Performance Analysis

In the scope of the SAFURE project, we utilised the Compositional Performance Analysis (CPA) [16] to provide a formal worst-case analysis for real-time communication systems, with a focus on real-time Ethernet. To be able to cover the upcoming standards in the Time-Sensitive Networking (TSN) group, the existing CPA models required extensions. These standars include the time-aware, burst-limiting and peristaltic shaper and frame-preemption. A detailled description of these standards and their implications have been given in D5.2 and D5.3. These standards, however, can only be applied

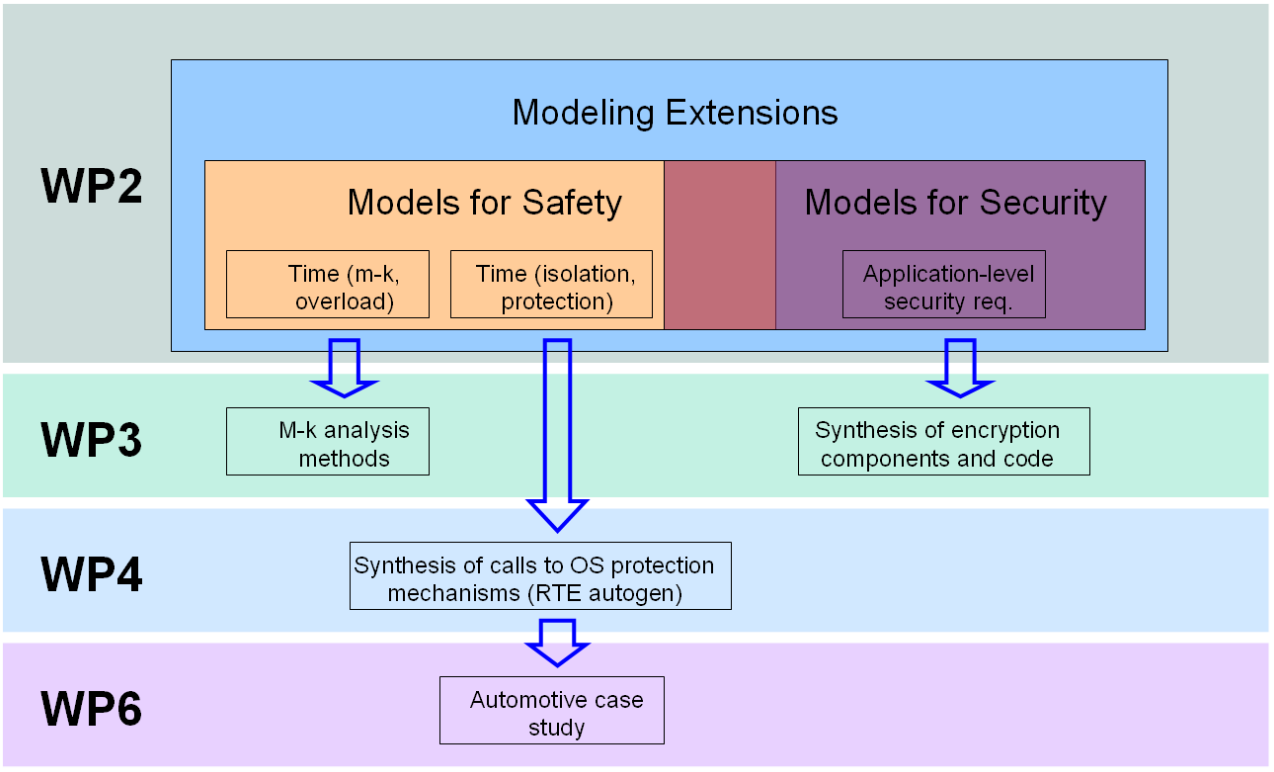


Figure 2.2: Modeling extensions as an input to analysis and synthesis

to static systems and are not suitable for dynamics, including topology, traffic or mode changes. With the trend towards fail-operational systems or simply a need for more efficient network utilisation, this is a highly desired feature. The software defined networking (SDN) paradigm is evaluated in the context of can be used to handle dynamics.

A general description of the CPA approach has been published in [16] in the context of SAFURE. CPA in the context of real-time Ethernet has been described in [27] The modelling extensions to CPA are described in detail in D3.2.

2.2 Design Patterns

The definition of modeling language extensions and the availability of modeling features for the defintion of constraints and (architecture) design elements that relate to safety and security is typically not sufficient for supporting the design stages of coplex embedded/CPS systems.

In the design of a feature-rich system, the architects are often faced with the problem of baing able to leverage at best the opportunities offered by the (new) modeling language and also on how to optimize the design and provide for better reusability and extensibility. The definition of architecture patterns addresses both issues. A pattern is a template of an architecture-level solution (or a set of possible solution, made parametric for better reuse) that addresses a common problem in an efficient way. Patterns are defined based on best practices, designers' experience or even the work of standardization bodies. In SAFURE, we provide a limited set of patters as demonstrators to show the possibility of using the provided modeling extensions in a practical design process, but also to support the definition of architecture models that are fundamental to the design of safety and security-critical systems.

2.2.1 Architecture Patterns for Safe and Secure Systems

Complementing the definition of the modeling extensions for the specification and analysis of safety and security properties, we analyzed the definition of architecture patterms for the development of mechanisms, libraries and systems aimed at providing or guaranteeing properties of satefy and security.

Among the analyzed and developed patterns, we provided for the definition of:

- Hardware Security Modules or HSMs
- Protection Kernels

The protection kernel pattern (the metamodel from D2.2 is shown in Figure 2.3) includes also the definition of all the typical features of an OS for mixed-critical systems.

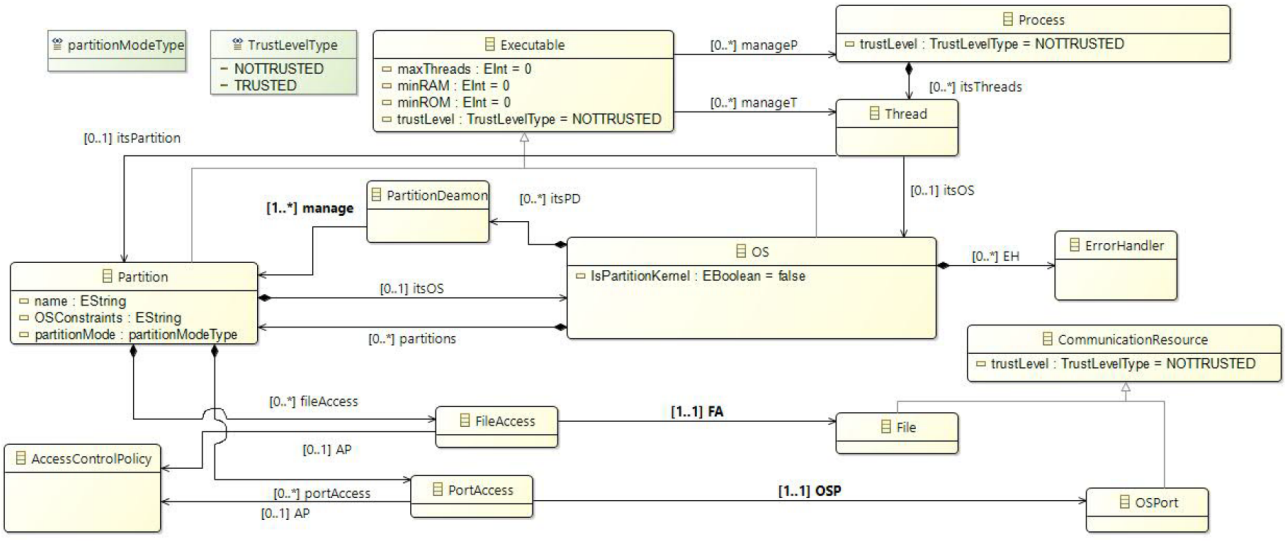


Figure 2.3: Pattern for a protection kernel

The metamodel diagrams like the one in Figure 2.3 formally describe the elements that are required for these critical architecture subsets or composites and the relationships among them. For protection kernels, of particular importance are the identification of elements like partitions and access control policies, that enable the application of the fundamental concepts of safety and security.

2.3 Conclusion

The modeling extensions and the architecture patterns defined in SAFURE are meant to be used as a guideline or support for the future definition of standards or for the definition of domain-specific modeling languages.

The value of the proposed models is partly in the capability of describing constraints and features of interest for mixed-critical systems, but especially, because of the fact that these modeling features have been used for different stages of Analysis and Synthesis, as belonging the the design and implmentation activities outlined in Chapter 1, and described in more details in the following chapters.

The application to analysis and implementation is a partial but convincing validation of the applicability and usefulness in the context of an actual development process of the introduced extensions.

Chapter 3 Design: Application Development

In this chapter, hints and best practices for the application development of mixed-critical systems are given. According to figure 1.1, the application development in SAFURE only contains the box ”Data Integrity Methodology”, which is described in the next section. The idea is that the integerity of data (e.g., the communication between applications) is crucial for the security of the overall system.

3.1 Data Integrity Methodology

We have presented different methods to preserve data integrity in Deliverables D3.1 and D3.2. Table 3.1 summarizes the algorithms that were implemented and examined within SAFURE. The algorithm type is either Message Authentication Code (MAC), Authenticated Encryption (AE), or Digital Signature (DS). MAC and AE are symmetric algorithm families and DS is asymmetric cryptography.

Algorithm	Type	Key lengths	Tag/Signature length	Performance
HMAC-SHA256	MAC	variable	256 bits	very fast
KMAC-128	MAC	variable	variable	fast
AES-CCM	AE	128, 192, 256 bits	variable	fast
AES-GCM	AE	128, 192, 256 bits	variable	fast
Poly1305-ChaCha20	MAC	256 bits	variable	fast
RSASSA-PSS Sig Gen	DS	1024 to 4096 bits	same as key length	very slow
RSASSA-PSS Sig Ver	DS	1024 to 4096 bits	same as key length	quite fast
ECDSA	DS	192, 224, 256, 384, 521 bits	double the key length	moderate
EdDSA	DS	256 bits	512 bits	moderate

Table 3.1: Data Integrity Methods

As a conclusion, the symmetric algorithms are the fastest, with HMAC-SHA256 and AES-GCM being the recommended algorithms in this class. For the asymmetic algorithms, we recommend RSASSA-PSS for applications with only few signature generations and many verifications. For all other applications, we recommend to use EdDSA.

3.1.1 Key Length Recommendations

The length of the used key has an impact on the security level, but also on the performance of the algorithm. Also, as noted in table 3.1, for some algorithms the chosen key length also determines the length of the authentication tag (for symmetric algorithms) or signature (for asymmetric algorithms). Table 3.2 summarizes the recommendations of different institutions on key lengths to be used medium- to long-term protection. In the DSA column, p represents the length of the key and q defines the group size.

We recommend 128 bits for symmetric algorithms, 256 bits for elliptic curve algorithms and at least 2048 bits for RSA and DSA.

Institution	Year	Symmetric	RSA	DSA	ECDSA
ECRYPT II [11]	2012	128 bits	3248 bits	$q >=$256 bits, $p >=$3248 bits	256 bits
NSA [22]	2015	256 bits	3072 bits	-	384 bits
NIST [9]	2016	128 bits	3072 bits	$q >=$256 bits, $p >=$3072 bits	256-383 bits
ANSSI [3]	2014	128 bits	2048 bits	$q >=$200 bits, $p >=$2048 bits	256 bits
BSI [10]	2017	128 bits	2000 bits	$q >=$250 bits, $p >=$2000 bits	250 bits

Table 3.2: Key Length Recommendations

3.1.2 Choice of Data Integrity Algorithm

In this section, some guidelines for the choice of data integrity algorithms in the domain of cyber-physical systems are given. The choice of the data integrity algorithm has an impact on the performance, key management, and overall security of the system.

1. **Symmetric vs. asymmetric:** Symmetric algorithms are much faster than their asymmetric counterparts. Therefore, it should be carefully evaluated whether it is necessary to use asymmetric cryptography. In cases where key distribution is a problem, asymmetric algorithms should be chosen. In most other cases, symmetric algorithms are the better choice.

2. **Space requirements:** If there is only little ROM space left on the device or if stack space (RAM) is very limited, symmetric algorithms usually better fit to these limitations. Also, it should be considered to use a space-efficient implementation of the respective algorithm.

3. **Speed:** Most speed optimizations come at the expense of a larger code size. When several algorithms are used in combination (e.g., hybrid encryption using AES with a key distributed using RSA), only the algorithms used for bulk data (AES in this case) should be optimized for speed.

4. **Key length:** The choice of key length influences the amount of (secure) storage and network traffic (for key distribution), but it also affects the run-time of algorithms. Therefore, the key length should be selected in order to offer good performance (speed, ROM and RAM usage) while still being secure enough for the application.

3.2 Conclusion

In this chapter, different methods for data integerity have been shown. The tables allow an application developer to decide which method and key length is best suited. Furthermore, the most important design choices are summarized.

Chapter 4 Design: Deployment

The Safure tools and methods for the analysis and synthesis of safe and secure components operate from descriptions of components and HW/SW architectures provided according to standard formats (examples are UML and AUTOSAR). These models are extended by including the modeling elements and features described in the previous section and in WP2 for the definition of safety and security features for mixed-critical systems.

As outlined in Chapter 1 (and illustrated in Figure 1.1), the deployment stage fo the design follows the Modeling and Application Development stages. It consists in the analysis of the models defined for the system and possibly in the synthesis of implementations, bridging the modeling stage of Figure 1.1 to the implementation stages in the following sections. The automatic synthesis of implementations has several advantages. It is cost-efficient, by relieving developers from the need to code the elements described in the design, but especially, has the advantage of providing provably correct and efficient implementations of the design model, in a flow that should ideally move towards correct-by-construction deployments.

The tools for the synthesis of implementations receive as input the models from WP2 and operate on the modeling extensions for security and safety to provide two different sets of code implementations as documented in WP2 and WP4.

- the synthesis of components or RTE code for the implementation of encryption according to the user specifications in an AUTOSAR flow
- the synthesis of an RTE implementation that makes use of the standard AUTOSAR OS features for timing isolation and protection to provide an implementation to a user request for preserving the criticality levels of components in a mixed-critical application

The tools and methods for the analysis of the safety and security properties of components include the following:

- Synthesis of mechanisms for the implementation of secure communication among components in an automotive environment
- Automatic generation of basic software (RTE) for the implementation of timing protection among components and tasks at different criticality levels.
- Analysis of the timing properties of networks
- Analysis of systems with temporary timing overloads

4.1 Synthesis

The synthesis tools are demonstrators that operate following a model-based design flow in which the system architecture is defined according to a standard modeling language, analysed for correctness, and then used to automatically generate system components (basic software and communication functions) that provide provably correct implementation of some functionality to the application components.

