SAFURE

D2.2
Architecture models and patterns for safety and security

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-D2.2
Work Package:	WP 2
Due Date:	Jan 2017 - M24
Actual Submission Date:	31st January, 2017
Responsible Organisation:	SSSA
Editor:	Marco Di Natale
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable is a document describing the selection of the modelling languages and tools for the definition of the automotive and telecommunication architectures of interest and the constraints that must be addressed to specify safety and security requirements (including timing constraints) and enable their automatic analysis or the synthesis of mechanisms that guarantee the required levels of timing, safety and security.
Keywords:	Modeling, Architecture patterns, MBD, MDA, AUTOSAR, security, safety, time

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

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

Editor

SSSA

Contributors (ordered according to beneficiary numbers)

TEC - Christina Petschnigg, Martin Deutschmann
ESCR - André Osterhues, Lena Steden
MAG - Stefania Botta
SYSG - Mikalai Krasikau, Sergey Tverdyshev
SYM - Jonas Diemer
TUBS - Leonie Ahrendts, Daniel Thiele
SSSA - Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, Youcheng Sun

Executive Summary

This deliverable provides the final results of the study on how to model application and platform constraints, metrics and properties that relate in the general sense to safety (including time) and security.
The document provides a summary of the state of the art from scientific research and standardization bodies. Based on the study of the common languages and tools, and the needs of the application context, extracted considering the findings of WP1, the proposals in research papers and other projects, and and the recommendations of standardization bodies, we define an initial set of modeling constructs for time, safety and security.
These constructs are initially represented as abstract, without consideration of an actual language or formalism. Next, they are expressed in UML/SysML and mapped on the AUTOSAR architecture or compatible with the AUTOSAR conceptual framework.
Together with the modeling recommendations, this document provides a description of selected architecture patterns (namely, a separation kernel with possibly a hierarchical scheduler and resource manager and a security encryption module modeled according to the standard requirements) that we expect to be recurrent in the design of mixed-criticality, safety- and security-sensitive embedded systems.
Finally, the report contains a discussion on how to connect the models developed in this work to analysis and synthesis methods and tools and provide a few examples to support the claim on the applicability of the proposed methodology.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This deliverable is the final result of WP2. It contains the background and analysis results and the presentation of the modeling features that are developed in the WP as an original contribution. In addition, the deliverable describes the possible use of the modeling features presented herein for the purpose of the the analysis and synthesis of safety and security components and mechanisms.
The purpose of this document is to provide a summary of the state of the art documents that are relevant for the objective of the workpackage, that is the definition of modeling features for the description of systems attributes and structure, constraints and properties that apply to the general context of systems with safety (including time) and security constraints. The definition of the modeling features is then made practical by describing the proposed representation of the required modeling concepts in AUTOSAR and UML/SysML.
The scope of the research is potentially huge, spanning across domains that include extremely vast areas of research. To limit the scope and provide meaningful results we restrict the analysis to the domain, the application fields and the case studies of the project. This means that the scope will be limited to embedded systems, with focus on automotive systems and concepts applicable to other CPS domains, including telecommunication. Whenever possible, precedence will be given to the needs of the case studies.
The state of the art analysis has been partitioned along two dimensions: according to the source (academic and technical papers, recommendations of standardization bodies and findings/results of other projects) and according to the domain (Safety, Time and Security). In addition, we strive at identifying the architecture patterns that are needed and/or are most likely going to offer potential for reuse in the systems that are targeted by SAFURE.
Next, based on the findings of the state of the art analysis, we identify the main gaps and needs, the opportunities for reusing or building upon existing standards and results and/or the possibility of providing a concrete definition of concepts that we believe are needed but have been limited to abstract descriptions.
The document structure is the following. Chapter 2 provides the results of the state of the art analysis. Chapter 3 provides a discussion of the guiding principles in the derivation of the modeling recommendations from the results of the state of the art analysis. It also defines the guidelines and the process in the definition of the modeling features and a summary of the analysis of the gaps in existing models and technologies. Chapter 4 contains a (preliminary) description of the abstract modeling concepts that are required. Chapter 5 contains the definition of the architecture patterns for safety and security critical systems.
Finally, Chapter 7 provides examples of concrete implementations of the needed modeling concepts in the modeling languages that are selected as applicable to the domains of interest.

1.1 Safety

A safety-critical system is a system whose failure or malfunction may result in death or serious injury to people, loss or serious damage to equipment or environment. Relevant activities in safety-critical systems are:

- Safety requirements identification
- Safety measures definitions
- System safety evaluation

With the increasing complexity of embedded safety-critical control systems, safety is a key issue in automotive system design and development. Safety standards such as ISO 26262 [83] for road vehicles, provide a reference lifecycle to achieve functional safety of E/E systems, based on hazards identification/mitigation and risk analysis.

The objective of functional safety is to reduce the probability of failures to a given acceptable rate in presence of malfunctioning behaviors. Established techniques for quantitative evaluation of dependability are applied for safety evaluation, like Fault Trees and Failure Mode and Effects Analysis.
Model-based development is a promising approach to handle upcoming issues with modern safety critical systems [103]: it allows to manage the system complexity and several methods and tools for model-based safety analysis have been developed.
In the automotive domain, the automotive industry adopts AUTOSAR as the reference architecture. Recently, safety extensions have been added in AUTOSAR [21] to develop safety-related sub-systems that comply with ISO 26262, complementing and integrating the specification of safety mechanisms in AUTOSAR. The extensions provide new concepts, including the decomposition of safety requirements, the traceability and the allocation of both safety requirements and safety mechanisms to elements of the system architecture. Traditionally, the concern of safety is with the consequences of failures; however, since security attacks can have catastrophic effects and can lead to violations of safety, security has been recognized as having an impact on safety. Security-aware systematic approaches to evaluate the effects of security issues on system safety are required [91].

1.2 Time

In CPS (Cyber-Physical Systems), a large number of safety constraints require that the system reacts within timing constraints to guarantee a timely reaction to a dangerous condition (such as the anti-lock braking system), or to guarantee that the system goes back into a safe state or simply to ensure stability of the controls that are implemented in the CPS. Given that time requirements play a special role in the general domain of safety constraints and requirements and dedicated models and languages have been dedicated to the specification and analysis of timing properties, they will be discussed in a separate section.
Multiple requirements belong to the general category of modeling languages for the specification and analysis of timing constraints

- specify all the events that pertain to the domain of time and the computation semantics with respect to time (timing semantics and timed events);
- ensure that a computation completes before its deadline (deadline constraints);
- ensure that a timing fault on a selected part of the application will not affect other parts of the application (timing isolation).

Several notable sources of information exist for the definition of timing properties and constraints as well as modeling languages and analysis and synthesis methods, and several conferences are dedicated to research in the domain of real-time systems. Among those, are the Real-time systems symposium (RTSS) and the Real-time and Embedded Application symposium (RTAS) and in general the other conferences that are part of the CPSWEEK. Research on real-time systems can also be found in design automation conferences (like DAC and DATE, that host sessions dedicated to automotive systems). More recently, a new line of research has explicitly targeted *Mixed-critical* systems, with models and analysis algorithms that are aimed at the analysis of systems in which a time- and safety- critical subsystem interacts with other non-safety critical systems by explicit communication or simply by sharing resources.
Among the modeling languages and standards, the AUTOSAR modeling language is especially relevant for automotive systems. Starting from its 4.0 release, AUTOSAR includes a metamodel definition for the specification of some timing features. The UML/SysML language that aims at the definition of a general system-level modeling languages for CPS has been extended for the needs of embedded and real-time systems by the MARTE profile [3]. Other options include other ADLs (Architecture Description Languages) such as EAST-AADL [1].
Finally, many projects have explicitly targeted the definition of timing models and requirements, the analysis of real-time systems and the development of mechanisms for predictable timing behavior. Among those the TIMMO and TIMMO-2 projects provided the groundwork on which the AUTOSAR timing extensions are based [2].

1.3 Security

The application of security methodologies and mechanisms to embedded systems follows the general guidelines of security in general purpose computing, with a few differences. In general, security refers to several aspects of systems. With respect to communication, security may refer to the following properties:

- ensure that a message comes from a trusted sender (**authenticity**);
- ensure that the information has not been modified on the route from the sender to the receiver (**integrity**);
- ensure that a message is not a replay (**freshness**);
- ensure that data is not read by an unauthorized entity (**confidentiality**);
- the sender cannot deny that he is the author/sender/origin of the message (**non-repudiation**);
- prevent denial-of-service (DoS) attacks from malicious entities that disrupt the communication capabilities (**availability**).

Authentication and integrity of sensitive data and protection from DoS attacks are necessary to protect correct and safe functionality of vehicle systems by guaranteeing that the received data comes from the right ECU (Electronic Control Unit) and has the correct value. Non-repudiation is usually of lesser importance in embedded systems.
In the context of existing standards (such as the Controller Area Network, CAN), the standard configuration of physical level implementations (the CAN protocol is broadcast and multi-master) offers very limited possibilities for the prevention of DoS attacks and a significant change to the fundamental layers of the communication standard would be required to deal with DoS attacks.
In the automotive domain, the security domains/requirements have been classified according to the architecture hierarchy:

- **Vehicle protection** Refers to the need of protecting the entire vehicle from attacks coming from external connections. The main objective is to protect the safety and integrity of the entire vehicle and the privacy of the driver.
- **Network protection** Refers to the need of protecting the network of components and to guarantee the integrity of communicated signals.
- **Node (ECU) protection** Refers to the need of protecting the data and state of the functions in execution on an ECU.

1.4 Definitions and Terms

Additional definitions and terms that are used throughout this chapter are listed in this section. A good reference source for security-related definitions is [107].

- **ASIL** Automotive Safety Integrity Level
- **Authentication** is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (because if a message is modified, the source has changed).
- **AUTOSAR** Automotive Open System Architecture
- **ASW** Application SoftWare
- **BSW** Basic SoftWare
- **CAL** Communications Abstraction Layer.
- **CPS** Cyber-Physical Systems, systems in which a computer system controls, interacts or monitors a physical system.
- **CSM** Communications Security Module.
- **Data integrity** is the property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

- **Data origin authentication** is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some (typically unspecified) time in the past. By definition, data origin authentication includes data integrity.
- **EAST-ADL** Electronics Architecture and Software Technology - Architecture Description Language
- **EMF** Eclipse Modeling Framework
- **Entity authentication** is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired).
- **FAA** Function Analysis Architecture
- **FDA** Function Design Architecture
- **Hazard** A hazard is a potential source of physical injury or damage to the health of persons caused by malfunctioning behavior of the item
- **Hazardous Event** A hazardous event is a combination of a hazard and an operational situation.
- **MAC** Message Authentication Code, a portion of a message that is added for the purpose of allowing verification of the message data.
- **Message authentication** is a term used analogously with data origin authentication. It provides data origin authentication with respect to the original message source (and data integrity, but no uniqueness and timeliness guarantees).
- **Operational situation** An operational situation is a scenario that can occur during the lifetime of a vehicle.
- **PDU** Protocol Data Unit.
- **RTE** Run Time Environment. A layer of software that is automatically generated by AUTOSAR tools to provide the implementation of communication and scheduling in AUTOSAR systems.
- **Safety goal** A property or condition of a system or subsystem that needs to be asserted in order to guarantee safety.
- **Safety relevant failure** failures that are identified during safety analyses to have the potential to lead to a violation of a safety goal
- **Transaction authentication** denotes message authentication augmented to additionally provide uniqueness and timeliness guarantees on data (thus preventing undetectable message replay).
- **Unilateral/bilateral authentication**: In unilateral authentication, one side proves identity. The requesting side is not even authenticated to the extent of proving that it is allowed to request authentication. In bilateral authentication, the requester is also authenticated at least (see below) to prove the privilege of requesting. There is an efficient and more secure way to authenticate both endpoints, based on the bilateral authentication described above. Along with the authentication (in the second message) requested initially by the receiver (in the first message), the sender also requests an authentication. The receiver sends a third message providing the authentication requested by the sender. This is only three messages (in contrast to four with two unilateral messages).
- **WCET** Worst Case Execution Time, of the code implementing a function or task executed without interruption on a given CPU considering all the possible states and input values.

Chapter 2

State of the art and Background

This chapter is dedicated to provide a summary of the state of the art with respect to the modeling requirements, languages and standards for safe, secure and time critical systems. The purpose of the state of the art is twofold. On one side, we aim at identifying modeling features (and analysis methods) that have already been defined and/or proposed and can be successfully reused for our needs. Also, we plan to identify commonalities in the approach or the requirements. At the same time we to plan analyze the standards to identify the constraints that apply to our modeling features. The second motivation is to look into proposed analysis methodologies and algorithms and to derive from them the modeling elements that are required to enable them.
The state of the art analysis will be provided in three main sections, the first section focuses on the results that have been published as academic or otherwise technical papers in conference proceedings or journals. This analysis will provide the scientific backbone, by collecting all the established methods and languages, and, at the same time, aims at the definition of the forefront of the research activities in this domain. The second part of the state of the art analysis focuses on the results of other projects, European, international or national, in the same domain as those of Safure. Finally, the third part deals with the analysis of the existing standards, the recommendations for modeling already developed by standardization bodies and the other standard constraints that apply to our modeling definitions.

2.1 Related Projects, Scientific research, Technical papers

As for all the other sections, the analysis of the scientific state of the art is divided in the three domains of safety, security and time.

2.1.1 Safety

There is an ever growing interest to implement functional safety in automotive industry to ensure the absence of unacceptable risks in modern cars.

SAFE project

The SAFE (Safe Automotive soFtware architEcture) project [134] targets the definition and implementation of best practices for the introduction of safety concepts in the automotive domain, with reference to the ISO 26262 [83].
In SAFE, the following objectives are considered:

- Failure error modeling and propagation to perform safety and cut-set analysis.
- Hardware and software COTS evaluation methods for safety test conformity and integration in safety systems
- Clarification of needs via explicit elicitation of safety requirements and tracing
- Specification of criteria and methods for architecture safety evaluation
- Generation of safety case documentation

Several recommendations in SAFE refer to the needs of automatic code generation or are specific of the modeling environment used for the description of the metamodel (and can therefore be made optional). Also, several recommendations are beyond the original scope and can be better characterized as "good modeling practices".

The project spans all the stages in the development cycle, in accordance with a model-based design methodology. Safety analyses are the central topic during the system development to identify safety relevant failures that can cause hazardous events, show implemented safety measures, and prove the effectiveness of the measures (see Figure 2.1). Safety relevant failures (**Safety critical failures** in the figure) are random hardware failures and systematic failures, typically representing design or coding errors, introduced by the development team. A **Hazardous Event** (in the figure) is defined as a combination of a hazard with an operational situation. **Safety Goals** are specified at the vehicle level to avoid the identified hazardous events. An ASIL (Automotive Safety Integrity Level) is assigned to each Safety goal, according to the severity of hazardous events.

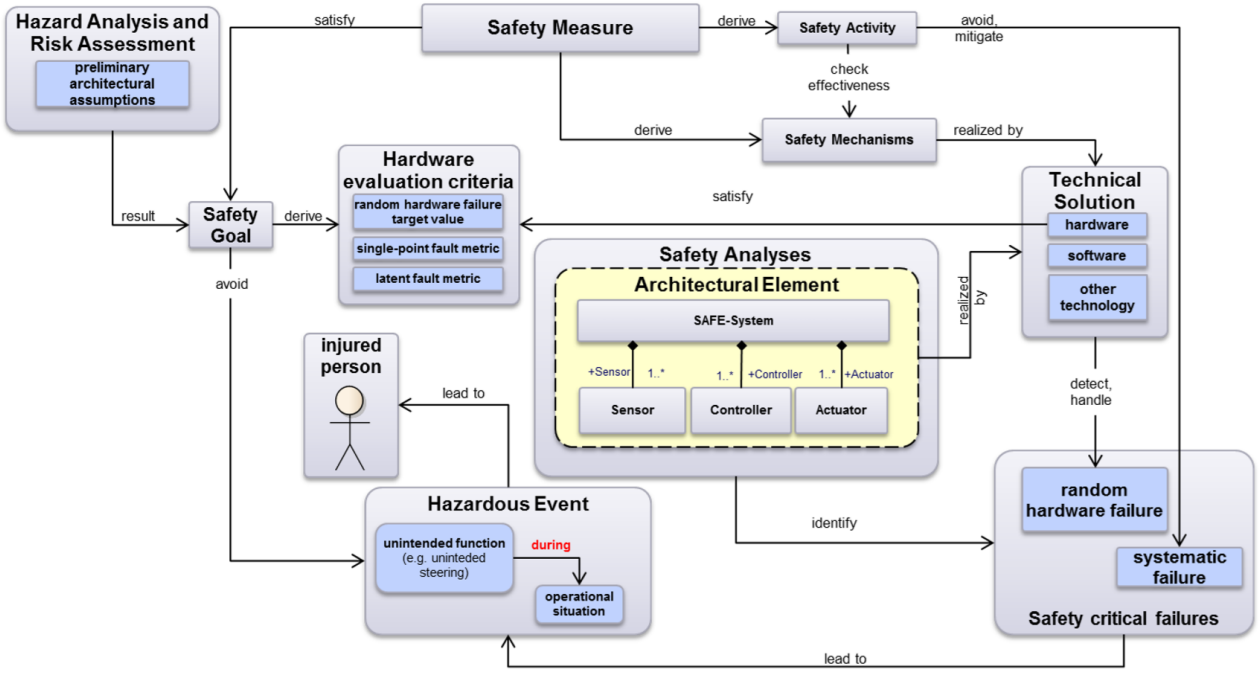

Figure 2.1: System development and Safety analyses ([137]).

Central to the ISO 26262 standard, is the concept of Item. In the standard, an Item refers to a specific system or array of systems that implements a function at the vehicle level to which the safety lifecycle is applied. Typically, the function is a safety-related function, with the potential to cause harm to people inside or outside the vehicle. The Item definition is used as input for the execution of hazard analysis and risk assessment.
Based on the information given in the Item and the results of the **Hazard Analysis** and **Risk Assessment** (in the figure), the safety goals are described as functional safety requirements and allocated to architectural elements of the item. Based on the ASIL allocated to the safety goals defined at the vehicle level, a hardware evaluation criterion (in **Hardware evaluation criteria**) for the affected hardware component is selected.
With reference to the SAFE system architecture shown in Figure 2.2, the vehicle level describes the context of the item as well as the architectural splitting up to several Items; the Item level describes the functionality of the item as well as the splitting up to several (sub)systems; the (sub)system level describes the architectural elements of the system. A system consists of components that are in general characterized by having a sensor, a controller and an actuator. The allocations of elements to software and hardware components, and the interfaces between the components are defined within this level; the Software level contains the architectural splitting of a software system to software partitions, software components and units; the Hardware level contains the splitting of the hardware system to hardware components and hardware parts.
The Item level contains different views of the item: the Item Feature view, that identifies all the safety relevant features of the item; the Item Element view, that identifies all the architectural elements that are used for the item; and the Item Failure view, that reports all identified failures caused by architectural elements or development team members.
The Functional Safety concept is among the main features of ISO 26262. The safety concept is initially created during the concept phase and subsequently refined during the product development at the system level. The Functional Safety concept describes the safety measures (Safety Measure in Figure 2.1) that are needed to avoid the violation of the safety goals (i.e., the features to detect and handle safety relevant failures). The Technical Safety concept identifies specific technical solutions to the safety measures identified in the Functional Safety

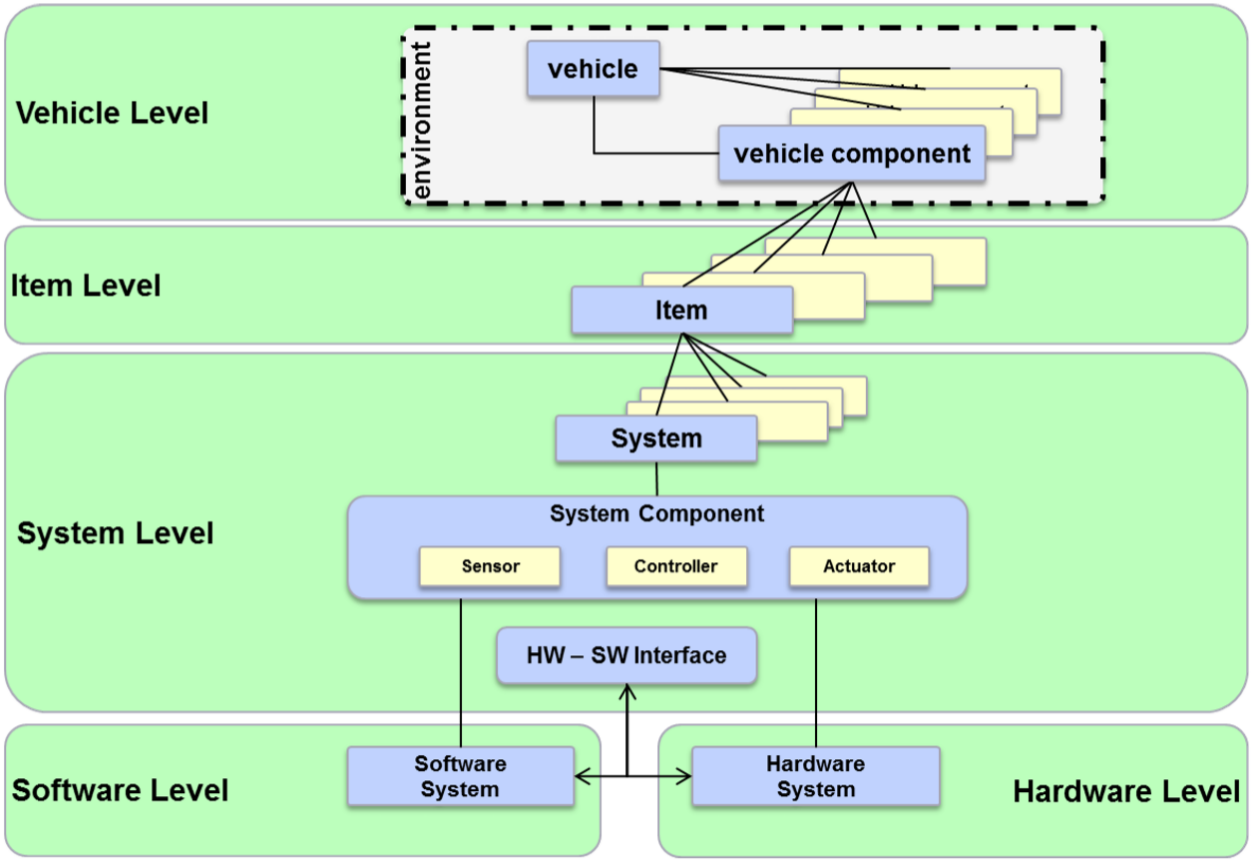

Figure 2.2: SAFE basic system architecture ([137]).

concept (Technical solutions in Figure 2.1).
In addition to safety measures, the Functional Safety concept describes fault tolerance mechanisms, and the allocation of the safety measures to the involved architectural elements.
Traceability of safety requirements is supported in SAFE by the concepts of requirement links and requirement allocations. They are both used for refining safety requirements and associating safety requirements with artifacts of the architecture.
In detail, the project targets the following safety-related activities :

- Hazard analysis and risk assessment
- Functional safety concept
- Specification of technical safety requirements, which is further divided into
  - Hardware safety requirements
  - Software safety requirements

At the end of the conceptual stage (Hazard analysis and risk assessment, Functional safety concept), the SAFE project produced a set of deliverables that define metamodels for the safety-case evaluation and the documentation of vehicle architectures (and/or subsystems).
The metamodel produced by the SAFE project is meant to be generic and not tied to any specific language or technology but adaptable to all of them, as shown in Figure 2.3.
The metamodel is based on experience gathered from the automotive domain and other domains, and on existing techniques and modeling languages, such as EAST-ADL for functional abstraction and AUTOSAR for the software component and hardware abstractions. Moreover, the ReqIF format (Requirements Interchange Format) is used for the specification of requirements.

The SAFE metamodel.

The SAFE metamodel is structured in the following packages:

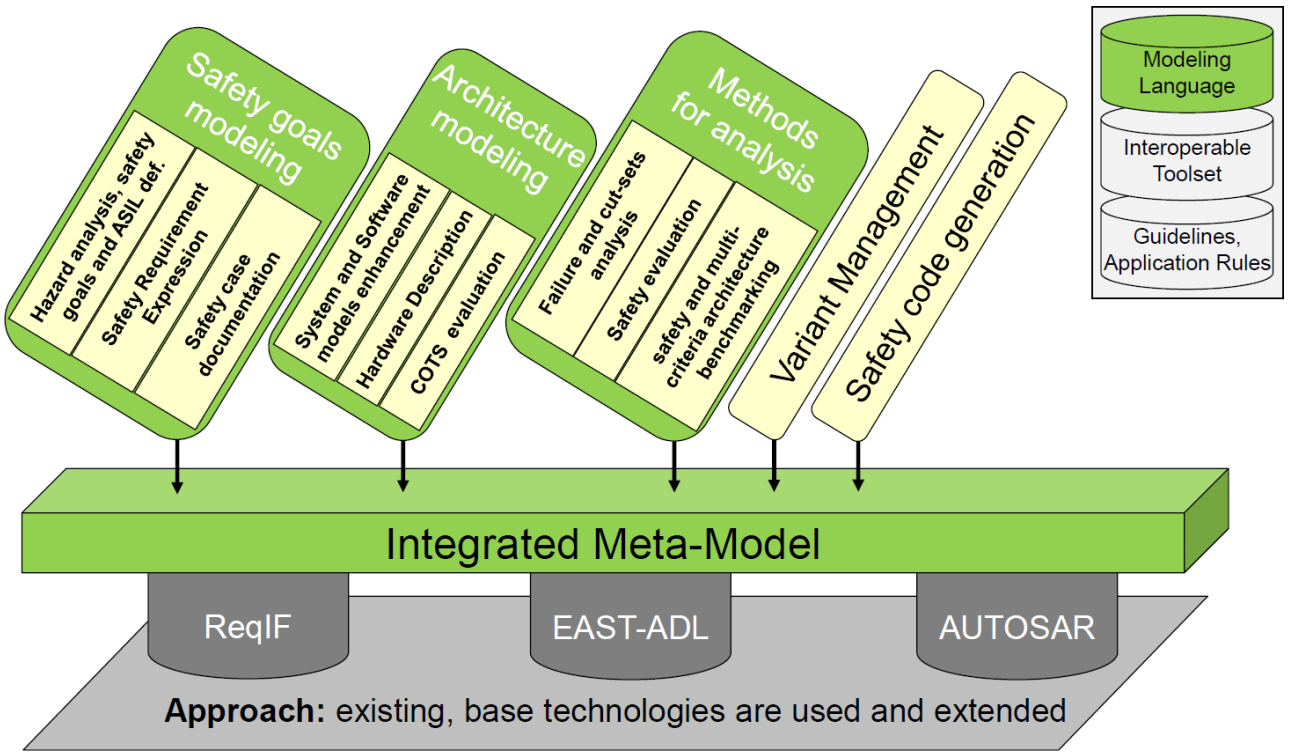

Figure 2.3: The SAFE metamodel structure and organization ([134]).

- CommonStructure: This is a technical package that defines the basic structures
  - AUTOSARInstanceRefs.:to enable the referencing of AUTOSAR InstanceRefs from the SAFE metamodel
  - DataTypes: for all the types of data
  - FormulaExpression: for the definition of a formula language to describe the error propagation
  - References: to enable linking and using external metamodels
  - SafetyExtensions: for the definition of several abstraction level-specific safety extensions of the SAFE metamodel
- Configuration: containing the definition of elements related to variant management
- ErrorModel: for the description of Basic component failures and the results of safety analysis
- Hardware: for the definition of safety extensions at the hardware level
- Hazards: for the definition of hazard, risk, event, controllability
- Requirements: provides links to a requirements perspective and extends safety elements enabling the requirements traceability that are necessary to fulfil a safety process
- SafetyAnalysis: for the safety analysis that aims at the identification and classification of malfunctions
- Software: for the definition of safety extensions at the software level
- System: for the definition of safety extensions at the system level

In particular, the SAFE metamodel uses elements from foreign metamodels (e.g. EAST-ADL, AUTOSAR), located in the package CommonStrucure/References. The referenced element appears in the SAFE metamodel with the same name as the name in the original metamodel.

SAFE metamodel for system and software level modeling.

Figure 2.4 summarizes the safety extensions and packages specified for the SAFE metamodel at the system and software level.

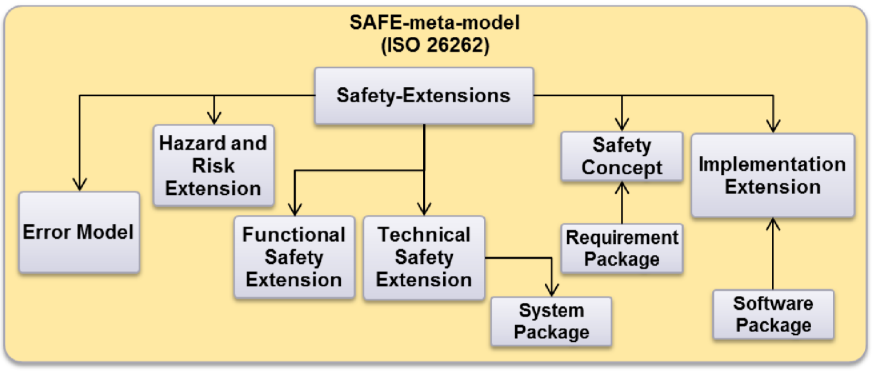

Figure 2.4: Safety extensions and packages specified at system and software level ([137]).

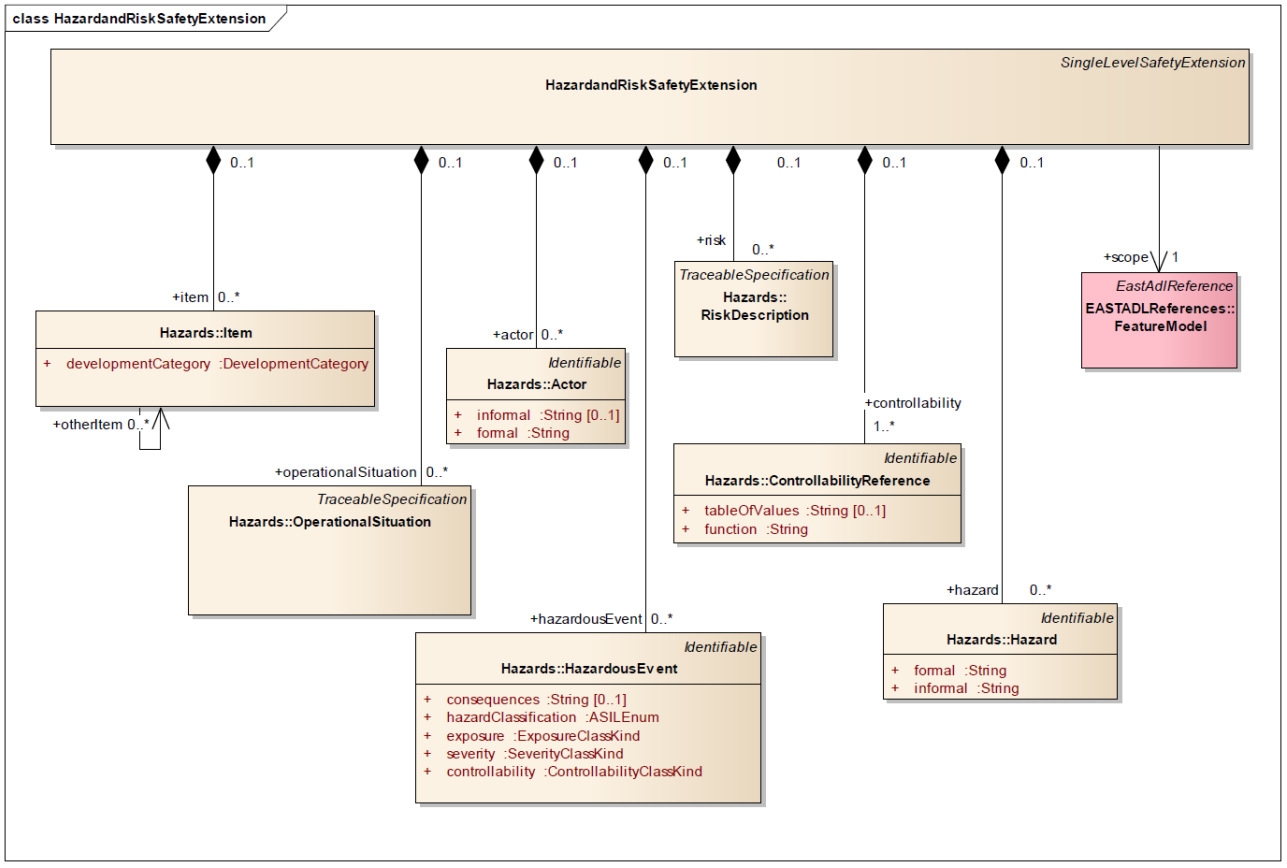

Figure 2.5: The SAFE metamodel for hazards and risks ([137]).

The starting point for the metamodel is the safety-related Item under development according to the definitions of the ISO 26262 framework. The set of modeling concepts is extremely large and will only be summarized here for the portions of interest.
As an example, the SAFE metamodel for hazards and risks modeling is shown in Figure 2.5. The item features are modeled by the feature model that is part of the vehicle level (**FeatureModel** class). The **Hazard** class identifies hazards. The **OperationalSituation** class specifies the operational conditions that include the driver (e.g, gas pedal position), the environment (e.g. nearby obstacles) and other participants (e.g. pedestrians). The **HazardousEvent** class represents an event that is the combination of a hazard with an operational situation. The attributes of the class are consequences, the hazard classification, exposure, controllability and severity. The **RiskDescription** class relates to the risk analysis. The probability of exposure is computed only for operational situations. The classification ranges from Incredible to High probability. The severity classification is taken from ISO 26262.

The SAFE metamodel for Functional safety is shown in Figure 2.6. This metamodel specifies the parts that are needed to model the functional safety concept defined in the ISO 26262.
The **FunctionalSafetyRequirement** class specifies the measures that are needed to avoid the violation of safety goals. In addition, the Functional Safety package describes the safe state for the specified goals, and the allocation of the safety measures to the involved architectural elements. The traceability of the features that causes the failure and the safety measure to handle the failure are part of the functional safety concept.

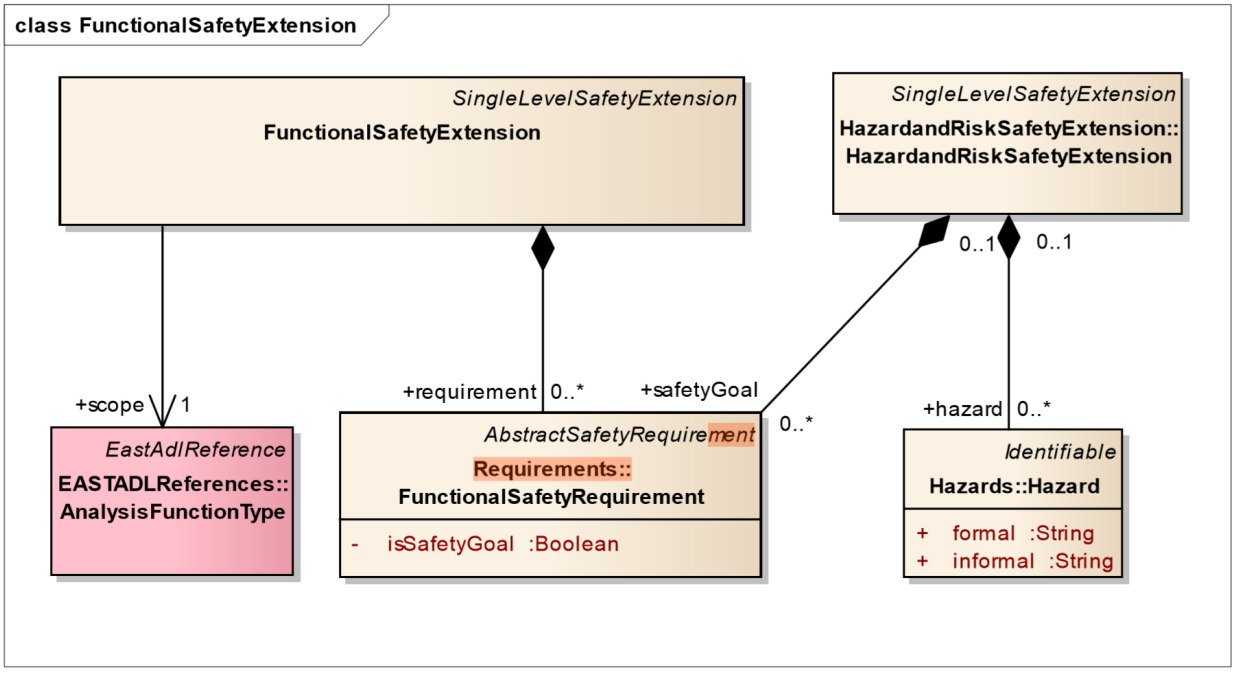

Figure 2.6: The SAFE metamodel for Funtional Safety ([137]).

The Technical safety metamodel (in Figure 2.7) specifies the safety mechanisms aimed at the detection and control of random hardware failures and at the avoidance or mitigation of systematic failures. The package contains the specification of the technical solution that is put in place to realize the specified safety mechanisms. Technical safety requirements are allocated to architectural elements and ensure traceability. The metamodel specifies the specific technical solutions based on the functional safety concept. It contains the hardware software interface (HW-SW Interface in Figure 2.2) specification (**HardwareSoftwareInterfaceSpecification** class) and the technical safety concept (**TechnicalSafetyRequirements** class).
The SAFE metamodel for the Error model is shown in Figure 2.8. The Error model is a container for all the artifacts that are needed to describe the error model of an architectural element: malfunctions, error types and error behaviors. Malfunctions and failures propagate through the complete system model, using the concept of fault failure propagation link (**FaultFailurePropagationLink** class).
Figure 2.9 shows the **Error model prototype** as specified in SAFE for a concrete function instance.
The SAFE metamodel in Figure 2.4 includes the packages:

- System Package
  contains the extensions that are needed to cover a safety architecture at the system level. In addition to the Functional safety extensions and the Technical safety extensions, the package contains the Safety measures and mechanisms to avoid, mitigate, detect or control safety relevant failures.
- Software package
  contains the elements for the definition of the safety architecture at the software level. The schema for a generic safety relevant architecture element at the software level is shown in Figure 2.10. In particular, software safety requirements are derived from the technical safety requirements and are drive the software development. Software partitions are identified to satisfy the independence of safety requirements. SW partitions (Software Partition 1, Software Partition 2) are designed with sufficient independence to ensure that a software feature realized in one partition cannot compromise a software safety requirement defined for any other software partition.

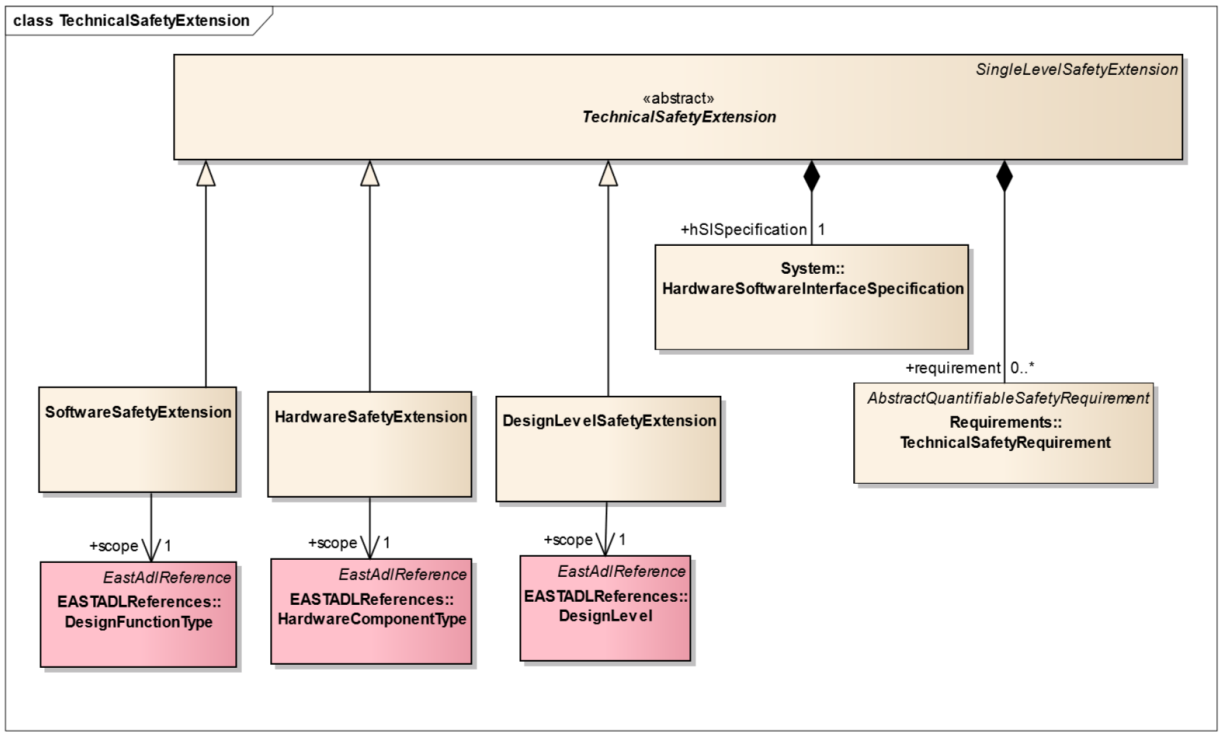

Figure 2.7: The SAFE metamodel for Technical Safety ([137]).

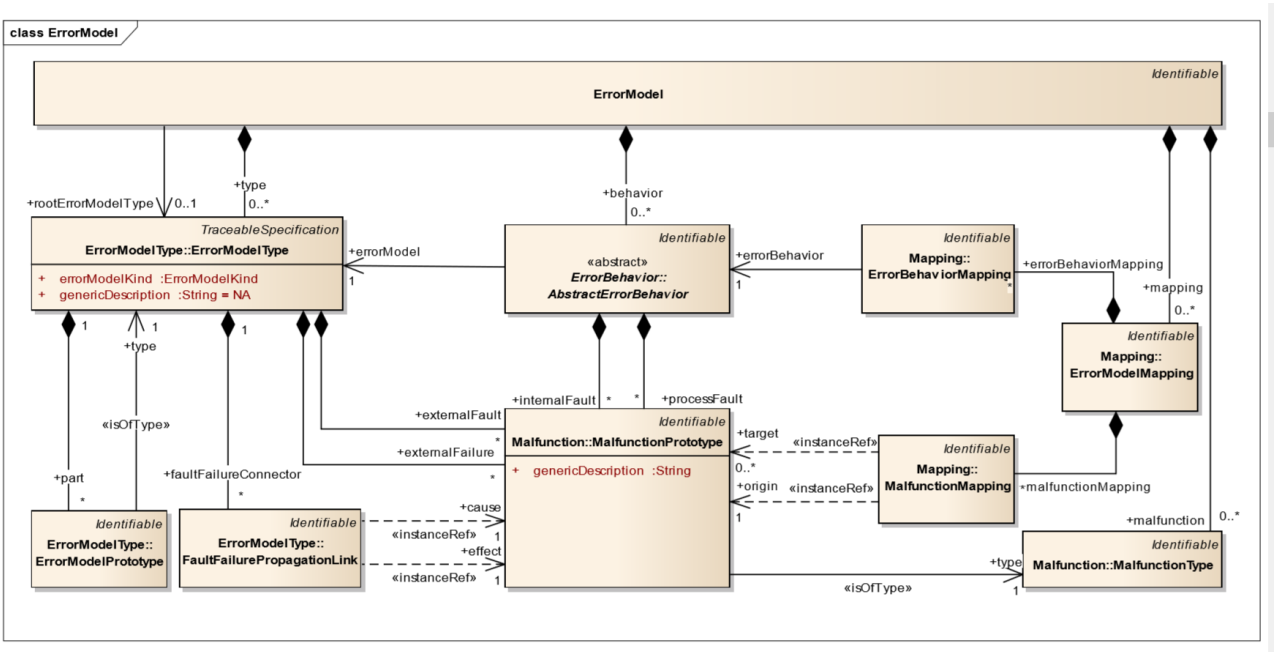

Figure 2.8: The SAFE metamodel for the Error model diagram ([137]).

In addition, software safety mechanisms are allocated to the architectural elements at the software level. Software verification is planned following the activities of software unit testing, software integration and testing, verification of software requirements, verification of calibration/configuration data, and others. AUTOSAR elements are integrated into the software architecture as shown in Figure 2.11 in which the AUTOSAR modeling entities appear in place of the generic concepts of Figure 2.10. Before the integration of the software components, each AUTOSAR SW-Component must qualified to provide evidence for its safe use.

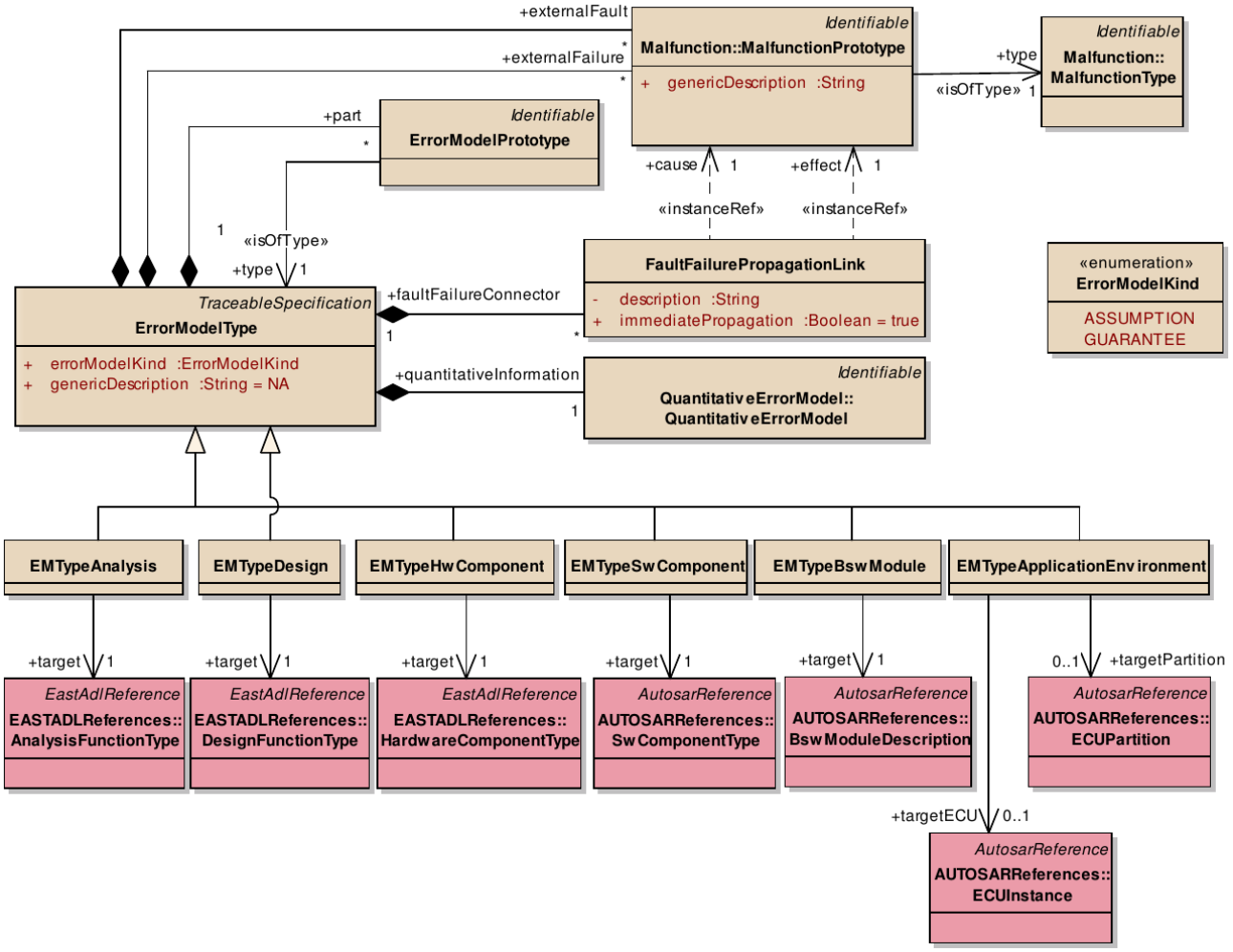

Figure 2.9: Error model prototype ([136]).

- Requirement Package
  contains the categorization of safety requirements into groups: Functional Safety Requirements, Technical Safety Requirements, Software Safety Requirements and Hardware Safety Requirements. Moreover, each safety requirement is assigned a sub-category that specifies the use case of the requirement (e.g, the sub-category *Process* means that the requirement describes safety relevant verification methods; and the sub-category *Product* means that the requirement specifies the technical solution to fulfil safety goals).

Safety goals are evaluated and classified according to ASILs. In the assignment of the ASIL level, three parameters are considered: exposure (how often people involved may be put at risk), controllability (how well the individuals involved can handle the problem) and severity (seriousness of the consequences). Safety goals are implemented in accordance with the classified ASIL, and each system design element inherits the highest ASIL from the technical safety requirements that specify mechanisms realized in the elements, as shown in Figure 2.12. If an element of an AUTOSAR specification is characterized by an ASIL level, the element is in the scope of an ISO 26262 development.

SAFE metamodel for Hardware modeling.

At the HW architectural level, hardware subsystems and components are split down to the level of the hardware elementary elements.
The basic steps in the hardware modeling are [135]:

- Capture the Hardware Technical Safety Concept;
- Complete the HW Component Failure Propagation on the Hardware Architecture;

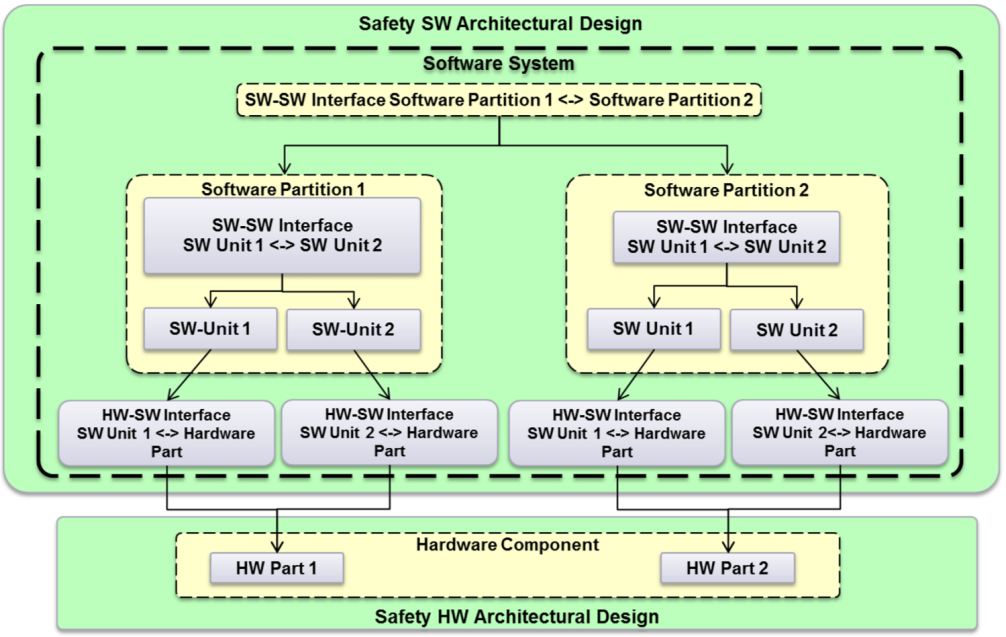

Figure 2.10: Software architecture element ([137]).

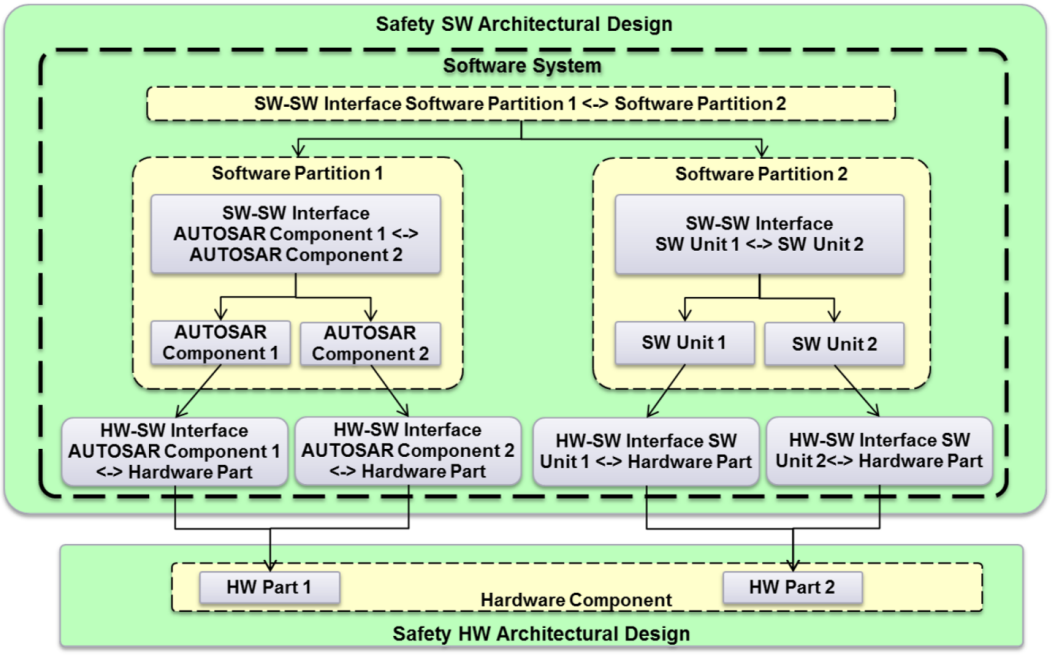

Figure 2.11: Integration AUTOSAR element and SW-architecture ([137]).

- Define the target values for all the HW Components and calculate the corresponding metrics;
- Define the Hardware Part Allocation and Malfunction;
- Develop the Electronics Schematic (by capturing all electronic Hardware Parts as Hardware Elements in AUTOSAR);
- Perform the Electronic FMEA and evaluate its contribution to the HW Component malfunctions;
- Verify the Component Metrics and the Probabilistic values.

In particular, in the Failure propagation on hardware architecture step, the following activities are executed:

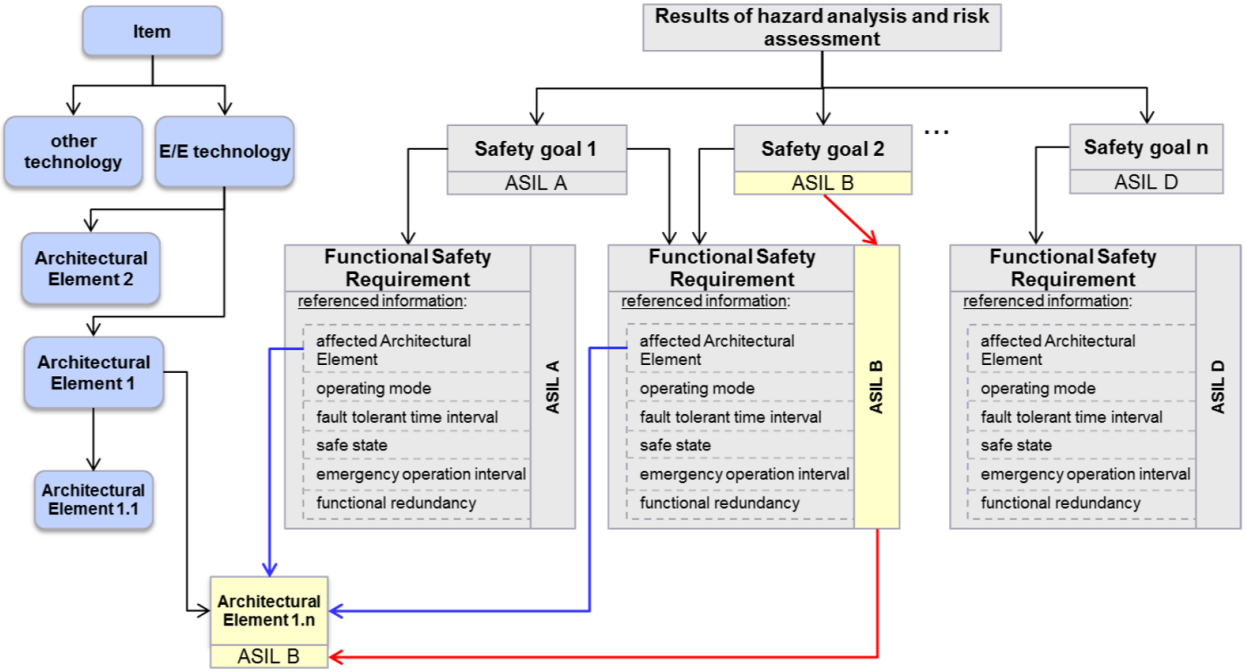

Figure 2.12: ASIL allocation to system design elements ([137]).

- 1) classify the character of the failure and the contribution of each fault by using the cut-set order and the coverage of a safety requirement with the specification of the diagnostic coverage of each safety mechanism;
- 2) tag each failure as Single Point, Residual, or Multiple Point Latent;
- 3) Identify the primary Hardware Safety Requirements based on the top-level malfunctions of the HW Architecture. The primary Hardware Safety Requirements shall prevent the occurrence of the malfunctions of the Hardware Components.

In AUTOSAR, the ECU Resource Template contains the elements to represent the hardware-related part of the system, as shown in Figure 2.13. An ECU is defined as a nested **HwElement**, connected by their **HwPin**s, and **HwPinGroup**s, to represent all **Hardware Part**s and to define a complete ECU electronic schematic at the hardware electronic design level.
The Hardware Package defines the extensions for the hardware part, as represented in Figure 2.14. It consists of the following sub-packages:

- FailureFormula
  contains all the equations that are necessary for the evaluation of the hardware architecture
- Failure
  describes the failure mode of the hardware component
- FailurePart
  describes the failure mode of each hardware part
- HWQuantitativeMeasure
  for the definition of the required quantitative safety analyses
- HWArchitectureMetrics
  for the calculation of hardware architectural metrics
- ProbabilisticMethods
  for the description of the residual risk of each safety goal due to random hardware failures. This package contains the definition of probabilistic metrics for random hardware failures (PMHF) and failure rate class methods (FRC).

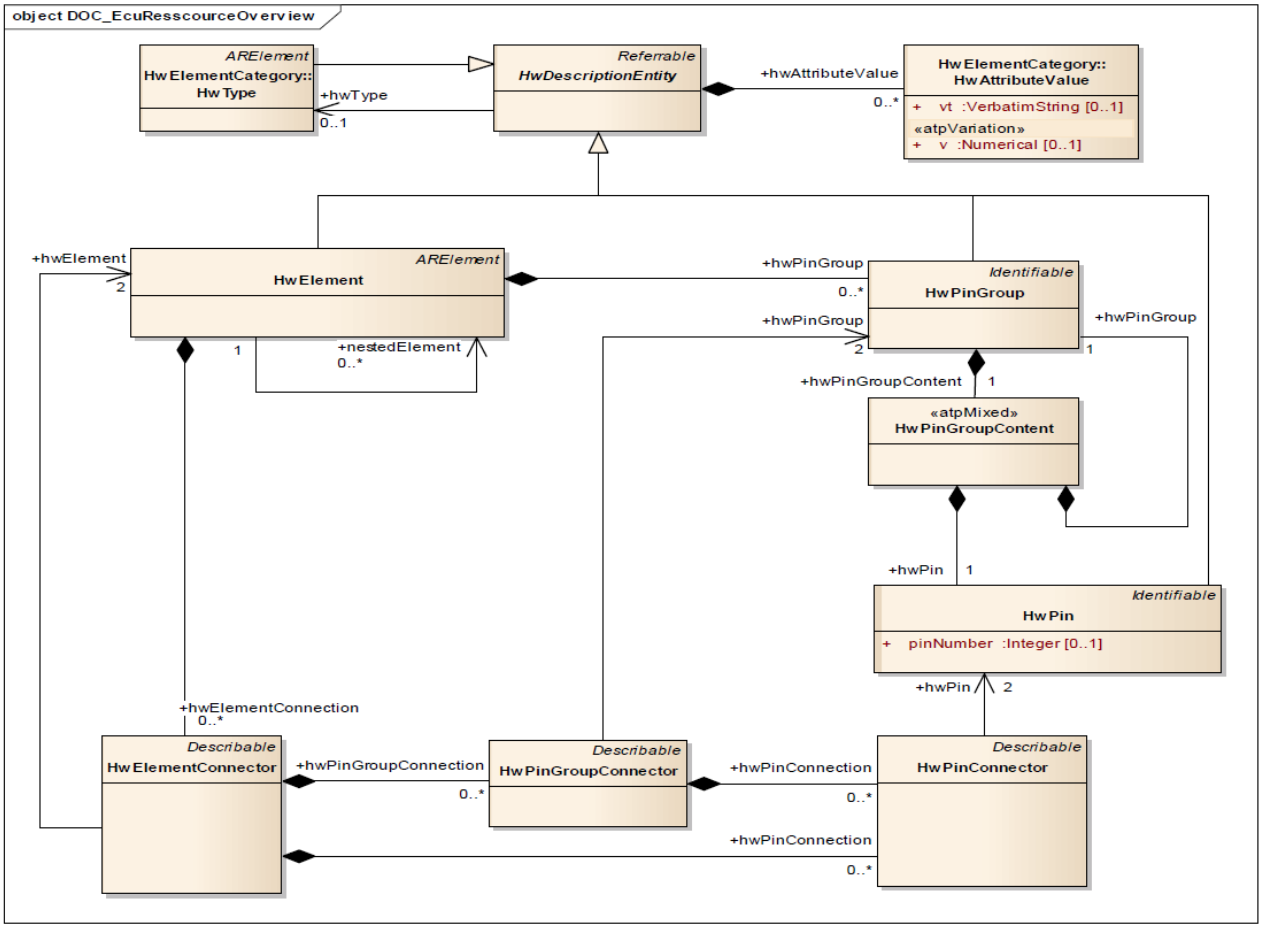

Figure 2.13: AUTOSAR ECU Resource Overview ([135]).

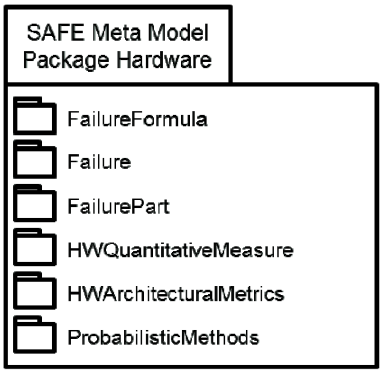

Figure 2.14: Hardware package overview ([135]).

As an example, Figure 2.15 shows the diagram for the **HWQuantitativeMeasure** package, which gives an overview about the quantitative analyses.
Given a Safety goal, the class **HWQuantitativeFailureAnalysis** represents the container for all the quantified failure analysis methods (**HWArchitecturalMetrics** class and **HWProbabilisticValue** class).
The **HWArchitecturalMetrics** class collects the hardware architectural metrics: the single-point fault metric, that describes the robustness of the hardware architecture to cope with single-point and residual fault (**HWSinglePointFaultMetric** class), and the latent fault metric, that describes the robustness of the hardware architecture to cope with multiple-point latent faults (**HWLatentFaultMetric** class).
HWProbabilisticValue is the class for storing the results of one of the two probabilistic methods: Probabilistic Metric for random Hardware Failures (**HWPMHF** class) or Failure Rate Class (**HWfailureClassContainer**

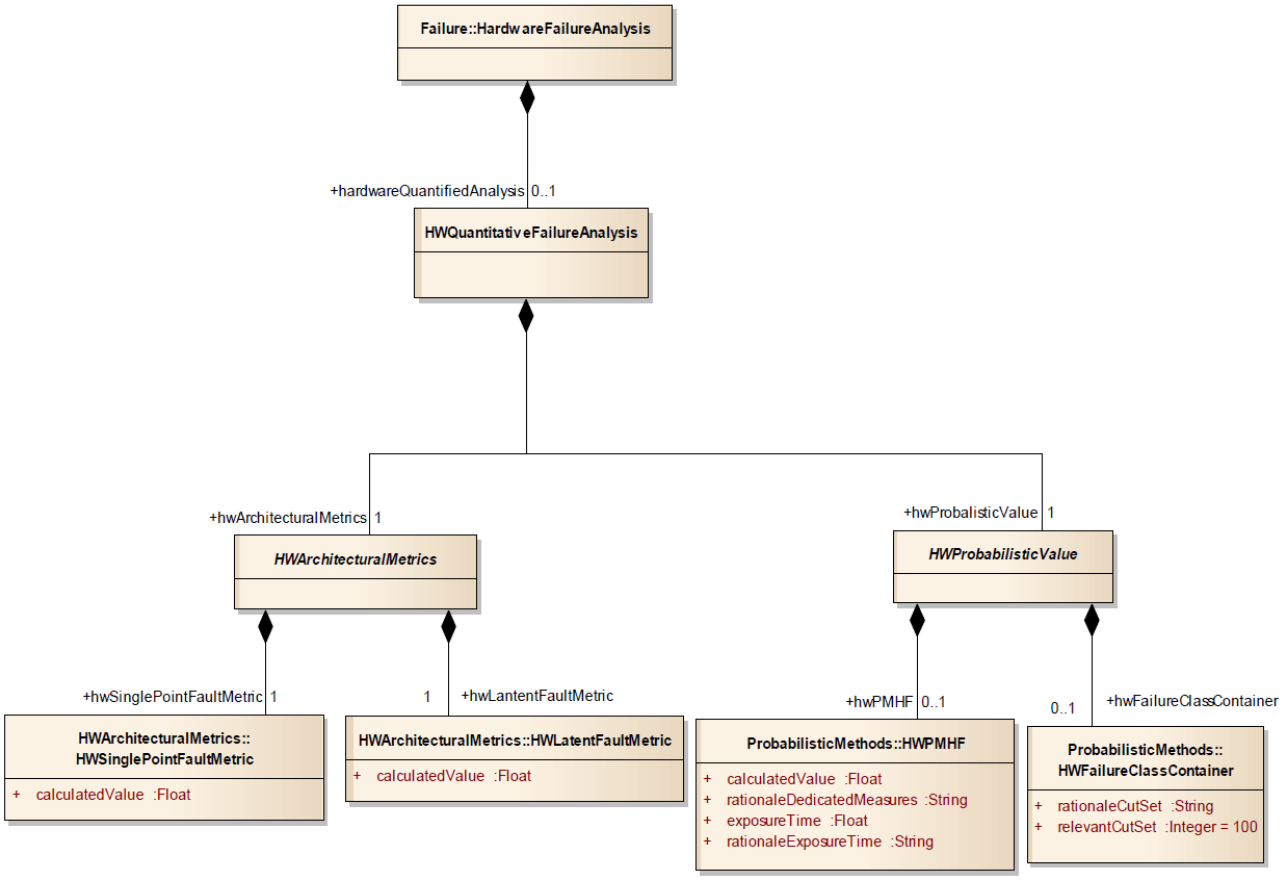

Figure 2.15: Hardware Quantitative Measure diagram ([135]).

class).
The attribute calculatedValue of the **HWPMHF** class is the result of the calculation of the PMHF. The attribute rationaleDedicatedMeasures shall allow defining a rationale for the dedicated measures applied to the design. The exposureTime attribute is the duration of the exposure used in the simplified computation of the PMH. The attribute rationaleExposureTime is for the Documentation of the rationale for the definition of the Exposure Time.
The **HWfailureClassContainer** class stores all the hardware element failure class results and the associated assumptions for the saving of the cut-set cut context as recorded in its attributes. The attribute rationaleCutSet provides a textual rationale for the number of relevant cut-sets and relevantCutSet stores the number of relevant cut-sets.

Threats and risk analysis

Safety-critical systems are vulnerable also to security threats. The identification of security threats as faults appeared as early as in 2004 in the paper by Avizienis et al. [29]. This paper represents the result of an effort to bring together the common strands of dependability and security. In the paper, faults are classified according to viewpoints: one of the viewpoints distinguishes between **nonmalicious** and **malicious faults** (faults introduced with the objective to cause harm to the system). Malicious faults are grouped into two classes: **Malicious logic faults** that encompass development faults and **Intrusion attempts** that are operational external faults. As new functionalities and technologies are introduced in automotive industry, a security-aware safety development process has become a crucial factor. In [104], the authors present an approach to safety evaluation that is a combination of the automotive hazard analysis and risk assessment (HARA) with the STRIDE approach typical of the security domain (a threat modeling approach that uses six security threat categories to review system design). During the hazard analysis and risk assessment (HARA), the evaluation of the safety is obtained by transforming the complete system error model into the input for a standard quantitative evaluation tool [51], and the safety analysis process follows a standard approach as shown in Figure 2.16. Problems caused by malicious attacks are not addressed by HARA within the ISO 26262 standard, although such attacks may

pre-empt a safety strategy.

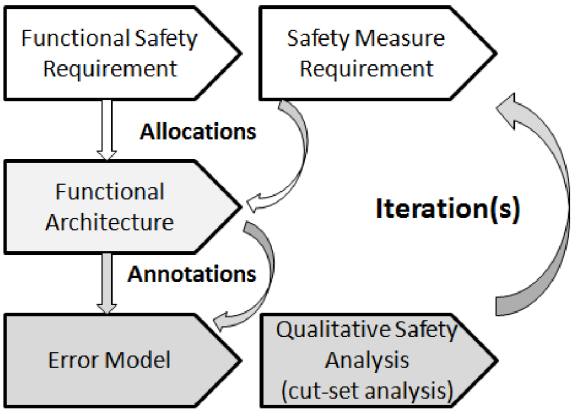

Figure 2.16: Overview of the safety analysis process ([51]).

The STRIDE threat model provides a way to methodically review system designs and highlight security design flaws. The proposed SAHARA (Security-Aware Hazard Analysis and Risk Assessment) method allows the evaluation of the impact of security issues on safety at the system level. In the same work, threats are quantified with reference to the ASIL analysis, according to the Resources and Know-How that are required to define threats and the Threats Criticality, see Figure 2.17. The impact of the threat on the system determines whether the threat is safety-related or not. If the threat is safety-related, it will be analyzed and the resulting hazards will be evaluated. The Common Criteria (CC) standard [50] was developed to facilitate the security evaluation of information technology products. This standard provides a common set of requirements for measuring the assurance of a product during a security evaluation. In particular, the CC addresses the protection from three main security aspects: unauthorised disclosure, modification, or loss of use.

Level	Required Resource	Example
0	no additional tool or everyday commodity	randomly using the user interface, strip fuse, key, coin,
1	standard tool	screwdriver, multi-meter, multi-tool
2	simple tool	corrugated-head screwdriver, CAN sniffer, oscilloscope
3	advanced tools	debugger, flashing tools, bus communication simulators

Level	Required Know-How	Example
0	no prior knowledge (black-box approach)	average driver, unknown internals
1	technical knowledge (gray-box approach)	electrician, mechanic, basic understanding of internals
2	domain knowledge (white-box approach)	person with technical training and focused interests, internals disclosed

Level	Threat Criticality	Example
0	no security impact	no security relevant impact
1	moderate security relevance	annoying manipulation, partial reduced availability of service
2	high security relevance	damage of goods, invoice manipulation, non-availability of service, privacy intrusion
3	high security and possible safety relevance	maximum security impact and life-threatening abuse possible

Figure 2.17: SAHARA method: Required Resources, Know-How and Threat Criticality ([104]).

In the EVITA Project [64], parameters for the risk analysis are determined using attack trees. The root of an attack tree is an abstract attack goal and its children represent possible attack objectives that could satisfy the attack goal. Attack objectives are decomposed into a number of attack methods and an attack method is

further decomposed into a logical combination (AND/OR) of attacks against one or more assets. The attack method with the highest attack probability can be identified and specific countermeasures can be applied to reduce the risk level. For safety-related security risks, the risk level is determined by a combination of the following measures:

- Severity,
- Attack probability and
- Controllability (potential for the human response to influence the severity of the attack).

The classification of Severity separates different aspects of the consequences of security threats: operational (interference with functions that are not safety-related), safety (interference with safety-related functions), privacy (driver privacy or reputation for manufactures) and financial (financial losses). These components may have different ratings from 0 to 4, as shown in Figure 2.18.

Security threat severity class	Aspects of security threats			
	Safety (S_S)	Privacy (S_p)	Financial (S_F)	Operational (S_o)
0	No injuries.	No unauthorized access to data.	No financial loss.	No impact on operational performance.
1	Light or moderate injuries.	Anonymous data only (no specific driver of vehicle data).	Low-level loss (~€10).	Impact not discernible to driver.
2	Severe injuries (survival probable). Light/moderate injuries for multiple vehicles.	Identification of vehicle or driver. Anonymous data for multiple vehicles.	Moderate loss (~€100). Low losses for multiple vehicles.	Driver aware of performance degradation. Indiscernible impacts for multiple vehicles.
3	Life threatening (survival uncertain) or fatal injuries. Severe injuries for multiple vehicles.	Driver or vehicle tracking. Identification of driver or vehicle, for multiple vehicles.	Heavy loss (~1000). Moderate losses for multiple vehicles.	Significant impact on performance. Noticeable impact for multiple vehicles.
4	Life threatening or fatal injuries for multiple vehicles.	Driver or vehicle tracking for multiple vehicles.	Heavy losses for multiple vehicles.	Significant impact for multiple vehicles.

Figure 2.18: Severity classification scheme ([64]).

The attack probability is computed in terms of the difficulty of mounting a successful attack, according to a parameter defined as "attack potential". The attack potential is a measure of the minimum effort to create and carry out a successful attack and it is obtained as the combination (sum) of a number of indexes related to several characteristics or aspects. The following aspects are analyzed: the Elapsed Time (total amount of time for the attack), the Specialist Expertise (the required level of knowledge to carry a successful attack), the Knowledge of the system (the specific expertise in relation to the system objective of the attack), the Window of opportunity (the required amount of off-line and on-line access to the system) and the Equipments (other equipment required). Different levels are identified for the previous factors and Table 2.19 associates numeric values with each level.
The attack potential that is required to exploit the attack is obtained by adding up the corresponding values from the table in Figure 2.19. This value is used in Figure 2.20 to classify the attack potential into 5 classes (Basic, Enhanced-Basic, Moderate, High and Beyond High), and to define a measure P of the attack probability using a numerical scale ranging from 1 to 5: the attack probability is higher for easier attack (Basic attack potential), and lower for more difficult attack (High attack potential).
An attack method may involve attacks against one or more assets combined with AND or OR relationships. We have an AND relationship if an attack method requires a conjunction of asset attacks; we have an OR relationship if an attack method can be implemented using anyone of a number of asset attack. For AND relationship, the combined attack probability is taken to be the lowest of the attack probabilities P_i for each contributing attack: $min\{P_i\}$.
For OR relationship, the combined attack probability is taken to be the highest of the attack probabilities P_i for each contributing attack: $max\{P_i\}$.
The Controllability represents the potential for the human response to influence the severity of the attack. Table 2.21 reports the classification for controllability (a qualitative measure is assumed).

Factor	Level	Comment	Value
Elapsed Time	≤ 1 day		0
	≤ 1 week		1
	≤ 1 month		4
	≤ 3 months		10
	≤ 6 months		17
	> 6 months		19
	not practical	The attack path is not exploitable within a timescale that would be useful to an attacker.	∞
Expertise	Layman	Unknowledgeable compared to experts or proficient persons, with no particular expertise	0
	Proficient	Knowledgeable in being familiar with the security behaviour of the product or system type.	3[1]
	Expert	Familiar with the underlying algorithms, protocols, hardware, structures, security behaviour, principles and concepts of security employed, techniques and tools for the definition of new attacks, cryptography, classical attacks for the product type, attack methods, etc.	6
	Multiple experts	Different fields of expertise are required at an Expert level for distinct steps of an attack.	8
Knowledge of system	Public	e.g. as gained from the Internet	0
	Restricted	e.g. knowledge that is controlled within the developer organisation and shared with other organisations under a non-disclosure agreement	3
	Sensitive	e.g. knowledge that is shared between discreet teams within the developer organisation, access to which is constrained only to team members	7
	Critical	e.g. knowledge that is known by only a few individuals, access to which is very tightly controlled on a strict need-to-know basis and individual undertaking	11
Window of Opportunity	Unnecessary/ unlimited	The attack does not need any kind of opportunity to be realized because there is no risk of being detected during access to the target of the attack and it is no problem to access the required number of targets for the attack.	0
	Easy	Access is required for ≤ 1 day and number of targets required performing the attack ≤ 10.	1
	Moderate	Access is required for ≤ 1 month and number of targets required to perform the attack ≤ 100.	4
	Difficult	Access is required for > 1 month or number of targets required to perform the attack > 100.	10
	None	The opportunity window is not sufficient to perform the attack (the access to the target is too short to perform the attack, or a sufficient number of targets is not accessible to the attacker).	∞[2]
Equipment	Standard	readily available to the attacker	0
	Specialised	not readily available to the attacker, but acquirable without undue effort. This could include purchase of moderate amounts of equipment or development of more extensive attack scripts or programs.	4[3]
	Bespoke	not readily available to the public because equipment may need to be specially produced, is so specialised that its distribution is restricted, or is very expensive.	7
	Multiple bespoke	Different types of bespoke equipment are required for distinct steps of an attack.	9

Figure 2.19: Rating of aspects of attack potential ([64]).

Values	Attack potential required to identify and exploit attack scenario	Attack probability *P* (reflecting relative likelihood of attack)
0-9	Basic	5
10-13	Enhanced-Basic	4
14-19	Moderate	3
20-24	High	2
≥ 25	Beyond High	1

Figure 2.20: Rating of attack potential ([64]).

Class	Meaning
C1	Despite operational limitations, avoidance of an accident is normally possible with a normal human response.
C2	Avoidance of an accident is difficult, but usually possible with a sensible human response.
C3	Avoidance of an accident is very difficult, but under favourable circumstances some control can be maintained with an experienced human response.
C4	Situation cannot be influenced by a human response.

Figure 2.21: Classification for Controllability ([64]).

Finally, in EVITA, Controllability, Severity and Attack probability are mapped to qualitative risk levels, according to the table in Figure 2.22. In the table, class $R0$ denotes very low risk levels, while class $R7+$ denotes *levels of risk that are unlikely to be considered acceptable* (high severity classes, high attack probability, low controllability) [64].

Controllability (C)	Safety-related Severity (S_S)	Combined Attack Probability (A)				
		A=1	A=2	A=3	A=4	A=5
C=1	S_S=1	R0	R0	R1	R2	R3
	S_S=2	R0	R1	R2	R3	R4
	S_S=3	R1	R2	R3	R4	R5
	S_S=4	R2	R3	R4	R5	R6
C=2	S_S=1	R0	R1	R2	R3	R4
	S_S=2	R1	R2	R3	R4	R5
	S_S=3	R2	R3	R4	R5	R6
	S_S=4	R3	R4	R5	R6	R7
C=3	S_S=1	R1	R2	R3	R4	R5
	S_S=2	R2	R3	R4	R5	R6
	S_S=3	R3	R4	R5	R6	R7
	S_S=4	R4	R5	R6	R7	R7+
C=4	S_S=1	R2	R3	R4	R5	R6
	S_S=2	R3	R4	R5	R6	R7
	S_S=3	R4	R5	R6	R7	R7+
	S_S=4	R5	R6	R7	R7+	R7+

Figure 2.22: Risk graph ([64]).