Two main examples of synthesis mechanisms have been developed. Both are based on the modeling extensions and the development flow that is typical of automotive applications, in accordance with the AUTOSAR standard.

4.1.1 Automatic generation of encryption components and code

This process (detailed in D2.2) starts from an AUTOSAR model in which selected communications have been annotated with security requirements. In a standard AUTOSAR process, components bear the responsibility of selecting the encryption mechanism and calling it before sending the data. In our flow, the user only needs to specify the security requirements associated with each communication, using the modeling tools of the standard AUTOSAR process, with the extensions described in D2.2 (as shown in Figure 4.1).

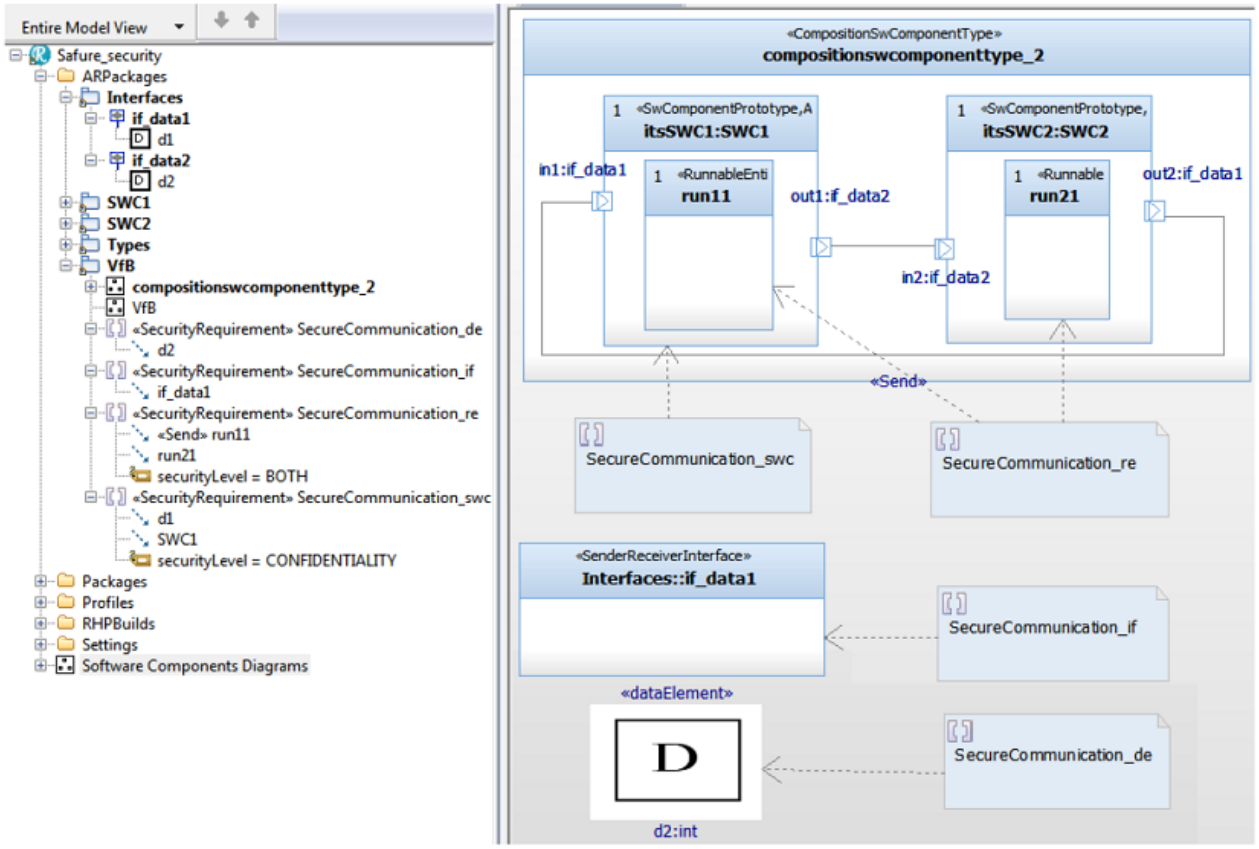


Figure 4.1: Specification of security requirements on communications

Next, the model with the security requirements information is exported using the standard AUTOSAR ARXML format and a parser with python processing scripts (purposely developed in Safure as a technology demonstrator) can be invoked to process the model and alternatively:

- Automatically generate a filter component that provides for the encryption of the data before communication (as shown in Figure 4.2).
- Customize the generation of the RTE code in such a way that encryption is performed before transmission.

When a filter component is generated, the code implementing the calls to the encryption functions is also automatically generated (right side of Figure 4.3). When an RTE implementation is generated, the RTE generator needs to be customized and the encryption is directly encoded in the code implementation of the communication API functions (left side of FIgure 4.3).

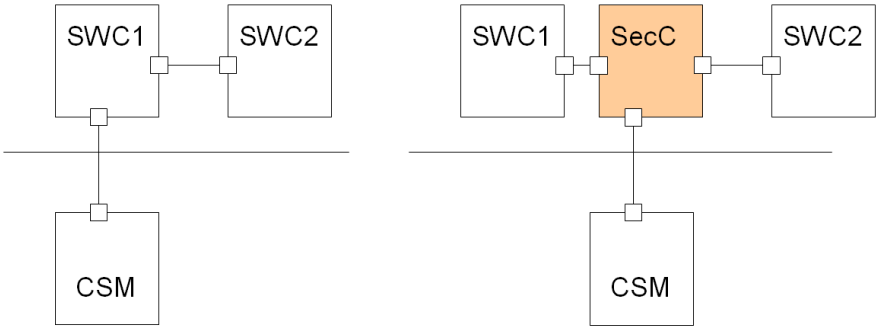


Figure 4.2: A security component performs the encryption automatically

```
FUNC(void, RTE_APPL_CODE) run0(void)
{
   ...
   Rte_Read_Port0in_GPSFilter(datum_buf);
   Csm_SymEncryptStart(cfgId, ..., sizeof(uint32));
   Csm_SymEncryptUpdate(cfgId, datum_buf, ... );
   Csm_SymEncryptFinish(cfgId, ...);
   Rte_Write_Port0out_GPSFilter(*datum_buf);
}
```

Generated RTE code

```
Std_ReturnType Rte_Write_gpsPort_GPS(uint32 datum) {
   ...
   Csm_SymEncryptStart(cfgId, ..., sizeof(uint32));
   Csm_SymEncryptUpdate(cfgId, datum_buf, ... );
   Csm_SymEncryptFinish(cfgId, ...);
   res = Com_SendSignal(sigID, datum_buf);
   ... // AUTOSAR COM
   return res;
}
```

Code in the generated encryption component

Figure 4.3: Generated code for the RTE implementation and the security component

To support our experiments, we implemented a CSM library compliant with the AUTOSAR standard. The library is implemented in C-language and relies on OpenSSL. The services implemented are symmetrical encryption/decryption and MAC generation/verication, to test confidentiality and integrity levels. The results have been presented in workshops, conferences and journal papers.

4.1.2 Automatic generation of OS calls for timing protection (isolation) in mixed-critical applications

This automatic generation process is described in detail in D6.5 and D6.6. The flow is outlined in Figure 4.4

The first step consists in the definition of a model with the extensions provided in WP2 for the specification of criticality levels assigned to the application runnables (functions). Next, the model is exported in the ARXML standard format, complemented by annotations for the modeling extensions. The model is imported in the open tool ARTOP, based on eclipse. In Artop, the model is completed by defining the model of the tasks and the mapping of the runnables, with their execution times into the tasks. The AUTOSAR model in Artop is processed by an RTE generation tool (a code generator) based on the open tool Acceleo. The generated code analyzes the criticality level of the tasks and the timing budgets and automatically adds to the task code the calls to the OS API for enforcing timing protection, by checking that the task execution never exceeds the maximum budget assigned to it. The generated code is compiled with the application code to produce the application that is targeted, for our case study on the multicore TC277 platform. If a timing fault occurs, the RTE code instructs the OS to automatically invoke a callback function, protecting other tasks from a time interference.

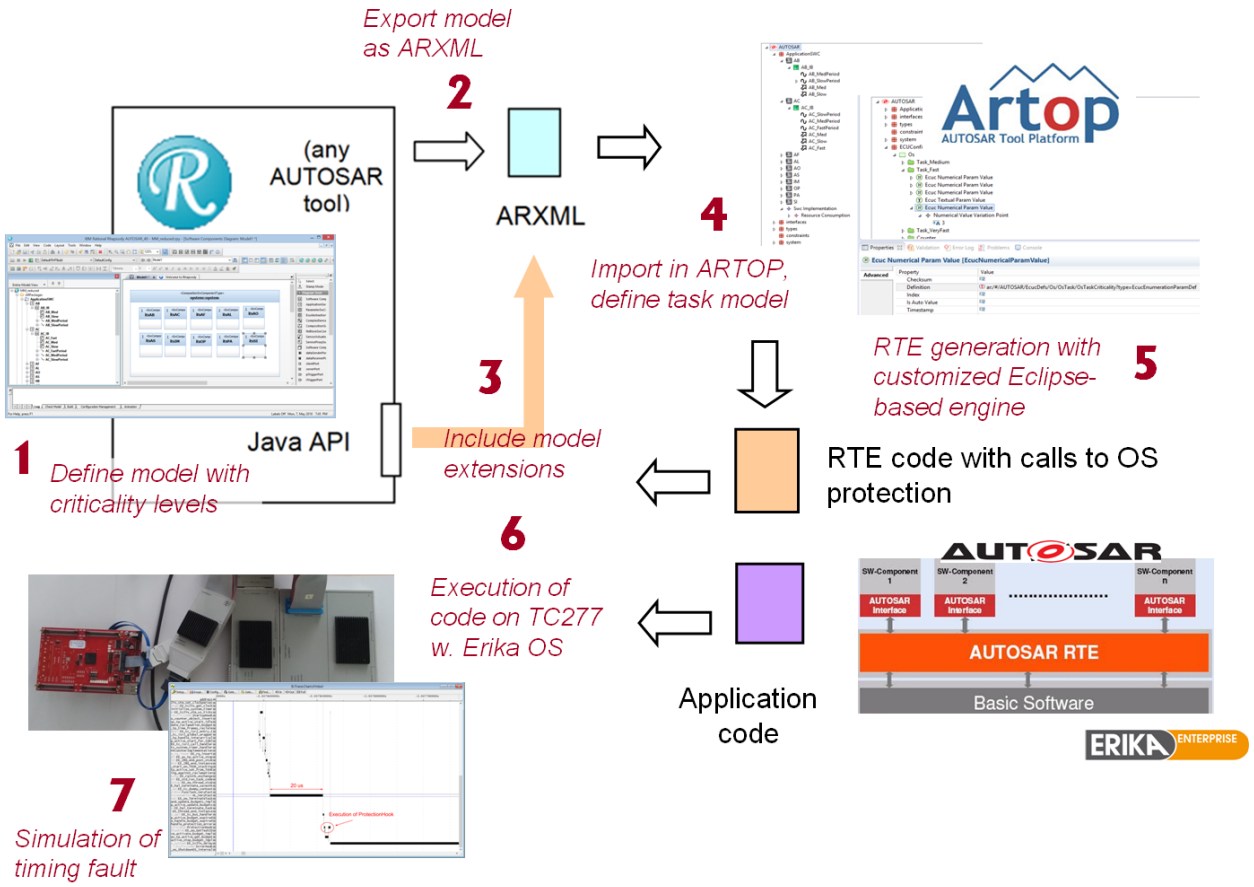


Figure 4.4: Flow for RTE generation for timing protection

4.2 Analysis

The formal worst-case timing analysis for real-time Ethernet standards have been partially implemented in the academic tool pyCPA, as well as in the commercial tool SymTA/S. While both tools have partially overlapping functionalities, they serve different purposes in academic research and industrial application. The pyCPA tool and its setup and usage for real-time Ethernet networks, including different TSN standards [18], has been described in detail in Deliverable D5.1. As for the deployment a commercial tool is required, we focus on the commercial tool SymTA/S here.

4.2.1 Timing Analysis of Tasks in overload conditions

Among the analysis methods of interest for systems with timing constraints, we explored methods for the analysis of systems with temporary overloads, or systems m-K, in which at most m deadline misses can occur for every K instances.

In WP2, we provided modeling features for the expression of such constraints in mixed-critical systems. The constraints can then be processed by a novel analysis method, described in detail in D3.2, and presented at the 2017 EMSOFT conference and the ACM Transactions on Embedded Systems. The analysis method is based on an MILP formulation of the constraints and allows to obtain an exact formulation for many cases of practical interest.

4.2.2 Timing Analysis of Ethernet Networks

In this subsection, we present design-time analysis approach that analyzes the deployment of an application on an Ethernet network. It employs a model-based worst-case analysis which computes worst-possible timing effects for a given network configuration in order to verify that a certain network configuration meets the timing requirements under all circumstances. By systematically modeling possible error scenarios, the analysis can also verify that timing is met under error conditions (like babbling idiots, i.e. faulty nodes sending unexpectedly). It utilizes the worst-case timing analysis for Ethernet networks developed in WP3 and implemented in WP5.

Figure 4.5 shows the overview of the timing analysis workflow.