The attempt in Evita of treating security faults, at least in principle, similar to faults in safety tree analysis is interesting. However, there are several issues with the approach. Clearly, the scope of the proposed method is to provide an early assessment, mostly of qualitative nature, but in consideration of a possible move towards a more accurate quantitative analysis it should be noted that the values in the table of the ratings of the attack potentials are actually a representation of a set of probabilities (that the attacker has the right level of expertise, or equipment and so on). Of course it is still an open issue on whether these factors can actually be expressed as probabilities with sufficient accuracy, but regardless of the problem in the definition of the values, what they are representing is indeed a measure of probability. If this is the case, clearly probabilities cannot be combined using sums, minimum and maximum operators. Hence, while the classification of the attack potentials and the definition of a threat tree is surely interesting, the numerical values and the methods to combine them show clear opportunities for improvement.

2.1.2 Time

A real-time computing system is a computing system which is characterized by the fact that for a successfully performed computation (1) the computational result needs to be functionally correct, and (2) the computational result must be delivered in time. The imposed timing constraints most often result from the interaction of the real-time computing system with its physical environment, for instance, if it serves as a control system. The dynamics of the controlled system then determine the permissible temporal behavior of the real-time computing system. According to [94], the two temporal requirements that need to be specified for a classical real-time system are (1) the *response time* i.e the maximal or exact distance between an event and the resulting response,

and (2) the *repetition pattern of an event type*. Further temporal restrictions for a real-time system, however, may be relevant such as the (a)synchronicity of events of different type. Also recent approaches specify probabilistic or weakly-hard constraints [124] for real-time systems.
To include timing constraints in the modeling of computing systems, a large variety of formalisms has been developed which may be classified and evaluated using an appropriate *taxonomy*. In the following paragraphs, we present the taxonomy proposed by [69].
Subsequently, major modeling languages are presented which are capable of expressing temporal characteristics of systems. They are described and evaluated according to the previously introduced taxonomy. Following [69], we address *operational time modeling languages* and *descriptive time modeling languages*. Operational languages use the concept of system state evolution to describe dynamic system behavior, whereas descriptive languages formally define static and dynamic properties of the modeled system.

Taxonomy

In this section, we present the taxonomy which has been proposed by [69] for the classification of modeling languages describing the dynamic behavior of computing systems.

Time Domains The time domain used by a time model may be discrete or dense, hybrid time models use both discrete and dense time domains.
A *discrete time domain* is described by a set of discrete points in time. It is a suitable time domain to describe clocked computing systems where the system behavior evolves on the basis of ticks which can be counted using the discrete set of natural numbers $\mathbb{N}$ or integers $\mathbb{Z}$. A *dense time domain* is a totally ordered set under $<$ such that for every two points t_1, t_2 with $t_1 < t_2$ there is a point t_3 such that $t_1 < t_3 < t_2$. A *continuous dense* set S has the property that any non-empty subset S_1 of S that has an upper bound must have a least upper bound in S_1. A prominent example for a continuous dense set is the set of real numbers $\mathbb{R}$. A *non-continuous dense* set is the set of rational numbers $\mathbb{Q}$. Real numbers are classically used in mathematics and are suitable to convert even incommensurable time units, rational numbers are used in numerical algorithms due to their finite amount of digits and are suitable to convert commensurable time units.

Ordering vs. Metric If the time domain has a metric structure, then a distance $d(t_1, t_2) \geq 0$ between any two points t_1, t_2 can be defined. A common and intuitive definition is $d(t_1, t_2) = \|t_1 - t_2\|$ for the popular time domains $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ (*quantitative temporal constraints*).
Alternatively, if the time domain has no metric or the modeling language does not allow to use metrics, events can be ordered by their occurrence (*qualitative temporal constraints*).
The full specification of real-time systems requires typically the use of metric constraints.

Linear vs. Branching Time Models A linear behavior is a system behavior which evolves from a given state at a given time in a unique manner, i.e. over a linear sequence of states. A branching behavior is a system behavior which evolves from a given state at a given time to different successor states depending (1) on the future inputs in case of a deterministic system, or (2) on arbitrary choices in case of a non-deterministic system. Since such a behavior can be depicted as a tree of system states, it is referred to as branching behavior. *Linear time models* describe temporal constraints for linear behaviors, whereas *branching time models* describe temporal constraints for branching behaviors.

Implicit vs. Explicit Time Reference Language with implicit time references do not use time quantities, e.g. specified temporal distances, in the formulation of timing constraints as do explicit languages. Rather they use implicit terms like “current time” or “sometime in the future”.
Implicit time references allow the compact and elegant modeling of time-invariant systems.

Time Advancement Due to the assumptions of a time model, i.e. the assumption of state transitions in zero time, it can happen that the behavior of the modeled system does not progress past a certain point in time. The reason is that an infinite computation is performed in finite time which is called the time advancement problem.
The time advancement problem can be solved by two approaches. In the *a priori approach*, the modeling language is defined in such a way that no time advancement problems can occur. In the *a posteriori approach*, time advancement problems are spotted once the system has been modeled and only then the model is adapted such that this physically impossible behavior is eliminated.

Concurrency and Composition The interaction of concurrently working system components may be *synchronous* and/or *asynchronous*. In the asynchronous case, in contrast to the synchronous case, the state of evolution of components is not coupled to distinct points in time and each component is allowed to perform its computation at its own unrelated speed.

Analysis and Verification The *expressiveness* of a time modeling language describes the ability of the language to classify behaviors by properties. The more properties a language can describe, the more it can differentiate between behaviors. Expressiveness and *decidability* are antagonistic, where decidability describes whether a specified model can automatically, i.e. algorithmically, be tested against its requirements.
Verification techniques may broadly be classified into exhaustive enumeration techniques and syntactic transformations. *Exhaustive enumeration techniques* explore automatically (1) an operational model of the system or (2) all possible system realizations which are conform to the required property. *Techniques based on syntactic transformations* successively apply logic deduction schemes until the requirements are shown to be equivalent to the system specification.

Operational Modeling Formalisms

In this Section, operational modeling formalisms are presented which are capable of describing the dynamics of computing systems. Operational modeling formalisms, in contrast to descriptive formalisms treated in Section 2.1.2, are based on an evolutionary view of the system where deterministic or non-deterministic state transitions determine the system behavior. This section is based on the comprehensive survey of Furia et al. [69].

Synchronous Abstract State Machines Synchronous abstract state machines are an alternative way to model dynamic systems if the classical mathematical theory of system dynamics is not suitable. This is usually the case if the system to be modeled is not sufficiently specified for an equation-based state-space representation or if it does not require such a detailed description.
Synchronous abstract state machines introduce a number of abstractions, particularly (1) a discretized time domain based on clock ticks, (2) a clock-synchronous system evolution, (3) a discretized state domain, and (4) zero-time state transitions.

- **Mealy- and Moore State Machines** Mealy state machines [106] and Moore state machines [110] are the classic formalisms to describe sequential computations. Both formalism are based on a discrete time domain and allow qualitative time modeling. Due to their simple structure, they are suited for automatic verification. The most important verification method is model checking.
- **Infinite-Word Finite-State Automata** Infinite-word finite-state automata extend the applicability of synchronous abstract state machines to the description of reactive systems. Reactive systems are characterized by non-termination, since they continuously interact with their environment, and also by non-determinism. Infinite-word finite-state automata can model these properties since they are non-deterministic finite-state automata capable of handling infinite input words under a defined acceptance condition. The most prominent example is the Büchi automaton.
- **Statecharts** Statecharts [75, 77], a graphical modeling formalism, allows to model the synchronous concurrency of finite-state automata which are composed in parallel. It also introduces mechanisms for hierarchical abstraction. Statecharts can model non-deterministic behavior by using (1) mutually exclusive transitions with the same input label, (2) states with timeouts, and (3) XOR compositions of modules. Various automatic analysis tools are available [76, 41, 71].
  Related to the classical statecharts are *UML statecharts* [117] and the modeling language *Esterel* [40].
- **Timed Automata** Timed automata are based on finite-state automata with the addition of real-valued clock variables which increase as time elapses [12, 38]. Timestamped inputs cause a state transition if clock-related constraints are fulfilled. A state transition may be associated with the reset of clock variables.
  Timed automata dispose of a continuous metric by the introduction of the real-value clock variables, however, at the same time the original discrete notion of time connected with input/state-sequences prevails. The semantics of deterministic as well as non-deterministic timed automata are based on linear time models which describe runs i.e. input/state-sequences. The verification problem can be reduced to a finite abstraction of a timed automaton [13], tools are available such as UPPAAL [97], Kronos [159].

Petri Nets Petri nets constitute a popular graphical modeling formalism for asynchronous and heterogeneous systems [122, 121]. A variant, transition diagrams, are part of the UML standard [117]. Petri nets focus on the act of communication, which is modeled by the flow of tokens, and not on the transferred data. This restriction to the consumption and emission of tokens reduces the complexity of the concurrent system behavior to be described [85].
Petri nets are capable of describing non-determinism inherent to concurrency since the transfer ("firing") of tokens between places via enabled transition happens non-deterministically i.e. the transfer may take place or not. Thus Petri nets describe branching-time behaviors. A Petri net has no metric structure such that only a qualitative temporal description in the sense of a total or a partial order of events (firing of transitions) can be defined.
In order to dispose of a time metric, so-called *Timed Petri nets* have been introduced [45]. A popular variant proposed by [108] annotates transitions with the minimum and maximum time that may pass between the enabling of a transition and the firing of a token.
Due to their good expressivity, which may even be enhanced by extensions, Petri nets are hard to analyze and verify and the related reachability problem may become undecidable. A comprehensive survey of tools is provided by [146].

Descriptive Modeling Formalisms

A descriptive modeling approach specifies a system by the formulation of its behavioral properties using a logic or algebraic formalism and abstracting from the functional and structural aspects of the system. The section on temporal logics is based on the survey of Bellini et al. [37]. The section on algebraic formalism is based on Furia et al. [69].

Temporal Logics Temporal logics are derived from modal logics. Formulas are built from atomic formulas, Boolean operators and temporal modal operators. The truth or falsity of a formula is time-dependent.
The expressiveness of a temporal logic increases with its *order*. *Point-based* temporal logics order events by the their instants of occurrence, and in order to express time intervals quantifiers ($\exists$, $\forall$) are used. *Interval-based* temporal logics are more expressive in the sense that the occurrence of an event may be related to a time instant as well as an time interval. Also relationships between time instants as well as time intervals can be specified. Logic formulas are either interpreted over linear (one future) or over branching *temporal structures* (several possible futures) which impacts the properties of decidability and executability of the logic. If a *metric of time* is defined, events can not only be *ordered* in time but also quantitative time constraints can be formulated. The problem of finding a decision procedure which identifies whether a temporal logic formula is satisfiable/valid or not, is called *logic decidability* problem. Logic decidability becomes often impossible with increasing order of the logic theory. If a *deductive system* exists, i.e a set of axioms and deduction rules, the prerequisite is given that the conformity of the specification with the requirements can be proven automatically. A temporal logic is *executable* if the specified system can be simulated. Time is *implicit* in a temporal logic if the truth of a formula depends on the instant of evaluation, otherwise it is *explicit*.

Algebraic Formalisms Process algebra is an algebraic formalism suitable to describe concurrent and reactive systems. The behavior of a system is defined by the (sequential, alternative, concurrent) composition of processes, i.e. the composition of elementary behaviors. Mathematically speaking, a process algebra is any structure satisfying the axioms given for the basic operators. A process is an element of a process algebra. By using the axioms, calculations can be performed with processes [86]. These calculations are the basis for formal system verification.
Basic process algebras are based on a discrete time domain and can specify qualitative time constraints. Extended versions, however, have a notion of dense time and/or dispose of a time metric metric. Algebraic expressions are evaluated over linear time structure, yet deterministic behavior can be modeled.

Heterogeneous Frameworks

Timing Augmented Description Language 2 (TADL2)

This section is based on the specification document for TADL2 [147].

Table 2.1: Comparing Features of Temporal Logics [37]

Logic	Logic order[1]	Fundamental time entity[2]	Temporal structure[3]	Metric for time/ Quantitative temporal constraints[4]	Logic decidability[4]	Deductive system[4]	Logic executability[4]	Ordering events[4]	Implicit Explicit[5]
PTL	P	P	L	N	Y	Y	Y	Y	I
Choppy	P	P	L	N	Y	(Y)	(Y)	Y	I
BTTL	P	P	B	N	Y	Y	Y	Y	I
ITL	P	I	L	N	(Y)	(Y)	(Y)	Y	I
PMLTI	P	I	L/B	N	(Y)	NA	NA	Y	I
CTL	P	P	B	N	Y	NA	NA	Y	I
IL	P	I	L	N	Y	NA	NA	Y	I
EIL	P	I	L	Y	Y	NA	NA	Y	I
RTIL	P	I	L	Y	Y	NA	NA	Y	(I)
LTI	2nd	I	L	N	Y	Y	NA	Y	(I)
RTTL	1st	P	L	(Y)	N	Y	NA	Y	E
TPTL	P	P	L	Y	Y	Y	NA	Y	(E)
RTL	1st	I	L	Y	N	NA	NA	Y	E
TRIO	1st	P	L	Y	N	Y	(Y)	Y	I
MTL	1st	P	L	Y	(N)	(Y)	NA	Y	I
TILCO	1st	I	L	Y	(Y)	Y	(Y)	Y	I

[1]P = propositional, 1st = first order, 2nd = second order;
[2]P = point, I = interval;
[3]L = linear, B = branching;
[4]N = no, (N) = no in the general case, Y = yes, (Y) = yes in some specific case, NA = not available;
[5]I = implicit, E = explicit.

For the timing modeling in the context of the SAFURE project, the work that has been undertaken in the TIMMO (TIMing MOdel) project and the follow-up TIMMO-2-USE (TIMing MOdel - TOols, algorithms, languages, methodology, and USE case) project is of high relevance.
The TIMMO project has proposed the Timing Augmented Description Language (TADL) and a complementary methodology which aimed at integrating timing aspects into an EAST-ADL2 and AUTOSAR 3.0 based development processes. The proposal has been successful and was incorporated into the AUTOSAR 4.0 standard. The TIMMO-2-USE project extensively evaluated existing timing modeling approaches and languages [147] and, driven by requirements from industrial use cases – especially from automotive domain, proposed the Timing Augmented Description Language 2 (TADL2) which is capable of expressing a wide range of required timing properties and timing constraints. In contrast to TADL, TADL2 includes symbolic timing expressions, multi-clock definitions, probabilistic timing information and timing constraints for different system modes [119] while still being compatible with EAST-ADL and AUTOSAR. For TADL2, a metamodel is available which easily integrates with EAST-ADL and AUTOSAR UML-based metamodels.
The suitability of TADL2 for real-world use cases combined with its possible integration in an industrial development process makes it a very attractive means of time modeling.
The conception of time in TADL2 is logical time which is related to the occurrence of events during the run time of the system. In fact, the time model of TADL2 is a specialization of the time model of the UML Profile for MARTE [72]. According to the TADL definition, *an **event** denotes a distinct form of state change in a running system, taking place at distinct points in time called occurrence of the event* [147]. An event may either be

(1) an event as defined in AUTOSAR (AUTOSAREvent),

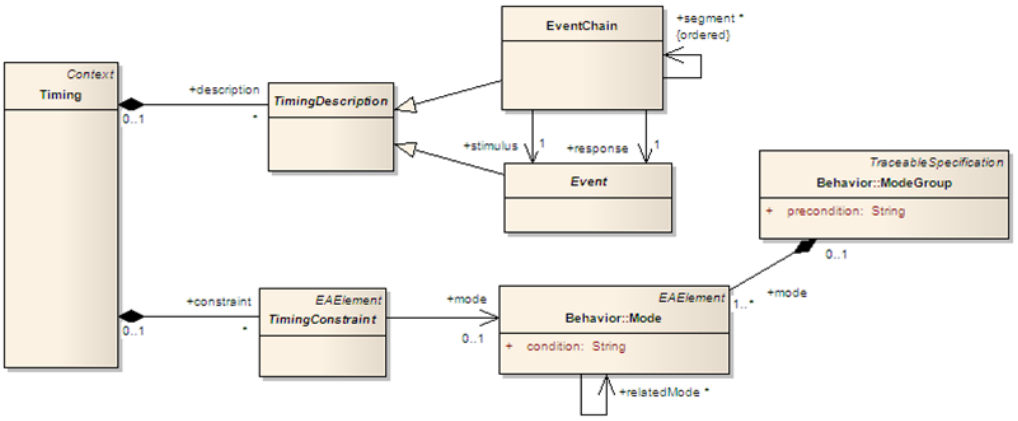

Figure 2.23: Organization of Basic TADL2 Elements [147]

(2) an event as defined in EASTADL (EASTADLEvent),

(3) an event occurring when the system mode is changed (Mode Event)

(4) any other event originating from the system environment (ExternalEvent).

Two events which have a causal relation, one event representing a stimulus and the other event the response, are summarized in an **event chain**. Events and their causal relations specified by event chains constitute the **description of timing** in the system. Timing requirements are formulated in so called **timing constraints** whose validity may depend on the system mode. A timing constraint references events, event chains and timing expressions. A **timing expression** defines a *duration* such as a delay, period, jitter or a tolerance duration.

While TADL has only an implicit notion of a **time base**, TADL2 is able to differentiate between a set of multiform time bases. A time base is characterized by a discrete and totally ordered set of instants which correspond to the occurrences of events the time base is associated with. The part highlighted in blue in Figure 2.24 illustrates that a time base is characterized by a dimension that defines the set of units that can be used to measure a duration. Each unit of the set can be converted to the other by using an appropriate factor and offset in a linear relation.
Time bases can be related to each other either statically or dynamically and the **time base relations** may be used to convert time expressions between different time bases, this is depicted in the red marked part of Figure 2.24.
A timing expression may either be a **ValueTimingExpression**, a **VariableTimingExpression** or a **SymbolicTimingExpression** – see the part of Figure 2.24 which is highlighted in green. A ValueTimingExpression is the simplest type of timing expression consisting of a constant float value in the unit of a time base. In contrast, a VariableTimingExpression represents a free variable or constant. A SymbolicTimingExpression is the most comprehensive type of timing expression which allows to define algebraic operations (addition, subtraction, multiplication, division) on VariableTimingExpressions from multiple time bases.
The vast majority of **timing constraints** in TADL2 refer to delays between a pair of events, repetitions of a single event and the synchronicity of a set of events. Moreover, **probabilistic timing constraints** and **weakly-hard timing constraints** are defined. Probabilistic constraints often refer to a timing property which has to fall in a certain time interval with a probability given by the associated probability distribution. A weakly-hard constraint imposes that the behavior must at least m out of k consecutive occurrences fulfill the constraint.

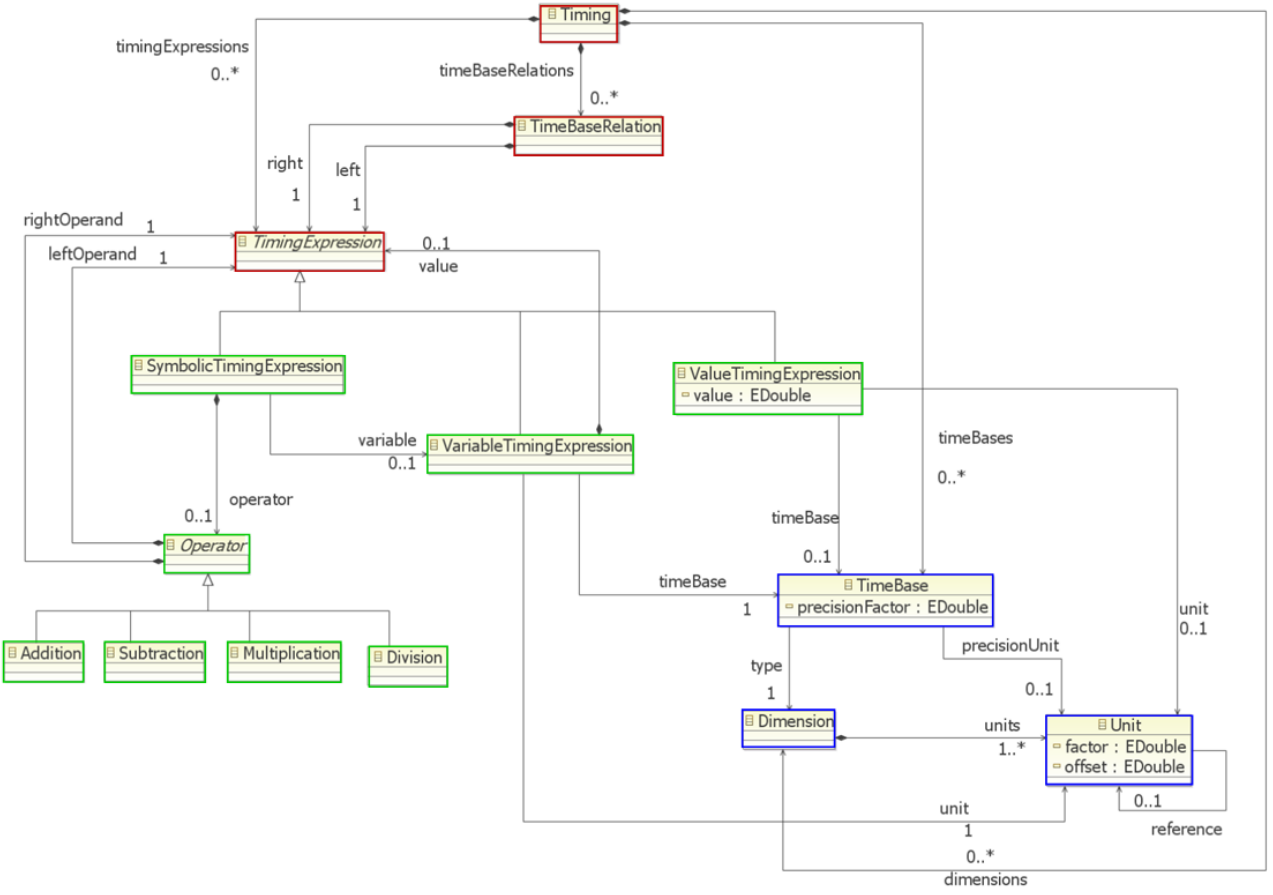

Figure 2.24: Timing Expression [147]

Mixed Criticality

In recent years, there has been a growing interest of the community on real-time research on the topic of scheduling and resource management for mixed criticality. The motivation for this research is the opportunity of integrating onto the same hardware platform and under the control of the same set of schedulers and resource managers computations with different criticality level.
One of the first references is the work in [130, 129] where the authors, inspired by the levels of criticality identified by the DO-178B and ISO26262 standards (with its SIL levels) propose a model in which computation tasks are identified by an activation pattern, a deadline, a priority, a *criticality level* (an integer value) and a vector of worst case execution time assumptions, one for each criticality level. The rationale is that more stringent criticality levels may require more pessimistic assumptions and lead to a larger estimate of the WCET. The immediate implication of this model is that each schedulable entity (tasks or threads in the papers in the literature, but the concepts can be extended to functions) is characterized by a vector of WCET values.
In subsequent papers this model has been retained or simplified, with the authors of other papers, such as [58] and [5] assuming only two criticality levels.
A different model has been presented in [32], where the authors also assume a degraded mode of operation of processors (such as, for example as a consequence of overheating) and study scheduling policies that allow to guarantee the critical load even in presence of reduced capabilities.
In essence, aside from the details of the scheduling policies, the model assumed by these papers is relatively simple.

Time budgeting

If a real-time application is executed on a distributed computing system, its tasks are typically mapped on several system resources for efficiency reasons. To verify whether the end-to-end deadline (time budget) of the application is met, a schedulability test has to be performed.
If a global scheduling policy is applied to the system, the end-to-end deadline can be directly verified by a global schedulability test. In the more common case of independent scheduling policies for the local resources, a schedulability test for each resource in the system has to be performed. A local schedulability test, however, requires a local deadline as input. This means that the global end-to-end deadline must be decomposed into local deadlines. The decomposition problem with respect to a given performance measure as an optimality criterion is, however, NP-hard. Therefore, heuristic approaches have been proposed which result in good sub-optimal solutions.
The literature on deadline composition differs in

- the assumed system model, e.g. scheduling policy
- the assumed task model, e.g. activation pattern, precedence constraints
- the assumed locality constraints with respect to task mapping, i.e. strict or relaxed
- the performance/quality measure to be optimized

Table 2.2 gives a survey over publications on deadline decomposition and indicates the investigated optimality criteria.

Publication	Optimality Criterion
Di Natale et al. 1994 [59]	maximizing the minimum task laxity
Abdelzaher et al. 1995 [6]	minimizing the maximum task lateness
Gutiérrez García et al. 1995 [70]	maximizing the scheduling index (largest distance between worst-case response times and deadlines)
Saksena et al. 1996 [138] [139]	maximizing the largest possible scaling factor for task execution times
Jonsson et al. 2002 [87]	success ratio (capability of finding feasible schedules) & minimizing the maximum task lateness & maximizing the minimum task laxity

Table 2.2: Deadline decomposition approaches

From a specification perspective, formal statements about global and local deadlines are required for hard real-time systems.
Also decomposition constraints which identify feasible decompositions need to be expressed. For instance, the basic path constraint for a linear chain of N tasks says that the sum of local deadlines d_j may not exceed the global deadline D: $\sum_{j=1}^{N} d_j \leq D$.
The specification language TADL2, for instance, is capable of formulating such path constraints.

Scheduling for Timing Isolation

The integration of functions with different safety criticality levels requires *freedom from interference* (ISO 26262 [83]). Freedom from interference is typically implemented via some kind of temporal and spatial isolation mechanisms. The former addresses the timing behavior of functions, while the latter is concerned with memory partitioning. Here, we focus on temporal isolation.
Various scheduling mechanisms for timing isolation have been proposed, with TDMA being the most prominent one. In the reminder of this subsection, we differentiate between scheduling mechanisms for timing isolation on processors and communication network (with a focus on Ethernet).

Processors In the context of functional architectures, ARINC 653 [9] specifies a time and space partitioning interface for real-time operating systems. In ARINC 653 software components run in partitions. Partitions must be isolated from each other temporally.
There are several ways for a scheduler to enforce timing isolation among tasks or processes. The simplest method is to partition the available time in slots (of equal or different size) and to assign the time slots to partitions (or processes) in a (usually periodic) controlled fashion. This method is applied in practice and it is quite simple, with the only possible problem of being relatively inflexible and requesting possible major reworks for changes and updates and being usually inefficient, since time slots need to be designed for the worst case and unused time typically cannot be reclaimed.
The scheduling literature has provided other scheduling approaches, under the general category of servers that can assign the processor time to computations with the property of isolation. One such policy is the Constant Bandwidth Server or CBS [7], recently implemented in Linux as SCHED_DEADLINE.
PikeOS [11] is a commercially available hypervisor, implementing an ARINC 653 compatible scheduler based on TDMA.
While a TDMA scheme provides timing isolation, it is very inflexible when asynchronous events such as interrupts must be processed. In the worst-case, the processing of an interrupt must wait until its corresponding time partition becomes available. One approach to address this problem is to serve asynchronous events at a higher priority than the TDMA partitions, e.g. [90], [153], [118], and [11]. This, however, implies that the handling of all asynchronous events must be certified according to the highest safety level to not compromise timing isolation.
This problem can be mitigated by applying (low-overhead) monitoring (e.g. [114], [113]), i.e. by enforcing an upper bound on the number of asynchronous events, which can interrupt other partitions [35], [34]. In this case, only the monitors require certification

Communication Networks Distributed time- and safety-critical systems require timing isolation at the interconnect. If the underlying interconnect offers time synchronization, TDMA can also be applied to communication networks. Among the automotive buses, this is the case in FlexRay [67], which offers a static segment for time-triggered communication, and MOST [48], which offers synchronous communication channels.
Time-triggering has also been applied to Ethernet. TTEthernet [28] allows to specify time-triggered segments for synchronous communication similar to FlexRay's static segment. The time-aware shaper (IEEE 802.1Qbv) [82] of the upcoming set of Ethernet TSN [81] standards also allows to schedule latency-critical traffic in time-triggered slot. In contrast to TTEthernet, IEEE 802.1Qbv does not support to schedule individual traffic streams inside their time-triggered slot, e.g. based on some offset relative to the beginning of the slot. Traffic streams can only be assigned to slots.
TDMA-based communication networks can provide low latency guarantees only if the entire distributed system including all nodes is fully synchronized to the network's TDMA schedule. Otherwise, there is *sampling delay*, which is an additional delay that must be considered when computing the system-wide end-to-end latency. Sampling delay occurs whenever data or frames miss their designated time-triggered slot and must wait a TDMA cycle for their next slot. Automotive systems are inherently hard to fully synchronize, e.g. the wide-spread CAN bus uses static-priority non-preemptive scheduling and introduces some jitter to its frames, there might be execution time jitter on ECUs, and in engine control there are angular-synchronized task with variable periods.
Apart from TDMA there are other approaches to provide (sufficient) timing isolation in real-time Ethernet networks. AFDX [8] is based on standard Ethernet (IEEE 802.1Q) and offers event-based communication. As an isolation mechanism, AFDX employs minimum distance shaping (bandwidth allocation gap in [8]) for traffic streams, which enter the network, limiting their maximum frame injection rate. Based on the maximum injection rate of all traffic streams, formal worst-case guarantees can be derived and no further shaping the core network is required. Ethernet AVB [79] defines standardized traffic shaping for standard Ethernet, which only supports priority-based arbitration. In AVB, a traffic class (priority level) is shaped by credit-based shaper at every switch in order to

bound the class' timing impact on other traffic streams. This permanent shaping can lead to increased worst-case latency guarantees [60]. The drawback of AVB's traffic shaper is that it is class-based, i.e. while the inter class interference of a misbehaving traffic stream is bounded by the shaper, the intra class interference is not. This can be mitigated by per-stream monitoring and filtering, which is being discussed in the context of Ethernet TSN in the IEEE 802.1Qci [80] standard.

Models for overload and weakly hard systems

Computing systems which have strict constraints on their timing behavior are usually called *hard* real-time systems. The attribute *hard* emphasizes the fact that hard deadlines for task completion times exist which must be met. Recent work in the real-time community has brought up the conceptual transition from hard real-time systems to *weakly-hard* real-time systems. Weakly-hard systems [73, 39, 124] exploit the fact that many functions are robust towards occasional deadline misses of tasks. This means that deadlines can be occasionally missed *without* compromising the quality of service or other performance measures. A typical example is a control algorithm which can tolerate an occasional loss of a sensor sample. To better quantify what "an occasional deadline miss" means, the so called (m, k) constraints are introduced which indicate that at most m deadlines may be missed in a given window of k consecutive task executions. In order to deem a system feasible, the (m, k) constraints of all tasks must be satisfied.
Typical worst-case analysis (TWCA) is a verification method which is capable of deriving bounds on the (m, k)-behavior of tasks, and it is applicable to a wide variety of systems with specific properties. It takes into account that (1) deadline misses originate from transient overload in the system e.g. caused by sporadic task activations, and (2) the extent of observed deadline misses depends on the amount and the temporal distribution of transient overload that is introduced.
In [125] TWCA is introduced to handle sporadic overload. It is extended by [126] to handle sporadic bursts. A weakly hard bound on the number of potential deadline misses depending on TWCA was introduced in [74] and was improved for tighter bounds in [158]. [123] demonstrates how TWCA can be used to analyze a real CAN bus with complex load patterns. In [68], the authors proposed a novel approach integrating timing and closed-loop verification that is based on a combination of LET (logical execution times) and TWCA.

Other timing models

Another interesting research trend deals with the analysis of tasks that are activated by event streams that cannot be characterized as periodic or simply sporadic (with a simple minimum interarrival time) but that are dependent from a physical process with a bounded dynamic.
An example are the automotive activities (for example in fuel injection applications) that are triggered at given reference points with respect to the position of the engine shaft.
The problem has been studied in several recent papers including [57] [42] [43]. In all these papers computations are defined as characterized by a set of possible execution times that are activated in correspondence to different engine rotation speeds. The activation events occur at reference positions of the engine shaft and are bound in their occurrence by the dynamic of the engine that is assumed to be with an acceleration bounded in a given range.

2.1.3 Security

The design of real-time CPS systems is a challenging task. This is mainly due to the complexity that originates from the wide range of extra-functional properties that need to be satisfied, while taking into account possible limitations of resources. The representation and analysis is even more complex considering that the extra-functional properties in these systems are tightly inter-connected and cannot be considered in isolation [152]. Due to the nature of real-time embedded systems (e.g. usage of sensors and actuators and interactions with the environment), timing properties in these

systems are of utmost importance. However, implications of other properties and aspects, such as security and safety, on timing properties should also be taken into account to ensure a correct design. Today, security is a requirement for an increasing number of embedded systems, ranging from low-end systems such as smartphones, networked sensors, and smart cards, to high-end systems such as routers, gateways, firewalls, storage servers, web servers, up to automotive and autonomous systems [155]. Technological advances that have spurred the development of these electronic systems have also ushered in seemingly parallel trends in the sophistication of security attacks. It has been observed that the cost of insecurity in electronic systems can be very high [92]. With an increasing proliferation of attacks, it is not surprising that security is felt as one of the largest concerns preventing the successful deployment of next-generation embedded systems [128].

Modern automotive electronics systems feature a relevant instance of high-end distributed real-time embedded system running over networked Electronic Control Units (ECU) communicating via serial buses and gateways. Most systems have not been designed with security in mind. Until very recently, cars have not been equipped with wireless networks, thus requiring the hacker to gain physical access to the car (usually using the On-Board Diagnosis connector). Therefore, common belief was that in the majority of the cases, there was little or no interest for hackers to compromise them. The only exception known so far has been the after-market community that tampers with engine calibrations to increase engine performance, manipulates mileage counter [15], or spoofs tachometers. The automotive industry has started to take actions to prevent these attacks. However, due to initiatives such as the *eCall* initiative in Europe[1], more cars are equipped with wireless units that are connected to the car's internal network. This enables hackers to use them as a remote entry point.

Recent research and news have shown that, as with many of these complex networks of systems, it is possible for external intruders to intentionally compromise the proper operation and functionality of these systems. Koscher *et al.* demonstrated that if an adversary were able to communicate on one or more of a car internal network buses, then this capability could be sufficient to maliciously control critical components across the entire car (including dangerous behavior such as forcibly engaging or disengaging individual brakes independent of driver input) [93]. These results raise the question of whether and how an adversary might be able to access a car internal bus (and thus compromise its ECUs) in the case of absent direct physical access. Checkoway *et al.* demonstrated that *external attacks* are indeed feasible [47]. Charlie Miller and Chris Valasek have recently demonstrated further remote attacks [109]. Furthermore, Checkoway *et al.* categorized external attack vectors as a function of the attacker ability to deliver malicious input via particular modalities: indirect physical access, short-range wireless access, and long-range wireless access [155]. Within each of these categories, they also characterized the attack surface exposed in current automobiles and their surprisingly large set of I/O channels.

However, unlike complex networks such as the Internet, where the issue of security has been extensively researched and funded, security issues surrounding complex networks of automotive systems have not been as readily studied. Moreover, these system's security vulnerabilities are increasingly being discovered and exploited. The state of the art processes, methods, and tools used for designing current automotive electronics systems focus on safety, reliability, and cost optimization. Methods and tools for the verification of the reliability of automotive electronics systems against random failures are commercially available. However, no security aspect is included as part of the hardware and software architecture development process and no standard communication protocol has any built-in provisions to prevent or mitigate attacks [99].

Recently, many research and industrial activities have started to take security into account in the early phases of the development cycle of automotive electronics systems, both by enforcing software programming standards that prevent software defects that may enable cyber-attacks [47], as well as by implementing security mechanisms for secure communication [98] including software delivery, installation and flashing [10][142].

[1] http://ec.europa.eu/digital-agenda/en/ecall-time-saved-lives-saved

On protocols and architectural mechanisms

In general, communication networks are vulnerable to unauthorized access when communications are performed without authentication. Authentication mechanisms ensure that sender and receiver identities are not compromised and thus, the sender and the receiver are indeed who they are claiming to be. Implicitly, authentication implies integrity, namely assuring that information has not been altered by unauthorized or unknown means [107]. Unfortunately, current communication network protocols, including Controller Area Network (CAN), FlexRay, MOST, and LIN have no authentication—or at best have CRC mechanisms to guarantee data integrity—and send their messages in the clear. Hence, room for fraudulent communications between ECUs exists [99].
There have been several publications demonstrating attacks on the authenticity of messages and nodes in embedded networks. Nilsson and Larson [116] detail the actions which an attacker may take, and demonstrate masquerade attacks on CAN using simulation. Additionally, they discuss the possibility of viruses transmitted over CAN and preventative measures. Hoppe *et al.* [78] and Lang *et al.* [96] demonstrate a combination of eavesdropping and replay attacks on CAN.
Some proposals for bus communication integrity and authentication have been proposed. Lin and Sangiovanni-Vincentelli have proposed a security mechanism to help prevent masquerade and replay cyber-attacks in vehicles with architecture based on Controller Area Network (CAN) [99]. They retrofitted the CAN protocol with software-only security mechanisms. The challenge here was to achieve high security level without introducing high communication overhead in terms of bus load and message latency because of the very limited data rates available (e.g., 500kbps). Furthermore, they introduced the concept of *counters* to implement time-stamping of the message authentication codes (MACs) in order to overcome the lack of global time in the CAN protocol. Finally, they focused on run-time authentication both in the system steady state—after ignition key-on and the security secret keys have been distributed to the ECUs—and during running resets experienced by some of the ECUs in the system—when counters are potentially out of synchronization.
Zalman and Mayer have proposed an alternative mechanism to prevent masquerade and replay that is based on a novel technique called *implicit* MAC [160]. The basic idea consists in computing the MAC on the time-stamped payload and then computing the CRC on the bundle composed of the payload and the MAC. However, a message containing the payload and the CRC only is transmitted, namely timestamp and MAC are omitted. The reason is that the implicitly included MAC bits are not transmitted over the physical channel but instead calculated on the receiver side. The receiver will follow the same reverse procedure for verifying the authenticity and integrity of the data. For this however, the receiver shall provide its own time stamp and add it to the received data. This implementation assumes the existence of a consistent time base to all the participants in the required communication.
A CAN packet does not include addresses in the traditional sense and instead supports a publish-and-subscribe communications model. The CAN ID header is used to indicate the packet type, and each packet is both physically and logically broadcast to all nodes, which then decide for themselves whether to process the packets. Authentication mechanisms have been proposed in the literature. The TESLA protocol uses a time-delayed release of keys for authentication [120]. A receiver can check the MAC after receiving the key used to compute the MAC. To guarantee security, the protocol needs to maintain global time and make sure that a receiver gets a message before the corresponding key is released. However, since in an automotive application the number of broadcast receivers is generally small (typically no more than five for any message), authenticated broadcast techniques proposed in [143][145][144] are preferred. In this approach, when an ECU sends a message, it computes a MAC for each distinct receiver over the message using a pair-wise shared secret key. Each MAC is then truncated down to just a few bits, and appended to the message.
When retrofitting traditional communication architectures such as those based on CAN, a major challenge is to ensure that system safety will not be hindered, given the limited computation and communication resources. Lin *et al.* have proposed an optimal Mixed Integer Linear Programming (MILP) formulation for solving the mapping problem from a functional model to the CAN-based

platform while meeting *both the security and the safety requirements* [100]. Security requirements consist in authentication and integrity in order to protect from masquerade and replay attacks whereas safety requirements consist of meeting end-to-end latency deadlines for safety-critical functional paths. Lin *et al.* group receiving ECUs in order to let them share one MAC in a message and therefore the bus load increment due to such a MAC. However, since ECUs can be compromised and critical computations may be affected by falsely accepted messages, Lin *et al.* organize ECUs into two or more receiving groups so that untrusted ECUs and safety-critical ECUs are mapped in different groups.
Chávez *et al.* [46] propose using RC4 encryption to provide confidentiality on CAN buses. However, Chávez *et al.* dismiss authentication and non-repudiation as unnecessary in these networks, under the assumption that message identifiers and error detection provide sufficient confirmation of the sender's identity. Incidentally, it must be observed that many experts agree that RC4 can no longer be regarded as secure. There are practical attacks that break an RC4 key in about 52 hours.
All these approaches assume an initial security critical key assignment and distribution, which is a crucial although, at least for the moment, overlooked aspect [154]. Carnevale *et al.* have proposed a hardware accelerator of the IEEE 802.1X-2010 Key Management scheme for automotive applications using the Ethernet backbone for in-vehicle communications [44].