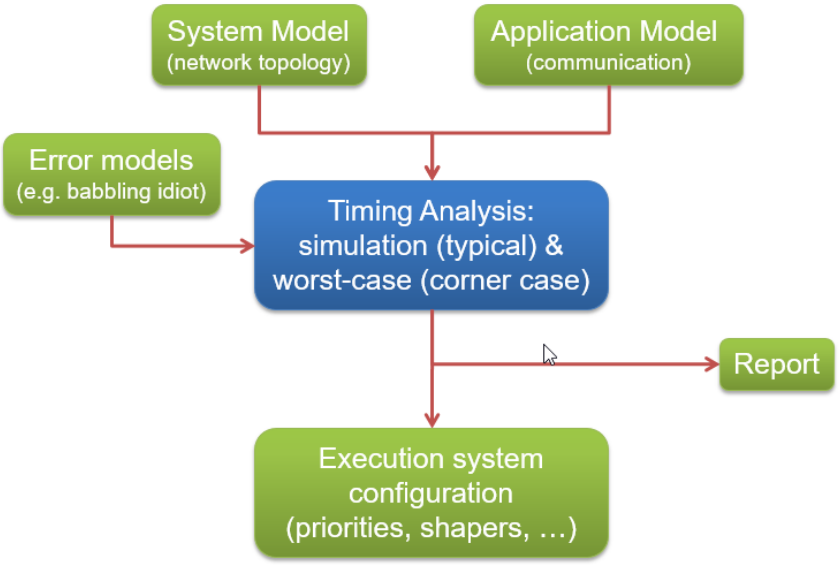


Figure 4.5: Overview of the SAFURE Framework

4.2.2.1 Analyzable Network Model

The proposed timing integrity methodology for networks relies on a model-based analysis of the network. The model consists of

- Topology model (switches, links, ECUs/nodes)
- Traffic model (messages, sizes, activation patterns, priorities)
- Constraints (message deadlines, buffer sizes)

The model of topology and traffic can be created manually in SymTA/S or imported from existing descriptions resulting from the network design process using industry standard formats (e.g. DBC, FIBEX, AUTOSAR), see Figure 4.6.

4.2.2.2 Design Rules for Constraint Specification

To avoid time consuming specification of constraints for individual model elements (e.g. deadline for each individual message), they can be generated according to design rules.

Suggested design rules:

- Worst-case load of all switch ports should be below 80%. This allows some headroom for uncertainties and avoids congestions in case of non-constant loads.

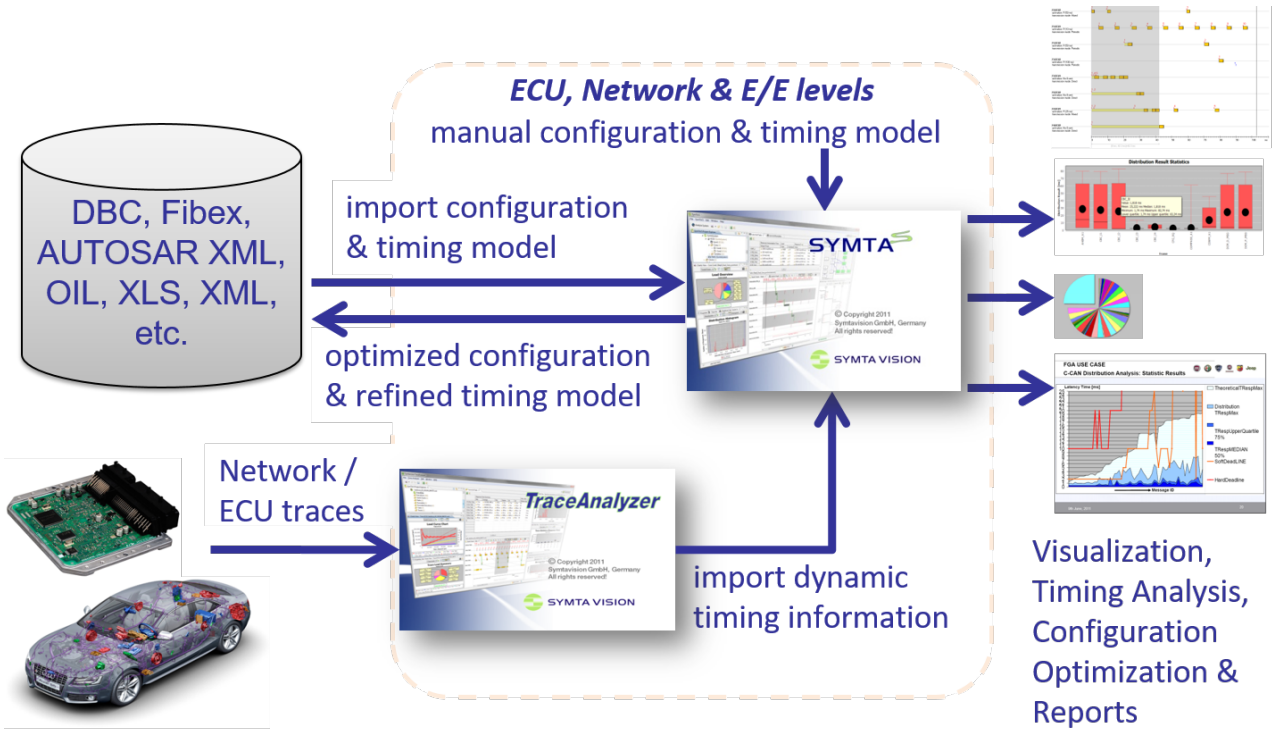


Figure 4.6: Symtavision Tool Suite integrated in automotive developing workflow

- Worst-case load of transmitting ECU ports should be below 50% in normal cases. This avoids single ECUs of spamming the network. This rule can be violated in special situations (e.g. 100Mbit-ECU connecting to Gbit-switch)
- Worst-case latency of periodic messages of the highest priority should be less than their respective periods.
- Worst-case latency of periodic messages of the medium priority should be less than two their respective periods.
- Worst-case latency of periodic messages of the lowest priority should be less than ten their respective periods.
- Worst-case buffer occupancy for all switch ports should be below the physical buffer capacity of the port according to the specifications of the switch. This avoids frame drops for all messages.
- Alternatively (if supported by the switch), the worst-case buffer occupancy can be constrained per priority, according to the priority-partitioning configured in the switch. The constraint should only be specified for priorities transmitting critical messages (where frame-drops are not allowed) or medium-critical messages where no end-to-end protection against frame drops is used (e.g. UDP protocol).
- Worst-case buffer occupancy for all ECU ports should be below the physical buffer capacity of the port according to the available TX memory of the ECU. This avoids frame drops/congestion for all messages.
- Alternatively (if supported by the ECU), the worst-case buffer occupancy can be constrained per priority, according to the priority-partitioning configured in the ECU. The constraint should only be specified for priorities transmitting critical messages (where frame-drops are not allowed) or medium-critical messages where no end-to-end protection against frame drops is used (e.g. UDP protocol).

4.2.2.3 Timing Analysis in Faultless Case

The first step of the analysis methodology is to ensure the timing requirements are met in the error-free case. For this, the worst-case timing analysis algorithm developed in WP3 is used on an unmodified model of the network. It computes upper bounds for the Ethernet port loads, message latencies and switch buffer occupancies that include the worst possible behavior according to the specified system. The computed bounds are compared to specified constraints.

In case all constraints are met, the network timing integrity in the error-free case is guaranteed. If some constraints are violated, the system configuration has to be adapted before evaluating error scenarios. This involves architectural changes and/or changes in the traffic model (which has to be communicated with function owners to ensure system functionality with reduced traffic requirements).

Design guidelines for handling constraint violations:

- Load violation: Optimize topology to avoid bottlenecks; Re-route parts of traffic using static routes (if possible); Reduce traffic by adjusting traffic model (e.g. increase period for periodic messages, if possible); increase transmission speed; use traffic shaping to avoid temporary overloads.
- Latency violation: increase constraint (if tolerable by functionality); reduce frequency of message transfer (if tolerable); increase message priority; optimize topology to reduce distance to receiver and/or reduce bottlenecks; increase transmission speed; implement traffic shaping for same- and higher-priority traffic to reduce long bursts of interference.
- Buffer occupancy violation: Implement traffic shaping to avoid bursts; re-partition buffer allocations; optimize topology; increase transmission speed of outflowing port.

The specific guidelines can be supported by analysis results (e.g. identification of bottlenecks, identification of bursts, ...).

4.2.2.4 Timing Analysis in Error Case of Babbling Idiots

The previous analysis only assured that there are no violations of timing in case all traffic (and components) are behaving as specified. In particular, this means that for instance a low-criticality message is sending only according to the specified message size and activation pattern.

To evaluate the network's integrity against a babbling idiot (e.g. due to a faulty ECU), the babbling traffic needs to be added to the traffic model. This is done by adding messages with very high payload and/or high activation frequency to each ECU that can potentially be a babbling idiot. The priority of these messages should be set to the highest value allowed by the corresponding ECU (assuming priority enforcement is available, e.g. by ingress policing at the switches).

Running the analysis on the modified system will show how critical messages behave under the presence of a babbling idiot. In case no constraint violations (of critical messages) are present, the system is robust against this type of error. If violations occur, they must be addressed by the guidelines above or (possibly in addition) by implementing more/better timing isolation mechanisms (e.g. traffic shaping implemented according to high criticality, switch-level traffic policing).

4.2.2.5 Interactions of Different Design Phases

There are two different points in time in the development phase, where SymTA/S plays a big role to ensure a correct timing behavior within an Ethernet network. At the one hand this is the very early development phase, where the system is conceptionally created and is still not implemented. In this case the customer can take his concept, model it in SymTA/S and can specify his timing for his conceptional system, to fulfill his requirements regarding timing, which he already knows. Or he can just ”play around” with some configurations to find bottle necks and weak points in his conecpt. On the other hand this is the verification phase, where the customer uses SymTA/S to ensure that

his already built system does not violate any timing constraints. The needed input parameters for SymTA/S is taken from measurements of his system then, like e.g. activation patterns or transmitted data lengths (Ethernet message sizes). Once the engineer has discovered a violation, he can go through the points from section 4.2.2.2 to optimize his system.

4.2.3 Thermal Security Analysis

Modern devices on the one hand show increased power density, while on the other hand wanting to minimize energy consumption for cost reduction or lifetime extension. To achieve this energy minimization, management systems are deployed in devices which monitor performance needs, temperature, power dissipation or energy consumption and control the device accordingly. These management systems introduce usage dependent dynamics in the device temperature and other related factors, as for example power dissipation or energy consumption. In the course of this project we identified security implications connected to the dynamics in the device characteristics. We analysed possible covert channels, as defined in D1.3, and side channels, as possible security issues.

4.2.3.1 Covert Channel Evaluation

In order to assess the security threat emanating from a covert channel, the channel capacity can be used as a key factor. Additionally, a threat scenario has to be defined (see use case definition in D6.1) to be able to make a qualitative evaluation of the channel capacity and do an experimental evaluation. This experimental evaluation has to not only contain experiments under controlled laboratory conditions, but also show the robustness of the channel and its performance under realistic conditions.

In D3.1, D3.2 and D3.3 we show the evaluation of the **Thermal Covert Channel**, a covert channel based on device temperature measurements. The capacity of the channel, reported in D3.1, can reach up to 1481 bps or 414 bps, using different derivation methods. The transmission rates under controlled laboratory conditions can reach up to 90 bps for less than 1% error probability. As outlined in D3.2, many of the environmental influence factors on the covert channel can be compensated using advanced encoding strategies. This was shown in an experiment where a SSH key was successfully leaked from a virtual machine. Similarly, we also analysed the power dissipation of a device, as a characteristic that is related to temperature. The analysis of this **Power Covert Channel** as defined in D1.3, reported in the dissemination material, showed that this channel yields higher capacity bounds in the order of 2 to 4 kbps. However, the Power Covert Channel showed to be less robust against external influence factors and only works under controlled conditions. In addition, in D3.2 and D3.3, the existence of a **Frequency Covert Channel** was shown. This covert channel yields capacities of less than 1 bps and higher error rates than the Thermal or Power Covert Channel. Yet, in contrast to the Thermal and Power Covert Channel, the Frequency Covert Channel does not rely on sensor measurements provided by the device but can be established only using timing information which makes mitigation harder.

4.2.3.2 Side Channel Data Leak

In D3.2 we briefly show the possibility for a side channel attack using thermal information, the thermal task inference. Using thermal information it is possible to distinguish different videos that are displayed on a mobile platform. This might allow an attacker to gather sensitive usage patterns to generate a user profile.

4.2.3.3 Considerations for Thermal Security

The thermal security analysis has shown that the threat level is highly dependent on the application scenario and the used devices. Therefore, it is not possible to provide quantitative rules but only give qualitative guidelines. In general, information on thermal characteristics are necessary to employ

the management systems for minimizing the energy consumption and guarantee thermal safety of the devices. However, the access to thermal information and related parameters should be subject to more strict access restrictions. This would eliminate most of the thermal security concerns for modern devices. Further, interferences to which the channels are highly susceptible (see D3.2 and D3.3) can be used to make the data leakage infeasible.

4.2.4 Vulnerability Detection for Multi-Cores

Specific algorithms for vulnerability detection for multicores due to resource clogging have been reported in D3.3, adapted to specific architectures in D4.2 and D4.3, and integrated on the automotive multicore use case in D6.5 and D6.6. In this section we provide details of the strategy followed for their integration on the corresponding hardware platforms and on the corresponding use case, and how these steps should be extrapolated to other platforms and use cases.

4.2.4.1 Integration on SAFURE hardware platforms

Vulnerability detection for multicores builds upon contention models for the access to (on-chip) shared hardware resources. The overall idea of this approach consists of devising tight upper-bounds to the impact on execution time that tasks running simultaneously can have on each other. To obtain those upper-bounds, we build on the Performance Monitoring Unit (PMU) so that, during operation, interference can be monitored interfacing the PMU to detect whether timing violations may occur.

We attempted to integrate this technology on the Qualcomm SnapDragon 810 processor, and integrated it successfully on the ARM Juno SoC and the Infineon AURIX TC27x processor family. The steps followed in the three platforms are as follows (see summary and flow in Figure 4.7):

- **STEP 1**. Reviewing the processor specifications to identify the major on-chip shared hardware resources that may have an impact on execution time if clogged. This basically includes those components in the memory path, which typically correspond to shared interconnects, shared cache memories, shared memory controllers and shared memories. For instance, in the case of the ARM big.LITTLE architecture implementations (the Juno SoC and the SnapDragon 810) this includes the interconnect between cores and shared second level (L2) caches, the L2 caches themselves, the bus connecting L2 caches with memory controllers and the memory controllers. In the case of the AURIX processor this includes the crossbar interconnect, and the shared data and code flash memories.

- **STEP 2**. Identifying those hardware configurations where contention can be controlled and/or upper-bounded, so that appropriate contention models can be built on top.

- **STEP 3**. Also building on processor specifications, we identified the events that can be monitored with the PMU that provide information about the shared hardware resources. Typically, this includes access counters, which may be further broken down into different types of accesses (e.g. hits/misses, read/write operations), as well as stall cycles counters.