On modeling

For modeling security features in general, several solutions based on UML have been offered. UMLsec is a UML profile for modeling security properties of computer systems in UML diagrams [88][89]. It is one of the major works in this area and also comes with a tool suite which enables the evaluation of security aspects and their violations. Although it seems promising, the extension only addresses few specific security requirements and cannot define complex or composed ones. This is a huge drawback because systems usually work in many domains and with other systems, which implies that the process must be able to work with dynamic and complex situations. UMLsec works with vulnerabilities and requirements. The way UMLsec models systems makes it very difficult to automatize because it is oriented to specification and not engineering. This problem exists because of the lack of a base meta-model.
A different UML-based approach for security and engineering processes is MDS [33]. It proposes a modular methodology for combining languages for modeling system designs with languages for modeling security. It uses transformations on security-enhanced models in order to generate implementations. One of the works that are based on this approach is SecureUML [148][102]. SecureUML can be used to model systems and design the security it needs. Unfortunately, the only security that can be described using this approach is role-based access control policies.
However, modeling security requirements in isolation (from other aspects of the system) is not enough and it becomes problematic to predict the impact on other extra-functional properties, especially for real-time embedded systems. For example, with reference to a biometric embedded system, Lloyd and Jürjens introduce a method to specify security requirements on UML models and check their satisfaction by relating model-level requirements to code-level implementations in [101]. UMLsec is used to include security requirements at model level, and the JML annotation language is used to relate code blocks back to the security requirements specification, thus enabling the evaluation of security requirement assurance. While this work constitutes a model-driven engineering approach for defining security requirements, it does not provide timing impacts of security implementation and does not automatically derive security implementation.
Saadatmand *et al.* have proposed and discussed benefits of extending MARTE, an extension of the UML modeling language for real-time and embedded systems design [131], with security annotations to cover the modeling needs of embedded systems [3]. This work focuses on providing UML stereotypes to specify confidentiality properties of message communication and related timing estimates. Later Saadamand and Leveque have proposed concepts and mechanisms that allow to model confidentiality and authentication requirements at a higher abstraction level and automatically derive the corresponding security implementations of the original component model into a secured one, taking into account

sensitive data flows in the system [132]. This allows the designer to work at a higher abstraction level and focus on sensitive data without having to think about the security implementation, which will be generated automatically. Furthermore, the resulting architecture ensures security requirements by construction and is expressed in the original meta-model and enables using the same timing analysis and synthesis as with the original component model.

The EVITA project

In the context of automotive security, the FP7 project EVITA[2] has to be mentioned, as it targets the design and verification of building blocks for secure automotive on-board networks. These are in turn meant to protect security-critical components against attacks. The project has been divided into five main stages:

1. Security requirements analysis
2. Secure on-board architecture design
3. Implementation of the prototype
4. Demonstration based on the EVITA prototype
5. Dissemination of project results

A meta-model to security was derived in the course of the EVITA project during the phase of architecture design [65]. The meta-model was devised making use of existing security and access control models. All concepts needed to specify security aspects and their relationships are defined. Selected security-relevant parts of the architecture and communication protocols were modeled by means of UML and Automata. Similar to the SAFE project, the EVITA meta-model is structured in different packages. These packages include:

- System Model, which defines all concepts allowing global system modeling;
- Security and Dependability Model, which defines security and dependability concepts;
- Fault Propagation Model, which defines faults, errors and failures;
- Trust Model, which defines trust concepts;
- Privacy Model, which defines privacy concepts.

Their dependencies are shown in Figure 2.25.
Figure 2.26 shows the EVITA trust model which expresses the following concepts. By trust, EVITA means the degree to which a trustor has a justifiable belief that the trustee will provide the expected function or service. *Trustee* is an entity or service that provides trust for the use of a function whereas *trustor* is an entity or service that uses a function with the expectation that it is trusted. Trustor and trustee are a *Stakeholder*, that is is a person, group, organization, or system who affects or can be affected by an organization's actions. Trust may have a *trust level*.
Trust risk is the potential risk of the system failure due to errors. Trust risk is the sum (over all relevant errors) of the negative impact of the failure (i.e., its criticality) multiplied by the likelihood of the failure occurring. Of course, the trust risk depends on the *Context*.
Trust policy is a quality policy that mandates a system-specific quality criterion for trust or one of its dimensions. System-specific quality criteria can also involve the system environment, the infrastructure in which it exists, and any assumptions about the system.

[2] http://www.evita-project.org

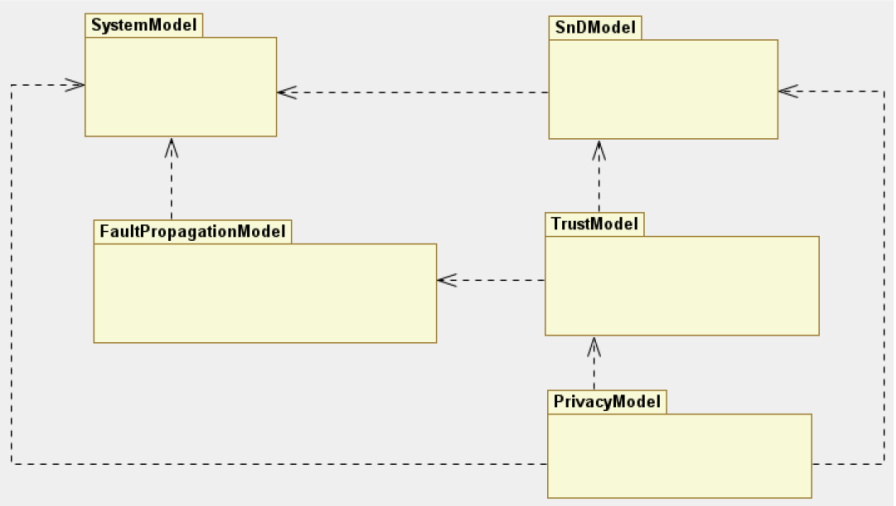

Figure 2.25: The EVITA modeling packages and their relationships

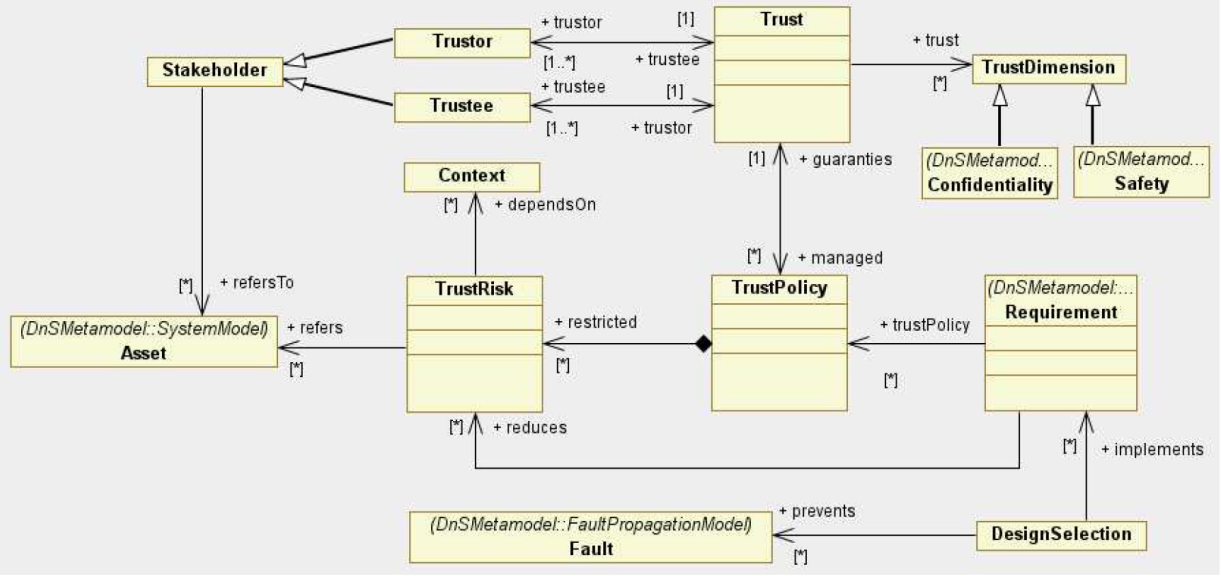

Figure 2.26: EVITA trust model

Requirement is a specification of a required amount of trust (actually a dimension of trust) in terms of a system-specific criterion and a minimum level of an associated quality metric that is necessary to meet one or more trust policies.
Design selection is a design decision that helps fulfill one or more trust requirements and/or reduces one or more system faults. Trust can be implemented as some combination of hardware or software components, manual procedures and services provided by either the application or the execution platforms. Design selection may be at different levels:

- Protocol level: this includes the design of protocols to be performed on embedded devices to achieve such goals as confidentiality, identification, data integrity, data origin authentication, and non-repudiation.
- Algorithm level: consisting of the design of cryptographic primitives (such as block ciphers and hash functions) and application-specific algorithms used at the protocol level.

- Architecture level: consisting of secure hardware/software partitioning, execution platform features and embedded software tactics to prevent, tolerate or remove faults.
- Hardware element level: it deals with the hardware design of the modules (the processors and co-processors) required and specified at the architecture level.
- Circuit level: it requires implementing transistor level and package-level techniques to thwart various physical-layer attacks.

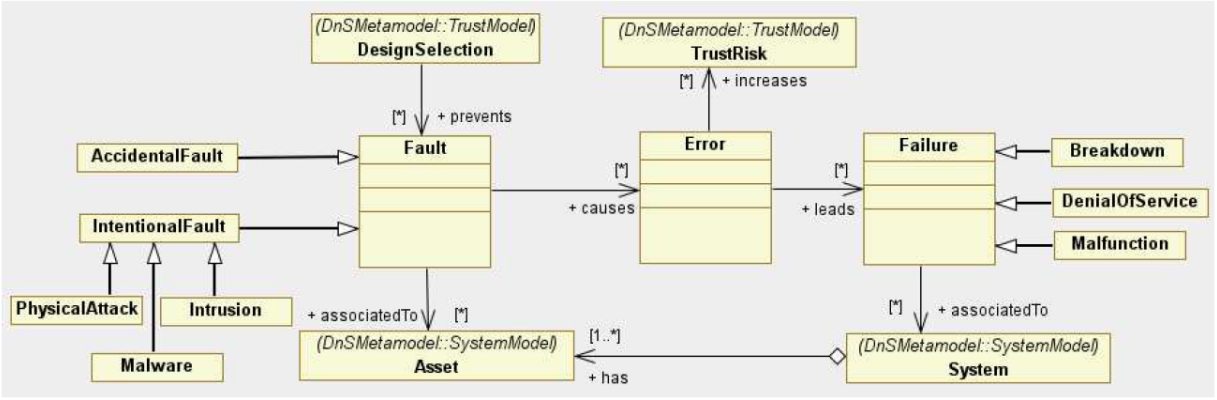

Figure 2.27: The EVITA model for Faults

With reference to Figure 2.27, the *Fault Progation* meta-model uses the "fault → error → failure" model. The fault model has a direct influence on security. If errors exist, they represent potential security lacks in the system. Security aspects will reduce the presence of fault. Traceability between security requirements and mistakes can be established. A *fault* is the adjudged or hypothesized cause of an *error*. Faults may be classified according to several criteria. EVITA classified them as accidental or intentional. Accidental faults can arise during either system development or operation. During development, accidental faults result from a bad design. During operation, they may be produced by the violation of an operating or maintenance procedure. Intentional faults fall into three classes: malware, physical attacks and intrusions. Typically an *attack* exploits a *vulnerability*, i.e., a fault that can be exploited of an asset to cause an intrusion. Attack may use physical means to cause faults such as power fluctuations, radiation, or wire-tapping.
An *error* is the part of the system state that may lead to a failure. An error is detected if its presence in the system is indicated by an error message or error signal that originates from within the system. Detected errors contribute to improving security and trust in the system. Errors that are present but not detected are latent errors.
Finally, *Failures* constitute the inability of the system, or some parts of it, to meet their specifications (both functional and non-functional requirements). Failures considered in the meta-model are: breakdown, denial of service and malfunction.
The EVITA meta-model comprises also several models of access control. Thanks to them, a user could specify access control rules in the model. All access control models are derived from a virtual class *Access Control* which *realizes* a given *TrustPolicy*. Figure 2.28 shows the UML model of the Role-Based Access Control (RBAC) policy [140]. The RBAC model defines mainly four entities which are modeled in UML as classes. The class *Role* that represents the role the user can play in the organization. This class can inherit from another class Role. Also, this class can belong to conflict sets (SSD for Static Separation of Duty and DSD for Dynamic Separation of Duty). This separation of duty states that a user cannot play at the same time two or more roles that belongs to the same conflict set. The class *User* that represents the user. A user can run one or mode sessions which will allow to activate one or more roles. The class *Permission* that defines which operation on which object

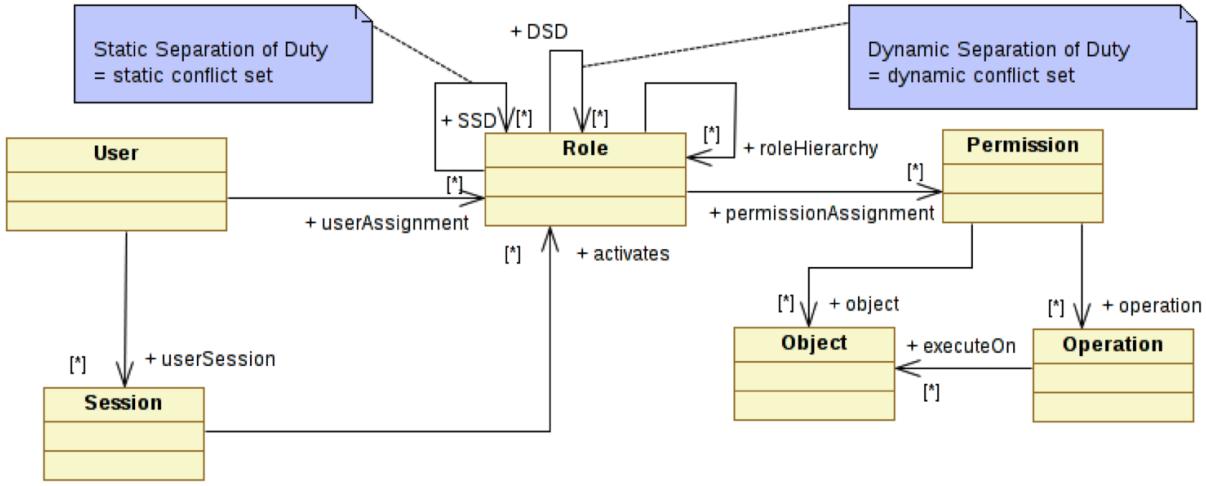

Figure 2.28: EVITA modeling of roles and access control policies

the user playing some role is allowed to perform. The class *Session* that is a means for the user to access the system.
Two mores classes are also defined. The class *Object* which represents the resources of the system that must be protected. The class *Operation* which represents the operations the system is able to perform. An operation can be executed on a instance of the class Object. The relationships *userAssignment* between the classes User and Role and *permissionAssignment* between the classes Role and Permission are established offline by the RBAC management which has not been presented in this document.
The meta-model makes it possible to define a security UML profile, very much like to SecureUML [88][89] and UMLSec [102], to be used in conjunction with other domain-specific UML profiles (e.g., automotive profile). The designer can thus apply several profiles on a model. The EVITA project has not defined any UML security profile.
It is worthwhile to notice that EVITA has also specified a set of security property predicates by means of the Fraunhofer SIT's Security Modelling Framework (SeMF). This led EVITA to define a number of Security Building Blocks (SeBBs), which constitute the means for security requirements refining. A number of SeBBs were incorporated into the EVITA architecture, notably those for encrypting/decrypting, generation/verification of a hash or digital signature and key generation.
Furthermore, another of the main contributions of the EVITA project consists in the definition of a *Hardware Security Module* (HSM). An HSM is a cryptographic co-processor that is integrated in the same chip as the application CPU for increased security, as it prevents an adversary from eavesdropping communications between HSM and CPU. Furthermore, the HSM has a programmable secure core (i.e., firmware) which guarantees more flexibility and reduced costs. There are three variants of HSM defined in EVITA, namely full, medium and light, which provide different levels of security and performance. These variants have been conceived to take into account the heterogeneity of on-board devices.
Figure 2.29 shows the architecture of the full EVITA HSM. Every HSM has its own internal CPU that can directly access its internal RAM and internal non-volatile memory (NVM). Separating the CPU prevents any malicious interference from the application CPU and the application software. The application CPU and its applications can access EVITA HSM only using the secure EVITA hardware function which enforces well-defined access (e.g., to prevent read-out of cryptographic keys).
The reduced variants medium and light lack one or more of the security features that are present in the full variant. Table 2.3 shows the security features provided by the three variants, respectively. Of course, if a variant does not support a security feature, then the related architectural component is missing.

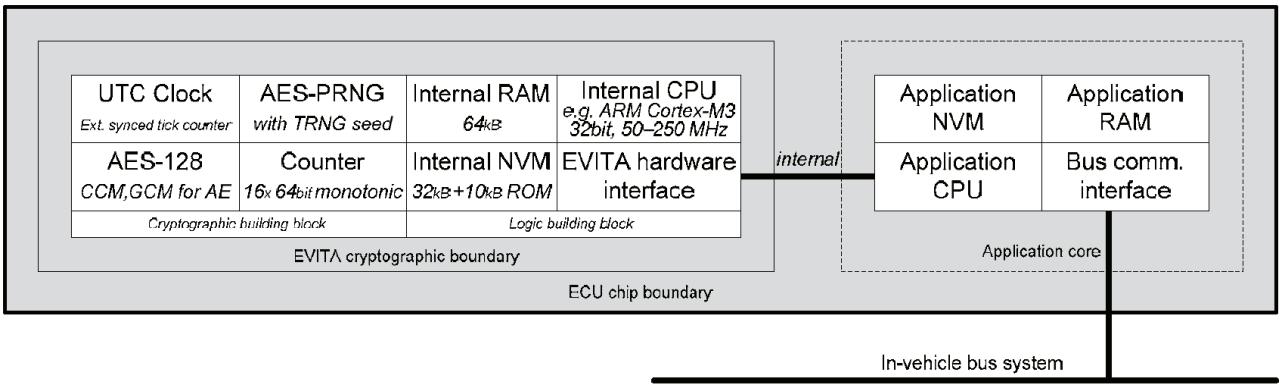

Figure 2.29: EVITA full HSM

	low	medium	full
Asymmetric cryptography (ECC-256)	-	-	x
Symmetric cryptography (AES-128)	x	x	x
Hash (WHIRLPOOL)	-	-	x
AES-PRNG with TRNG seed	x	x	x
Counters	-	x	x

Table 2.3: Security features of the HSM variants

To fix ideas, the full HSM can be used for V2X applications, i.e., vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications, whereas the medium HSM can be used at the ECU level. Finally, the light HSM can be used at sensor and actuator level.
HSM is the key component for the EVITA security protocols including key distribution, policy management and access control, secure bootstrapping, secure over-the-air firmware updates, secure transport and storage [66].
The HSM is abstracted primarily by the *Cryptographic Services* (CRS) module, which offers an API for various cryptographic functions. CRS provide an interface to all basic cryptographic functionalities and primitives and are therefore not a module but rather a library that can be integrated by a security module or security program that requires cryptographic services. Thus, CRS is stateless and CRS users have to manage the corresponding security credentials (e.g., keys, passwords, or random numbers) and the context of every operation at its own.Some functions have not yet been available in CRS (e.g., for creating cryptographic keys). Figure 2.30 shows a model of CRS.
Particularly useful is also the concept of *CryptographicObject* (see Figure 2.31) that represents an instance of a cryptographic key, a hash value, a MAC value, or a random value used in cryptographic operations, protocols, and algorithms. Note, a cryptographic object itself does not store its context, and thus, it is used linked with a context object (e.g., a security credential). A cryptographic object is further specialized for each *CryptoType* (e.g., *SymmetricKey*). It can handle its cryptographic content directly (e.g., using a char array for a key) or it may represent just a reference to a cryptographic object that actually resides protected in a security hardware.

The EURO-MILS project

The EURO-MILS consortium[3] issues guidelines and criteria for protection profiles for systems with virtualization. These profiles provide recommendations for operating systems that execute different

[3]http://www.euromils.eu

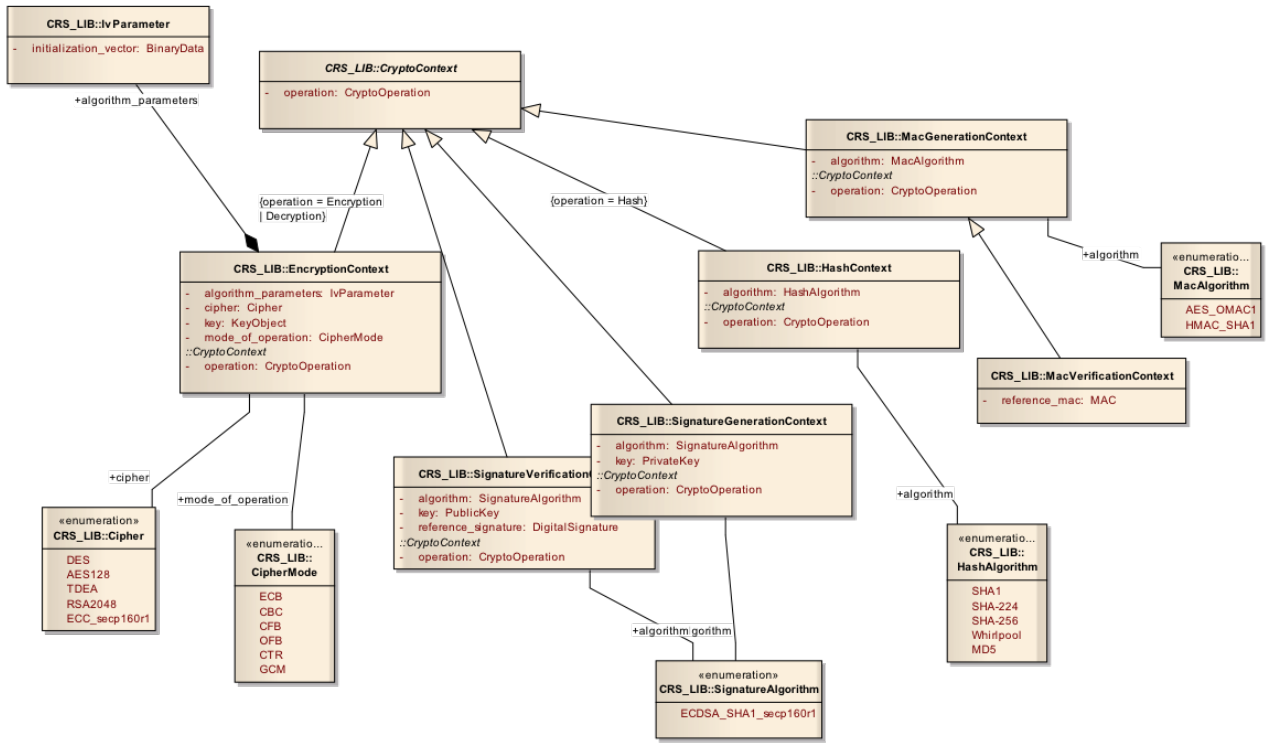

Figure 2.30: The Evita Cryptographic Services

applications with separation of concerns ensuring that applications with malicious or faulty behavior do not affect other applications.
The target is an operating system providing a *separation kernel with real-time support*. The architecture of the system is outlined in the document as in Figure 2.32.
The operating system implements the mechanisms to assign resources to partitions, provides the execution environment to applications and implements the communication between partitions.
The kernel maintains configuration data for the system security policies and the description of all its managed entities (identity, resource usage, security attributes of entities).
A partition is a logical unit maintained by the kernel and retains control over a set of assigned resources (physical memory space, IO space, CPUs, time allocated on the CPU and interrupts). Partitions can be of type user or system and communicate under the supervision of the kernel using communication objects. Each partition has a set of access rights on each communication object.
The security services provided by a kernel are defined as TSS:

- TSS_SSA for separation in space among the applications in different partitions and from the OS. Partitions are assigned resources in space, including memory ranges and a set of CPUs, an I/O (memory) space and interrupts.
- TSS_STA separation in time.
- TSS_COM provision and management of communication objects.
- TSS_MAN management and access to kernel and kernel data (invokability of system API).
- TSS_SPT self-protection and accuracy of security functionality.
- TSS_AUD generation and treatment of audit data.

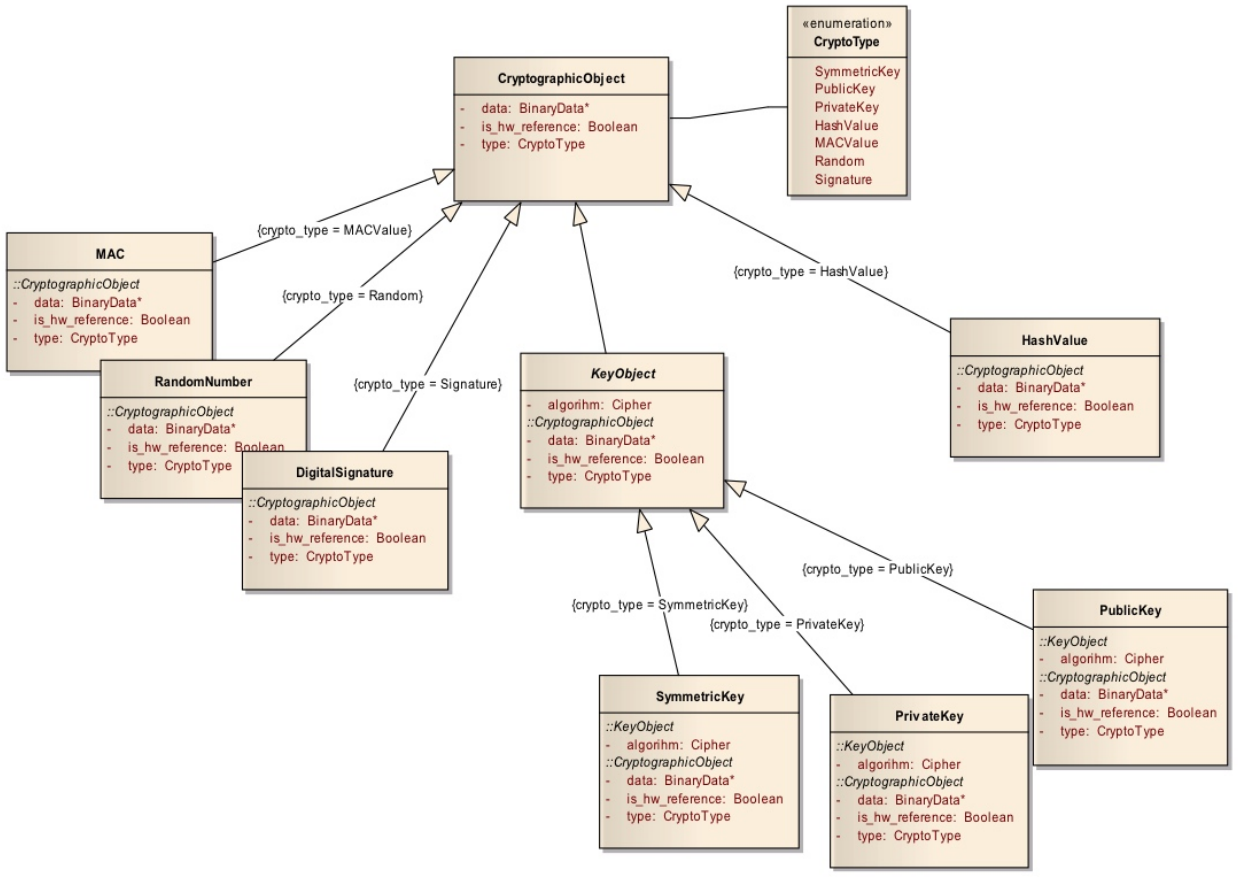

Figure 2.31: Cryptographic objects

The recommendation lists the primary and secondary assets to be managed, the subjects and roles involved in interactions with the system and its components, the security threats, the security policies and objectives.

2.2 Standardization bodies and Best practices

2.2.1 Safety

Standards are available to developers of safety-critical systems. In particular the "IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)" is an international standard of rules for systems comprised of electrical, electronic, and software components applied in industry. The standard provides the assurance that safety-related systems will offer the necessary risk reduction required to achieve safety. Central to the standard are the concepts of safety life cycle, risk and safety functions and safety integrity levels.

ISO 26262

In the Automotive application field, "ISO 26262: Road vehicles — Functional safety" [83] is the adaptation of IEC 61508 specific to the application sector of electrical and electronic systems in the road vehicle industry.
"ISO 26262:

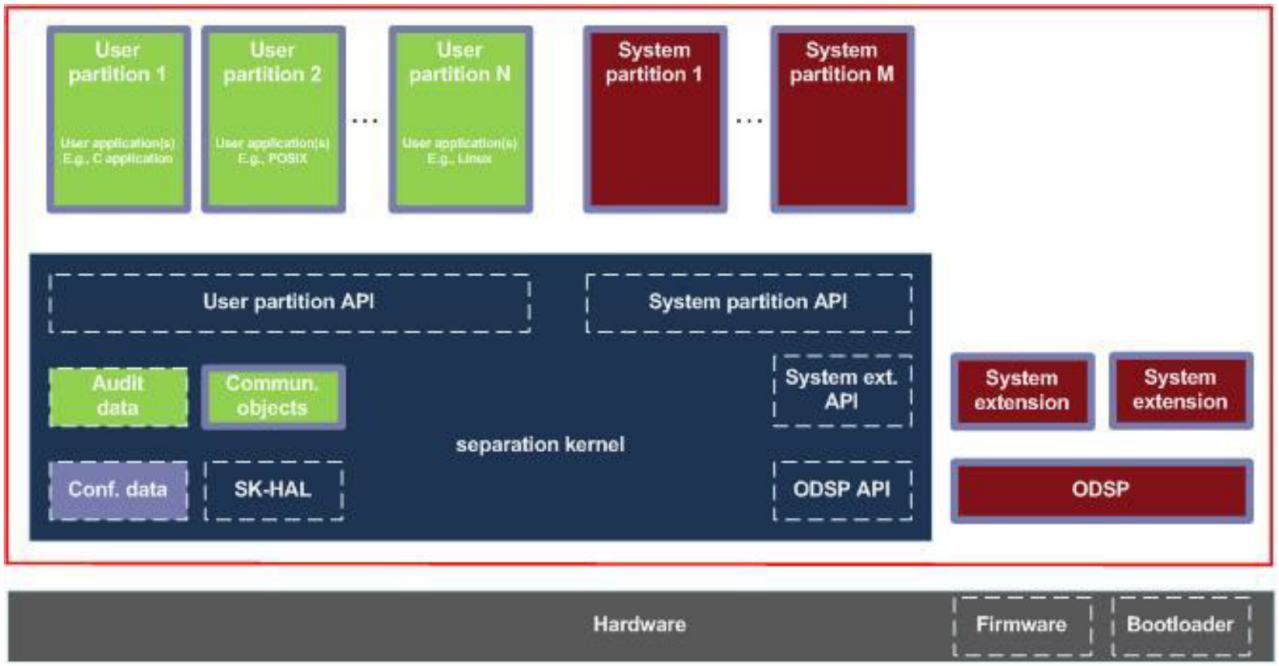

Figure 2.32: The architecture of an operating system with isolation according to ([4]).

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

The overall structure of ISO 26262 is shown in Figure 2.33. It is structured into the following parts:

1. Vocabulary;
2. Management of Functional Safety;
3. Concept Phase;
4. Product Development: System Level;
5. Product Development: Hardware Level;
6. Product Development: Software Level;
7. Production and Operation;
8. Supporting Processes;
9. ASIL-oriented and Safety-oriented Analyses;
10. Guidelines on ISO 26262.

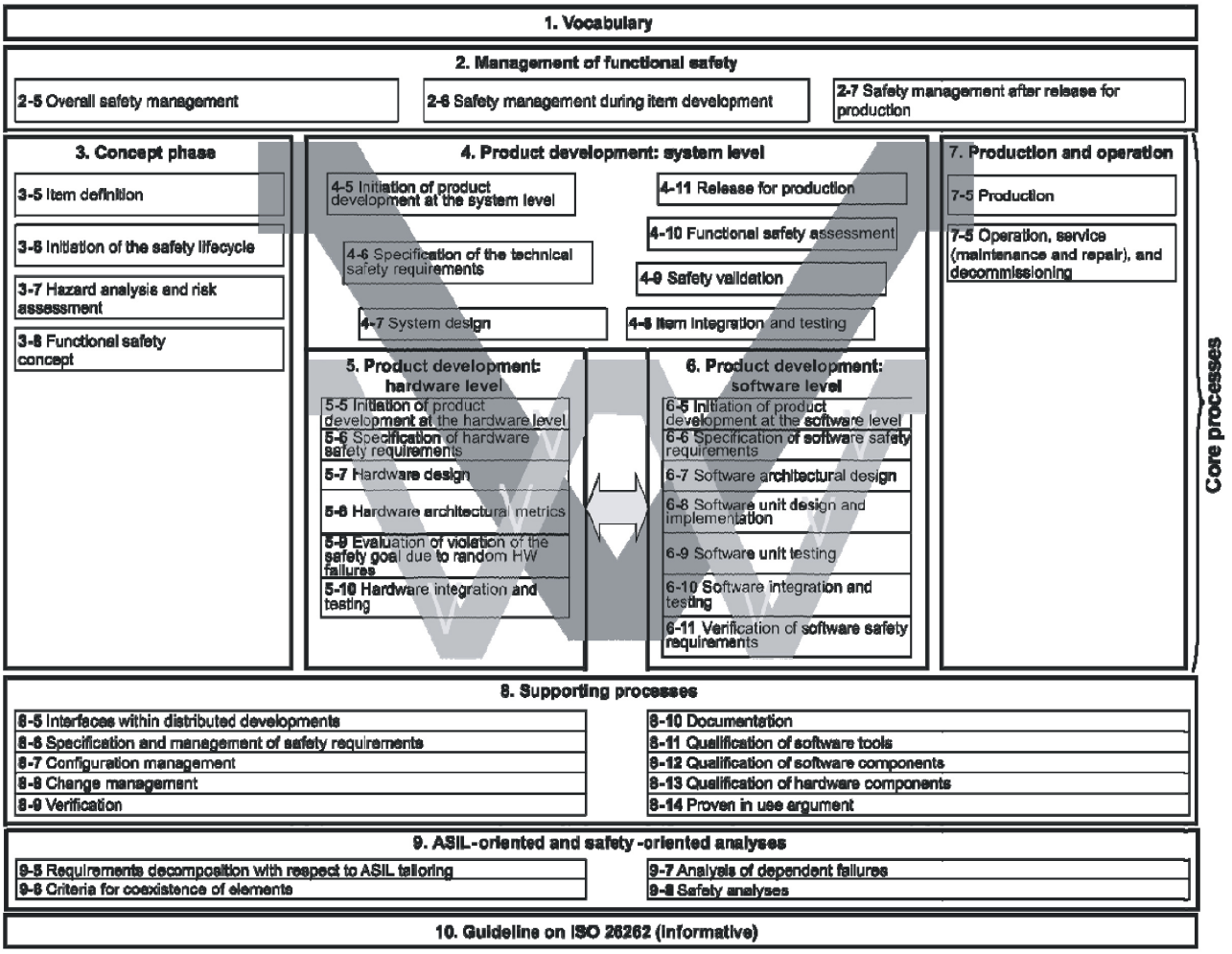

Figure 2.33: Overview of ISO 26262.

The standard comprises 10 parts and covers the product development phase, ranging from the specification, to design, implementation, integration, verification, validation, and production release. It is is based upon a V-Model as a reference process model for the different phases of product development. The production of a functional safety case is a requirement for compliance with the standard. The lifecycle starts from a description of the item. Basic rules are:

- Develop the item from the system perspective.
- Develop the item from the hardware perspective, based on the system design specification.
- Develop the item from the software perspective, based on the system design specification.
- Plan production and operation, and specify the associated requirements, during product development.

The standard contains a part dedicated to terminology (Part 1) and a part dedicated to guidance on applying the standard (Part 10). In the following a short description of the other parts is given.

2. Management of Functional Safety

Functional Safety is the part of the overall safety of a system, or piece of equipment, that depends on the system or equipment operating correctly in response to its inputs, including the safe management of likely operator errors, hardware failures and environmental changes.

The objective of Functional Safety is freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment). This lifecycle encompasses principal safety activities during concept, development and production phases.
Referring to the figure, the following clauses are identified:

- Overall Safety Management
  the outcomes of this clause are a set of organization-specific rules and processes for functional safety, evidence for the competence and qualification of the persons in charge of carrying out the activities and evidence of a proper quality management system.

- Safety Management during Item Development
  this clause aims at the definition of safety management roles and responsibilities, and the definition of the requirements on the safety management, regarding the development phases.

- Safety management after release for production
  this clause defines the responsibility of the organizations and persons responsible for functional safety after release for production. This concerns activities for maintaining the functional safety of the item in the lifecycle phases after release.

3. Concept Phase

This part is organized in the following clauses:

- Item definition, Initiation of the safety lifecycle
  The goals of these clauses is to define and describe the item and support an adequate understanding so that each activity of the safety lifecycle can be performed

- Hazard Analysis and Risk Assessment
  The hazards of the item shall be systematically determined, with techniques such as checklists and FMEA, in terms of the conditions or events that can be observed at the vehicle level. The effects of hazards shall be identified for relevant operational situations. All identified hazards shall be classified with respect to severity, probability of exposure or controllability. ASIL shall be determined for each hazardous event using the proper combination of the previous parameters. A safety goal shall be determined for each hazard, and expressed in terms of functional objectives.

- Functional Safety Concept
  The goal of this clause is to derive the functional safety requirements, from the safety goals, and to allocate them to the preliminary architectural elements so to ensure required safety.

4. Product Development: System Level

Many clauses are identified. Basically, the objectives of this part are:

- determine and plan the functional safety activities during the subphases of the system development, included in the safety plan.

- develop the technical safety requirements, which refine the functional safety concept considering the preliminary architectural design.

- verify through analysis that technical safety requirements comply to the functional safety requirements. The response of the system or any of its elements to stimuli, including failures shall be specified for each technical requirement, in combination for each possible operating state.

5. Product Development: Hardware Level

This part consists of the following clauses: Initiation of Product Development at the Hardware Level, Specification of Hardware safety requirements, Hardware design, Hardware Architectural Metrics, Evaluation of Violation of the Safety Goal due to Random HW Failures, Hardware integration and testing.
For example, the scope of the Initiation of Product Development at the Hardware Level clause is to determine and plan the functional safety activities during the individual subphases of hardware development, which is included in the safety plan (see Figure 2.34). This activity includes the Hardware implementation of the technical safety concept; the Analysis of potential faults and their effects; and the Coordination with software development.