- **STEP 4**. Test whether the corresponding events in the PMU can be effectively accessed and understood. This required developing small microbenchmarks with known expected behavior, so that values to be read are known (with high precision) beforehand. Then, by running those microbenchmarks and comparing the readings of those events through the PMU with expected values, we understood whether the behavior of the PMU was as expected.

- **STEP 5**. Test whether those configurations identified in STEP 2 can be properly set. This required reusing some microbenchmarks and creating others to test whether problematic behavior for contention modelling, which should have been disabled with the appropriate configuration, was effectively disabled.

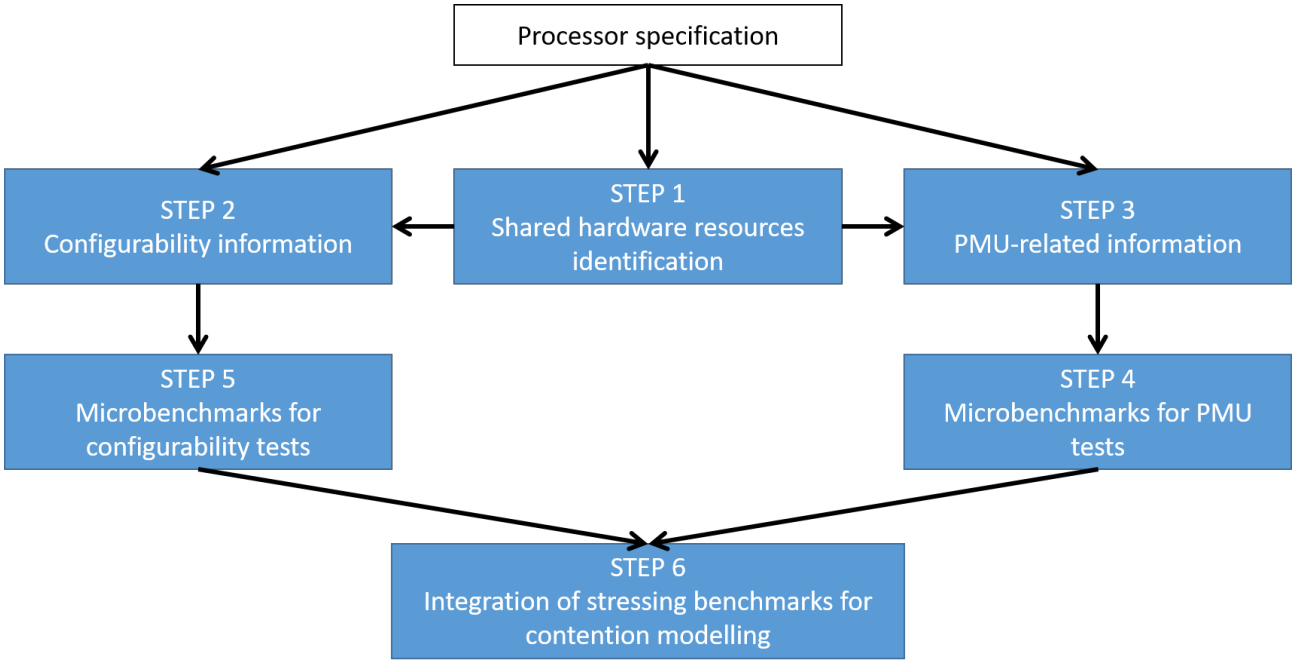


Figure 4.7: Steps for the integration of the contention model on a hardware platform.

- **STEP 6**. Once the configuration and PMU have been mastered, the final step consisted on developing appropriate stressing microbenchmarks allowing to use shared hardware resources intensively to quantify the impact on execution time that contention in each resource may have. This information – in the form of contention cycles per resource and access type – is later used to upper-bound the impact that contention may have in the execution time of any given task under analysis when executed jointly with other tasks.

During this process, we experienced a series of difficulties related to the specific characteristics of each hardware platform. They are described in the following paragraphs.

Qualcomm SnapDragon 810. During STEP 3, we realized that specifications only included information about a limited subset of events, which challenges devising precise contention models for some platform uses. In particular, events that can be monitored span from the core to the shared L2, but events in the memory bus and memory controller are not publicly documented. Thus, if execution time is sensitive to the use of those resources, the contention model may provide overly pessimistic contention bounds due to lack of detailed information.

During STEP 5, we discovered that the appropriate configuration for contention modelling – where prefetchers are disabled – did not work, which became a roadblock for the integration of our technology on this platform. Further experiments during STEP 6 on the default configuration corroborated this fact, which led to dismissing this hardware platform for this technology.

ARM Juno SoC. While this platform is very similar to the SnapDragon 810, thus bringing the same limitations for STEP 3, STEP 5 was executed successfully. This allowed configuring this platform so that contention is tightly modelled if data fit in L1 caches. Hence, although results could be better with more complete documentation, they already allowed providing upper-bounds to contention in some shared resources and hence, enable the use of this platform (with caution) for critical real-time tasks.

Infineon AURIX TC27x. The application of the steps above on this platform revealed, during STEP 4, that PMU events identified during STEP 3 had a different behavior than expected. In particular, specifications provide little detail on the events monitored and, differently to most architectures, the AURIX processor typically counts stall cycles instead of accesses to the shared hardware resources. However, once we adapted our model to the type of events available in the AURIX PMU, the assessment of all the steps was successful and an accurate and reliable contention model was developed.

4.2.4.2 Integration on SAFURE use cases

While originally the contention model was intended to be used on the SnapDragon 810 processor and the telecom use case, the failure to use this platform (also for other SAFURE technologies) led to a platform shift. Hence, we integrated this technology on the AURIX platform used for the automotive multicore use case.

For that purpose, we prepared a software test so that end users (Magneti Marelli) could verify that their hardware platform behaved as expected. This required preparing those tests carried out in STEPS 4, 5 and 6, together with a brief set of instructions for their use. Few iterations allowed to pass this one-time test.

Once the platform was verified, we provided the end user with an instrumentation software package so that they could collect measurements of their application easily, as well as with instructions on how to use it and what results to collect. Again, after few iterations, relevant excerpts of the use case were evaluated and results, as reported in D6.5, prove that contention can be properly modelled and results are tight and low. Hence, this technology provides end users with evidence on the contention that their use case can experience when executed on a multicore, which is needed for scheduling purposes and for overrun detection.

The application of this technology to other parts of the use case was, after that, carried out by the end user on its own, since test and integration steps need to be carried out just once.

4.2.4.3 Beyond SAFURE platforms and use cases

The result of this process is a set of tools and strategies that allow using this technology beyond SAFURE use cases and for other hardware platforms. In particular, the same process followed for the specific hardware platforms considered and use case evaluated can be carried out again *as they are* on other platforms and use cases. However, in order to ease the process and due to the value of this technology, we have reached an agreement with a commercial tool vendor (Rapita Systems Ltd.) to integrate this software technology on their toolset (Rapita Verification Suite) for its commercial exploitation. This work, which was not part of SAFURE DoW, is currently ongoing and expected to complete in the timeframe of 1 year, as described in D7.5.

Some relevant observations regarding such an automation of the integration process are as follows:

- The integration on other use cases for already analyzed platforms is a low risk activity, since challenges risking the applicability of this technology mostly relate to hardware platforms, not to use cases. Moreover, the software package will be commercialized together with consultancy services for its proper integration and verification.
- The porting of this technology to other similar hardware platforms (e.g. AURIX TC29x and TC3xx families) is also a low risk activity since differences across processor families of the same vendor typically neither compromise configurability nor PMU available events.
- The porting of this technology to different hardware platforms is an intrinsically risky activity, since the particular characteristics of the platform or documentation available can easily determine whether the porting will be successful or a failure. In general, the higher the openness of the platform, the easiest the porting, since information related to configurability and PMU utilization will be more detailed. Whenever information is limited, those hardware platforms that intrinsically favor controllability and predictability (e.g. processors from vendors targetting critical real-time systems) are also good candidates.

4.2.5 Timing Interference Analysis

In previous Deliverables, we pointed out that the recent shift of the safety-critical and time-critical industries toward multi-core COTS processors for size, weight and power (SWaP) [8] as well as

efficiency reasons, introduced some major concerns with regards to the **time predictability** requirements imposed by the regulation standards [19, 20, 24].

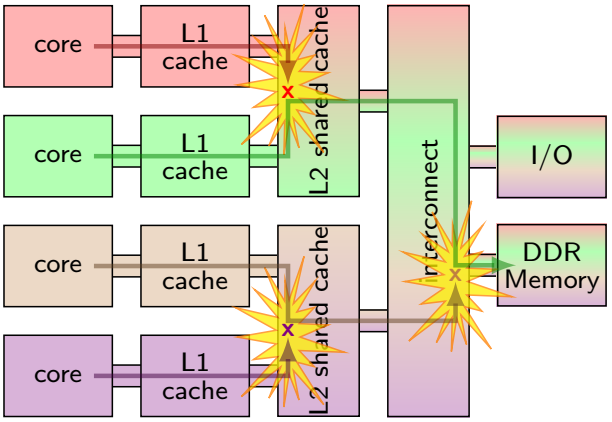


Figure 4.8: Timing Interference in multi-core architectures

At hardware level, as shown in Figure 4.8, concurrent accesses on shared hardware resources are arbitrated, introducing jitter at application level defined as **timing interference** [12]. Such interference, caused by electronic competition on shared hardware resources, are breaking the timing isolation principles required by the industry standards. Therefore the certification authorities has required the industry to analyze and characterize **timing interference**.

4.2.5.1 Tools Developed as part of SAFURE

As part of the SAFURE project, we developed different tools related to timing interference characterization and regulation:

- METrICS: Measurement Environment for Multi-Core Time Critical Systems, detailed in Section 3.3 of Deliverable D4.2.
- xTRACT Visualizer: expert Timing and Resource Access Counting Trace Visualizer, detailed in Section 3.7 of Deliverable D4.2.
- BB-RTE: Budget-Based RunTime Engine, detailed in Chapter 4 of Deliverable D4.3.

4.2.5.2 Measurement Environment for Multi-Core Time Critical Systems (METrICS)

METrICS [14], partly developed in the context of the SAFURE project, is a toolsuite dedicated at performing fine-grain time and resource access measurements in safety critical systems, allowing us to actually measure timing interferences and search for the causes of these interferences.

Performing such kind of profiling in a time-critical context is challenging, as the monitoring technology involved should have a minimal impact on timing. Therefore, classical profiling tools relying on interrupts, system calls, kernel modules and so on could not be used due to their timing intrusivity level. Also sampling techniques are prohibited as safety-critical systems are focusing on the worst case runtime.

The METrICS environment allows us to collect various measurements during the execution of safety critical applications, including full execution time distribution as well as shared hardware access information.

METrICS consists of several components appearing in Figure 4.9, and fully detailed in Section 3.3 of Deliverable D4.2. It is built on top of PikeOS, the Real-Time Operating System provided by SYSGO.

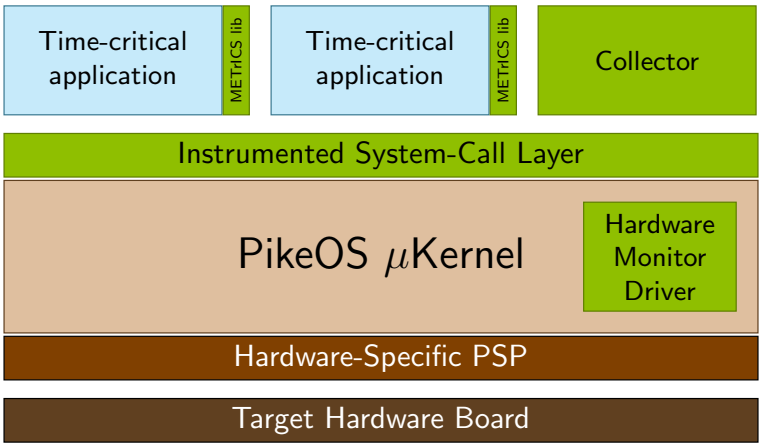


Figure 4.9: Architecture of the METrICS measurement tool

METrICS have been developped with minimizing the timing impact of the environment in mind. Time intrusiveness results, presented in Section 3.4 of Deliverable D4.2, have shown a probing time of less than 110ns in 98% of the cases.

4.2.5.3 expert Timing and Resource Access Counting Trace Visualizer (xTRACT)

With METrICS, we collect large distribution of traces with collected timing and resource access information. Such traces could be very large, with over 110GB of data being collected every two weeks.

Fully automating the data mining to analyze the collected results would be great, but some degree of expertise is necessary to properly filter out the results. We therefore provided xTRACT, a GUI providing the expert with different ways to visualize the collected data, observe the impact of timing interference, and study the correlations between timing and shared hardware resource contention.

xTRACT is coupling statistical filtering tools developped around the open-source pandas module in python, together with statistical vizualization technology based on matplotlib (for scalable rendering) or d3.js (for interactive rendering).

A portfolio of the vizualization available as part of xTRACT is presented in Section 3.7 of Deliverable D4.2.

4.2.5.4 Budget-Based RunTime Engine (BB-RTE)

Within the SAFURE project, we developed the **Budget-Based RunTime Engine** [13]: a regulation solution relying on budgeting the number of shared hardware accesses to guarantee time properties.

The principles, fully described in Chapter 4 of Deliverable D4.3 consist of determining, through analysis, a maximum budget in terms of shared hardware resource access for the low-critical tasks, such that when respecting this budget, high-critical tasks deadlines are guaranteed.

The BB-RTE process, depicted in Figure 4.10, involves two major steps: First, an **offline characterization and analysis** step during which all the budgets are determined. Second, an **online regulation step**, performed by the runtime engine presented in Section 5.2.

The platform characterization step analyses the target hardware platform, relying on a set of low-level (assembly code) Stressing Benchmarks. Each of these benchmarks is responsible for stressing a particular hardware resource with various load levels. Collecting, thanks to METrICS, both the Performance Monitor Counters allowing us to count the number of accesses to the hardware resource as well as the ones corresponding to the observed runtimes allows us to to determine the maximum available bandwidth in terms of accesses to this particular resource.

The critical application characterization step analyses the usage made by high-critical applications of the shared hardware resources. We also, thanks again to the stressing benchmarks, analyse the

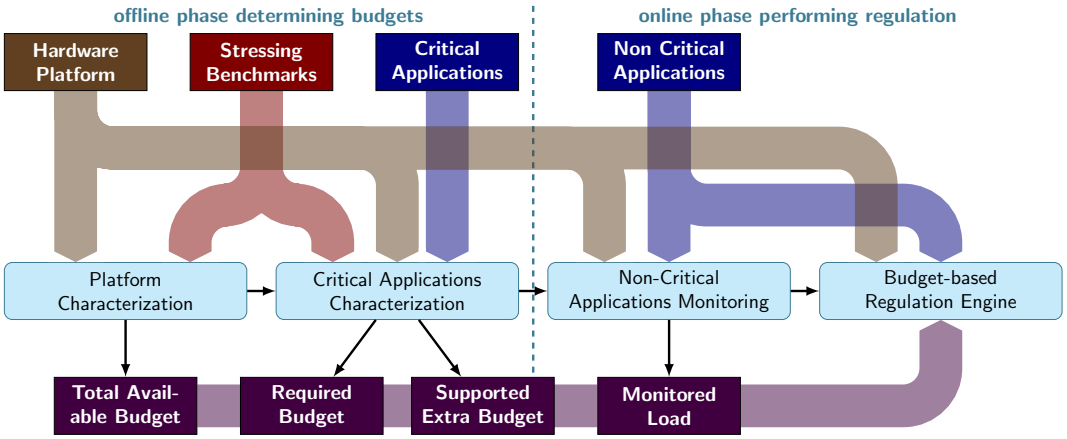


Figure 4.10: Timing integrity process for mixed time-critical systems

level of extra resource access supported by our high-critical applications before being significantly slowed down.

Furthermore details on how the different budgets are determined are provided in Section 4.3 of Deliverable D4.3. We relied on the xTRACT Visualizer tool to observe the corresponding pareto optimum values.

4.3 System Configuration Tools for PikeOS

In Chapter 3.2 ”Network Integrity” of Deliverable D3.3 we have introduced an implementation of VPN on PikeOS to provide a secure communication channel outside the target. We also have described there how PikeOS time partition separation can be used to mitigate threats like timing covert channels due to the sharing of resources such as L2 cache and interconnects. In this section we describe the concrete PikeOS configuration applied for the VPN implementation for mitigating the timing covert channels.

4.3.1 Time Partition Configuration

As it has been described in delivery D3.3 timing covert channels can be eliminated by using a time scheme where trusted and untrusted partitions are not executing concurrently at the same time. This implies using of not only resource partitioning but also time partitioning. Additional L2 cache and TLB covert channels can be complexly eliminated by adding an additional sandwich partition between the trusted and untrusted partitions.

Lets assume that a user application execution time is more critical for us than a network communication performance. Then we decide to have a user_application and vpn_fp ration like 15:3 and also to have a one schedule unit for the sandwich partition.

PikeOS CODEO configuration tool allows to flexibly setup time windows where each time partition is active. Figure 4.11 shows integration project implementing the chosen schedule scheme.

The scheme is represented in two ways.

- Graphical
  Here time windows are placed in a sequential order.
- Numerical
  Here a system integrator can specify parameters like *start* and *offsest* ans also *Time Partition ID*.

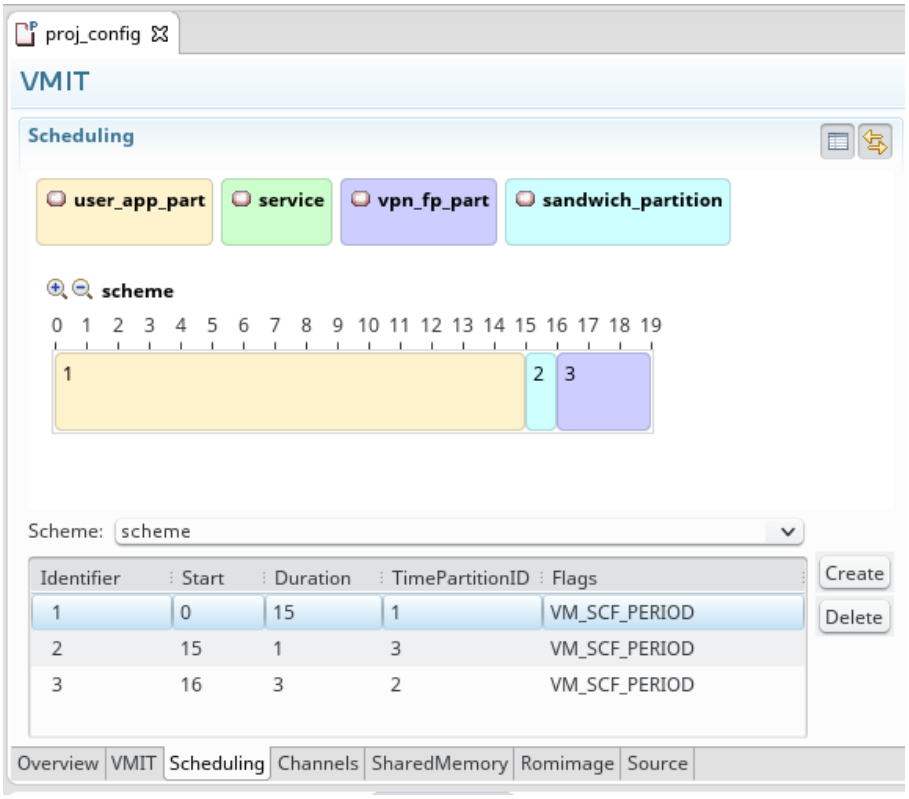


Figure 4.11: Time Partition Scheduling Scheme

A set of time windows creates a scheduling period which can be understood as a repeatable schedule pattern. By variating of number time windows and their length it is possible to create a schedule patterns suites a particular requirement.

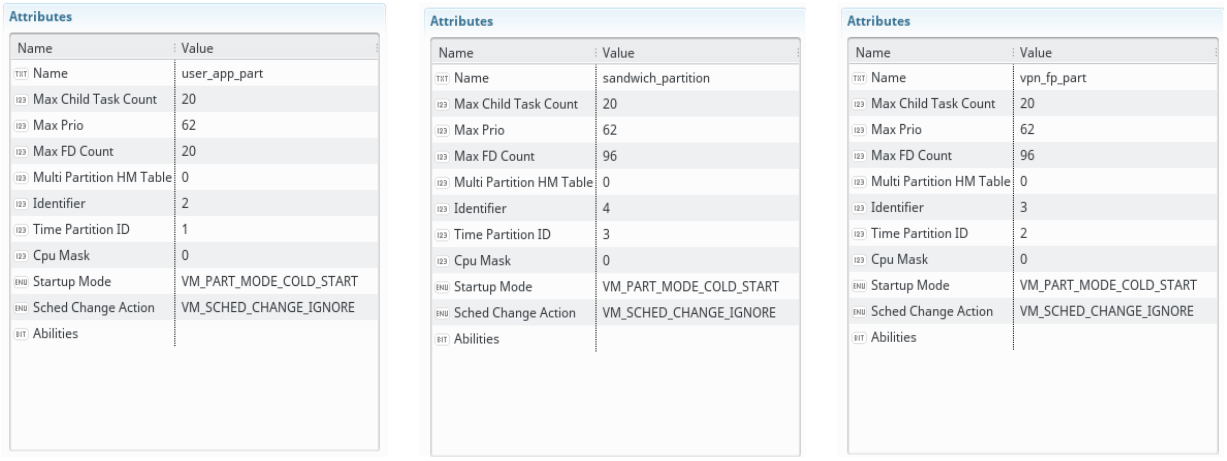
Lets focus on three following resource partition:

- textbfuser_app_part
- textbfvpn_fp_part
- textbfsandwich_part

As it can bee seeing at the picture, a time partition and a resource partition assigned are highlighted by the same color.

The assignation can be done by setting a partition attribute *Time Partition ID* to the ID number of the corresponding Time Partition. Table 4.1 shows the resource partitions with their attributes. *Time Partition ID* of each partition contains ID of the corresponding Time Partition.

In the deliverable D4.3 Section 3.2, we have described the timing covert channels for the PikeOS VPN implementation and the mitigation measures that can be applied to counter these threats. In this section we have presented the concrete configuration using the PikeOS configuration tool that implements the mitigation mechanism described in Section 3.2.2.2 of deliverable D4.3. By applying the configuration described above for the VPN implementation, we can remove the timing covert channels that exist due to the sharing of hardware resources such as L2 cache and interconnects.



Attributes

Name	Value
Name	user_app_part
Max Child Task Count	20
Max Prio	62
Max FD Count	20
Multi Partition HM Table	0
Identifier	2
Time Partition ID	1
Cpu Mask	0
Startup Mode	VM_PART_MODE_COLD_START
Sched Change Action	VM_SCHED_CHANGE_IGNORE
Abilities	

Attributes

Name	Value
Name	sandwich_partition
Max Child Task Count	20
Max Prio	62
Max FD Count	96
Multi Partition HM Table	0
Identifier	4
Time Partition ID	3
Cpu Mask	0
Startup Mode	VM_PART_MODE_COLD_START
Sched Change Action	VM_SCHED_CHANGE_IGNORE
Abilities	

Attributes

Name	Value
Name	vpn_fp_part
Max Child Task Count	20
Max Prio	62
Max FD Count	96
Multi Partition HM Table	0
Identifier	3
Time Partition ID	2
Cpu Mask	0
Startup Mode	VM_PART_MODE_COLD_START
Sched Change Action	VM_SCHED_CHANGE_IGNORE
Abilities	

Table 4.1: Resource Partition attributes

4.4 Conclusion

In this chapter, deployment methodologies were presented during the design phase to help developers to better design mixed-critical systems in terms of safety and security. In essence that were analyses that can be anchored in development processes, configuration tools that can be used by such developers or a support for the synthesis of such systems. A special emphasis was placed on analyses, as they are an essential part of the design phase of a development process. These includes timing, or thermal analyses, for example.

Chapter 5 Execution: OS & Microarchitecture

In Deliverable D3.1, we stated that SAFURE was aiming at ensuring integrity of Safe and Secure systems, especially focusing on the Data Integrity, Timining and Resource Sharing Integrirty and Temperature Integrity aspects.

Previous chapters were dealing with these aspects at design time, this chapter focuses on runtime especially on the requirements at operating system and microarchitecture level. Especaill, the analysis phases performed at design time and presented in Chapter 4 enable us to monitor Integrity at runtime and implement protection mechanisms that will avoid safety or security issues.

The SAFURE framework, previously presented in Figure 1.1, couples measurement techniques (runtime monitoring) with protection mechanisms and some runtime decision engines, impacting online scheduling. These mechanisms are presented in this chapter highlting, for each of these, which aspects of the integrity is focused.

5.1 Ensuring Data Integrity: Protection Mechanisms

Data Integrity refers to assuring and maintaining the accuracy of data. As a consequence protection techniques aim at preventing unintentional changes to information.

At operative system level the AUTOSAR standard introduces the concept of OS-Application as a way to enforce *memory protection* and the concept of timing monitors to enforce *timing protection*. The presence and usage of these protection schemes inside automotive systems is mandatory to target high-level ASIL applications[20].

For the SAFURE Automotive Use Case, it was decided to realize protection mechanisms that are **OS-independent**, so they have been realized as part of the microcontroller firmware architecture. These mechanisms are still strictly related to the OS functionalities, but simply they will not be part of the OS itself. In this way we can apply memory protection and timing protection also on legacy systems to guarantee safety requirements.

5.1.1 Memory Protection Unit

The memory protection unit (**MPU**) allows the creation of SW partitions and it has been built upon the Memory Protection Unit of the Aurix Microcontroller. The Figure 5.1 shows the Memory Protection Unit of the Tricore Aurix microcontroller.

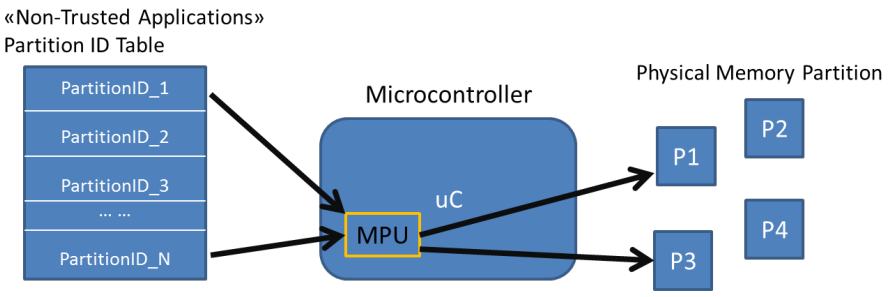


Figure 5.1: Memory protection Unit

The main characteristics are summarized in these points:

- The user can configure one or more partitions IDs.
- Each partition ID must be mapped on a configurable protection set.
- These protection sets will be mapped on the MPU of the microcontroller.
- The MPU will physically create the memory regions specified by the protection sets. An access to a memory region from another un-authorized partition will generate a Memory Protection Error.

5.1.2 Timing Protection driver

The Timing Protection driver (**TPROT**) implements the Autosar Timing Protection Model upon the OS Hook Routines and upon the OS APIs . The Figure 5.2 shows the OS-Independent Timing Protection Model.

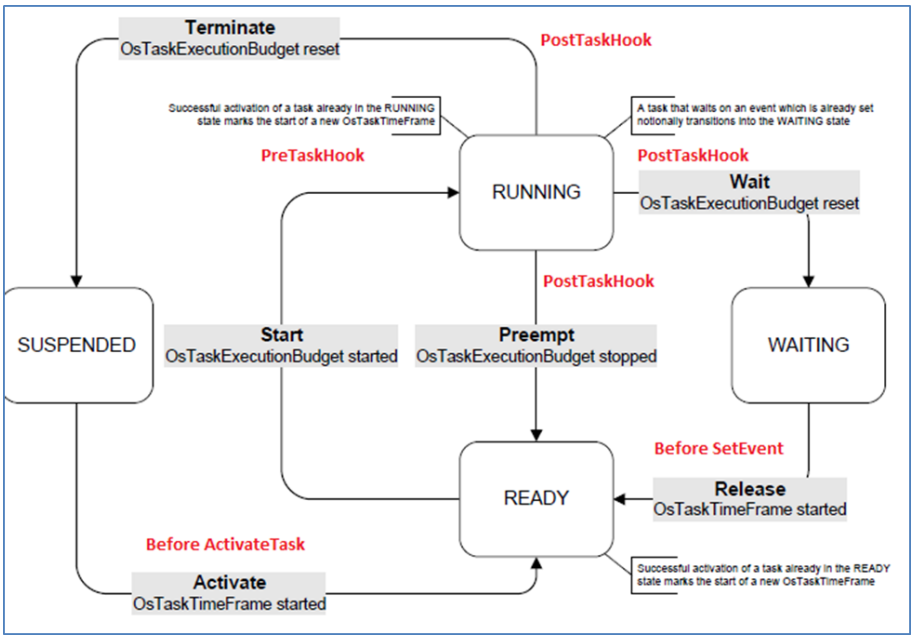


Figure 5.2: OS-Independent Timing Protection Model

The red-colored notes in the above figure represent the interaction points between OS and TPROT. Upon these interfaces, the TPROT functions must be called to realize the three protection schemes according to AUTOSAR OS:

- *Execution Time Protection*
- *Locking Time Protection*
- *Inter-Arrival Time Protection*

Please refer to the Deliverables **D4.1** and **D4.3** for other information on *Freedom of Interferences* for mixed-critical system and specifically for other details on memory protection and timing protection.