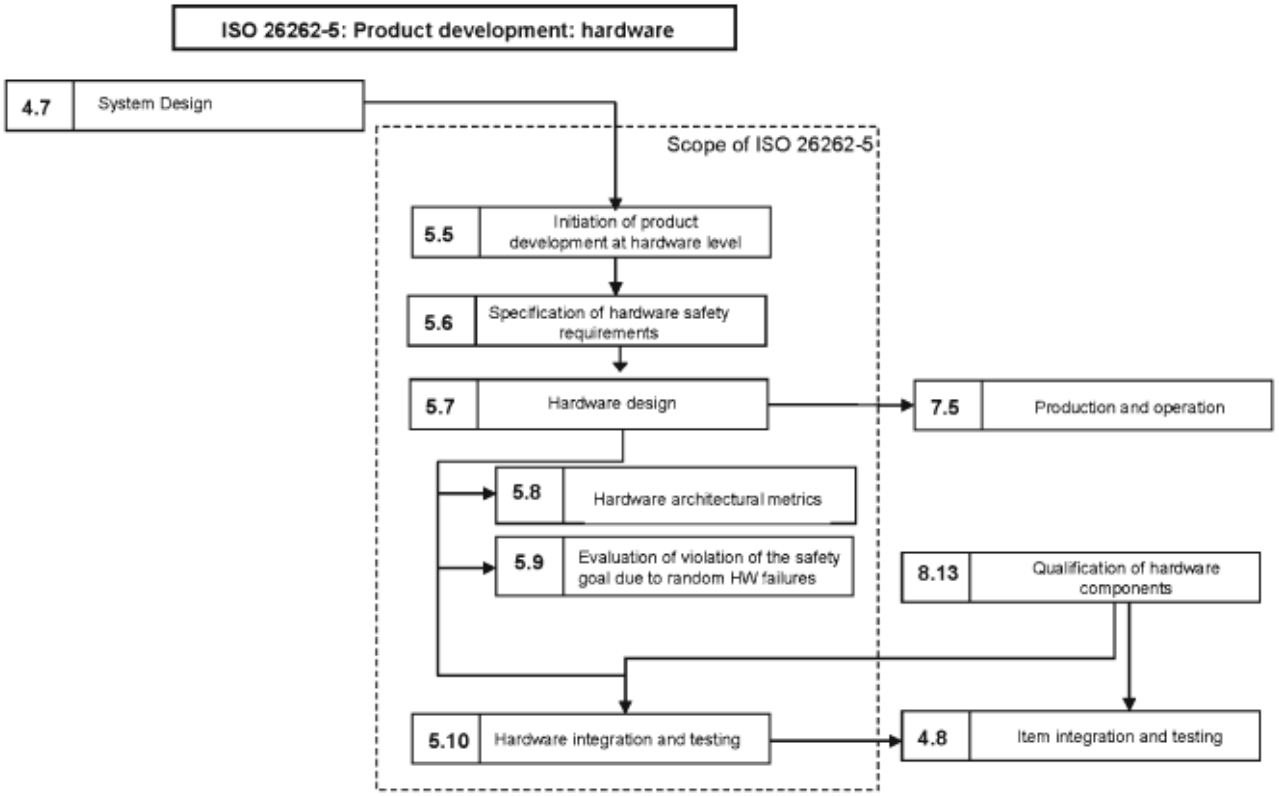

Figure 2.34: Reference phase model for the development of a safety related item[83].

The scope of Hardware Architectural Metrics clause is to infer if the residual risk of safety goal violation, due to random hardware failures of the item, is sufficiently low. Hardware Architectural Metrics aims to evaluate the hardware architecture of the item against the requirements for fault handling as represented by the hardware architectural metrics. The considered metrics are: diagnostic coverage, single point faults metric and latent fault metric.
The scope of Evaluation of Violation of the Safety Goal due to Random HW Failures is to infer if the residual risk of safety goal violation, due to random hardware failures of the item, is sufficiently low.

6. Product Development: Software Level

The reference phase model for the software development process for an item is shown in Figure 2.35. For example, the objective of the Software architectural design clause is to develop a software architectural design that realizes the software safety requirements and to verify the software architectural design. The software architectural design shall be verified by using the software architectural design

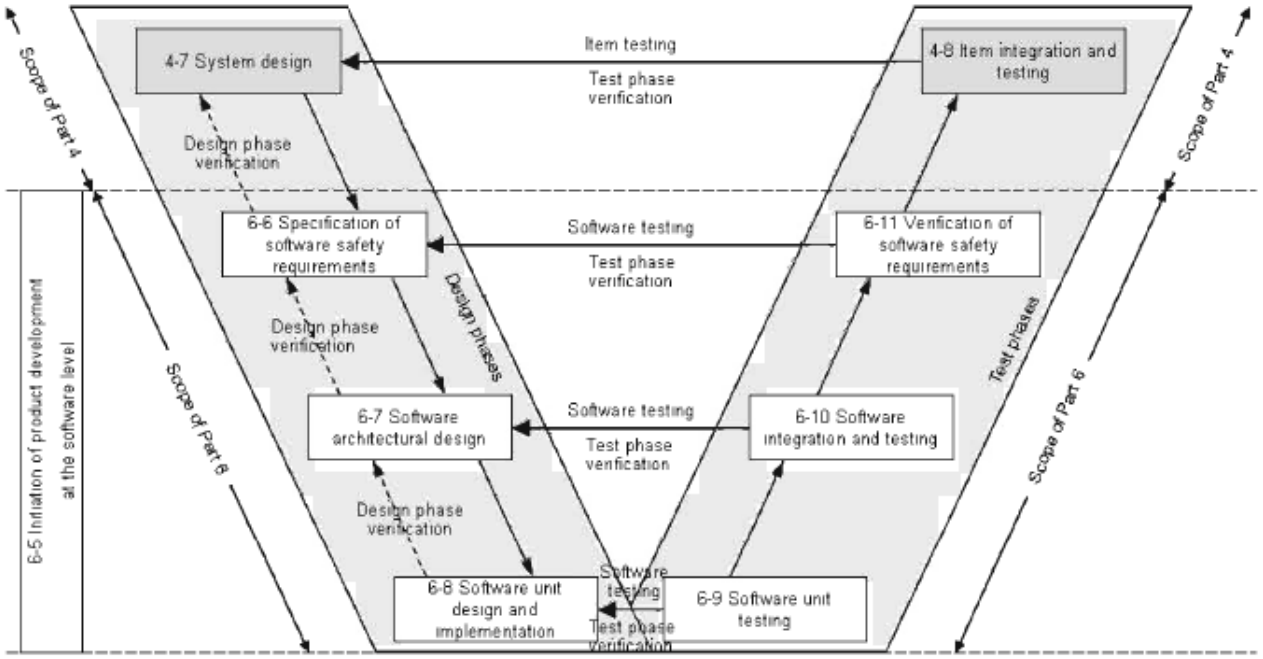

Figure 2.35: Reference phase model for the software development [83].

verification methods shown in Figure 2.36.

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Informal verification by walkthrough of the design[a]	++	+	o	o
1b	Informal verification by inspection of the design[a]	+	++	++	++
1c	Semi-formal verification by simulating dynamic parts of the design[b]	+	+	+	+
1d	Semi-formal verification by prototype generation / animation	o	o	+	+
1e	Formal verification	o	o	+	+
1f	Control flow analysis[c, d]	+	+	++	++
1g	Data flow analysis[c, d]	+	+	++	++

[a] Informal verification is used to assess whether the software requirements are completely and correctly refined and realised in the software architectural design. In the case of model-based development this method can be applied to the model.

[b] Method 1c requires the usage of executable models for the dynamic parts of the software architecture.

[c] Control and data flow analysis can be carried out informally, semi-formally or formally.

[d] Control and data flow analysis may be limited to safety-related components and their interfaces.

Figure 2.36: Methods for the verification of the software architectural design [83].

7. Production and Operation

This part specifies requirements on production, operation, service, and decommissioning. In particular the Production aims at developing a production plan for safety-related products and to ensure that the required functional safety is achieved during the production process.

8. Supporting Processes

This part consists of the following clauses: Interfaces within distributed developments, Specification and management of safety requirements, Configuration management, Change management, Verification, Documentation, Qualification of software tools, Qualification of software components, Qualification of hardware components, Proven in use argument. For example, Qualification of software tools requires using appropriate tools to support project activities. The objective of Verification is to ensure that all work products are correct, complete, and consistent; and that all work products meet the requirements of ISO 26262. The objective of Documentation is to develop a documentation management strategy so that every phase of the entire safety lifecycle can be executed effectively and can be reproduced.

9. ASIL-oriented and Safety-oriented Analyses

This part includes the activities on Requirements decomposition with respect to ASIL tailoring, Criteria for coexistence of elements, Analysis of Dependent Failures and Safety Analyses. The evaluation for dependent failures is fundamental in order to identify any single cause that could bypass or invalidate the independence or freedom from interference between elements of an item required to comply with its safety goals.

AUTOSAR Safety Extensions

In the automotive industry, the adherence of an AUTOSAR standardized software architecture to ISO 26262 with respect to functional safety is required. The document "Specification of Safety Extensions" [21] covers safety extensions that shall enable ISO 262626 development in an AUTOSAR context. These extensions allow a standardized exchange of safety information and provide the basis for consistent management as required by ISO 26262.
In addition to safety mechanisms provided by the AUTOSAR standard (e.g. memory partitioning, end-to-end-protection), additional requirements need to be addressed for functional safety:

- Safety requirements must be clearly distinguishable from other requirements
- Safety integrity levels must be assigned to AUTOSAR elements following the schema of ISO 26262. It is worth mentioning that the requirements prescribe the definition of a SIL level to every attribute, which is most likely unnecessary. The other sections of the document clarify that at least they should be defined for hardware systems (ECUs for example) and SW components.
- Decomposition of safety requirements, Traceability of safety requirements and Safety measures must be allowed according to ISO 26262;
- Safety measures and safety mechanisms as required by ISO, intended as an abstract (general) way to reference any safety measure of the system architecture.

These requirements are addressed by using existing metamodels concepts. The metamodel for the description of safety requirements in AUTOSAR is shown in Figure 2.37. Each safety requirement must be allocated to an element of the system architecture, i.e. to a component in the HW, SW architecture, or both (HW and SW).
Requirements are modeled by the **StructuredReq** class, and the traceability of requirements, according to the ISO 26262, is guaranteed by the inherited abstract class **Traceable**.
The **category** attribute (inherited in **StructuredReq** by class **Identifiable**) is used to specify the type of safety requirement (safety goal, safety functional, safety technical, safety software, safety hardware and safety external). Safety requirements are expressed as part of the standard AUTOSAR XML format.

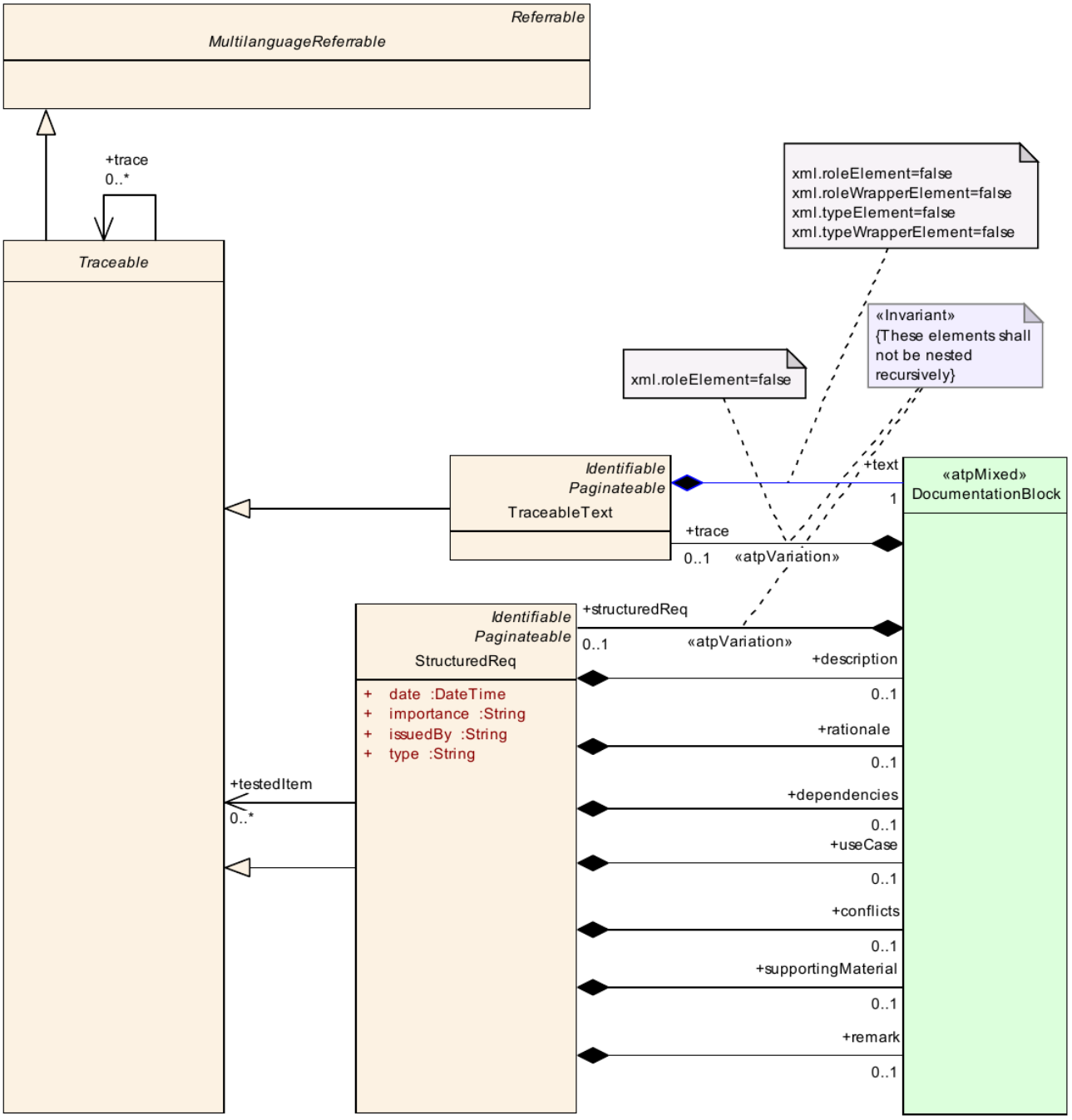

Figure 2.37: Safety requirements metamodel ([24]).

Figure 2.38, taken from [21], shows a hierarchy of safety requirements and the allocation to system architecture elements. The ASIL assignment is inherited through the lower levels. In the figure, a technical safety requirement is allocated to μC, a hardware safety requirement is allocated to Watchdog and a software safety requirement is assigned to a sub-component of μC .
Figure 2.39, taken from [21], shows the abstraction of the different safety mechanisms available in the software stack and the ECU hardware. Safety mechanisms can be implemented as hardware components (as is the case of the Watchdog SM) in the figure, by basic software components, as is the case of the partitioning safety mechanism (a concept very close to that of a protection kernel) or even at the application level.
Clearly, the definition of ASIL levels for a subset of the AUTOSAR elements brings into attention the need for a consistency analysis and/or a set of rules to (re)evaluate the ASIL levels that are resulting

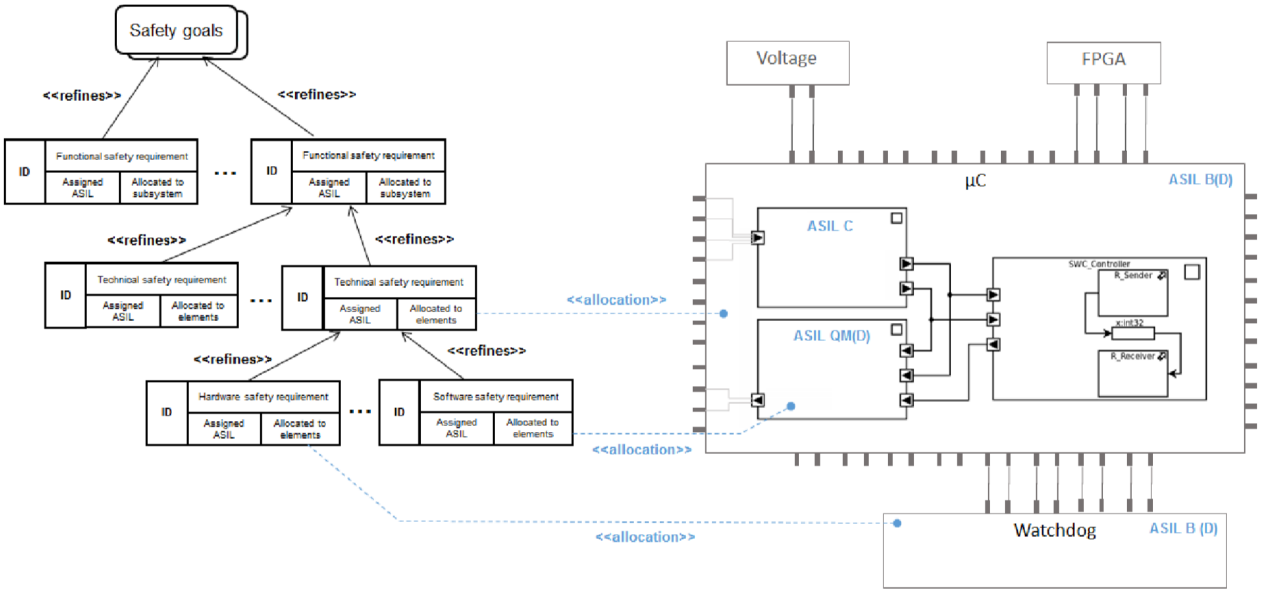

Figure 2.38: Hierarchy of safety requirements and allocation to system architecture elements ([21]).

from mapping of AUTOSAR components onto partitions and then onto ECUs.

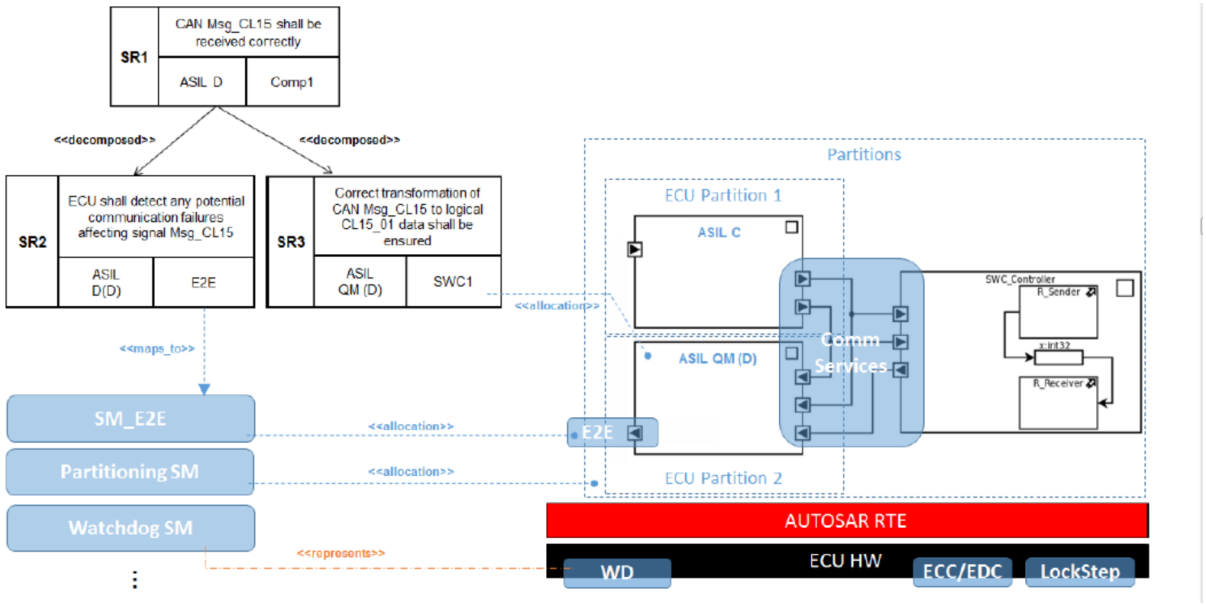

Figure 2.39: Safety measures, safety requirements and allocations to elements of the architecture ([21]).

Some functional safety-measures are not enforced or delivered in AUTOSAR. For example, the AUTOSAR Specification does not define the use of techniques for risk analysis, such as Hazard Analysis (HARA) [19].
Recent works on AUTOSAR and ISO26262 are, for example, [127], where tools to automatically implement semi-formal safety requirements are presented, and traceability information is used for safety case documentation, and [141], where a tool is provided to automatically extract relevant functional requirements for given safety scenarios.

2.2.2 Time

AUTOSAR is a software architecture standard in the automotive domain which enables by the definition of software layers and interfaces, the design and integration of software modules by independent developer teams. Due to the importance of real-time guarantees in cyber-physical systems, the AUTOSAR standard considers from version 4.0 timing relevant aspects. These are specified in the document *AUTOSAR Specification of Timing Extensions* [23]. The consistent consideration of timing properties and timing constraints in the AUTOSAR model allows to perform subsequent timing analysis and validate that all timing related requirements are met, as proposed in [25].

Timing Description and Timing Constraints

The description of timing in AUTOSAR is based on the Timing Augmented Description Language (TADL) as proposed in the TIMMO project. Therefore, the notion of time in AUTOSAR is event-based. An **event** describes a certain system behavior which occurs at a certain point during run time (temporal dimension) and at a specific location (local dimension). Since AUTOSAR defines different abstractions of the software architecture (views), observable event types (**TimingDescriptionEvent**) are associated with different views.
The causal relationship between events is described with the concept of **event chains**. An event chain formed by two (consecutive) events A and B expresses that event A is a stimulus for event B. In general, an event chain can be composed of elementary event chain segments and may contain branches and junctions.
The notion of **timing constraints**, which formalize timing requirements that are associated with the system, is based on events and event chains.
An **Event Triggering Constraint** restricts the temporal pattern of event occurrences by indicating e.g. period, jitter, minimum interarrival time.
Latency and synchronization timing constraints refer to restrictions on event chains. The latency constraint specifies the duration between the occurrence of the stimulus event and the occurrence of the response event where the causal relationship between the events is defined by the underlying event chain. Latency is adequate to constrain the maximum reaction period (e.g. elapsed time between stimulus and first response) or the maximum age of the stimulus event (e.g. elapsed time between latest stimulus event to response).
In the scope of the AUTOSAR Timing Extension or TIMEX, there must be a causal dependency between the source and the target event of a latency constraint. In case there is no such dependency, an offset constraint can be specified instead.
A synchronization constraint restricts the amount of time (tolerance) within which events must occur to satisfy the synchronization requirement.
With respect to executable entities, AUTOSAR defines **Execution Order Constraints** and **Execution Time Constraints**.
AUTOSAR timing extension are integrated in the common UML meta model by integrating timing properties and requirements in the different AUTOSAR templates.

Timing Views

The AUTOSAR views and the corresponding timing views are

- VFB Timing and Virtual Function Bus View,
- SW-C Timing and Software Component View,
- System Timing and System View,
- BSW Module Timing and Basic Software Module View,
- ECU Timing and ECU View.

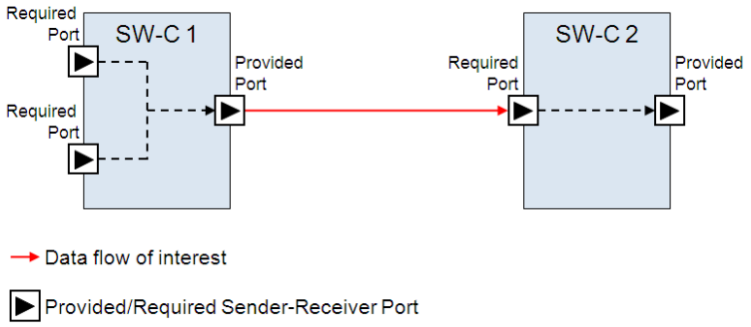

Figure 2.40: Scope of the VFB Timing [23]

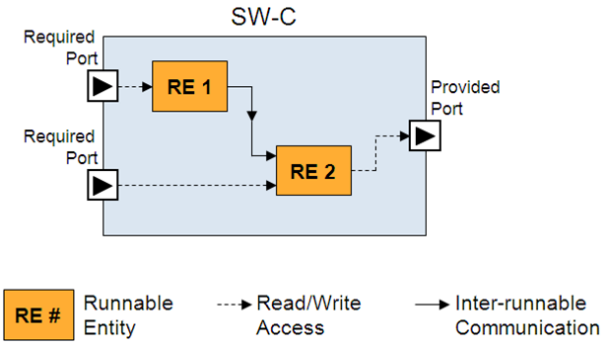

Figure 2.41: Scope of the SW Component Timing [23]

VFB Timing The VFB timing View is an abstraction, which describes at a logical level the communication between the SW components that interact with each other in order to exchange AUTOSAR Services. The VFB View has no notion of the mapping of SW components to ECUs and the concrete communication technology, therefore SW components are treated as black boxes and there is no notion of execution time or messaging latency times, therefore any analysis is abstract and based on the verification of the basic consistency of the requirements. A timing description will therefore relate either

- (1) to a single SW component specifying the timing constraints (e.g. latency) between the SW component's receiver ports and provided ports or,
- (2) to a composition of SW components and their interconnection specifying the timing constraints (e.g. end-to-end-latency) between receiver ports and provided ports.

An event that is observable in the VFB abstraction are classified as *TDEventVfb*.
Figure 2.40 shows an example of a typical scenario for the analysis, where the flow of information between ports and the end-to-end paths are the subject of the analysis..

SW Component Timing The SW Component View details the VFB view by including the internal behavior of SW components, i.e. the composition of SW component of runnable entities, in the abstraction. In the SW Component Timing events are either of type *TDEventVfb* and *TDEventSwcInternalBehavior*.
Figure 2.41 shows an example of a scenario for this type of description where the role of runnables (internal functions triggered in response to events) is specified.

System Timing At the system abstraction level, the employed hardware and its interconnection (bus, network) is known. The mapping of SW components to ECUs has been decided and the necessary

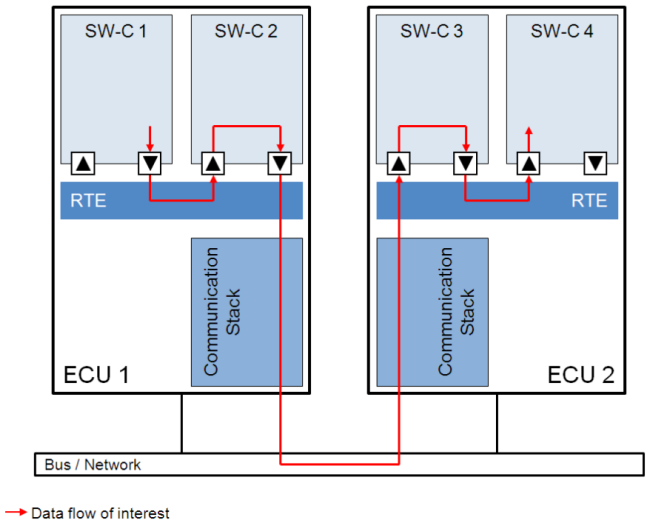

Figure 2.42: Scope of the System Timing [23]

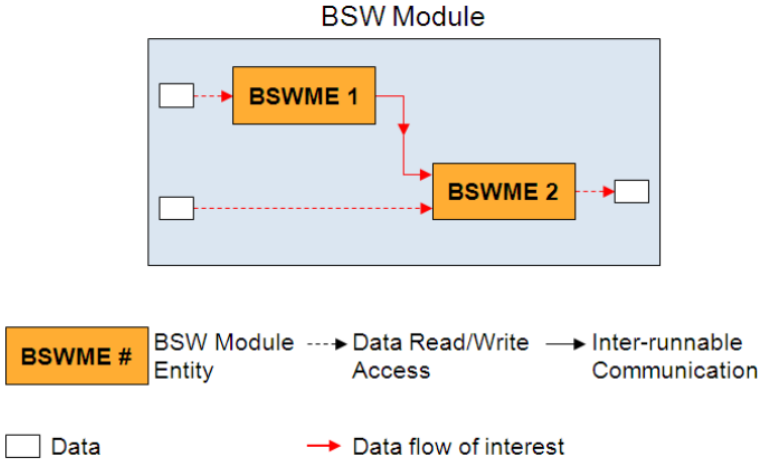

Figure 2.43: Scope of the BSW Module Timing [23]

RTE is provided, which masks the complexity of inter-ECU and intra-ECU communication over the distributed system. In the SystemTiming view, timing properties and constraints may therefore not only relate to VFB Timing or SW Component Timing aspects but, additionally, to concrete signals and frames. Events are of type *TDEventVfb*, *TDEventSwcInternalBehavior* and/or *TDEventCom*. For illustration refer to Figure 2.42.

BSW Module Timing A Basic Software Module (BSW module) is a collection of software files that define a certain basic software functionality present on an ECU [18]. A BSW module is, similar to a SW component, composed of individual BSW entities which describe the internal behavior of the BSW module. The event class associated with this timing view is *TDEventBswInternalBehavior* and mostly focuses on state changes related to the execution of an BSW entity. Figure 2.43 shows an example highlighting the capability of analyzing the communications over the network in an information/event flow and the cooperation of remote functionality hosted by different ECUs.

ECU Timing The ECU timing view is similar to the system timing abstraction, however, only the individual ECU as an element of the entire system is considered. The main focus of this view is to describe BSW modules, their internal behavior and the interaction of BS modules with each other.

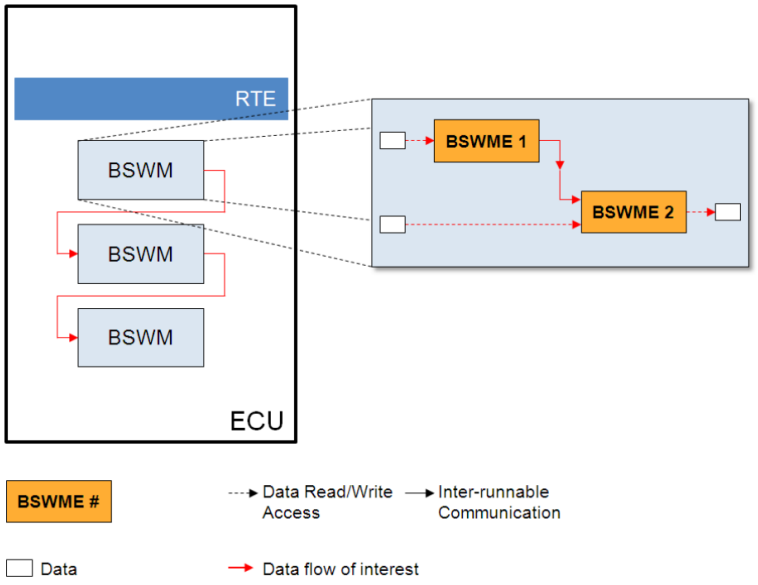

Figure 2.44: Scope of the ECU Timing [23]

Any TimingDescriptionEvent may be used in the ECU timing view.
For illustration refer to Figure 2.44.

AUTOSAR modeling of Timing Constraints and properties

Formally, the AUTOSAR modeling of Time is summarized in the Metamodel elements of Figure 2.45. A timing extension consists of a set of timing descriptions and timing constraints. The latter are further characterized as requirements or guarantees.
A Timing Description or timing attribute applies to different entities in AUTOSAR, as summarized in Figure 2.46. The element TimingDescriptionEvent and its specializations are used to describe the occurrence in a given time instant of an event within the system. For example, this can be the start of a RunnableEntity or storing a frame in the hardware buffer of a communication controller. An overview of the different event types is given in the metamodel snapshot of Figure 2.46.
Events are further refined with respect to their applicability to operations (requests or reponses) or data communication, as shown in Figure 2.47. Figure 2.48 shows the details of events that apply to operations.
Other timed events apply to the definition of the internal behavior of components, as shown in Figure 2.49, such as, for example to the activation, start and completion in the execution of runnables.
AUTOSAR Timing descriptions are used to define formulas for the expression of timing constraints and timing properties. We omit the full description of the metamodel and the syntax for timing description. To give an example, Figure 2.50 shows the types of time expression constraints that apply to the event occurrence in time.
The element TimingDescriptionEventChain is used to specify a causal relationship between timing description events and allows to identify causality chains in the system and to refer to the event chain as a whole, attaching latency (deadline) constraints to it.
Figure 2.52 shows an example of an event chain to which an end-to-end constraint applies. The event chain can refer to a specification at any level (Vfb, Component or System).
Finally, a section of the metamodel is dedicated to the expression of timing constraints, in their most general form. The table in Figure 2.53 provides the list of the constraints that are supported by the AUTOSAR specification and the metamodeling elements to which they apply.
Constraints are roughly divided in constraints that apply to the event arrivals, defining their dependencies and periodicity (or time relation, the corresponding metamodel snapshot in Figure 2.54); and

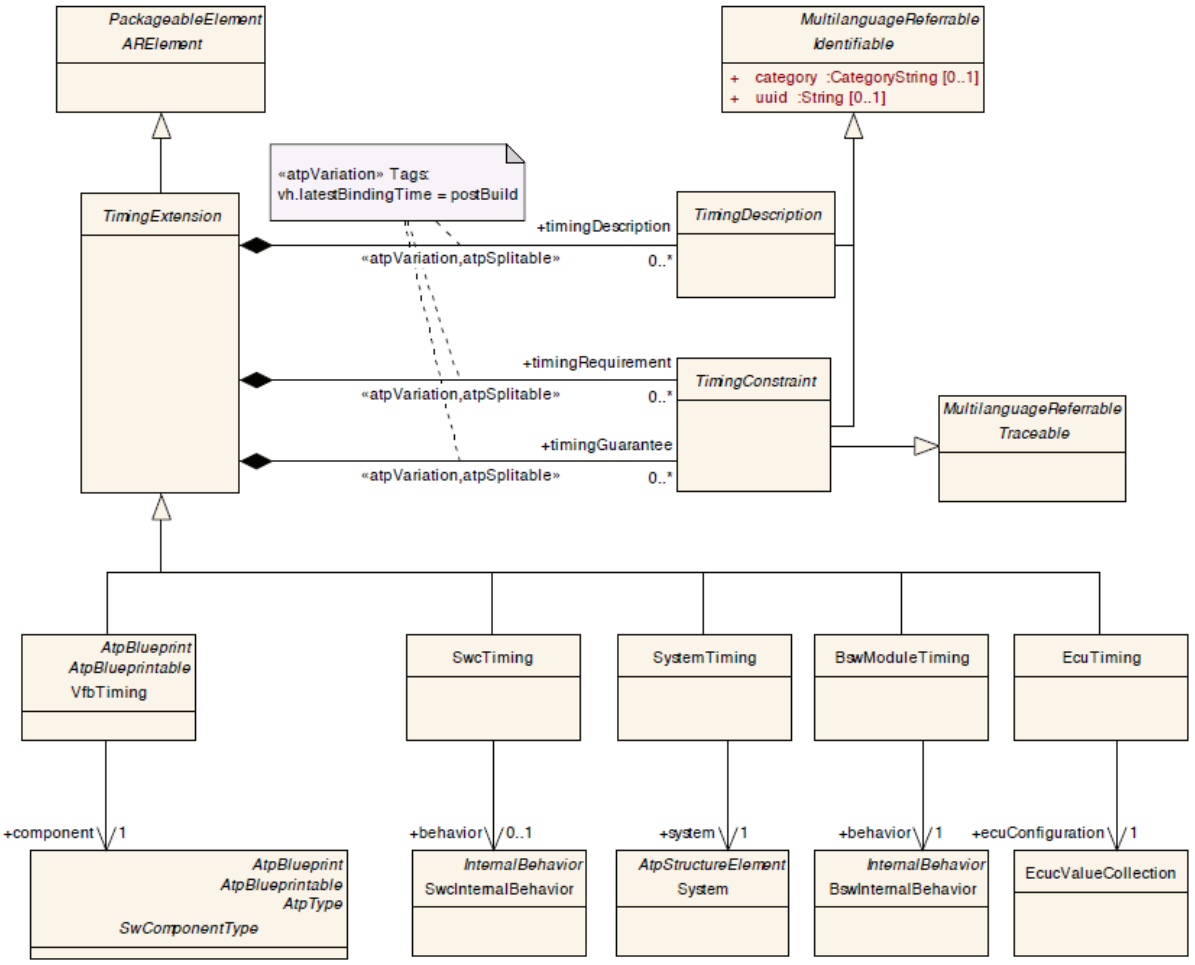

Figure 2.45: The AUTOSAR framework for timing extensions.