5.2 Ensuring Shared Resource and Timing Integrity

In Section 4.2.5.4, we presented the **Budget-Based RunTime Engine** (BB-RTE) [13], that aims at determining a maximum budget in terms of shared hardware resource access for the low-critical tasks, so that high-critical tasks have their deadlines respected.

As depicted in Figure 4.10, BB-RTE also involves an **online application monitoring** phase, as well as the associated **online regulation runtime engine**, ensuring the resource sharing and timing integrity.

5.2.1 Run-Time Monitoring

The run-time monitoring performed by BB-RTE on the low-critical applications is very similar to the one used offline to perform the high-critical application characterization: Thanks to METrICS [14], we gather the number of per-resource accesses performed by the low-critical applications on each shared hardware resource.

When, during a timeslot, the number of shared hardware resource accesses reach the available budget computed during the offline characterization phases (see Section 4.2.5.4), we trigger the run-time engine for it to deal with the risk of a high-critical deadline miss.

5.2.2 Real-Time Scheduling

The expected Run-Time Engine behavior, from a scheduling point of view, is shown in the example of Figure 5.3: During the first timeslot, the low-critical task consumes all its access budget prior to terminating its workload. We therefore expect the run-time engine to suspend the task until the next timeslot; during the second timelsot, however, the low-critical task is able to terminate without consuming all the budget. As a consequence, the run-time engine should not interfere with the task.

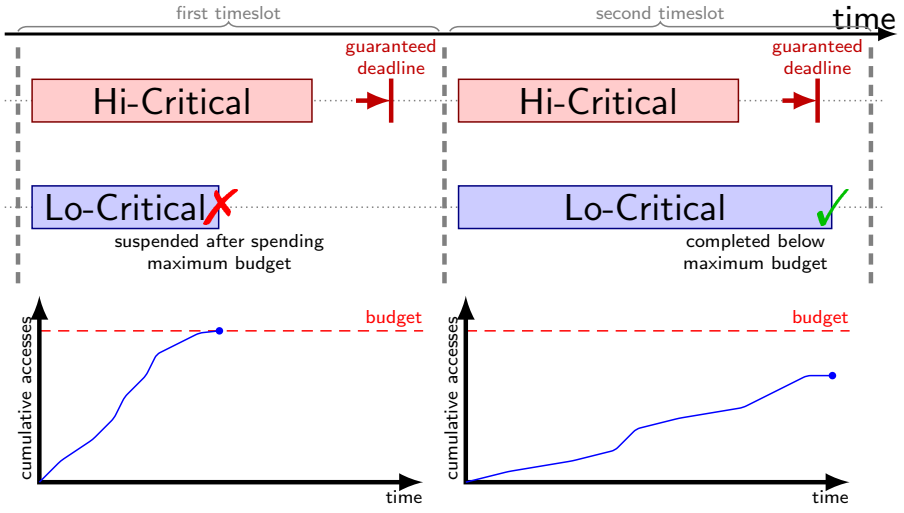


Figure 5.3: BB-RTE: Monitored Information VS Expected Run-Time Engine behavior

Proceeding this way allows us to guarantee, for each timeslot, that the timing requirements of the high-critical task will be met, whatever the behavior of the low-critical task is. Also, the only scheduling action exepected from the run-time engine is to suspend some currently running tasks.

5.2.3 Run-Time Engine

We have build the BB-RTE run-time engine on top of our METrICS characterization environment. The original METrICS architecture, previously presented in Figure 4.9, has been updated to add both the required online monitoring and scheduling action features.

The new architecture appears in Figure 5.4, the most notable change being the METrICS collector replaced by the BB-RTE run-time engine.

Within METrICS, the collector was a PikeOS native partition in charge of 1) defining the shared memory space where each instrumented application will save its collected measurements; 2) configuring specific measurement scenarios (selecting which resource accesses will be counted); 3)

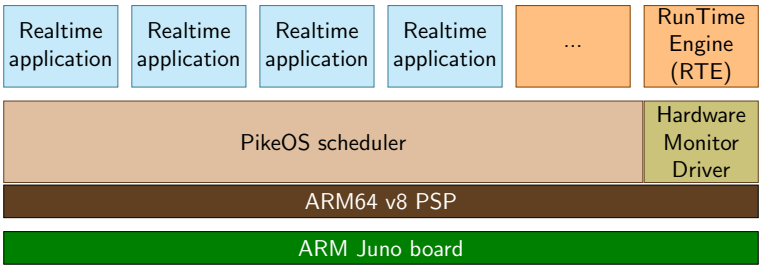


Figure 5.4: METrICS infrastructure in the context of SAFURE BB-RTE

launching the measurement campaign (relying on PikeOS scheduling schemes); 4) transfering the content of the shared memory to the host computer at the end of the measurement to postprocess the collected results

The collector took a particular care not to interfere with the target application operational scheduling, not being schedulable at all when the application is effectively running, and therfore not altering its real-time behavior.

Within the SAFURE project, as we expect the run-time engine to take some scheduling decisions, and therefore to have an impact on the timing, it was no more critical for the collector not to interfere with the application schedule. Also, instead of gathering the collected information into a shared memory for it to be dumped at the end of the execution for an offline postprocessing stage, our run-time engine needs to process the gathered results online, compare them to the budget and optionally suspends some tasks. As a consequence, the BB-RTE native partition now runs as a background process in time partition 0 of PikeOS.

As comparing the collected metrics to a pre-determined budget is performed in just a few comparisons, the timing impact of the BB-RTE while not performing any regulation remains acceptable. When some regulations are required, a budget being spent by non-critical tasks the BB-RTE partiton relies on PikeOS scheduling features to temporarily suspend the non-critical tasks.

An evaluation of the BB-RTE run-time engine has been realized in Section 4.5 of Deliverable D4.3, showing some limitations of the approach with regards to aperiodic tasks whose timing behaviors are poorly captured by the budgeting technique.

5.3 Ensuring Temperature Integrity

Temperature can also cause interference between criticalities. For-instance, a Lo-Critical application can heat up the platform causing Dynamic Thermal Management (DTM) to be triggered. A subsequent Hi-Critical application may get lower service due to this DTM trigger and may even miss its deadline. This violates the certification requirement of isolation between criticalities and therefore, the effects of such thermal interference must be mitigated to ensure temperature integrity.

This section covers the steps taken to add thermal protection to the avionics prototype. The avionics prototype detailed in Deliverable **D4.2**. Deliverable **D3.2** covered the theoretical basis of this scheme and Deliverable **D3.3** covered the application of the scheme on a synthetic mixed-critical taskset based on the avionics prototype, running on an x86 platform. The thermal protection scheme, called Thermal Isolation Servers, is also explained in [2].

In this document we first explain how temperature measurement functionality was enabled in Section 5.3.1. As explained in the section, adding temperature measurement functionality required more effort than initially anticipated. Therefore, the integration of thermal protection could not be completed. In Section 5.3.2, we outline the steps necessary to the adapt the thermal protection scheme for the avionics prototype. This section also presents the results of the initial thermal calibration tests.

5.3.1 Measuring Power & Temperature as part of the Run-Time Engine

As described in Sections 4.2.5.2 and 5.2.3, METrICS relies on the ISA-level timebase to collect precise timing information, and on Performance Monitor Counters to count down the number of shared hardware resource accesses.

Despite both these features being architecture-dependent, mostly all architectures support these features one way or another. This allowed us to make sure that METrICS could easily be portable to different architecture with a limited cost. Nowadays, METrICS successfully runs on top of PikeOS for ARM-v8, PowerPC-e500 and PowerPC-e6500 architectures.

When it comes to measuring **current**, **power** or **temperature** with hardware probes however, every manufacturor has is own implementation approach, and portability can hardly being ensured.

We evaluated the feasibility of gathering such information on the WP4 avionic prototype based on PikeOS 4.1 running on top of the ARM-v8 Juno Board from ARM. In this section, we will present the alternative way the platform allows us to access such information, as well as the difficulties encountered. Finally we will conclude with an assessment on how it could be ported to other architectures.

5.3.1.1 Probing power and temperature on the Juno board

Figure 5.5 represents the ARM Juno motherboard. Beyond the two multi-core clusters of Cortex A53, and Cortex A72 ARM-v8 cores and the Mali GPU, the Juno motherboard embbeds some additional processors, not directly programmable by the user: The System Control Processor (SCP) that is part of the SoC, and the Motherboard Configuration Controller that is off-chip on the motherboard.

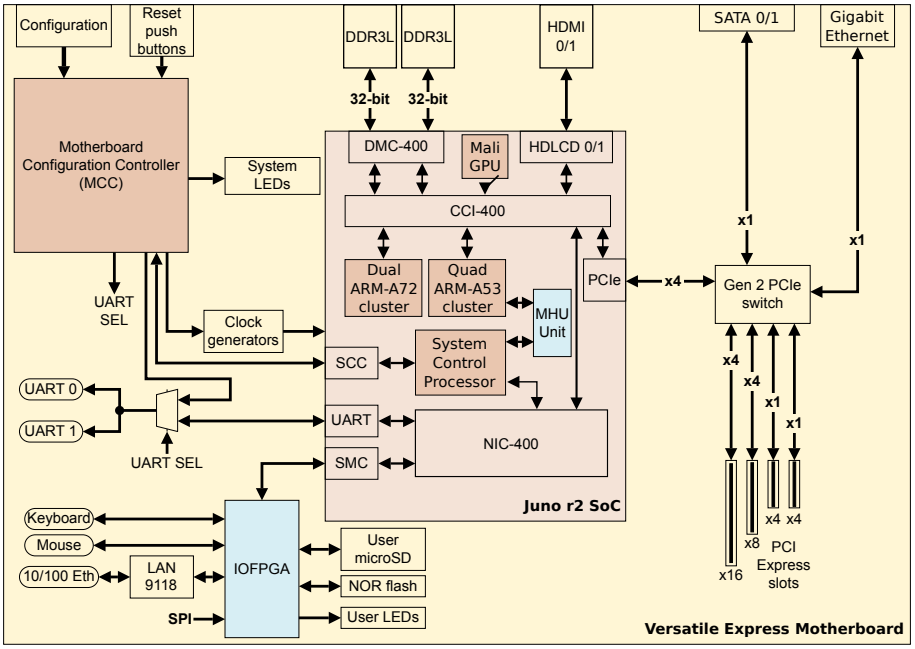


Figure 5.5: Versatile Express Juno Board

Unfortunately, accessing power and temperature information is not directly doable from the computation cores, and we had to rely on these not-directly programmable extra cores and other motherboard peripherals.

During the Juno board boot sequence, the MCC processor of the motherboard is powering up the SoC SCC processor, which is in charge of pursuing the boot process at SoC-level and starting the applicative cores. Therefore, power usage of the applicative cores are controlled by the SCP, itself being controlled by the MCC.

Focusing on power and temperature probing features, two components have access to different sets of information: The SCP processor of the SoC, and the IOFPGA component of the motherboard.

Information about the versatile motherboard and the IOFPGA component can be found in [6], whereas the information on the SoC and the SCP can be found in [5].

5.3.1.2 Motherboard IOFPGA Component

The IOFPGA component of the Juno motherboard is in charge of providing access to low-bandwidth peripherals that the Juno SoC does not directly control through an SMC interface.

The IOFPGA also provides energy registers that measure the instantaneous current consumption, instantaneous voltage suppliedevelopedaneous power consumption, and cumulative energy consumption, for different parts of accessibleJuno SoC and the Juno Motherboard, through memory-mapped APB energy meter registers.

As these registers were memory-mapped, we developed straightforwardly an ad'hoc driver to map this memory region to a virtual address accessible to the application, so that METrICS timing and PMC information could be extended with power information. The results of the execution of an example application with these features turned on is shown below:

```
Starting PowerTemp
 - Getting access to IOFPGA memory regions
   - Physical address: 1c010000
 - Mapping IOFPGA memory region to local address space
   - Virtual address: 503eb000
Probing IOFPGA power information
IOFPGA monitors:
  A72:  0.062992 A   0.909988 V   0.054948 W   0.294994 J
  A53:  0.115637 A   0.911837 V   0.104135 W   0.644928 J
  SYS:  0.869908 A   0.908755 V   0.787316 W   6.832029 J
  GPU:  0.044619 A   0.913687 V   0.038380 W   0.359071 J
IOFPGA monitors:
  A72:  0.062992 A   0.910604 V   0.054985 W   0.351152 J
  A53:  0.052562 A   0.911837 V   0.046681 W   0.693915 J
  SYS:  0.781866 A   0.908755 V   0.707872 W   7.556856 J
  GPU:  0.044619 A   0.913687 V   0.038380 W   0.397166 J
IOFPGA monitors:
  A72:  0.065617 A   0.909988 V   0.054948 W   0.407165 J
  A53:  0.052562 A   0.911837 V   0.046681 W   0.742580 J
  SYS:  0.784494 A   0.908755 V   0.710970 W   8.282830 J
  GPU:  0.044619 A   0.913687 V   0.035981 W   0.435525 J
```

The trace above shows three successive collections of the power information, for the different parts of the ARM-v8 architecture: A72 correspond to the values related to the high-performance 2-core cluster, A53 to the values related to the low-power 4-core cluster, GPU to the Mali graphical unit, and SYS to the versatile motherboard.

The columns correspond respectively to the current in Ampere, the voltage in Volt, the power in Watt and the energy (accumulated power) in Joules. However, there is no temperature information. Motherboard-level temperature was used to be available in Juno revision 0 and 1, but is no more available with revision 2. Anyhow, mother-level temperature is not precise enough to observe covert temperature channel attacks.

Despite being easy to implement, the access to the IOFPGA is not sufficient to observe proper temperature measurements.

5.3.1.3 SoC System Control Processor (SCP)

As its name suggests, the System Control Processor is in charge of controlling the whole System on Chip. In addition to configuration and booting, the SCP is also in charge of power related features such as the DVFS and the DTM. As a consequence, the SCP has access to a wide scope of power and temperature related probes, but it is not programmable.

To access SCP information from the applicative cores (A72 & A53), it is needed to pass through the MHU Unit (Message Handling Unit), that implements a proprietary communication protocol.

The specification of this protocol through the MHU unit is defined in [4]. It encompasses the concepts of both physical and virtual channels, that are used together to support data transfer and request signaling for the messaging interfaces. The whole system relies on unidirectional memory-mapped channels, each composed of three registers (STAT, SET & CLEAR) as well as an interrupt line to the receiver. The communication data flow passes through another set of dedicated memory-mapped buffers.

This protocol has therefore being implemented on top of PikeOS, defining all the associated uncachable physically mapped memory regions, and the results of the execution of an example application with these alternative features turned on is shown below:

```
Starting PowerTemp
 - Getting access to MHU (Interface) memory regions
   - Physical address: 2e000000
 - Mapping MHU interface memory region to local address space
   - addresses: scp_to_ap=0x503fb200 ap_to_scp=0x503fb300
 - Getting access to MHU (Registers) memory regions
   - Physical address: 2b1f0000
 - Mapping MHU registers memory region to local address space
   - Virtual address: 503fc000
Probing SCP/MHU power information
SCP monitors:
  A72:  0.073000 A   0.905000 V   0.004239 W  29.173000 C
  A53:  0.126000 A   0.907000 V   0.000124 W  29.616000 C
  SYS:  0.898000 A   0.904000 V   0.000839 W  29.111000 C
  GPU:  0.060000 A   0.912000 V   0.006438 W  28.505000 C
SCP monitors:
  A72:  0.073000 A   0.904000 V   0.004193 W  29.000000 C
  A53:  0.122000 A   0.907000 V   0.001209 W  29.383000 C
  SYS:  0.809000 A   0.903000 V   0.003388 W  29.107000 C
  GPU:  0.060000 A   0.912000 V   0.006438 W  28.501000 C
SCP monitors:
  A72:  0.073000 A   0.905000 V   0.004239 W  28.770000 C
  A53:  0.122000 A   0.907000 V   0.001209 W  29.598000 C
  SYS:  0.808000 A   0.903000 V   0.002811 W  29.177000 C
  GPU:  0.060000 A   0.912000 V   0.004041 W  28.634000 C
```

Again, the trace above shows three successive collections of the power information, for the different parts of the ARM-v8 architecture: A72 correspond to the values related to the high-performance 2-core cluster, A53 to the values related to the low-power 4-core cluster, GPU to the Mali graphical unit, and SYS to the versatile motherboard.

This time, the columns correspond respectively to the current in Ampere, the voltage in Volt, the power in Watt and the temperature in Degree Celcius. The temperature is provided with a 3-digit precision (actually in millidegree), that is sufficient to observe temperature variation for temperature covert attacks.

The implementation however was much more complex than for IOFGA, requiring months of development and debugging, espcially as the documentation was really sparse, the MHU protocol quite complex, and the documentation on the physical addresses was incorrect.

5.3.1.4 Conclusion

In this section, we presented two alternative ways the Juno board can assess power and temperature. Unfortunately, the more core-specific the probing feature is, the more accurate the temperature information is, whereas for current and power consumption, the generic versatile-level in enough.

From the METrICS point of view, that focuses on portability, this is clearly an issue as it requires some implementation effort to target even another ARM-v8 architecture. Portability is also very tied to the level (and accuracy) of the documentation to access the temperature / power information. If the access to the memory-mapped information of the IOFPGA was quite straightforward, with just a MMU tweak to provide access to the memory range, the access to the SCP via the MHU protocol tooks several months of development, especially debugging as the documentation was 1) unclear without any example, 2) incorrect on one physical address value compromising the whole collection.

As a consequence, assessing the effort required to port the power / temperature colleciton features to another architecture is tricky, from one or two weeks (memory mapped information) to up to 3-6 monthes (for poorly documented protocol information).

5.3.2 Thermal Protection

The thermal protection scheme has two primary components:

1. Building the thermal model based on thermal calibration tests.
2. Based on the model, determine a temperature threshold for throttling the execution for Lo-Critical application.

Note that component 2 is different from the baseline scheme of Thermal Isolation Server (TIS) detailed in Deliverable **D3.2**. In this section, we will first provide brief overview of TIS scheme and the calibration tests for its integration. We will then overview the adaptions necessary for integration of this scheme on the avionics prototype. Lastly we will provide first results of the initial thermal calibration tests.

5.3.2.1 Thermal Isolation Servers Overview

TISs statically scheduled periodic resources with an associated *thermal budget* which encapsulates the maximum temperature increase caused by tasks executed by the server. The thermal budget is a function of the server period, utilization and the core the server is running on. For detailed explanation, please refer to Deliverable **D3.2** or the corresponding publication [2].

In the mixed-critical context (avionics application), the basic idea of thermal protection is the following:

1. Run the Hi-Critical application without any thermal control/throttling.
2. Determine the maximum additional temperature increase such that DTM is not triggered. We call this $\mathbf{\Lambda^{Lo}}$.
3. Mapping the Lo-Critical application to TISs such that deadline constraints are met and the aggregate thermal budget of all servers does not exceed $\mathbf{\Lambda^{Lo}}$.

To enable these individual components, the following thermal calibration tests are required:

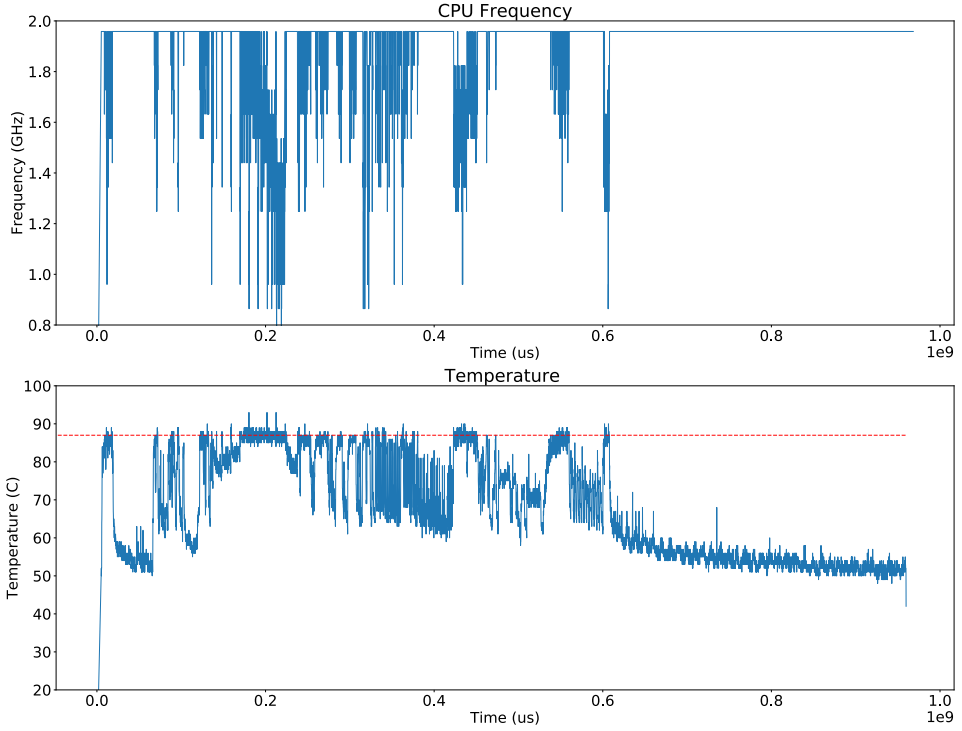


Figure 5.6: Thermal throttling observed on the Dragonboard 810 which has big.LITTLE architecture similar to the JUNO Board. Thermal throttling triggers when the maximum temperature across all cores/sensors reaches 87° C.

Determine DTM trigger temperature:
This is straightforward if it is documented in the SoC reference manual. If not, it can be determined by conducting the following calibration test: Execute a *load*[a] application on all cores without any preemption or idle time. During execution, monitor both operating frequency and temperature by reading the corresponding on-chip thermal sensors. Observe the temperature when there is a sharp drop in operating frequency. This temperature is the point where DTM is triggered. This temperature is called T^{Δ}

[a]For an x86 platform, an infinite loop with floating point operations was sufficient to cause DTM trigger

Determine the maximum temperature caused by Hi-Critical application:
Execute the Hi-Critical application on one of the cores and leave the remaining cores idle. This calibration test should be run over several hyperperiods such that the temperature at the start and end of the hyperperiod approaches the same value (thermal steady state). Furthermore, the frequency of temperature measurements should be high such that temperature peaks are not missed between measurements. The frequency required depends on the power density of the platform. As a general recommendation, based on the power density of modern computing platforms, a sense frequency of 10ms should be used. The temperature profile of this calibration test is called $\boldsymbol{T}^{\mathbf{Hi}}(t)$.

Figure 5.3.2.1 shows thermal throttling observed on the Dragonboard 810 platform. From these two calibration tests, we can determine the maximum allowed temperature increase for Lo-Critical

application. This can be computed using:

$$\Lambda_i^{\mathrm{Lo}} = T^{\mathrm{Hi}} - \max_t T_i^{\mathrm{Hi}}(t) \qquad \forall i \in \{1, 2, \dots m\} \tag{5.1}$$

where m is the number of cores and X_i represents the i^{th} element of a vector. For component 3, we need to determine the thermal model of the platform. This includes the thermal models of the self-core (the core executing the tasks) and the thermal effect on the neighboring cores. Details on developing and applying these models are given in SAFURE Deliverable **D3.3**. However, they are summarized here for completeness.

Transient Model:
load application executed periodically with a period of $10s$ and computation time of $5s$ running on core $i \in \{1, 2, \dots m\}$. All other cores idle. Test duration $60s$. Temperature measurements are taken after every 10ms.

Steady-state Model:
Core i executing *load* application for 40 minutes where $i \in \{1, 2, \dots m\}$. All other cores idle. This is followed by all cores idle for 20 minutes. Temperature measurements are taken after every minute.

Inter-core Model:
Cores (i, j) active for 40 minutes where $i, j \in \{i \neq j : 1, 2 \dots m\}$. All other cores idle. This is followed by all cores idle for 20 minutes. Temperature measurements are taken after every minute.

Using the Transient, Steady-state and Inter-core model, a given server configuration can be simulated to determine its resultant temperature increase; and it can be verified that the temperature increase does not exceed $\mathbf{\Lambda^{Lo}}$.

5.3.2.2 Adaption for the Avionics Prototype

This section proposes adaptions to the thermal protection scheme to fit the avionics prototype. Note that only a proposal for a given scheme is presented here. As indicated in the start of this section, the scheme has not been implemented. Two assumptions that held for the implementation of TIS on the x-86 platform (**D3.3**) do not hold for the avionics prototype. They are:

1. Instead of having a single *load* application, the individual Lo-Critical tasks are different and will likely have different thermal behaviors. This *thermal heterogeneity* of tasks has to be considered in the designed thermal protection scheme.

2. Instead of having a per-core thermal sensor, the JUNO board has one thermal sensor in the A53 cluster. This is a significant limitation and does not fulfill the requirement of a per-core thermal sensor (**D1.2** S1-NF-031).

To adapt to these limitations, it is necessary to make the thermal protection scheme reactive rather than proactive. Each time partition is assigned a maximum temperature budget. During the execution of a partition, the temperature is monitored. If the assigned budget is exceeded, the Lo-Critical application is suspended.

To formally state this, we need to define the following cooling function:
$\mathrm{cooling}(\mathrm{T_{init}}, \mathrm{u})$: Upper-bound of Temperature u idle cycles after an initial temperature of T_{init}. This function can be determined using thermal calibration tests.

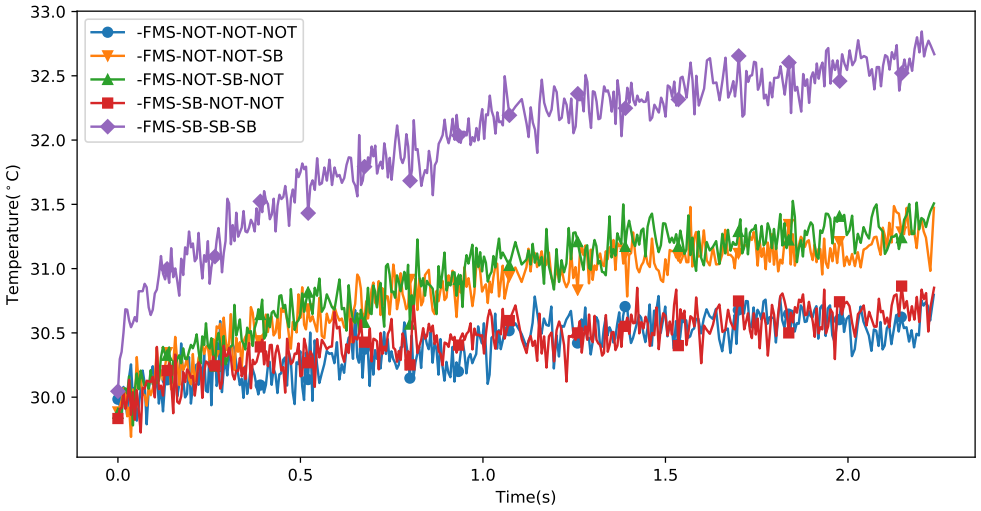


Figure 5.7: Temperature of the A53 cluster in JUNO board for different execution configurations

Using the determined cooling function, we can compute the platform temperature at time t when the initial temperature at time $l < t$ is $T(l)$ and only the Hi-Critical application executes between $(l, t]$. It is assumed that the HI-Critical application starts execution at time 0. The hyperperiod of the Hi-Critical application is represented by L.

$$T^{\text{Hi}}(t, l, T_l) = \text{cooling}(T_l - T^{\text{Hi}}(l \mod L), t - l) + T^{\text{Hi}}(t \mod L) \tag{5.2}$$

Now, let us suppose that there are n time partitions within the hyperperiod of a Hi-Critical application. The starting time of the i^{th} partition is represented by s_i. The maximum feasible temperature at the start of a partition i is represented using $T_i^{\max}$. This can be computed as:

$$T_i^{\max} = \max\{T_{s_i} : \max_{s_i \le t \le s_i + L}\{T^{\text{Hi}}(t, s_i, T_{s_i})\} < T^{\Delta}\} \tag{5.3}$$

Once $T_i^{\max}$ $\forall i \in \{1, 2, \ldots n\}$ are determined, the following thermal protection scheme can be implemented:

Run-Time Protection scheme:
In frame i, execute the Lo-Critical application as long as temperature does not exceed $T^{\max}_{(i \mod n)+1}$. If this threshold is exceeded, suspend the execution of the Lo-Critical application until the temperature at the start of a future frame j is less than $T_j^{\max}$.