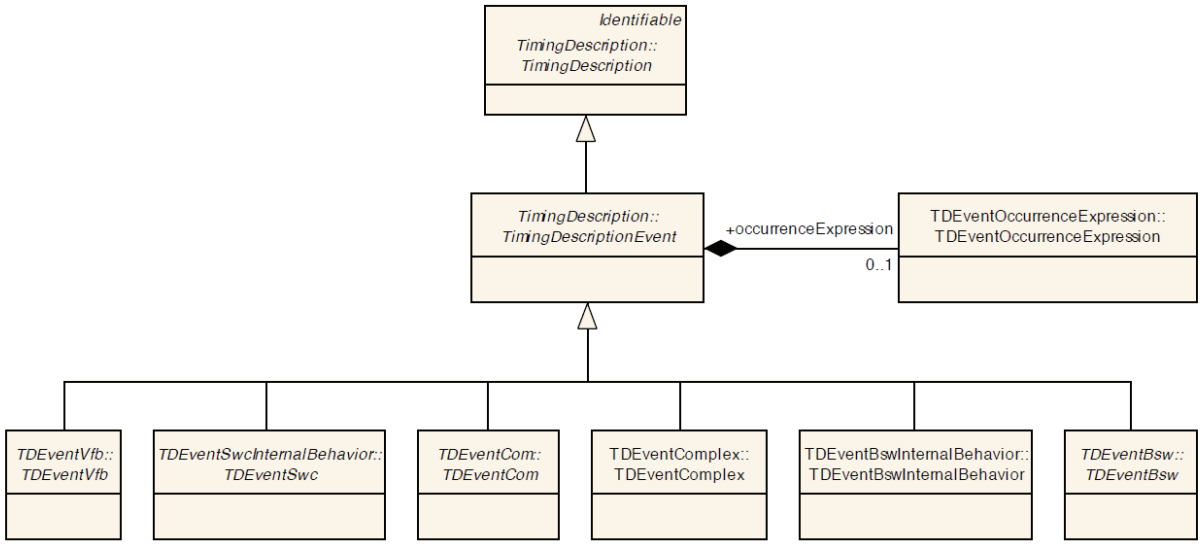

Figure 2.46: Timing descriptions in AUTOSAR

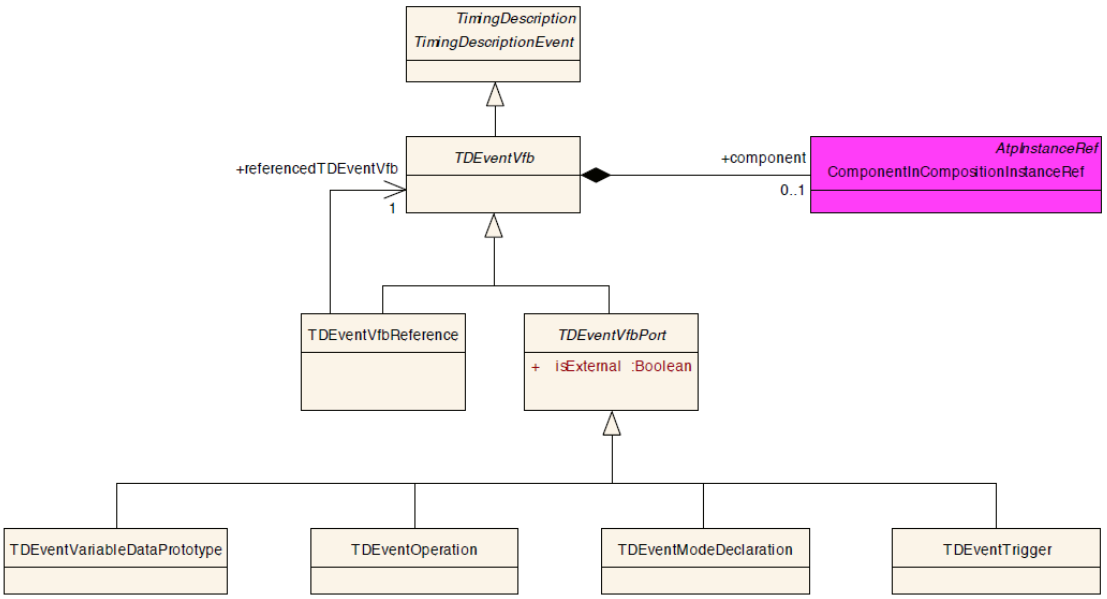

Figure 2.47: The general classification of timed events

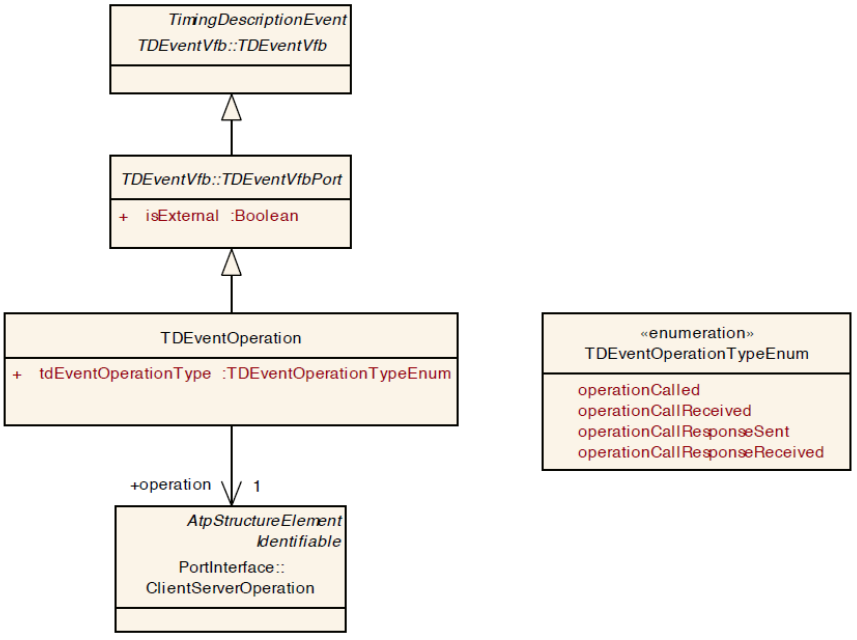

Figure 2.48: Timed events applicable to operations

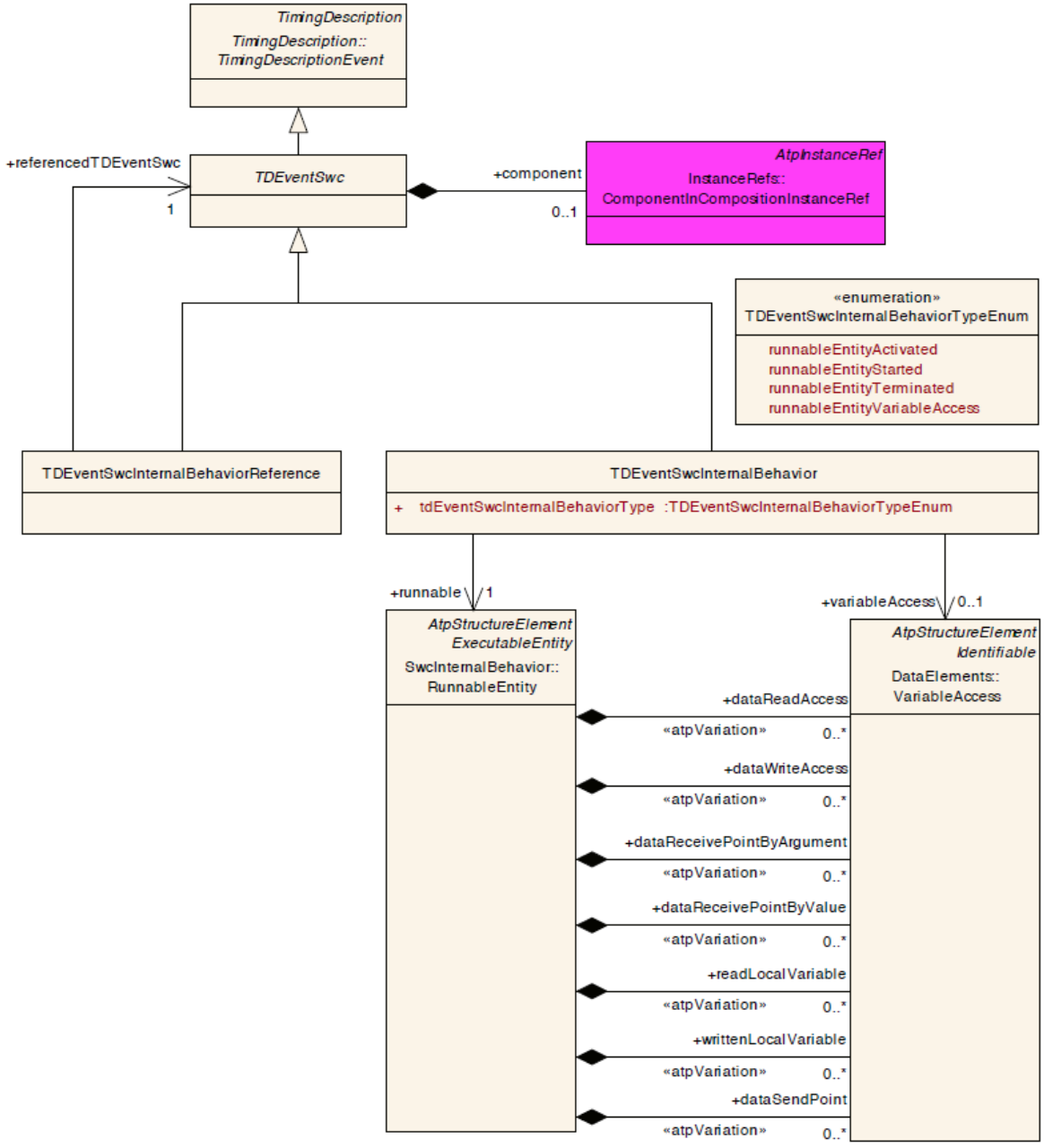

Figure 2.49: Timed events applicable to component behaviors

```
ExtUnaryFuncName  :  'TIMEX_value' |
                     'TIMEX_occurs' |
                     'TIMEX_hasOccurred' |
                     'TIMEX_timeSinceLastOccurrence' |
                     'TIMEX_angleSinceLastOccurrence'
                     ;
```

Figure 2.50: Timing descriptions for the definition of event arrivals

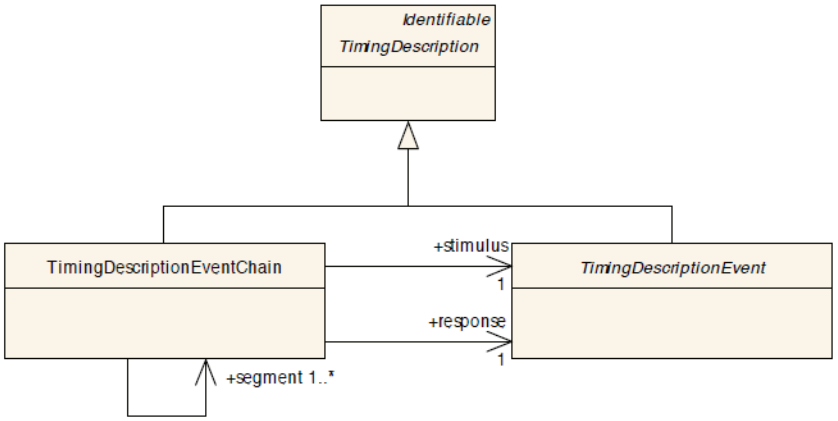

Figure 2.51: Timing Descriptions for event chains

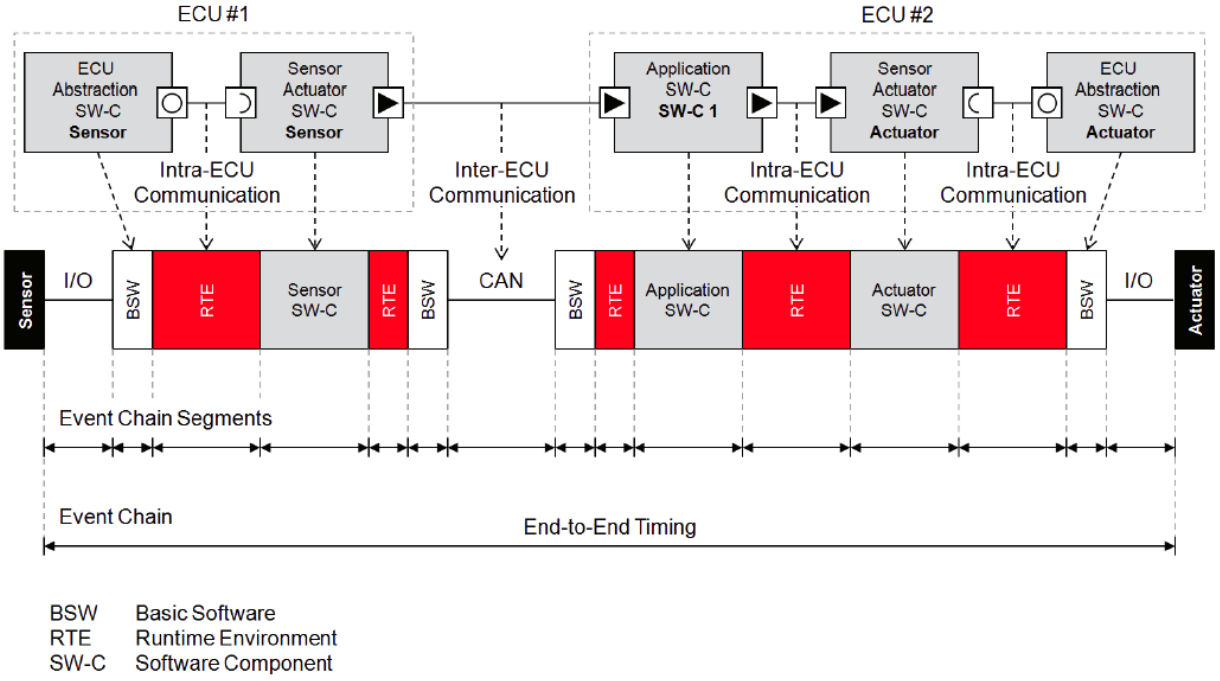

Figure 2.52: An example of an end-to-end chain combining execution of computations and transmission of messages.

Constraint	*Imposed on*	*Use Case*
Event Triggering	`TimingDescriptionEvent`	Specification of an activation Model
Latency Timing	`TimingDescriptionEventChain`	End-to-End path latency (in reaction or max age semantics)
Age	`TimingDescriptionEvent`	Restriction
Synchronization Timing	`TimingDescriptionEventChain`	Restrictions for forks and joins of event chains
Synchronization Timing	`TimingDescriptionEvent`	Restriction
Offset Timing	`TimingDescriptionEvent`	Restriction
Execution Order	`ExecutableEntity`	Restriction
Execution Time	`ExecutableEntity`	Restriction

Figure 2.53: The AUTOSAR entities that are provided for the expression of timing constraints

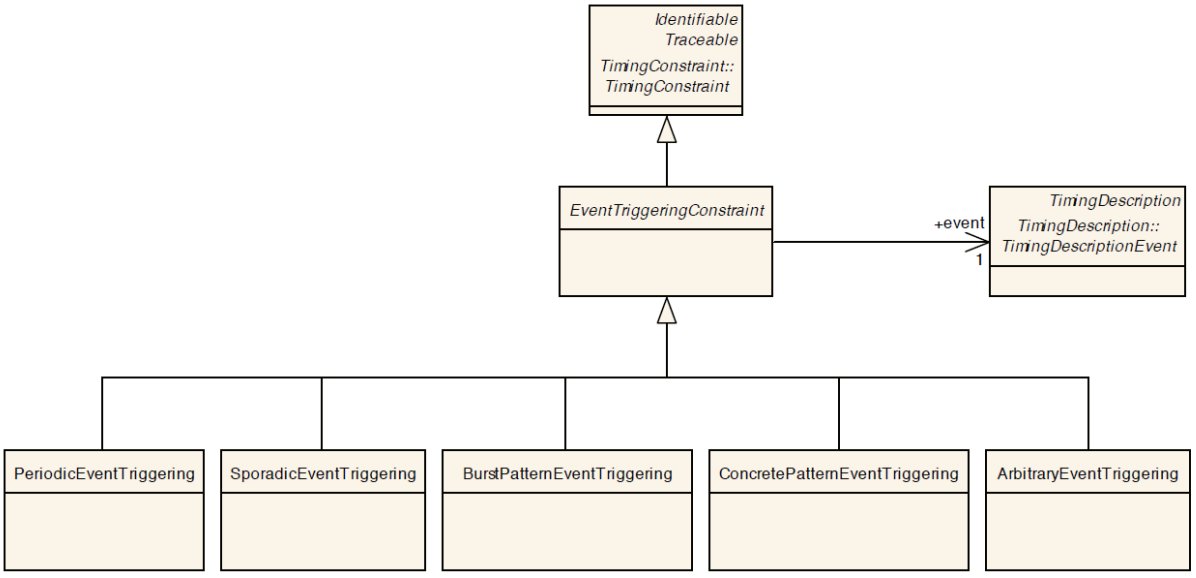

Figure 2.54: Event arrival constraints

constraints that apply to latencies (or deadline specifications).
LatencyTimingConstraint (shown in Figure 2.55) specify latency constraints. The element LatencyTimingConstraint is used to specify the amount of time that elapses between the occurrence of any two timing description events. For example, this can be the time it takes for a packet of data on a bus or network to get from one designated point to another, or the time it takes for a task to be executed on a processor. In the timing specification a LatencyTimingConstraint is associated with one TimingDescriptionEventChain, and specifies the minimum and/or maximum time duration between the occurrence of the stimulus and the occurrence of the corresponding response of that chain.
Figure 2.56 shows the constraints that can be used to enforce an order of execution between executable entities.
The element ExecutionTimeConstraint specifies an execution time constraints in terms of the minimum and maximum execution time assumed for executable entities.
An ExecutionTimeConstraint references the ExecutableEntity for which the execution time shall be constrained. The ComponentInCompositionInstanceRef referenced by component defines the component instance, which contains the RunnableEntity. Figure 2.57 shows the main entities in the definition of execution time constraints.
The table in Figure 2.58 provides the detailed list of the attributes that allow for the specification of an execution time constraint.

AUTOSAR timing analysis

In the development of E/E architectures, many functions are time critical due to safety requirements, and some other functions have timing constraints for certain performance guarantee. The AUTOSAR specification document for Timing Analysis [25] describes the timing properties and different methods for timing analysis, the recommended approach to specify timing requirements and how to conduct the timing analysis in various scenarios.

Timing properties and methods for timing analysis

Two main timing properties referred in the AUTOSAR standard are execution/transmission times and response times.
The execution time represents the duration taken by a schedulable entity (e.g. function or runnable) to complete its execution on an ECU without interference from other schedulable entities. Similarly,

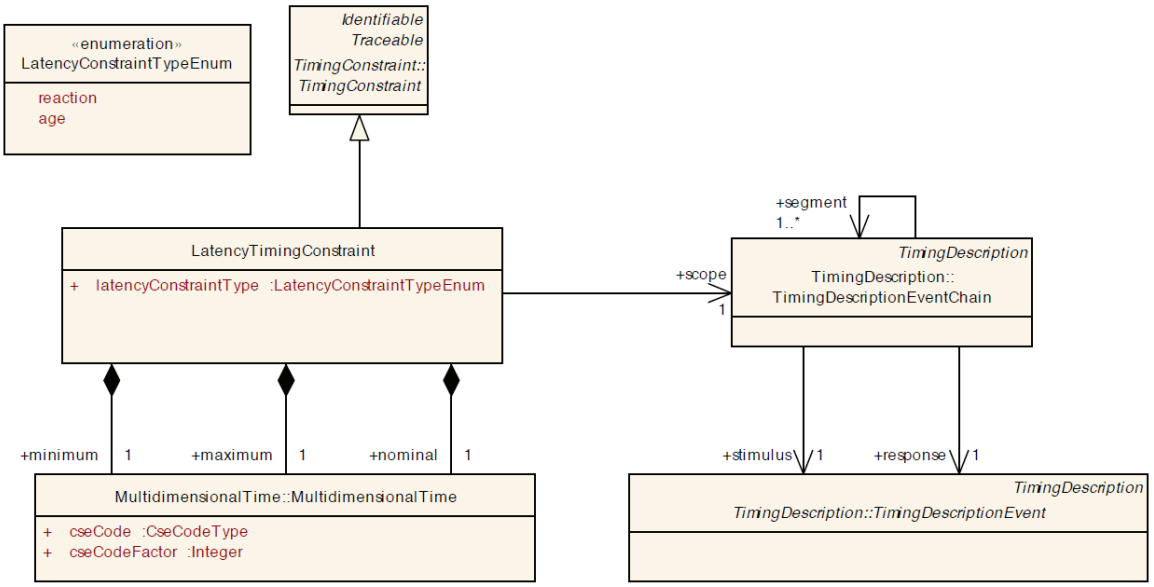

Figure 2.55: Latency constraints

transmission time applies to the context where a signal/message/frame that is transmitted through a network. The execution time is a quantitative property and can be characterized by its worst-case, best-case or average-case representation in AUTOSAR.
Different from the execution time, the response time represents the time a schedulable entity takes to complete its execution/transmission when there are other schedulable entities on the ECU /network. Still, its value can be denoted by the best-case, worst-case and average-case. When a sequence of schedulable entities are considered, the end-to-end response time corresponds the elapsed time from the activation of the first schedulable entity till the time that the last schedulable entity terminates its execution.
For timing analysis, the methods can be roughly classified into three main approaches: analytical calculation (analysis), simulation and measurement. The analysis and simulation approach can be also regarded as model-based. The model-based/measurement-based criterion is closely related to the stage the method can carry out.
Timing properties obtained from different methods should be consistent. For example, the worst-case execution time calculated from a perfect static code analysis tool should never be exceeded by the measured value from the system.

Timing Requirements

The timing requirement is one key factor for the development and integration of automotive E/E systems, and it must be identifiable and traceable from a requirement specification via a supplier's performance specification to a test and integration documentation. Timing requirements are introduced at the very beginning of the development cycle in the form of textual descriptions. The AUTOSAR TIMEX [23] extends the AUTOSAR System Template and defines the standardized format for the exchange of a system description within the development process.
Within the automotive distributed system, the timing analysis can be viewed as a tool to assure the desired temporal behavior during the mapping of a function network to a component network. The timing requirement is decomposed hierarchically. In a first step, the overall timing budget can be split into component-internal and networking parts, where a component usually refers to an ECU. As soon as the whole network communication is available, the worst-case timing demand can be quantified.
The Generic Methodology Pattern (GMP) [119] provides guidelines for timing requirement decomposition, and is applicable to each AUTOSAR timing view. GMP basically performs the following

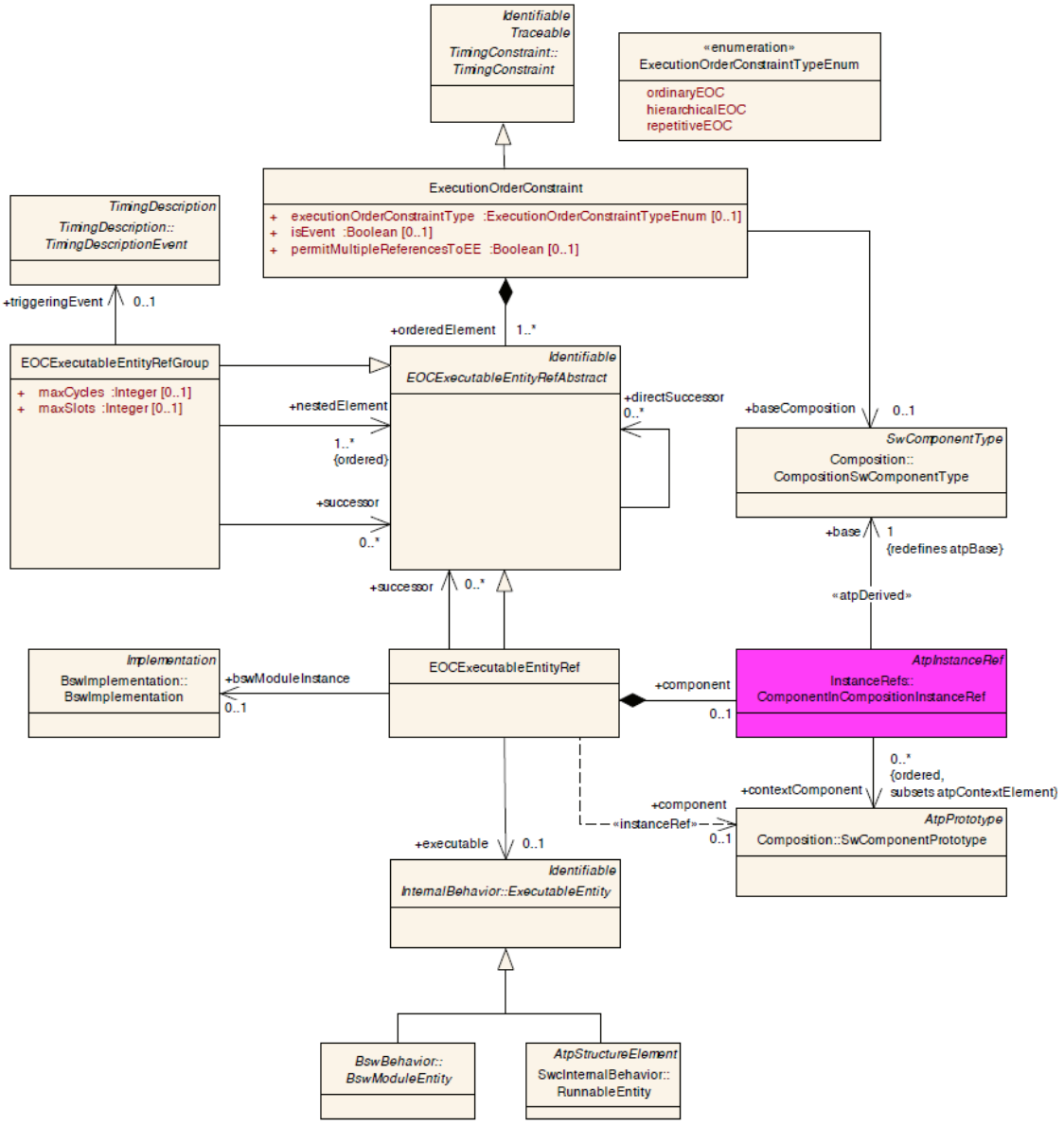

Figure 2.56: AUTOSAR modeling for the enforcement of an order of execution

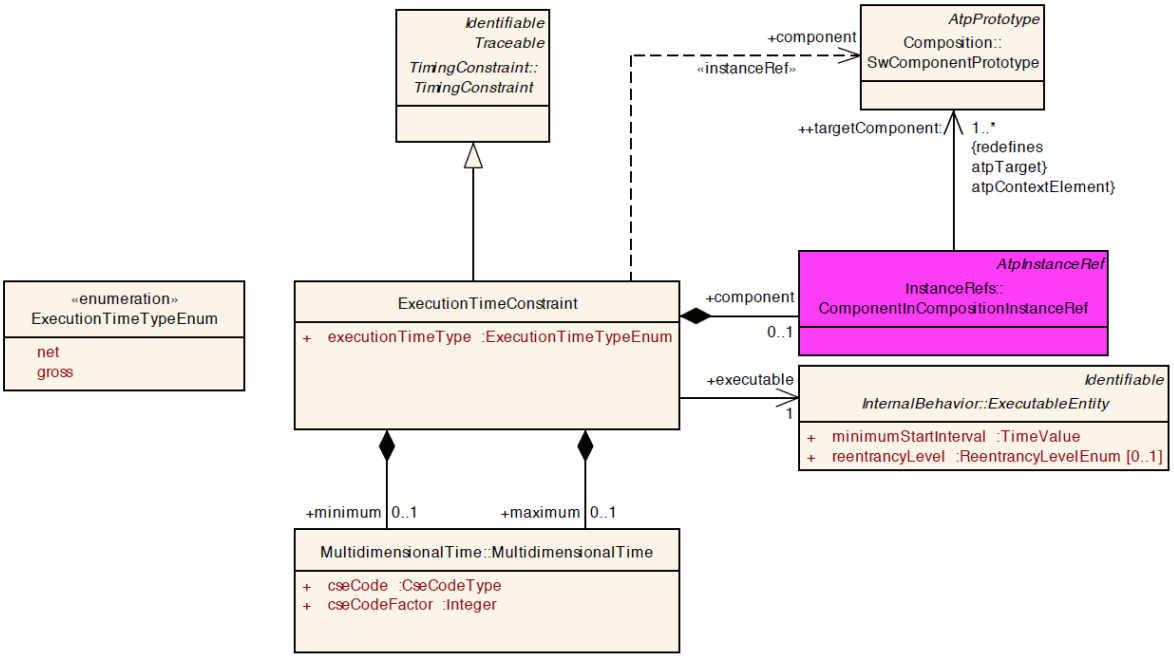

Figure 2.57: Execution time constraints

Class	**ExecutionTimeConstraint**			
Package	M2::AUTOSARTemplates::CommonStructure::Timing::TimingConstraint::Execution TimeConstraint			
Note	An ExecutionTimeConstraint is used to specify the execution time of the referenced ExecutableEntity in the referenced component. A minimum and maximum execution time can be defined. Two types of execution time semantics can be used. The desired semantics can be set by the attribute executionTimeType: The "net" execution time is the time used to execute the ExecutableEntity without interruption and without external calls. The "gross" execution time is the time used to execute the ExecutableEntity without interruption including external calls to other entities. The time to execute the ExecutableEntity including interruptions by other entities and including external calls is commonly called "response time". The TimingExtensions provide the concept of event chains and latency constraints for that purpose. An event chain from the start of the entity to the termination of the entity with according latency constraint represents a response time constraint for that executable entity.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,TimingConstraint,Traceable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
component	SwComponentPrototype	0..1	iref	The component that containts the referenced ExecutableEntity for the ExecutionTimeConstraint. If the entity is in a basic software module no component must be provided.
executable	ExecutableEntity	1	ref	The referenced ExecutableEntity for the ExecutionTimeConstraint.
executionTimeType	ExecutionTimeTypeEnum	1	attr	
maximum	MultidimensionalTime	0..1	aggr	The maximum execution time.
minimum	MultidimensionalTime	0..1	aggr	The minimum execution time.

Figure 2.58: Attributes of an execution time constraint

steps.

- Create a solution that describes the architecture without any timing information.
- Attach timing requirements to the solution. For instance, a timing requirement in the AUTOSAR SwcTiming view is a timing requirement that can be modeled by a timing constraint attached to events or event chains.
- Create, analyze and verify the timing model.
- Describe timing properties and timing requirements for the next level (timing view).

Several approaches based on Architecture Description Languages (ADLs) can be used to narrow the gap between the requirement specification in natural language and the implementation phase modeled in AUTOSAR. There are UML-based ADLs like SysML (UML specialization for System Modeling) and MARTE (UML specialization for Modeling and Analysis of Real-Time and Embedded systems). More domain specific approaches include AADL and EAST-ADL. EAST-ADL2 and its timing extension TADL2 allow the functional specification with precise timing models. Moreover, TADL2 and AUTOSAR TIMEX share the same base concepts. More details on TADL2 have been shown in the previous section.

Scenarios for Timing Analysis

The timing analysis in the automotive distributed system can be performed at the level of a single ECU or the network level. The AUTOSAR specification [25] introduces a series of scenarios, which are also called use-cases, for guiding the timing analysis.
On the ECU level, the scheduling of tasks and interrupts together with the execution times of various code fragments define the timing behavior of the software for a ECU. Scenarios that can be encountered for the timing analysis upon an ECU are briefly introduced in below.

- Create timing model of the entire ECU. In this case, all relevant timing information for an ECU is collected and the timing model of the entire ECU is created.
- Collect timing information of a SW-C. All relevant timing information of an selected SW-C is collected.
- Select an ECU supplier.
- Validate timing after SW-C integration. This happens when in the already existing ECU system, one SW-C is replaced by a new one. From timing analysis point of view, the new version must be ensured to satisfy the given timing constraints.
- Validate the timing of a defined system. The timing must be validated to ensure the schedulability of a system and that all given timing constraints must be satisfied. The trigger event for this scenario can be the change of a timing constraint of the change of the RTS/OS configuration.
- Debug timing. When there is unexpected or inconsistent behavior that can result a timing problem in the system, its cause must be tracked down, understood and isolated.
- Optimize timing for an ECU. While there is presence of timing violation, resource bottlenecks or the need to add further functionality into an already heavily loaded system, a better solution that fulfils all timing and resource requirements is required.
- Optimizing scheduling. The main idea is to find a modified schedule configuration that fulfills a certain optimization goal regarding scheduling.

- Optimizing code. The software code and the deployment of code have the important impact on timing, and different activities related to this can be performed. It should be aware that such a timing optimization can also affect other aspects of the system such as memory and re-usability, thus, conflicts with safety and security concerns.

- Verify timing models and compare timing properties. The model based approaches are used in the early design phase, and when the real system becomes available, timing properties like execution times and response times gathered in the model based methods have to be compared with measured time values.

  On the network level, heterogeneous network types are in use, and the timing analysis focuses on the network communication. The parameter configurations (e.g. size of signals, transformation pattern) and the selected protocols (e.g. CAN, FlexRay) for the communication define the timing behavior on each individual communication network. Scenarios for timing analysis of a network are listed in below.

- Integration of a distributed function. Considering an existing E/E automotive architecture consisting of several ECUs via several communication networks, the communication demand of a new functionality should be verified that the legacy and additional communication both fulfill the performance and timing constraints.

- Design of the new developed network. This concerns the design and feasible integration of a domain specific network into the existing automotive platform architecture.

- Remapping an existing function. This is the validation for the communication on a network after changing an existing function from a ECU to another one within in the network.

Tool Support and Best Practices

Based on the timing extensions [23] the AUTOSAR Timing Analysis document [25] describes various use cases for ECU- and network-level timing analysis. These can be seen as suggestions for best practices for timing analysis.
On paper, [23] and [25] solve the issue of modeling timing, specifying timing constraints and analyzing the timing behavior (and thus checking the constraints). However, this methodology is currently only used sparsely in practice, and there is no end-to-end tool support.
Currently, modeling (or at least export) of system descriptions in AUTOSAR (e.g. ECUs, SW-Cs, Tasks, Runnables, Buses, Frames, ...) is sometimes available. However, these models typically lack most timing information (e.g. task execution times) and also no constraints are specified. For timing analysis, this missing info is currently supplied from other sources, e.g.:

- Timing information (e.g. task execution times) is obtained from estimates, budgets, or measurements and imported in a timing analysis tool in a custom format (e.g. table-based).

- Constraint information is provided manually and imported in a timing analysis tool in a custom format (e.g. table-based).

- Some constraint information can be derived from other design files, such as a runnable order constraint, which can be derived from a Matlab/Simulink model (in certain cases).

To summarize, the formal and consequent use of AUTOSAR for specification of timing and timing constraints is still evolving. This is not due to a lack of expressiveness in AUTOSAR, but rather due to the immaturity of end-to-end timing processes in the automotive industry.

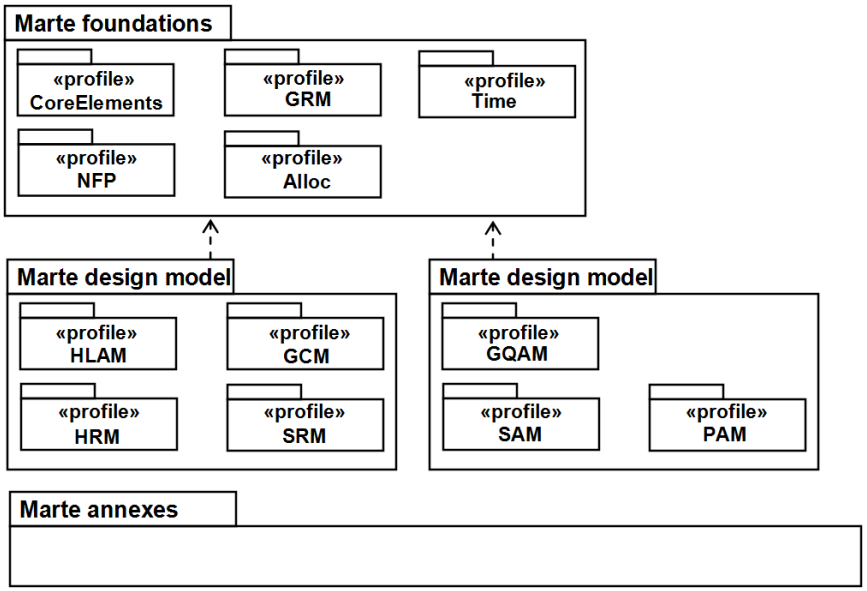

Figure 2.59: The packages in the MARTE profile

UML Modeling and MARTE

UML has been developed outside the context of embedded systems design and it is clear from previous sections how it does not cope with the modeling of resource allocation and sharing, nor with the minimum requirements for timing analysis. In fact, almost nothing exists in standard UML (the same could be said for SDL) for modeling (or analyzing) nonfunctional aspects, nor scheduling or placement of software components on hardware resources can be specified and analyzed. The MARTE profile for the Modeling and Analysis of Real-Time Embedded Systems for UML [3], enhances the standard language by defining timed models of systems, including time assumptions on the environment and platform dependent aspects like resource availability and scheduling. Such model extensions should allow formal or simulation-based validation of the timing behavior of the software.
The OMG Real-Time embedded systems MARTE profile, aims at substituting a number of proposals for time-related extensions that appeared in recent years (such as the OMG SPT profile [**?**]). In order to better support the mapping of active objects into concurrent threads many research and commercial systems introduced additional non-standard diagrams. An UML profile is a collection of language extensions or semantics restrictions of generic UML concepts. These extensions are called stereotypes, and indicated with their names in between guillemets, as in ÂńTimedEventÂż. The profile concept is itself a stereotype of the standard UML Package. The MARTE profile defines a comprehensive conceptual framework that uses stereotypes built on the UML meta-model providing a much broader scope than any other real-time extension and applies to all diagrams. MARTE consists mostly of a notation framework or vocabulary, with the purpose of providing the necessary concepts for schedulability and performance analysis of (timed) behavioral diagrams or scenarios. However, MARTE inherits from UML the deficiencies related to its incomplete semantics and, at least as of today (2015), it lacks a sufficiently established practice. The current version of the profile is based on extensions (stereotyped model elements, tagged values and constraints) belonging to four main framework packages, further divided into subpackages (as in Figure âĂŐ2.59).
Of the four frameworks, the Foundations package contains the fundamental definitions for modeling time, in the Time subpackage the definitions for time, clocks and timed events. The GRM package contains the generic resource specification and usage patterns, and the NFP package the stereotypes for non-functional properties. The Design model contains the extensions for modeling concurrency and resources in the GCM, SRM and HRM packages. The Analysis Models package contains specialized concepts for modeling schedulability (SAM) and performance (PAM) analysis. In MARTE, the time

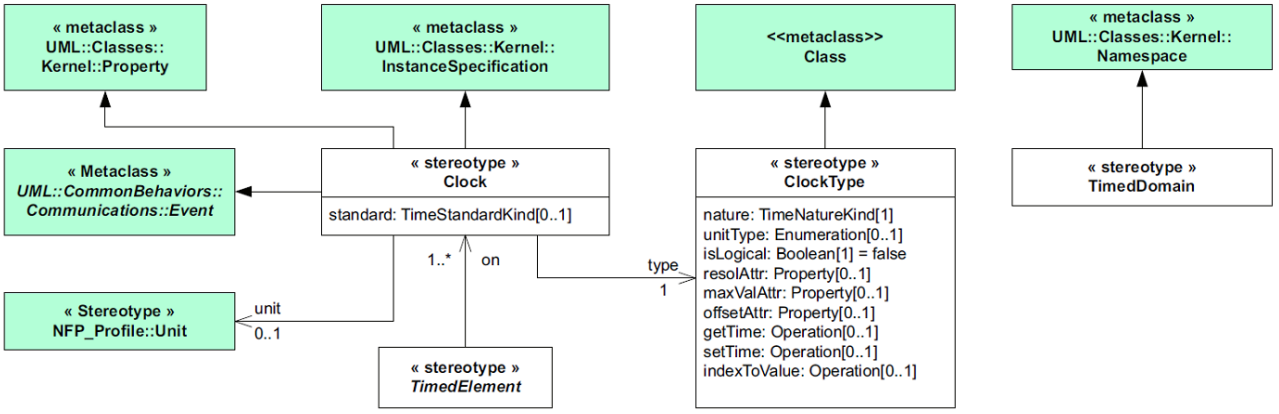

Figure 2.60: The definition of clocks in the TRM of the MARTE profile

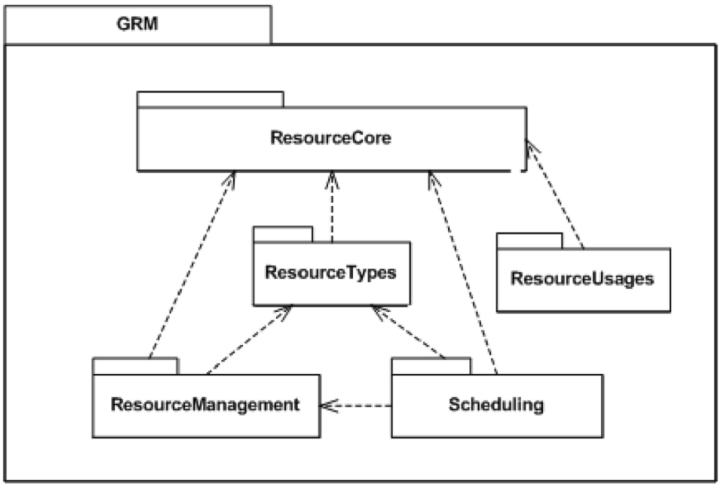

Figure 2.61: The main packages for the definition of resources

model provides for both continuous and discrete time models, as well as global and local clock, including drift and offset specifications. The profile allows referencing to time instances (associated with events), of Time type and to the time interval between any two instances of time of Duration type in attributes or constraints specifications inside any UML diagram. The time package (some of its stereotypes are shown in Figure 2.60) not only contains definitions for a formal model of time, but also stereotyped definitions for the two basic mechanisms of timer and clock. Timers can be periodic, they can be set or reset, paused or restarted and, when expired, they send timeout signals. Clocks are specialized periodic timers capable of generating Tick events.
Time values are typically associated with Events, defined in UML as a âĂIJspecification of a type of observable occurrenceâĂİ (change of state). A pairing of an event with the associated time instance (time tag) is defined in the MARTE profile as a TimedEvent. The GRM resource model package defines the main resource types as well as generic resource managers and schedulers (its main packages with their relationships in Figure 2.61 and a detail of some stereotypes in Figure 2.62.)
In the MARTE Profile the mapping between the logical entities and the physical architecture supporting their execution is a form of realization layering (synonymous of deployment). The semantics of the mapping provides a further distinction between the Âńdeploys Âż mapping, indicating that instances of the supplier are located on the client and the ÂńrequiresÂż mapping, which is a specialization indicating that the client provides a minimum deployment environment as required by the supplier.
The GRM profile package is used for the definition of the stereotypes for software and hardware resources. The software resource modeling is much more detailed and comprehensive than the hardware

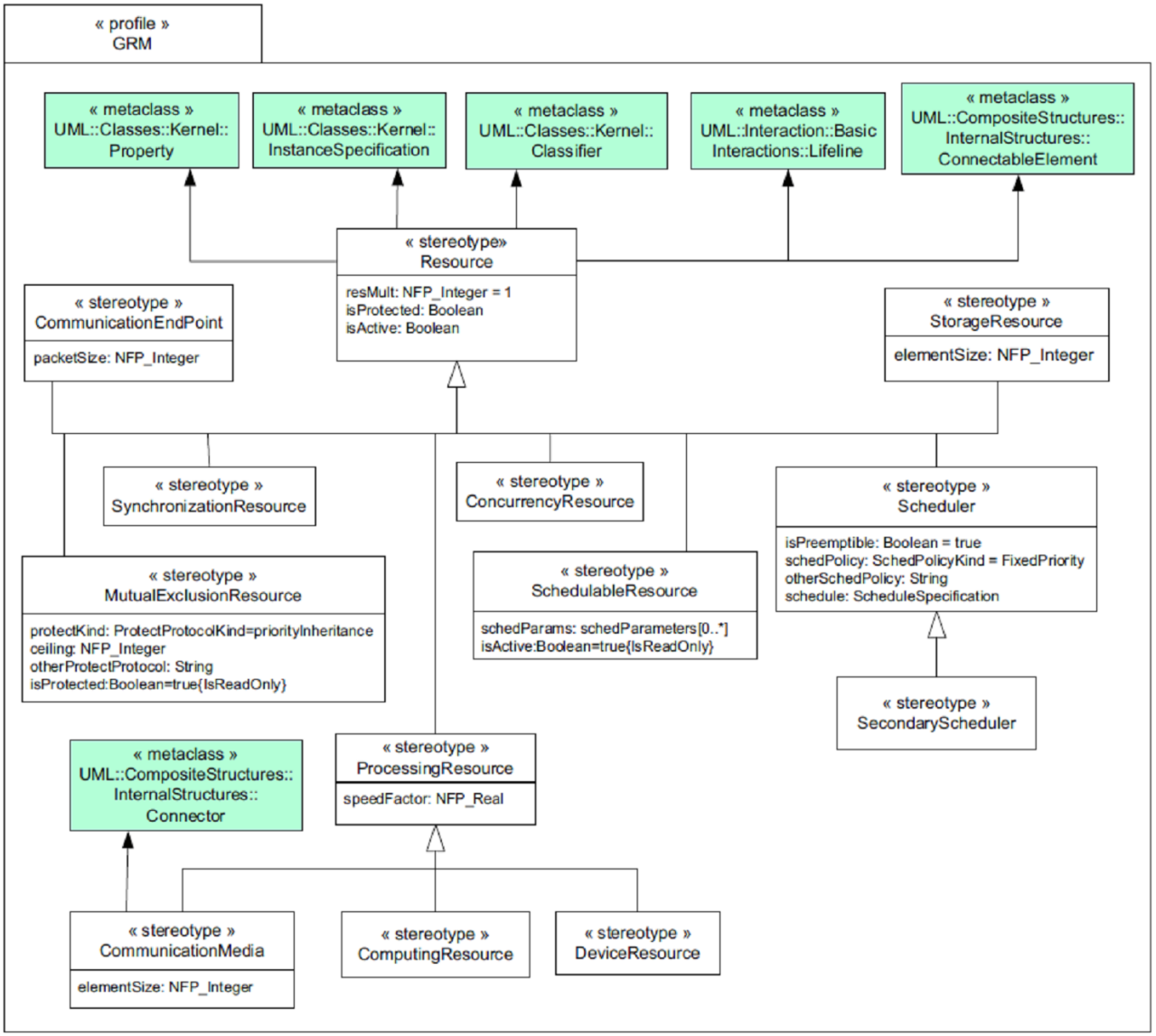

Figure 2.62: The definition of a resource in MARTE

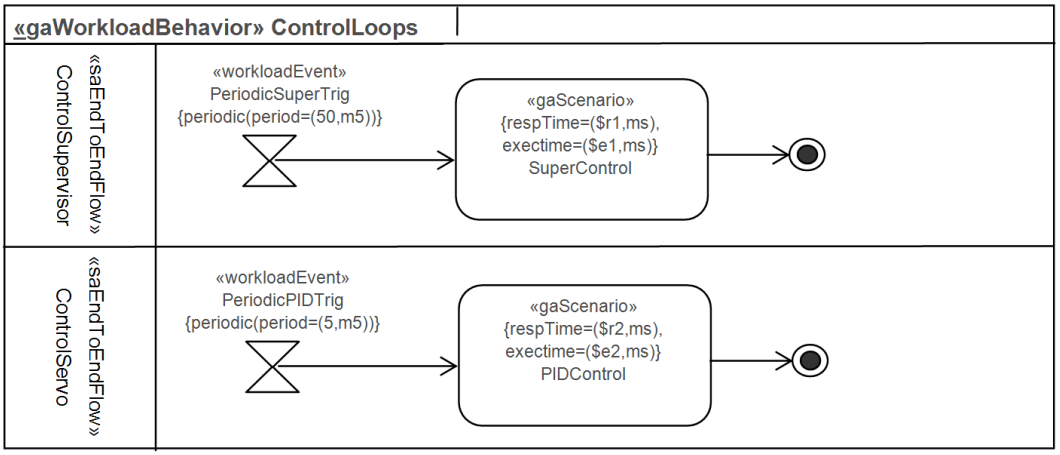

Figure 2.63: A schedulability analysis scenario in MARTE

modeling package, which only contains stereotypes for the basic concepts.
The Schedulability analysis model is based on stereotyped scenarios. Each Scheduling situation is in practice a sequence, collaboration, or activity diagram, where one or more trigger events result in actions to be scheduled within the deadline associated with the trigger. Rate Monotonic Analysis (RMA) is the method of choice for analyzing simple models with a restricted semantics, conforming to the so-called task-centric design paradigm. This requires updating the current definition of UML actions, in order to allow for preemption (which is a necessary prerequisite of RMA). In this task-centric approach, the behavior of each active object or task consists of a combination of reading input signals, performing computation and producing output signals. Each active object can request the execution of actions of other passive objects in a synchronous or asynchronous fashion.
Figure âĂŐ2.63 shows an example with three activities that are logically concurrent, activated periodically, and handle a single event. The MARTE stereotypes provide for the specification of the execution times and deadlines, and, as long as the active objects cooperate only by means of pure asynchronous messages, possibly implemented by means of memory mailboxes, a kind of protected (shared resource) object, simple schedulability analysis formulas can be used.

2.2.3 Security

AUTOSAR issued a specification regarding mechanisms for secure onboard communication [22]. In AUTOSAR, the standardization efforts have been dedicated to the definition of mechanisms that provide for authentication at the level of the network, that is under the assumptions that

- the end points of the communication are internal Electronic Control Units (ECUs) or complex sensors or actuators (considered equal to ECUs),
- end points are assumed to be trustworthy; they are trusted to behave according to their specifications and are not manipulated,
- the generation, distribution, storage and usage of keys is assumed to be secure and protected against manipulation, eavesdropping, or any other leakage to the attacker, and
- the attacker is assumed to have full access to or even full control over the communication bus.

The AUTOSAR specification is focused on the domain of network communication, to protect the information exchanged over the communication buses.
The main objectives of the AUTOSAR mechanisms are to provide information integrity and authenticity. Confidentiality is not considered at the same level of importance. Furthermore, denial-of-service attacks are unfortunately very hard to prevent (this applies not only to automotive use cases, but

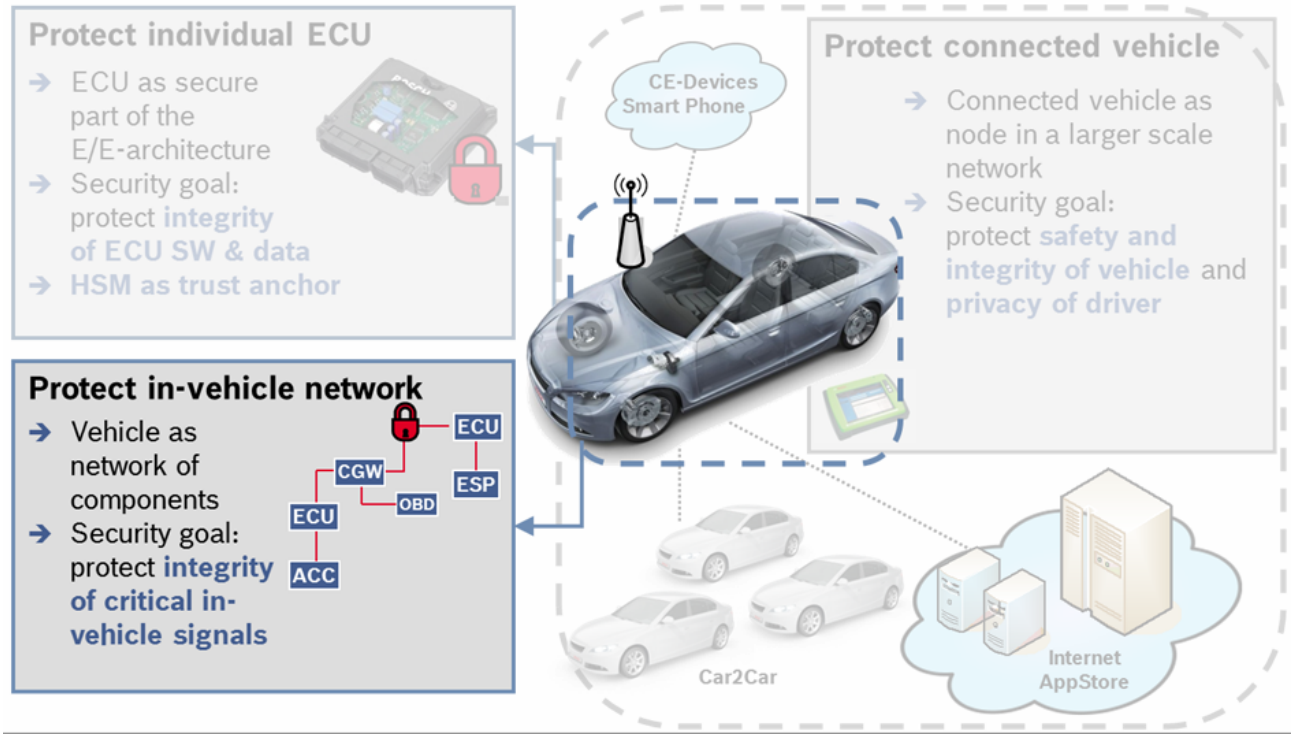

Figure 2.64: The three domains for security in automotive systems (from [22]).

the general case). One mitigation is the usage of redundancy (i.e., additional ECUs). In the automotive domain, the problem is usually neglected, because (i) additional ECUs lead to higher costs and (ii) usually these attacks have no benefit for the attacker. They are only considered if the (passenger) safety is affected.

Authenticity verification in AUTOSAR includes means to check data freshness so that the receiver of a message has the proof that the transmitted message is not a message that has been recorded and replayed by an attacker.

The specification is based on the assumption that mainly symmetric authentication approaches with message authentication codes (MACs) are used. The choice is based on the observation that symmetric authentication requires much smaller keys than asymmetric approaches and can be implemented compactly and efficiently in software and in hardware. However, the specification is also abstracted to possibly allow for asymmetric authentication approaches. One advantage of asymmetric approaches is that the key distribution is easier although they are much more computationally demanding than symmetric ones.

From the architecture standpoint, AUTOSAR requires both the sending ECU and receiving ECU to implement three software modules: the SecOC, the CSM (Crypto Service Manager) and CAL modules (Figure 2.65). The SecOC module is integrated with the communication services and in particular with the module that is in charge of routing the PduR (Protocol data unit at the routing level) on the sender and receiver side. The SecOC modules on both sides interact with the PduR module by implementing a given API.

The Crypto Service Manager, defined in [20] provides synchronous and asynchronous services to basic cryptographic functionalities for use by all software modules, application or basic services. The CSM provides a standardized interface to access these functions. The CSM module is meant to be used by application components that require encryption for their data.

Depending on the type of data or messages to be protected, different levels of protection can be required by the application (depending on the criticality of the information and the hazards in case of compromised data). The architecture solution therefore allows for configurability to adapt to different

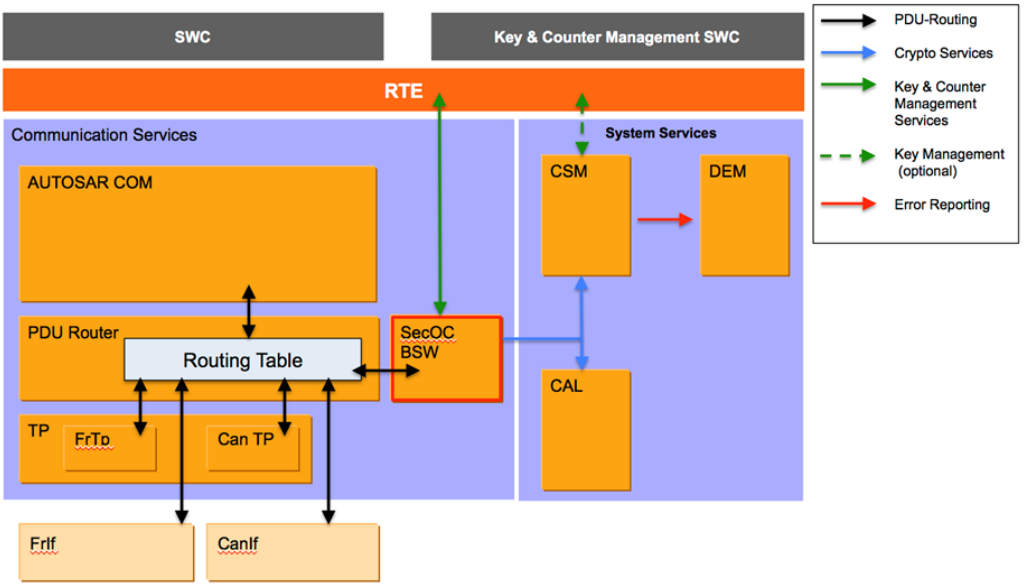

Figure 2.65: The SecOC component module in AUTOSAR (from [22]).

security needs.

SecOC

In this AUTOSAR architecture, the PDU Router (PDU stands for Protocol Data Unit) is responsible to route incoming and outgoing I-PDUs (I-PDU stands for Interaction Layer Protocol Data Unit; an I-PDU carries signals [151]). In the case of PDU with security requirements, the security levels on reception and transmission should be obtained by forwarding the request to the SecOC module. The SecOC module shall then add or process the security relevant information and shall propagate the results in form of an I-PDU back to the PduR. The SecOC module shall support all kind of communication paradigms and principles that are supported by the router, including multicast communications, transport protocols and the gateway functionality. To guarantee message freshness, the SecOC modules on the sending and receiving side maintains freshness values (e.g. a freshness counter or timestamp) for each Secured I-PDU of different type, i.e. for each secured communication link.
On the sender side, the SecOC module creates a secured I-PDU by adding authentication information to the outgoing authentic I-PDU. The authentication information comprises of an authenticator (e.g. Message Authentication Code or MAC) and an optional freshness value. Regardless of the fact that the freshness value is included in the Secure I-PDU payload, the freshness value is considered during generation of the authenticator. When using a freshness counter instead of a timestamp, the freshness counter is incremented prior to providing the authentication information to the receiver side.

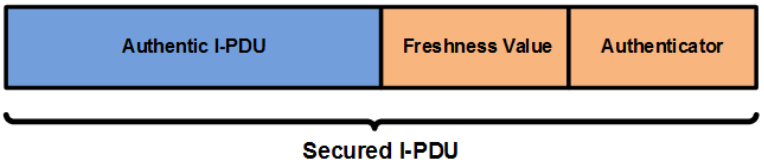

Figure 2.66: The additional fields in a secure I-PDU (from [22]).

On the receiver side, the SecOC module checks the freshness and authenticity of the authentic I-PDU by verifying the authentication information that has been appended by the sending side SecOC module. To verify the authenticity and freshness of an authentic I-PDU, the secured I-PDU provided to the receiving side SecOC should be the same secured I-PDU provided by the sending side SecOC

and the receiving side SecOC should have knowledge of the freshness value used by the sending side SecOC during creation of the authenticator.

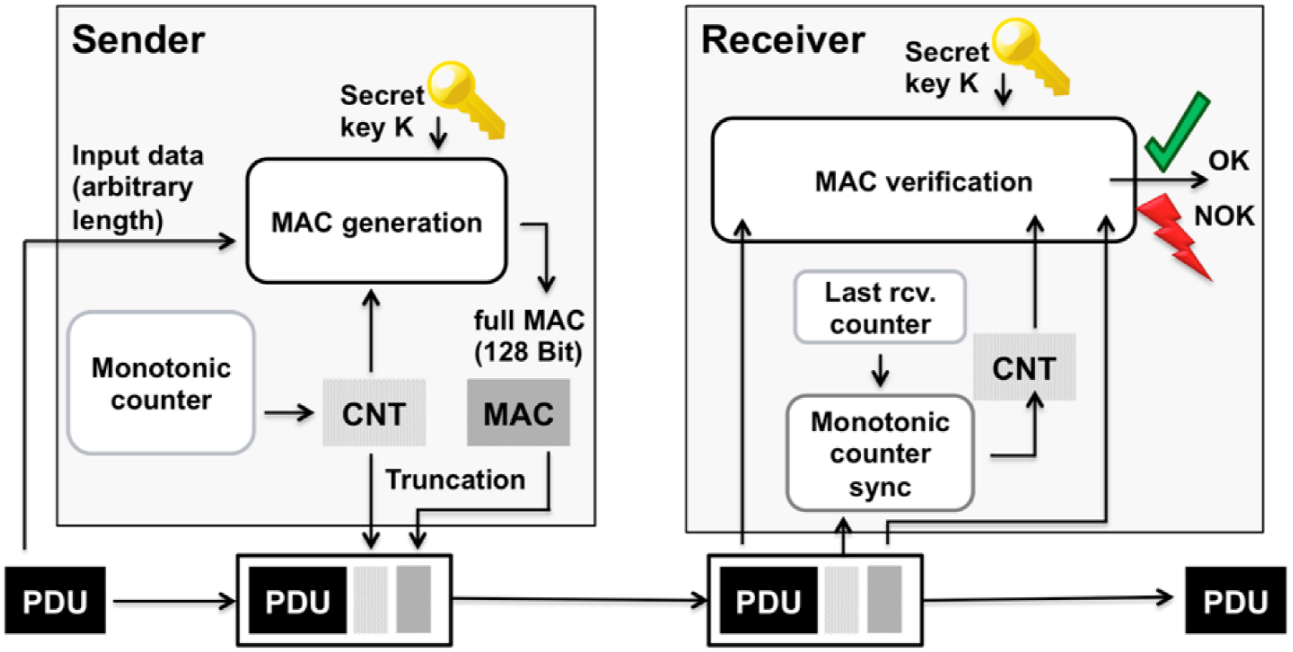

Figure 2.67: The security flow (from [22]).

The length of the authentic I-PDU, the freshness value and the authenticator within a secured I-PDU may vary from one secured I-PDU to another.
The authenticator (e.g. MAC) refers to a unique authentication data string generated using a key, and computed on the data identifier of the secured I-PDU (parameter SecOCDataId), the I-PDU data, and the complete freshness value.
Depending on the authentication algorithm (parameter SecOCAuthAlgorithm) used to generate the authenticator, it may be possible to truncate the resulting authenticator (e.g. in case of a MAC) generated by the authentication algorithm. Truncation may be desired when the message payload is limited in length and does not have sufficient space to include the full authenticator.
The authenticator length contained in a secured I-PDU (parameter SecOCAuthInfoTxLength) shall be specific to a uniquely identifiable Secured I-PDU to provide flexibility across the system (i.e. two independent unique Secured I-PDUs may have different authenticator lengths included in the payload of the secure I-PDU). This allows for fine grain configuration of the MAC truncation length for each Secured I-PDU.
If truncation is possible, the authenticator should only be truncated down to the most significant bits of the resulting authenticator generated by the authentication algorithm. Figure 2.68 shows the truncation of the authenticator and the freshness values respecting the parameter SecOCFreshnessValueTxLength and SecOCAuthInfoTxLength. AUTOSAR recommends the use of Message Authentication Codes (MACs) as a basis for authentication (e.g. a CMAC [63] based on AES [84] with an adequate key length) for resource-constrained systems with a static (predetermined) set of participants. In addition, the standard proposes to always use a key length of at least 128 bit with the exception of those cases in which a MAC truncation is required. Truncation increases the risk of false positives (i.e., collisions). From the perspective of an attacker, the number of trials depends on the throughput of the bus (e.g., CAN) and the hardware (e.g., HSM hardware). For low speed buses and non-critical data, a truncation down to 32 bits is feasible. For guidance the standard refers to appendix A of [63] in which MAC sizes of 64 bit and above are considered sensible.
If the verification of the secured I-PDU fails and SecOCFreshnessValueSnycAttempts is configured to a value greater than 0, the SecOC module shall reevaluate the secured I-PDU using a different freshness

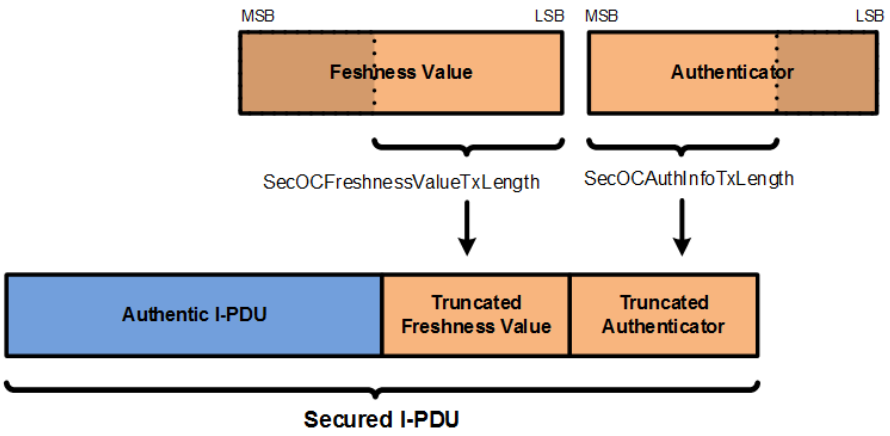

Figure 2.68: The additional fields in an PDU with truncation options (from [22]).

value before considering the received data as non-authentic (e.g. counter or time de-synchronization is suspected to be the reason of the failed authentication verification). The number of verification attempts using a different freshness value before considering the received data as non-authentic shall be limited by SecOCFreshnessValueSnycAttempts.
If a Secondary Freshness Value is configured, the freshness value previously described in this document is referred to as the Primary Freshness Value. If a Secondary Freshness Value is configured for a secured I-PDU and the authentication verification fails for that PDU using the counter value corresponding to the Primary Freshness Value, authentication verification shall be re-attempted using the value corresponding to the Secondary Freshness Value.
In the event the counter value corresponding to the Primary Freshness Value fails authentication verification and the counter value corresponding to Secondary Freshness Value results in successful authentication verification, OEM-specific software should utilize the SecOC_FreshnessValueRead and SecOC_FreshnessValueWrite interfaces to replace the counter value corresponding to the Primary Freshness Value with the counter value corresponding to Secondary Freshness Value.
Adaptation in case of asymmetric approach
In case of an asymmetric approach using digital signatures instead of MACs, some adaptations have to be made:

- Instead of a shared secret between sender and (all) receiver(s), a key pair consisting of public key and secret key is used. The secret (or private) key is used by the sender to generate the signature, the corresponding public keys is used by (all) receiver(s) to verify the signature. The private key must not be feasibly computable from the public key and it shall not be assessable by the receivers. The SecOCKeyUsageRestriction parameter is not needed for the asymmetric approach since the restriction is implicitly enforced by the usage of different keys.

- In order to verify a message, the receiver needs access to the complete signature/output of the signature generation algorithm. Therefore, a truncation of the signature as proposed in the MAC case is not possible. The parameter SecOCAuthInfoTxLength has to be set to the complete length of the signature.

- The signature verification uses a different algorithm than the signature generation. So instead of “rebuildingâĂIJ the MAC on receiver side and comparing it with the received (truncated) MAC” as given above, the receiver/verifier performs the verification algorithm using the Data-ToAuthenticator (including full counter) and the signature as inputs and getting a boolean value as output, determining whether the verification passed or failed.

Crypto Service Manager

The Crypto Service Manager, or CSM, is a module that provides a set of cryptographic services (with possible support from the HW) to application components and to other basic software modules, including the SecOC itself (as in Figure 2.69). The CSM module provides a set of standardized cryptography functionality, based on the availability of a software library ("Basic Crypto Routines" in the figure) or on the availability of a hardware module (such as the HSM or the "Crypto HW" label in the figure). Figure 2.69 shows the position of the CSM module and its relationship with the application layer and the microcontroller abstraction providing access to a possible Crypto HW module.

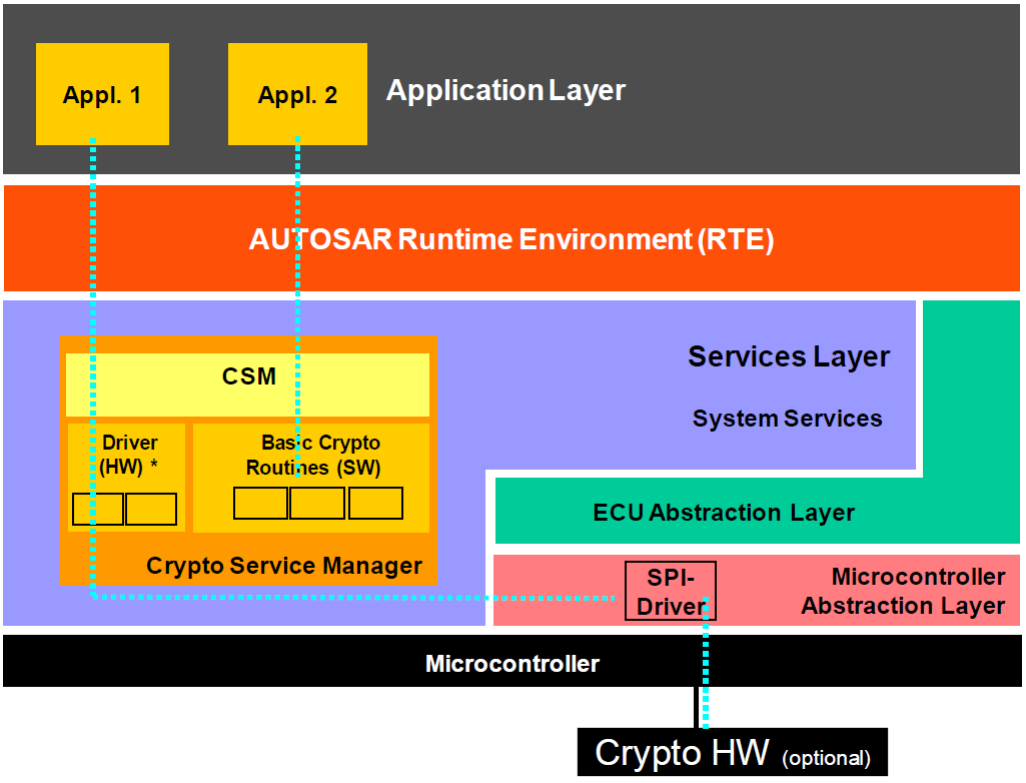

Figure 2.69: The architecture-level definition of the CSM module and its relationship with other services in AUTOSAR (image taken from [20]).

The hardware modules in the Crypto HW will typically include secure key storage capabilities. This means that keys are completely managed from within the HW module and not visible outside (so that they cannot be read or altered by malicious software). To enable this HW support, the programmer level API of the CSM will refer to opaque key types interpreted as key handles instead of raw key data. Furthermore, parameters that are used for key output are INOUT to allow the specification of a target handle. The AUTOSAR standard document defines quite a rich set of functionalities with a large set of API callable functions, organized according to a set of interfaces. The provided interfaces are:

- General interfaces
- Hash interface
- MAC interface
- Random interface
- Symmetrical block interface

- Symmetrical interface
- Asymmetrical interface
- Signature interface
- Compression/Decompression interface
- Checksum interface
- Key generation interface
- Key derivation interface
- Key exchange interface
- Symetrical key update interface
- Symmetrical key extract interface
- Symmetrical key wrapping interface
- Asymmetrical key update interface
- Asymmetrical key extract interfaces
- Asymmetric key wrapping interface

The document provided by the AUTOSAR is oriented to programmers and does not contain any reference to modeling features. However, it is worth noting that there are several issues with respect to the intended use of the CSM module and the AUTOSAR methodology or philosophy. This intended use is confirmed by examples of service invocation from the application components as described in the AUTOSAR document "AUTOSAR Software Component Template" [16].
In particular, AUTOSAR Services can be seen as a hybrid concept between basic software modules and software components. Software components that require AUTOSAR services use one of the standardized AUTOSAR interfaces to communicate with them. The dependency of a software component from an AUTOSAR service is modeled by adding required/provided ports (hereinafter referred to as "service ports") to the software component. The interface of a service port needs to be one of the standardized service interfaces (well defined in the AUTOSAR documentation) which is indicated by having its *isService* attribute set to *true*. The interface is related to *ServiceNeeds* via several levels of indirection.
Furthermore, the internal behavior of the software component shall contain a *SwcServiceDepencency*, which is used to add more information about the required service.
The meta-model of the *SwcServiceDepencency* is shown in Figure 2.70. For example,

- *SwcServiceDepencency* is used to associate ports to a given *ServiceNeeds* element;
- *ServiceNeeds* are used to provide detailed information on what the software-component expects from the AUTOSAR Services. For instance, *CryptoServiceNeeds* can be used to specify a maximum length for the key used in cryptographic services;
- *RoleBasedPortAssignment* is used to map a service port to a specific *ServiceNeeds*. The attribute *portPrototype* is used to refer a service port of the software component.

Many other classes derived from *ServiceNeeds* are not shown in the figure. See [16] for a complete list. So, if software component *swc* wants to use the MAC cryptoservice, then the designer has to add client ports to *swc*; assign to each port the required interface (*CsmMacGenerate* and *CsmMacVerify*

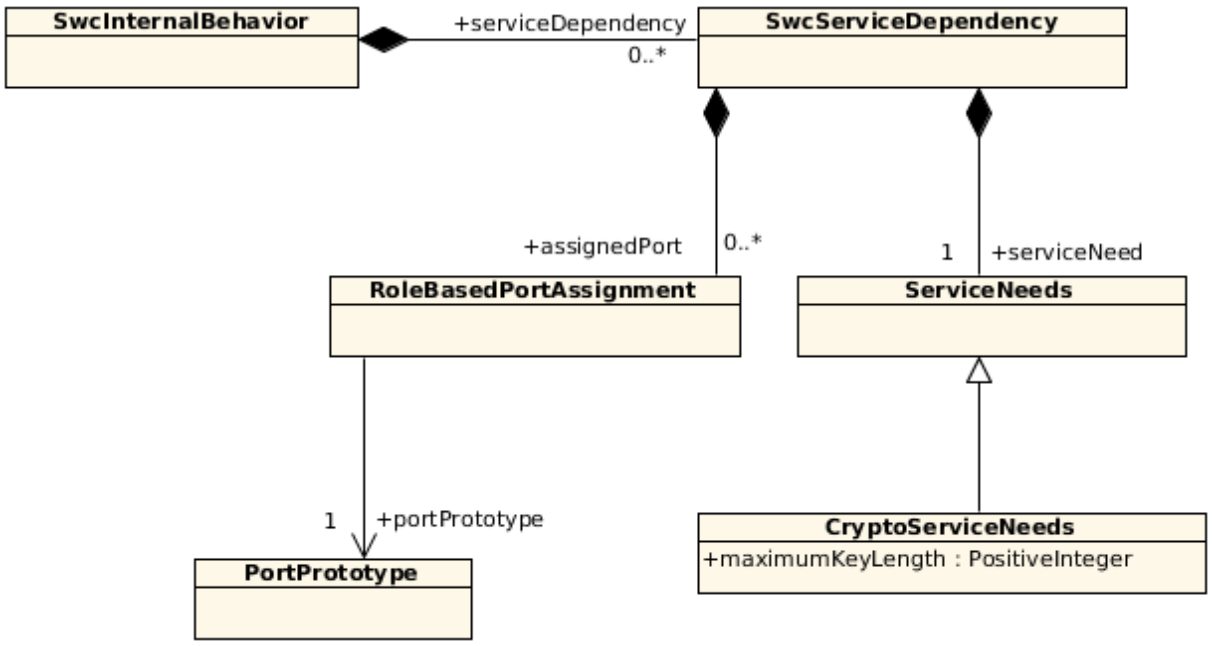

Figure 2.70: Service dependency meta-model from metamodels in [16].

[17]); set the *isService* attribute of the interfaces to true and add *SwcServiceDepencency* to the internal behaviour element of *swc*.
First and foremost, direct access from the application SW components to these services breaks one of the typical assumptions of AUTOSAR, that is, that all accesses to services are mediated by the RTE. Most of the AUTOSAR concepts (communication and scheduling, for example) assume that application components are provided with a specification of the required service (such as characteristics of the activation event, order of execution, mutual exclusion and communication needs for the components and their runnables), but the selection of the scheduling policies and the communication services that need to be used to satisfy these requirements is up to the RTE generator.
With the crypto services, this is not true. It is somewhat expected that application components encode directly the "how" rather than the "what", meaning that application components can freely use the crypto services and it is their responsibility to use them correctly.
This bring another issue. Sometimes, the best selection of the encryption method for communication over networks depends on the signal to message packing and the characteristics of the network itself. However, this information is not available to application component behaviors that are not even aware of the component placement in the physical system. This is why the CSM is also meant of be directly accessed by the SecOC (besides the application components), to allow for the computation of the MAC for the network messages.
In our modeling options, we will consider the availability of the CSM services, but we will provide a modeling style in which the application provides for the high level requirements that apply to the confidentiality of their communications.

AUTOSAR model

Figure 2.71 shows the recommended model for the functionality and attributes that characterize the SecOC module and the secure PDU.
The model contains the following definitions (from [22]):

SecuredIPdu: I-PDU that contains payload of an authentic I-PDU supplemented by additional authentication information (freshness counter and authenticator).

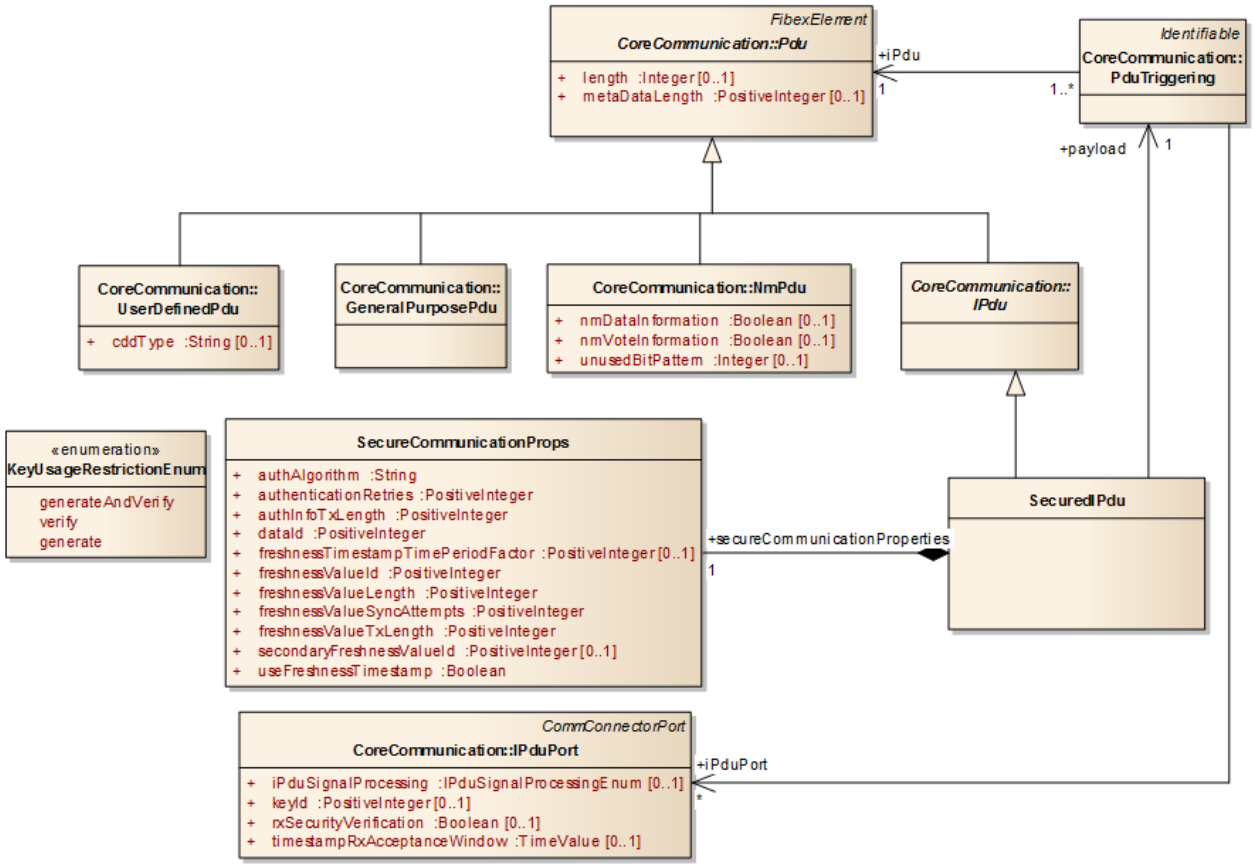

Figure 2.71: The security model in AUTOSAR.

payload: Reference to a PDU that will be protected against unauthorized manipulation and replay attacks.

SecureCommunicationProps: Properties used to configure SecuredPdus.

authAlgorithm This attribute defines the authentication algorithm used for MAC generation and verification.

authInfoTxLength This attribute defines the length in bits of the authentication code to be included in the payload of the

authenticated Pdu.

dataId This attribute defines a unique numerical identifier for the SecuredPdu.

freshnessValueId This attribute defines the Id of the freshness value. The freshness value might be a normal counter or a time value.

secondaryFreshnessValueId (0..1) This parameter defines the Id of the Secondary Freshness Value. The Secondary Freshness Value might be a normal counter or a time value.

freshnessValueLength This attribute defines the complete length in bits of the freshness value. As long as the key doesn't change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter.

freshnessValueSyncAttempts Loss of synchronization shall not lead to authentication fail when the number of authentication attempts remain in the defined acceptance range.

This attribute defines the number of authentication attempts to perform before flagging a received message as unauthentic. Each additional MAC verification retry shall use a

freshness counter value from the defined acceptance range (e.g. when using counter by incrementing the counter window; when using time by decreasing time stamp windows). The LSB of the counter/time that was not transmitted with truncated counter value in the message should be incremented/decremented in the defined range.

freshnessValueTxLength This parameter defines the length in bits of the freshness value to be included in the payload of the secured I-PDU. This length is specific to the least significant bits of the complete freshness counter. If the parameter is 0, no freshness value is included in the secured I-PDU.

freshnessTimestampTimePeriodFactor (0..1) This attribute defines a factor that specifies the time period for the freshness timestamp. It holds a multiplication factor that specifies the concrete meaning of a freshness timestamp increment by one on basis of microseconds.

useFreshnessTimestamp This attribute specifies whether the freshness value is generated through individual freshness counters or by a timestamp. The value is set to TRUE when timestamps are used.

authenticationAttempts This attribute defines the additional number of authentication attempts that are to be carried out when the generation of the authentication information failed for a given SecuredIPdu. If zero is set than only one authentication attempt is done.

If SecOCUseFreshnessTimestamp is set to TRUE,

- the SecOC module shall use a freshness timestamp to generate the freshness value,
- the parameter SecOCFreshnessTimestampTimePeriod shall be used to configure the resolution, and
- an acceptance window shall be defined for each receiver of a secured I-PDU using SecOCRxAcceptanceWindow.

The freshness value shall not roll over or overflow for the life of the key used to generate/verify corresponding authenticators.
The functions that are required to be implemented by the SecOC are listed in the standard document. Among those are:

```
Std_ReturnType SecOC_Transmit(PduIdType id, const PduInfoType* info)
```

A service to request authentication and transmission of an authentic I-PDU.

```
Std_ReturnType SecOC_AssociateKey(uint8 keyID, const SecOC_KeyType* keyPtr)
```

To associate a given key value to a given key id (see also parameter SecOCKeyID).

```
Std_ReturnType SecOC_FreshnessValueRead(uint16 freshnessValueID,  uint64* counterValue)
```

To read a specific freshness value value residing in the SecOC module.

```
Std_ReturnType SecOC_FreshnessValueWrite(uint16 freshnessValueID, uint64 counterValue)
```

To write a specific freshness value residing in the SecOC module.