5.3.2.3 Initial Thermal Calibration Test Results

To determine the thermal model, several calibration tests were run with different settings of the stressing benchmark application. In this section, we show results of the tests run with the following stressing benchmark setting:
Buffersize = 2^{21} Bytes , Stride = 64 Bytes, NOPs per load = 16.
Following tests were conducted:

1. FMS-NOT-NOT-NOT: Core 0 executing FMS. All other cores idle.
2. FMS-NOT-NOT-SB: Core 0 executing FMS. Core 3 executing stressing benchmark. All other cores idle.

3. FMS-NOT-SB-NOT: Core 0 executing FMS. Core 2 executing stressing benchmark. All other cores idle.
4. FMS-SB-NOT-NOT: Core 0 executing FMS. Core 1 executing stressing benchmark. All other cores idle.
5. FMS-SB-SB-SB: Core 0 executing FMS. All other cores executing stressing benchmark.

Figure 5.3.2.3 shows the temperature of the JUNO board for the different execution configurations. The tests were not sufficient to determine the thermal model, as the temperature steady state is not characterized. However, they do give us insights into the thermal characteristics of the JUNO board. It appears that the execution of the stressing benchmark on core 3 does not significantly increase temperature. However, the execution on core 1 or core 2 increases temperature. Furthermore, core 0 and 1 appear to have similar thermal characteristics. As expected, the execution of the stressing benchmark on cores 1, 2, and 3 simultaneously causes the highest increase in temperature.

5.4 Conclusion

This chapter focused on mechnisms to ensure system integrity at the operating system and micro-architecture level, dealing both with direct aspects (data protection and timing considerations), as well as possible side-channel attacks using temperature as a source of threat.

The next chapter will more particularly focus on safety and securty aspects tied to the network infrastructure, and especially related to data integrity and time integrity.

Chapter 6 Execution: Network

In this chapter we give an overview of the work performed for the network aspects of the SAFURE framework, for both the safety and security. In addition to the work performed, we give an outlook for potential future work and recommendations.

As shown in figure 1.1, the network element consists of deterministic networks, protocol extensions, and anti-counterfeiting measures.

6.1 Deterministic Networks

For open discussion on security threats and mitigation strategies (in particular for automotive industry), we suggest using Common Criteria (CC) [1] as a tool to structure such a discussion. The CC have various benefits: they provide a generic framework that can be tailored towards automotive use, they have been applied to numerous projects in various industries, including safety-critical applications, and, finally, they provide a thorough catalogue of security functions and security assurance methods that target completeness and correctness of a security solution with scalable assurance levels. CC have even been argued for applicability to automotive systems before and even set in context to ASIL. The common criteria allow for the following workflow:

- An organization seeking to acquire a particular type of security product develops their security needs into a protection profile, then has this evaluated and publishes it.
- A developer takes this protection profile, writes a security target, that claims conformance to the protection profile and has this secutiry target evaluated.
- The developer then builds a target of evaluation (or uses an existing one) and has this evaluated against the security target.

For our particular case, the target of evaluation is a deterministic network consisting of Ethernet switches and the wire harness that connects the switches to each other as well as the wire harness connecting the user to the target of evaluation.

6.2 Protocol Extensions

The Time-Sensitive Networking (TSN) group is still in the process of desigining new standards. These standards require a careful analysis in order to be evaluated for their integration into the real-time domain. Within this project we have taken first steps towards providing formal analysis methods for existing and developing standards. For some of these standards, complete analyses are now available within the pyCPA framework, while others can be simulated in the OMNeT++ framework. Furthermore, we extended existing standards, such as the IEEE802.1CB-2017 by allowing temporal reduncancy, either as an alternative to spatial reduncancy or in combination with it.

In addition to evaluating existing standards of the TSN group, we have been working on researching alternative ways of enforcing traffic shaping. The preliminary results are currently under submission. Further details, as well as the venue, are omitted due to the double-blind revision process.

6.3 Anti-Counterfeiting Measures

In a network environment, attackers have an incentive to manipulate the network configuration and the application binaries. As security is a moving target with new vulnerabilities and sometimes even new attack methods coming up, it is desirable for the OEM to be able to update the firmware of the network devices. A Secure Update mechanism can effectively prevent manipulated firmware.

Another entry for attackers is the communication between network devices, which is often unprotected. With Secure Communication, the confidentiality as well as the integrity of messages can be preserved.

For details, refer to chapter 2 in Deliverable D5.2. Furthermore, section 3.2 in Deliverable D3.2 gives an overview of cryptographic algorithms and their performance (ROM size, RAM usage, execution time).

6.3.1 Secure Updates

In order to create a secure update system, the update data (i.e., firmware binaries and/or configuration files) need to be protected either by an authentication tag (when using a symmetric MAC algorithm) or a digital signature (when using asymmetric signatures), which is attached. Optionally (e.g. if the firmware contains intellectual property), the data can also be encrypted. On the receiver side, the authentication tag or signature is then decrypted and verified. The new firmware is flashed into memory only if the verification returns no error.

6.3.2 Secure Communication

To protect the communication between different network nodes, the same cryptographic algorithms as for secure updates can be deployed. Integrity protection can be realized by MACs or digital signatures, and confidentiality can be preserved using block or stream ciphers Due to different requirements, the exact selection of algorithms, key sizes, and key handling depends heavily on the use case. For safety-critical mesages, it should be considered that each crypto algorithm inherently adds some latency to the message. In general, symmetric algorithms have much smaller latencies (and also less complexity) than asymmetric algorithms. For some use cases, it might be necessary to use special hardware (e.g. cryptographic coprocessors) in order to meet the required data latency and/or throughput.

6.4 Conclusion

Future networks will have to be able to cover safety and security aspects, as both have an increasing significance. While some parts of the SAFURE framework focused on one of these aspects, we managed to include both safety and security simultanioulsy, e.g. in the combined automotive use-case, covering both end-to-end encryption and integrity protection, as well as deterministic network scheduling. With this work we laid the foundation for future systems to build on the SAFURE results, which can be used to guide the research into a direction of entwined safety and security for networks.

Chapter 7 Summary

This document presented a framework for various methods and developed concepts to help system developers ensure safety and security in the design and verification phases of their embedded systems. These concepts ranged from development strategies to tools that can be used in the various phases of mixed critical system development.

The individual SAFURE partners validated their respective results using demonstrators from various industrial sectors (telecommunications and automotive) and presented their results in the Deliverables D6.3 to D6.6. The findings derived from these results were compiled in this report and used to design a framework. This framework should serve as a basis for future and similar projects from industry and research.

Figure 7.1 concludes with an overview of the SAFURE partners and to what extent their cooperation and expertise have been incorporated into the framework. Due to their expertise they were decisive to develop the Deliverables and to draw adequate conclusions for the framework from collaboration with other partners in SAFURE.

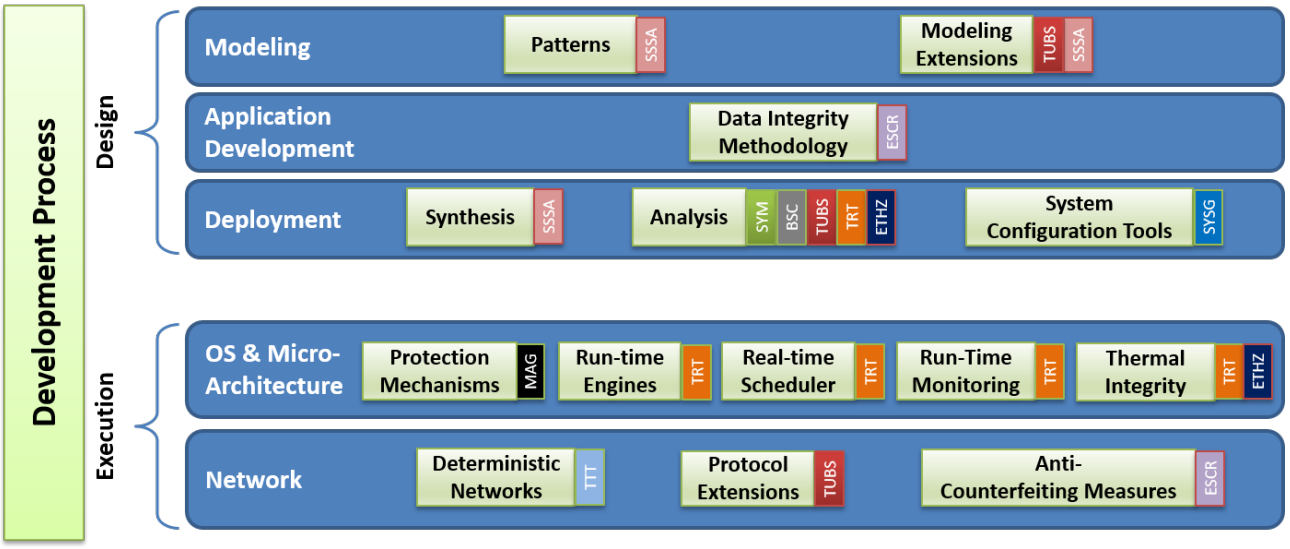


Figure 7.1: Overview of the SAFURE Framework and SAFURE Partners

Chapter 8 List of Abbreviations

AE	Authenticated Encryption
BB-RTE	Budget-Based RunTime Engine
CC	Common Criteria
CPA	Compositional Performance Analysis
CPS	Cyber Physical System
DS	Digital Signature
EC	European Commission
DVFS	Dynamic Volatage and Frequency Scaling
DTM	Dynamic Thermal Management
ISA	Instruction Set Architecture
GPU	Graphics Processing Unit
MAC	Message Authentication Code
METrICS	Measurement Environment for Multi-Core Time Critical Systems
MHU	Message Handling Unit (component of the Juno SoC)
MMC	Motherboard Configuration Controller (part of the Juno motherboard)
MPU	Memory Protection Unit
PMC	Performance Monitor Counters
SCP	System Control Processor (part of the Juno SoC)
SoC	System on Chip
TSN	Time-Sensitive Networking
xTRACT	expert Timing and Resource Access Counting Trace Visualizer

Bibliography

[1] Common Criteria (ISO/IEC 15408): Information technology – Security techniques – Evaluation criteria for IT security, 2009.

[2] Rehan Ahmed, Pengcheng Huang, Max Millen, and Lothar Thiele. On the design and application of thermal isolation servers. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):165, 2017.

[3] ANSSI. Mécanismes cryptographiques - Règles et recommandations, Rev. 2.03. Technical report, Agence nationale de la sécurité des systèmes d'information, February 2014.

[4] ARM Limited. ARM Compute Subsystem SCP, Version: 1.1, Message Interface Protocols, Aug 2016.

[5] ARM Limited. Juno r2 ARM Development Platform SoC, Revision: r2p0, Technical Reference Manual, Aug 2016.

[6] ARM Limited. ARM VersatileTM Express Juno r2 Development Platform (V2M-Juno r2) - Technical Reference Manual, Nov 2017.

[7] AUTOSAR Consortium. Autosar classic platform 4.3.1 standard, May 2018.

[8] Thomas G. Baker. Lessons learned integrating COTS into systems. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS '02, pages 21–30, 2002.

[9] Elaine Barker. SP800-57: Recommendation for Key Management – Part I. Technical report, National Institute of Standards and Technology (NIST), January 2016.

[10] BSI. TR-02102-1: Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Technical report, Bundesamt für Sicherheit in der Informationstechnik, February 2017.

[11] ECRYPT II. ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012). Technical report, European Network of Excellence in Cryptology II, September 2012.

[12] Sylvain Girbal, Daniel Gracia Pérez, Jimmy Le Rhun, Madeleine Faugère, Claire Pagetti, and Guy Durrieu. A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In *Proceedings of the 34th Digital Avionics Systems Conference*, DASC'2015, 2015.

[13] Sylvain Girbal and Jimmy Le Rhun. BB-RTE: a budget-based runtime engine for mixed and time critical systems. In *Embedded Real Time Software and Systems (under review)*, ERTS '18, 2018.

[14] Sylvain Girbal, Jimmy Le Rhun, and Hadi Saoud. METrICS: a measurement environment for multi-core time critical systems. In *Embedded Real Time Software and Systems (under review)*, ERTS '18, 2018.

[15] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, 1994.

[16] Robin Hofmann, Leonie Ahrendts, and Rolf Ernst. Cpa compositional performance analysis. In Jürgen Ha, Soonhoi; Teich, editor, *Handbook of Hardware/Software Codesign*, pages 1–31. Springer, Dordrecht, 2017.

[17] IBM. Rational rhapsody overview page, May 2018.

[18] IEEE Time-Sensitive Networking Task Group. IEEE Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html.

[19] International Electrotechnical Commission. IEC 61508: Functional safety of electrical, electronic, or programmable electronic safety-related systems, 2011.

[20] International Organization for Standardization (ISO). ISO 26262: Road Vehicles – Functional Safety, 2011.

[21] International Organization for Standardization (ISO). ISO/IEC 33001: Information Technology – Process Assessment – Concepts and Terminology, 2015.

[22] NSA. Fact sheet suite b cryptography, August 2015.

[23] National Institute of Standards and Technology (NIST). FIPS PUB 140-2: Security Requirements for Cryptographic Modules, 2001.

[24] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (IMA) development guidance and certification considerations.

[25] Youcheng Sun and Marco Di Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s), 2017.

[26] VDA QMC Working Group 13 / Automotive SIG. Automotive SPICE® – Process Reference Model, Process Assessment Model, 2015.

[27] Haibo ZENG, Prachi JOSHI, Daniel THIELE, Jonas DIEMER, Philip AXER, and Rolf ERNST. *24.4 Packet-Switched Networks: Ethernet*, page 2540. SPRINGER, Apr 2017.