Chapter 3

Guiding principles and Gap Analysis

This section exploits the results of the state of the art to define the methodology and the steps that have been used to identify the modeling features of interest for Safure.

3.1 Guiding principles

Based on the analysis of the state of the art and according to the need of keeping the scope of the WP and of the deliverable documents manageable, the developments of WP2 are defined as follows.

- The reference concrete modeling languages are assumed to be AUTOSAR and UML/SysML.
- A metamodel is provided for all the modeling concepts that are identified as required but missing from existing standards or project proposals (as a result of the Gap analysis). These metamodels are identified and defined in the next chapter. Metamodels are created using Eclipse/EMF editors and made available in text form or as code.
- We identify the existing metamodels (as part of academic papers, projects or project deliverables) that we believe can be reused and integrated.
- For all the metamodels, we try to provide a proposed implementation as AUTOSAR extensions (by linking and integrating with existing AUTOSAR Metamodels) or as UML/SysML profiles, preferably defined on top of the MARTE profile packages.
- Whenever possible or practical, we provide a sample definition of these profiles and/or additional concepts using commercial or open source tools. Among commercial tools, Rhapsody from IBM is preferably used for its capability of supporting both AUTOSAR and UML modeling.

3.2 Gap analysis

In this section we identify what are the modeling needs that are not satisfied by existing standards or outstanding proposals from projects. These gaps are identified based on the analysis of the project requirements, the state of the art of the scientific research, best practices and requirements from standards and the input of the partners. They are first expressed informally and then formalized as metamodels, using the modeling features of Eclipse.

3.2.1 Safety

For the modeling of concepts related to classical safety, we consider as adequate the modeling recommendations of the SAFE project. However, in SAFURE we strive at highlighting the concept of security attacks resulting in faults. There are three main set of elements that have been identified as possible extensions.

The Evita project identifies the need for the definition of security threats in a tree and to characterize each threat in terms of its possible impact on security. This is reflected in the generic metamodel description in which threats are defined as types of faults. However, there is no explicit modeling features that allows to characterize the threats in a hierarchy of dependencies that allows to assign to a security hazard its impact, likelihood and controllability attributes.. In addition, security threats should be characterized with respect to their attack potential using appropriate attributes.
The definition of the propagation of safety and security faults (and the same applies to taint analysis) requires that the causal dependencies in data streams (data dependencies between items that are written from inputs that are read) is explicitly identified in the component model(s). This should be done at the highest possible level in the architecture design to allow the analysis to be performed at the system level, abstracting from the detailed behavior of the component functions.
Finally, the prescriptions of the safety extensions of AUTOSAR (including, for example, the possibility of associating a SIL level to software and hardware components) has not yet been translated into modeling recommendations.

3.2.2 Time

In additions to the time concepts that are available from the standards, we believe the following concepts are required

- Models to define timing constraints for weakly-hard systems, that is, systems where deadline misses are tolerated upon condition that they occur according to a predictable and bounded pattern, and in general to specify timing requirements for overload conditions.
- Models for the definition of **time budgets** assigned to computations in a system characterized by end-to-end constraints.
- Patterns for specifying the need for **Timing isolation** and the algorithms and architecture elements that can be used to guarantee and enforce it.
- Models to describe (sub)systems with timing constraints at different criticality levels, leveraging the concepts of mixed-criticality as they apply to real-time systems.

Mixed criticality

The concept of mixed criticality in scheduling and timing analysis is based on simple concepts. It requires the definition of a criticality level that is associated to the schedulable entities. In most research papers those are tasks, but the migration of the concept to AUTOSAR and, in general, applicability to model-based flows in which tasks are derived from (smaller) functional units recommends that the criticality level is also associated with high-level functions or reactions.
The criticality level is expressed as a boolean or integer value in most papers and may correspond to different values of estimated execution times, possibly up to one for each level.

Timing Isolation

The requirements for time isolation in critical systems should be highlighted and the analysis of the capability of a system to support these requirements should be supported by providing means for the designer to express the system features that allow to check for timing isolation.
This means that adequate patterns should be provided for the modeling of operating systems, scheduling and resource management policies that aim at time isolation. These features should be associated with the definition of identifiers for the known policies with their properties and attributes.
A list of the policies that support timing isolation with the corresponding parameters are:

- Time-triggered scheduling, with the definition of the slot size, the slot assignment policy (possibly the full schedule) and possible time reclaiming algorithms
- Server policies including CBS (and possibly other)

Time budgets

In hard real-time system modeling, often a maximum *end-to-end latency* is specified which constrains the maximum time that is allowed to elapse between the occurrence of an event initiating a computation and the provision of the computational result. This function-related end-to-end latency has to be split in time budgets if, from an implementation perspective, the computation is performed by a linear chain of tasks (which may be distributed over several resources) and not by a single task only. A time budget has then to be assigned to each task in the chain such that the end-to-end latency constraint is still respected. In the timing modeling language TADL2 a *comparison constraint* is used to verify that the sum of the individual time budgets (local latencies) is smaller or equal to the end-to-end latency.

Weakly-hard systems

Likewise in the modeling of weakly-hard real-time systems, *end-to-end weakly-hard constraints* of the form (m, k) are specified where m denotes the maximum tolerated number of exceeded end-to-end-latencies in any sequence of k consecutive computations. As in the case of hard real-time systems, when moving to the constraint definition at a lower hierarchical level at which the sub-structure of the system becomes apparent, a form of budgeting has to be done. Here it is necessary to split the end-to-end (m, k) constraint into (m, k) sub-constraints for each involved task in the task chain. So far no equivalent to the above mentioned comparison constraint in TADL2 exists which identifies permitted/excludes non-permitted solutions to the (m, k) budgeting problem, as it is far more involved than the simple time budgeting problem for the end-to-end latency.

Other overload conditions

Other overload management conditions (beyond the weakly hard model) require that in general the user may be able to specify two bounds. These refer to the maximum number of timing violations that can be tolerated in a given time interval (this could be related to the n-over-k model of weakly hard systems) and the maximum lateness (delay from a deadline) that can be tolerated for each computation.

3.2.3 Security

With reference to Section 2.1.3, we have identified a number of concepts that are required but that are missing from existing standards, project proposals and scientific literature. In this section we briefly introduce and describe them. In doing this we use the same section structuring as Section 2.1.3.

On mechanisms and protocols

As to in-vehicle communications, standards and research have mostly focused on the authenticity and integrity requirements. Actually, most of especially the safety-relevant use cases rely on authentic and trustworthy information from sensors, between ECUs and to actuators. Therefore the primary scope is protecting and assuring authenticity and integrity of messages (transmitted data) and sender authentication (entity). Potential future topics of interest include confidentiality of communicated data and key exchange.
At the moment, the AUTOSAR Secure Onboard Communication concept does not contain default use cases requiring message confidentiality. However, for confidentiality and privacy reasons, some data has to be kept confidential while being sent over the in-vehicle bus. For example when initializing an in-vehicle security system, keys should be exchanged confidentially. Also keys and credentials for

other in-car systems may be communicated in the vehicle and personal data has to be kept confidential for privacy reasons. Furthermore, increasing connectivity of vehicles and at the same time increasing integration of personal devices like smartphones into the vehicle environment will lead to a growing need for solutions providing privacy for various data.
State-of-the-art solutions for ECUs with higher security requirements use cryptographic functionality implemented in hardware. EVITA's Hardware Security Modules (HSM) constitutes a relevant example that supports both AES symmetric encryption and Elliptic Curve asymmetric cryptography. Due to the standardized layered architecture, such hardware peripherals can easily be made available in an AUTOSAR stack. Furthermore, one possibility to combine confidentiality with integrity/authenticity is the usage of the Galois/Counter Mode (GCM). Although there is currently no standard AUTOSAR CRY/CSM interface, many HSM manufacturers support AES-GCM in hardware (i.e., offering an acceleration for the carry-less multiplication that is used to calculate the GHASH authentication tag).
The problem with confidentiality is related to the conjunction of the communication model of automotive applications and the limited bandwidth offered by customary bus technology. e.g., CAN bus. Typically, a CAN packet does not include addresses in the traditional sense and instead supports a *publish-and-subscribe* communications model. The CAN ID header is used to indicate the packet type, and each packet is both physically and logically broadcast to all nodes, which then decide for themselves whether to process the packets. If the confidentiality of a packet has to be protected, then we have two possibilities. The first one consists in encrypting the packet by means of the secret key that the sender shares with every receiver. Since the structure of an automotive application is relatively static, possible receivers of a packet could be defined at design phase. However, this solution would require to encrypt and transmit the packet as many times as the number of receivers. Unfortunately, most on-board networks cannot bear an increase of the bus traffic. Nowadays most CAN networks are utilized by almost 100%. An alternative solution consists in organizing receivers in a *group*, give them a shared *group-key* and encrypt packets by means of that key. Re-keying mechanisms should be conceived to both periodically refresh that key as well as to revoke and re-distribute it in order to evict compromised receivers [61][62]. Organizing automotive devices in groups for secure communication and key management has been proposed by EVITA, too [66].
Furthermore, in order to satisfy the identified security properties, namely confidentiality, integrity and/or authenticity, cryptographic algorithms have to be used. Using these algorithms has a cost in terms of timing, performance and, in general, power consumption. Therefore, choosing an appropriate security mechanism is critical in order to ensure the satisfaction of the timing requirements of the system while fulfilling the security requirements. For this reason, and to take into account the timing costs of different security mechanisms, the impact of security algorithms on the system performance must be evaluated. In order to perform such an evaluation, we may take inspiration from, for example, Daidone *et al.* who evaluated the impact of security on performance of wireless sensor networks under the IEEE 802.15.4 communication standard [54][55][56]. The performance metrics to be evaluated could be the processing time or the bandwidth consumption whereas the performance factors may include cryptographic algorithms, encryption modes, security requirements, and the size of security parameters (length of keys or fingerprints). These evaluation results could be used to enrich some time estimation toolkit in order to achieve an overall timing and schedulability analysis that takes into consideration security requirements too. Some results in this field have been already achieved [156][157].

On modeling

From the analysis of the state of the art, it is clear that in terms of security, AUTOSAR models focus on the mechanisms that should be implemented as part of the BSW layers and are therefore to be considered as architecture patterns.
However, AUTOSAR mostly disregards the application level, that is, how the designer of an application with security concerns should specify that its communications need to be suitably protected. This specification can hardly be defined as a simple set of security-related attributes applicable (for

example) to component ports, since the AUTOSAR specification matter-of-fact implies the selection of mechanisms based on the availability of resources (bandwidth and processing time). These are in turn functions of the time specification of the application and the availability of resources.
Therefore, there is no other way but to offer AUTOSAR/UML extensions at the application level that handle, **at the same time**, time, safety and security requirements. For this purpose, the evaluation of the performance impact of security is crucial.
These requirements should be applicable to the elements of the model, that is:

- In AUTOSAR, to runnables and any port interaction, of sender/receiver or client server type, as well as on events.
- In UML/SysML to operations and any port interaction, on standard and flow ports, as well as on events.

These attributes should be defined by means of suitable stereotypes.
For secure on-board communication, the definition of a set of different security levels would simplify the configuration of a car security system. Security experts define the different security levels and the associated properties. For every message with security protection needs, the appropriate level may be selected. The applicable security level may depend on the properties or kind of the message which needs to be protected.
The envisioned process requires a set of (possibly automatic synthesis) tools that bridges the gap between application-level specifications and the selection of architecture features.
More in general, modeling should be not limited to just the communication aspect. Rather, modeling should address security as much as possible. For instance, well-known security engineering best-practices make it possible to harden the software components. These include, for example, using safe string libraries, diligent input validation, and checking function "contracts" at module boundaries. Modeling should allow the designer to require the employment of these practices. For instance, a class diagram, possibly extended by means of proper stereotypes, may mandate the use of a `SecureString` class instead of a customary `String` library.
In a similar fashion, modeling should address other aspects of the system. According to the "defense in depth" principle, just hardening the software components is in general not sufficient. Another design countermeasure consists in reducing the attack surface. Consider the Bluetooth vulnerability documented by Checkoway et al. [47], for example. Using a safe `strcpy` library function would certainly harden the Blueetooth implementation component. However, further security improvements could derive from requiring that, in contrast to current procedures, the Bluetooth component will respond to pairing requests only after user interaction.

3.2.4 Architecture Features

There are several concepts at the architecture level that have been clearly identified as recurring elements or reusable patterns.
Among them, we focus on the concepts of protection kernel and hardware security module.
The concept of a protection kernel (often associated to a hypervisor or other hierarchical resource manager) is definitely gaining ground to provide the possibility of integrating multiple applications, or tasks, or functional components onto the same platform with functional and time isolation. In the network domain, software defined networking (SDN) can be used to implement such a resource manager.
The hardware security module is extensively discussed in the Evita project and, based on the project recommendations, several commercial implementations are starting to appear. In Evita a detailed metamodel of the data and the policies of the HSM are presented. However, a model for the HSM architecture pattern is not explicitly presented and offered.

Chapter 4

Abstract Modeling Concepts

This chapter contains the (semi-)formal definition of the modeling elements that are required for the analysis and the system representations in Safure. In the definitions of this chapter, the modeling elements are defined as abstract, that is, not necessarily tied to any specific (commercial or established) modeling language. To this purpose, the modeling elements are represented using the metamodeling features of Eclipse. However, even if they are meant to be abstract and independent, our modeling concept are undoubtedly influenced by existing standards, especially AUTOSAR and UML. Also, in the following, we assume the typical AUTOSAR methodology consisting in the separation of functional modeling and platform modeling, and the definition of an implementation of the functional concepts on the given platform by an explicit mapping level that defines the structure of threads and communication resources or messages.

4.1 General concepts

For the purpose of safety and security analysis it is important to identify the data dependencies and to find out which data items can be possibly affected by malfunctioning and/or corrupted components and functions. These dependencies should apply to data-oriented communication and service-oriented interactions, and should highlight what output value is computed based on what input value or what computation function may be affected as the result of a sequence of calls.
Given that this information can be potentially useful for all types of analysis (safety, security and time), including for example, the taint analysis summarized in the following sections, it is included in a dedicated general section.
Almost all behavioral modeling languages already have constructs to express these dependencies and specify exactly the analytical or procedural dependency among values. This is true, for example in all the behavioral diagrams of UML. However, we claim that these behavioral diagrams and descriptions are often at a level of granularity that is too fine for the analysis that we would like to enable at the system level, where architecture-level or system-level diagrams are more suited.
For example, in AUTOSAR, there is an indication of which runnable (an atomic specification of behavior with no internal details) reads from or writes onto ports, but no explicit declaration of the actual dependencies.
To this purpose, Figure 4.1 shows the proposal for the explicit definition of a data dependency between data elements that are read (input) and produced (output) by a given function.
Quite simply, we only recommend that the description of the indication that a generic executable reads or writes a data item (from a port) is also complemented with a set of value dependencies. The value dependencies are indications of causality dependencies implemented by the executable and connect one produced value with all the input values it can possibly depend upon.
Another general set of relations that are needed in the Safure models for multiple reasons (safety, security and time analysis) is the complete set of mapping of the functional elements onto the platform elements (including the basic software services and the hardware). Both UML/SysML and AUTOSAR

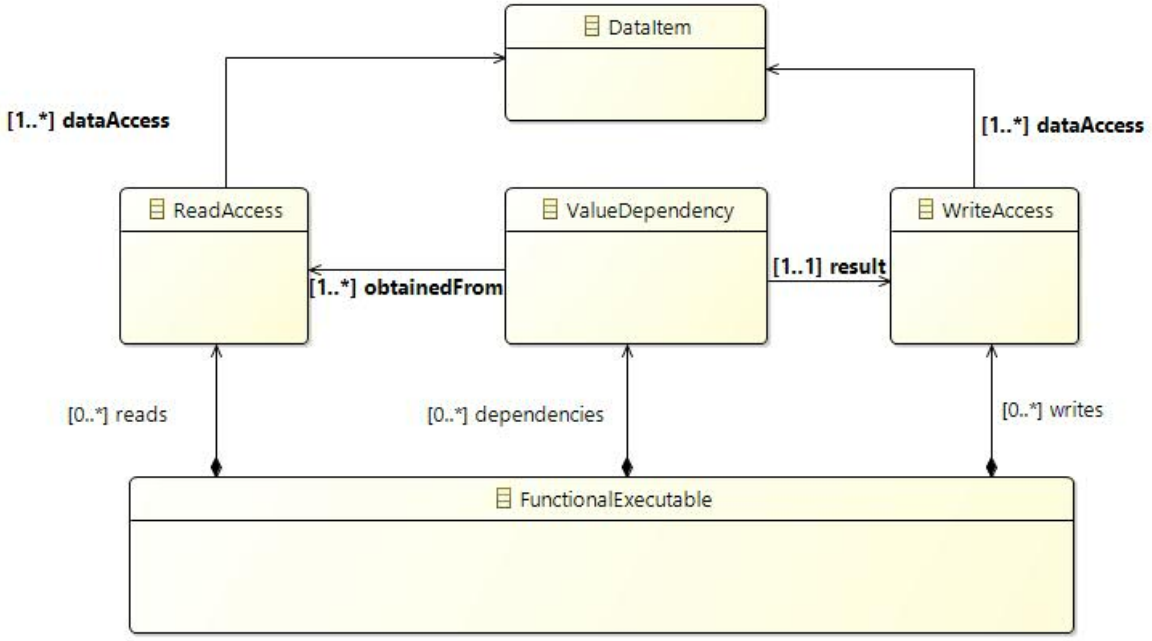

Figure 4.1: Expressing dependencies in data processing

provide the main concepts of mapping that are required.
The concepts for mapping that are required in Safure are identified in Figure 4.2. Most of them are already part of the SysML or AUTOSAR standards, with the possible exception of the consideration of partitions in execution under the control of a separation kernel and the mapping of data that is communicated on a communication resource.

4.2 Safety

The required safety concepts mostly correspond to the formalization of the concepts already highlighted in Evita (but not made explicitly available as part of a metamodel). They are summarized in Figure 4.3. The Figure shows the explicit connection of security faults with hazards and the proposed Evita classification of security hazards with respect to privacy, operational or financial risks, and the set of attributes that characterize attacks and the corresponding Faults. In this case, together with the qualitative definitions provided in Evita (the attributes with the enumerated types), we also plan for attributes that carry actual numerical values, corresponding to the typical safety concepts of mean time to fault and mean time to recovery. If these values could be computed, classical analysis methods for safety could handle the security attacks in an homogeneous way. In Evita, the introduced concepts are the standard concepts of Fault, Error and Failure, with the latter characterized by the associated set of Hazards, with attributes defining in a qualitative way (as integer levels) the safety, privacy, financial, and operational integrity levels. In addition, Denial of service and a security malfunction are added as special types of failures. Also, Intentional Faults are recognized as possible types of faults and are further characterized by attributes, such as the controllability (from C1 to C4), the amount of expertise that is required to carry a possible attack, the required knowledge for carrying a successful attack and the opportunity for the attacker (all defined qualitatively as enumerated). These qualitative measures are tentatively used in Evita to perform quantitative analysis. This is still not possible in a formal way, at least using the typical measures of safety analysis. However, we recognized that such a homogeneous treatment is desirable, and therefore allow the characterization of Intentional Faults with the typical attributes of traditional faults, such as the mean time to attack and the mean time to recovery.

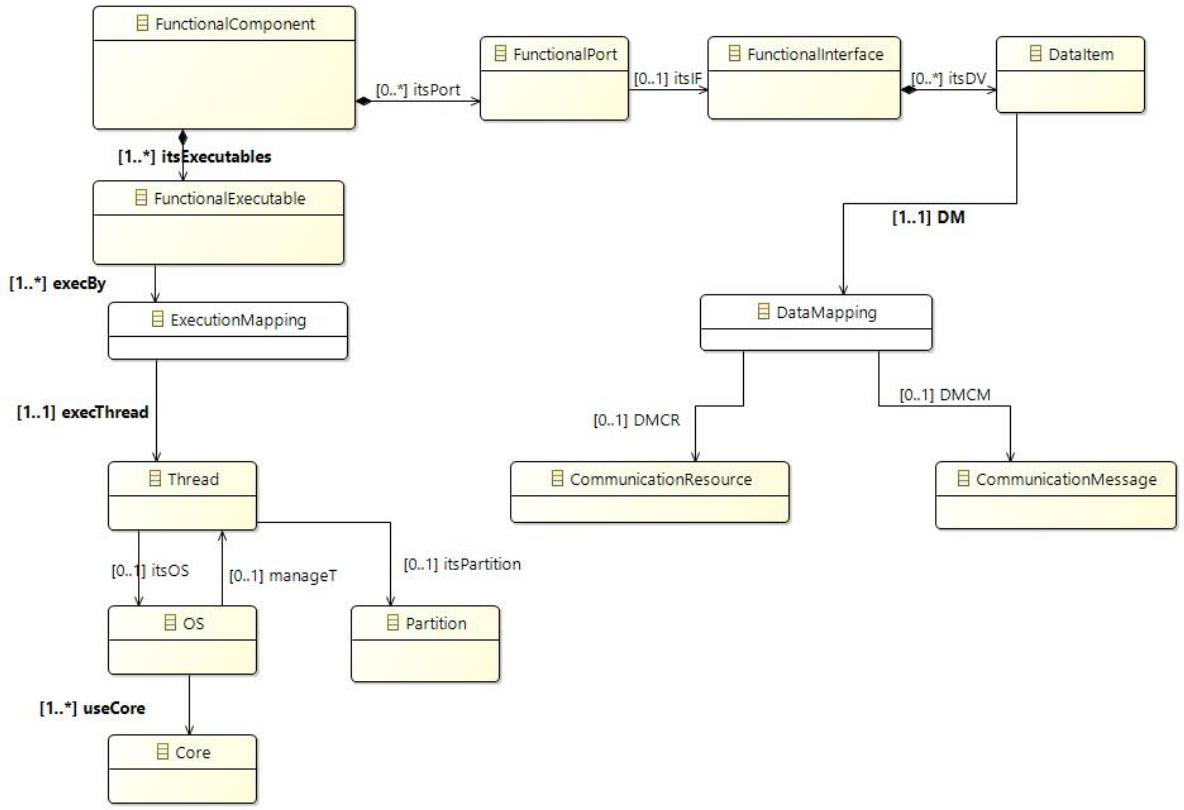

Figure 4.2: Mapping execution and data

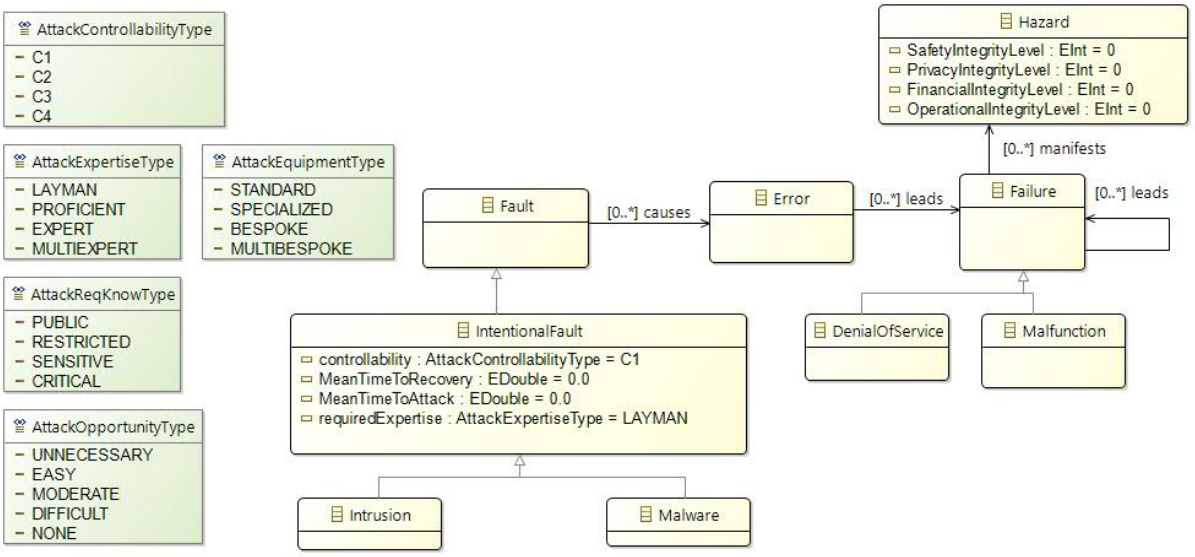

Figure 4.3: Metamodel of additional concept connecting attacks to faults and hazards

4.3 Time

Figure 4.4 shows the main entities of interest for the description of the timing extensions of SAFURE. First, timing specifications are classified as belonging to the two main types of timing constraints (or assertions) and timing assumptions. Assumptions are timing properties that are assumed to be true about the model of the system that we want to verify or analyze, while assertions are properties that we want to be guaranteed based on the system design the has been developed. For example, in timing analysis, the maximum time that a function requires for execution may be an assumption (given that the hardware has been selected and the execution of the function in isolation analyzed and estimated in some way). A deadline that is imposed on its termination may be an assertion that we may want to demonstrate being always guaranteed, upon an adequate selection of a scheduling policy and an assignment of priorities. This is in contrast with the AUTOSAR classification that provides no distinction between the timing characteristics that are assumed to be true in the system (such as the periodicity of activation events, for example), and the timing properties that need to be verified (assertions) such as deadlines.
The two types of timing assumptions that are considered are assumptions on execution times and assumptions on event arrival times (further detailed in the following). Activation assumptions apply to timing events. Timing events are further possibly belonging to a timing chain. Timing execution assumptions apply to schedulable entities and pertain to the estimate that the execution of a given functionality will be bounded by a given value or characterized in some way (for example in a probabilistic way).

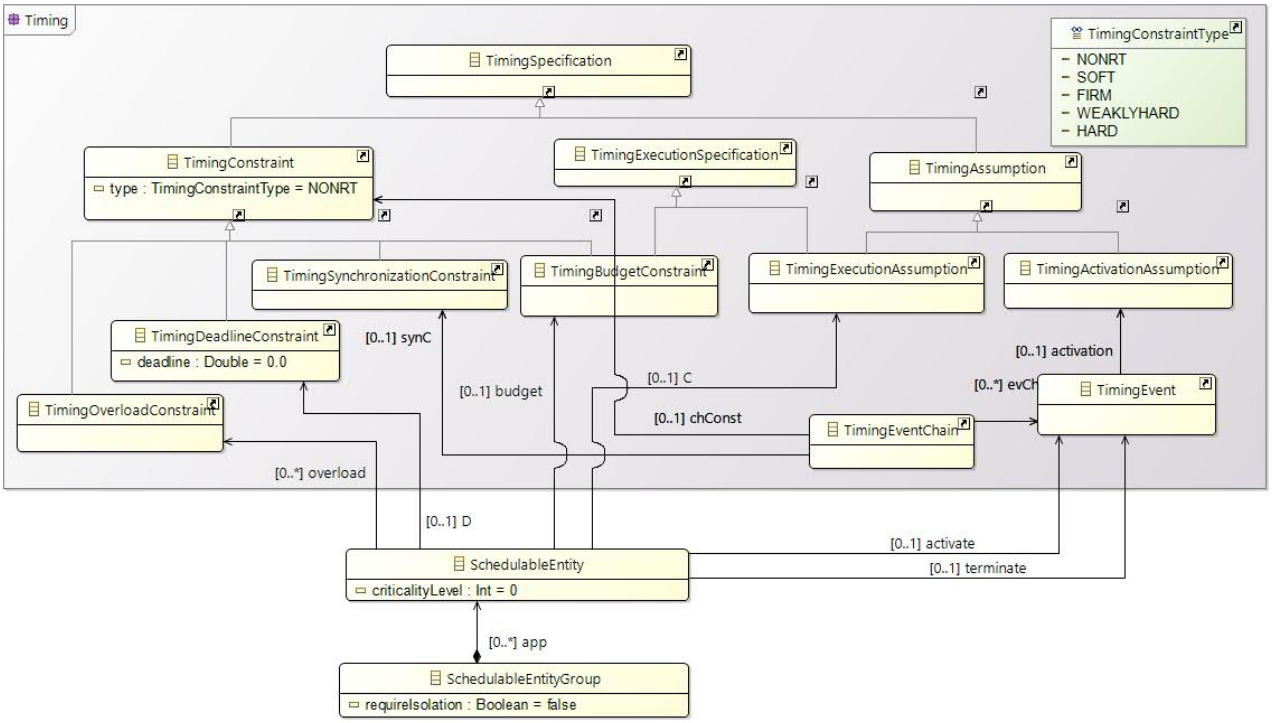

Figure 4.4: The modeling concepts for the representation of time constraints and assumptions

Timing Constraints are of four types: **execution time constraints** (timing budgets), **deadline constraints**, with a deadline attribute, **overload constraints**, that provide more information to the expected worst-case behavior when deadlines are trespassed, and **synchronization constraints** that apply to event sets. Timing constraints are characterized according to the severity of the time constraints as being of type NRT (or non-real-time, no consequence for missed deadlines), SOFT (a missed deadline decreases the value of the system), FIRM, HARD or WEAKLY HARD.

In contrast to AUTOSAR, we provide for a distinction between the estimated execution time of an executable entity (or WCET) that is defined as an assumption, and the maximum execution time that is assigned to an executable entity or a set of them (or timing budget), acting as a constraint.
The set of constraints that apply to the system being possibly in an overload condition (with deadline misses) is specified in Figure 4.5.
Timing Constraints and assumptions may be applied to schedulable entities and the schedulable entities are characterized by a criticality level. In agreement with the recent literature in real-time research, we assume this parameter to be an integer value (higher values mean higher criticality). Also, schedulable entities may belong to a group (an application, a set with functional dependency, a set with a common supplier or a set with common criticality level) for which time isolation is requested if sharing resources with other groups (or schedulable entities).

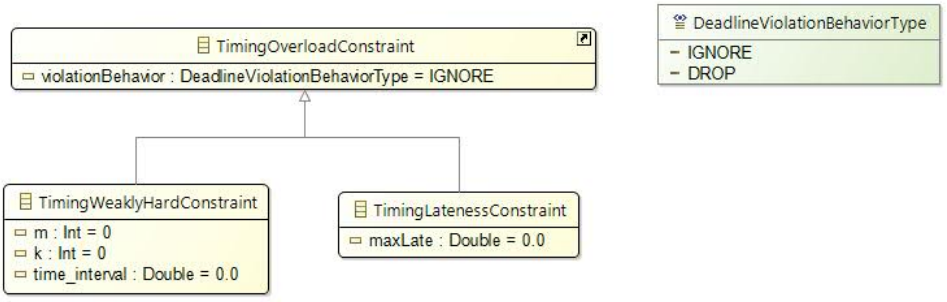

Figure 4.5: The modeling concepts for the representation of behavior in overload conditions

The overload constraints that are considered are of two types: a weakly hard model, and a maximum lateness model. In addition, the overload constraint needs to be characterized by an attribute that restricts the system behavior when deadlines are missed, by allowing schedulable entities to continue (IGNORE) or terminates them at the deadline (DROP).
The weakly hard model is characterized by three parameters. The first (denoted as m for consistency with the literature on the subject) represents the required number of instances that should not miss the deadline. The second and the third specify the context to which the m parameter applies. If it applies to a set of instances, then the k parameter is used (at least m instances every k consecutive ones must meet their deadline). If it applies to a time interval, the third parameter is used.
The concept of Timing execution specification is further refined (as shown in Figure 4.6) to allow the definition of function and task execution times and budgets. In the simplest model, three specifications can be foreseen, a definition based on the maximum only, a minimum and maximum, and a full specification of a (probability) distribution.
However, to allow for different time specifications for different criticality levels, or different execution modes, or even to allow for more complex timing specification patterns, the model provides for multivalued specifications by order (in array form) or by key.

4.4 Security

In this section we summarize the main modeling security concepts of SAFURE. In doing this, we build upon the results of the EVITA project and, in particular, EVITA's meta-models (see Figure 2.25). As described in Chapter 2, EVITA has defined a number of models, many of which refer to basic concepts such as trust and attacks [65]. However, EVITA has essentially focused on intra-platform security issues whereas, in contrast, SAFURE focuses on the security of remote or in-vehicle communication and its relationship with performance and safety.
The main concepts of the SAFURE security modeling are reported in Figures 4.7 and 4.8. Intuitively,

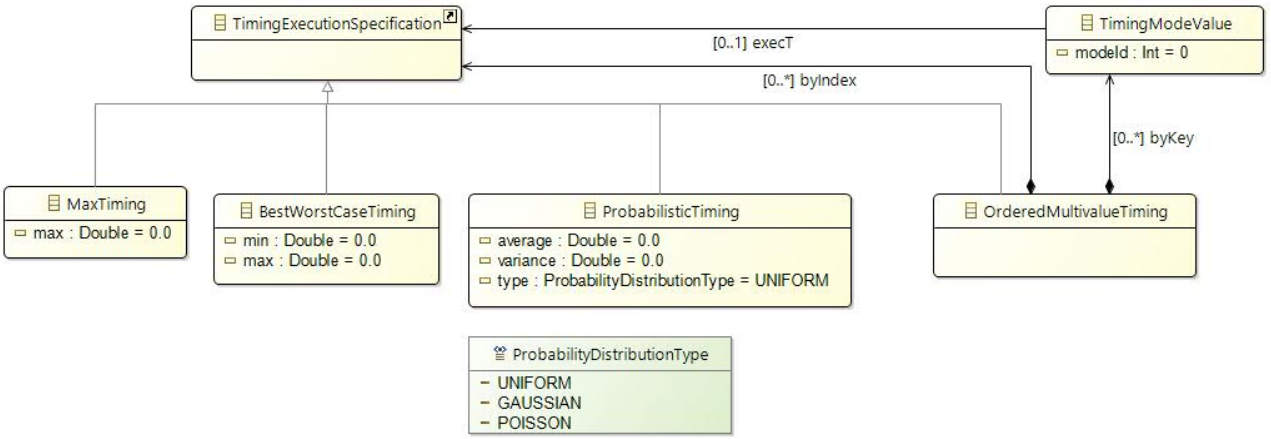

Figure 4.6: Specification of execution time assumptions or constraints (budgets)

Figure 4.7 shows security concepts at the functional level whereas Figure 4.8 shows SAFURE's in-vehicle secure communication model.

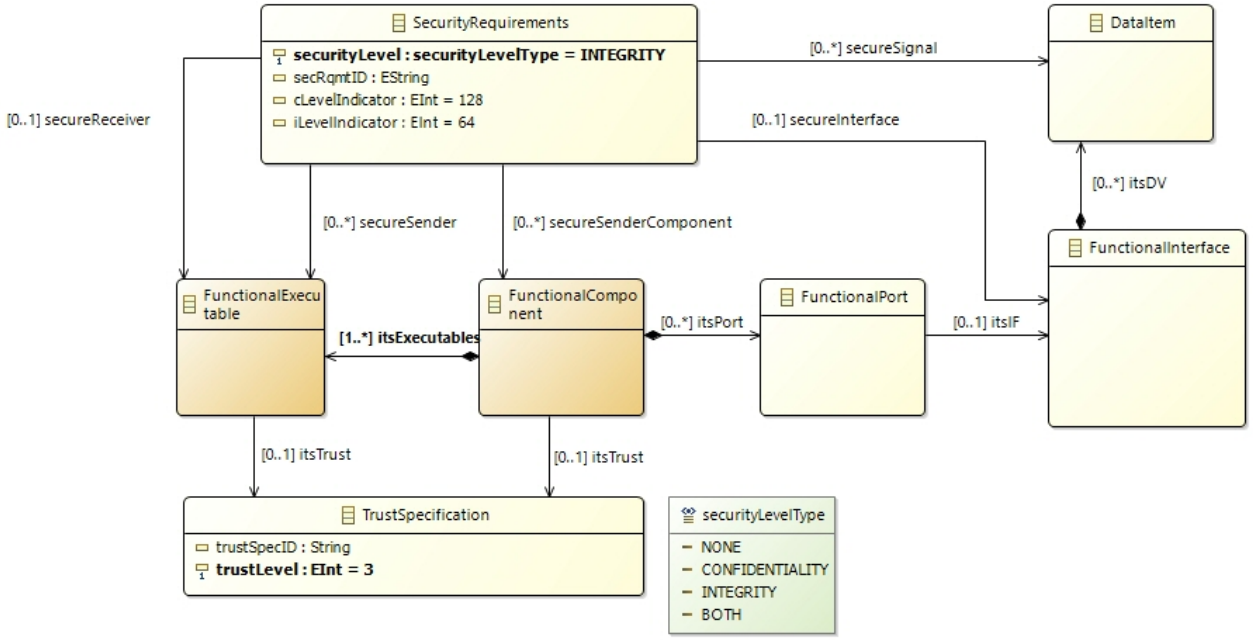

Figure 4.7: The modelling concepts for the representation of the functional elements for security.

With reference to Figure 4.7, SAFURE defines two main concepts at the functional level: the trust level of a functional element and the security requirements of communications between elements.
A functional element, either a component or an executable, may be associated to a *trust specification* which specifies to what extent the element can be trusted to provide the expected function, or service with respect to attacks targeted to compromise the functionality of the element. A trust specification consists of: i) a *trust specification identifier* (trustSpecID), which identifies the specification, and ii) a *trust level* (trustLevel) which provides an indication of the extent to which the element can be trusted. The trustLevel is an attribute of type *trustLevelType* that corresponds to an integer in the range 1 to 5, being 1 the highest trust level and 5 the basic one.
The notion of trustLevel recurs in other projects. For example, it is similar to EVITA's *attack potential*

which is a measure of the effort required to create and carry out an attack [64]. The attack potential is computed by considering different factors including:

- *Elapsed time*, the amount of time needed to identify a vulnerability, develop an attack method and execute the attack;
- *Expertise*, fields and level of expertise required;
- *Knowledge*, specific knowledge of the system under investigation required;
- *Window of opportunity*, amount of access to the target required for the attack; and, finally,
- *Equipment*, equipment required to carry out the attack.

A high attack potential corresponds to a low probability of successful attacks, and consequently to a high trust level.

Factors like Required Resources and Required Know-How are considered also in the SAHARA (Security-Aware Hazard Analysis and Risk Assessment) method for defining threats criticality [104]. Similarly, in the Common Vulnerability Scoring System (CVSS) [52], defined as an open standard to assess vulnerability in software systems, the Exploitability metrics represent a measure of vulnerability and quality of a components. In this case, the factors used are: the *Attack vector*, the context by which the attack is possible (e.g. network access, or local access); the *Attack complexity*, that is, the knowledge of the system required by the attacker; *Privileges Required*, the level of privileges an attacker must possess; *User interaction*, whether the vulnerable system can be exploited without interaction from any user other than the attacker; and, finally, *Scope*, the impact on resources beyond the component. CVSS has been used recently in [111] to assess vulnerability of components security analysis of automotive architectures at the system-level.
In addition to the factors influencing an attack listed above, the trust level of a software element can be related to the software development process through which the element has been specified, designed, implemented and tested. As a consequence, the ASIL of the component can contribute to the estimate of the trust level of the component.
In SAFURE, *security requirements* can be specified on communications among elements. More specifically, it can be specified on communications of a given sender element, communications containing certain data items, as well as communications specified by a executables receiving from a given interface (secure interface). As in EVITA, a requirement is a specification of a required amount of trust (actually a dimension of trust) in terms of a system-specific criterion and a minimum level of an associated quality metric that is necessary to meet one or more trust policies. More precisely, the class Requirement in EVITA's Trust meta-model (see Figure 2.25) is a generalization of SAFURE's secure communication requirement. A security requirement contains four attributes:

- a security requirement identifier,
- a security level,
- a confidentiality level indicator, and
- an integrity level indicator.

A *security requirement identifier* (secRqmtID) identifies the security requirement. A *security level* (securityLevel) specifies the desired secure communication requirements. It is of type *securityLevelType*, an enumerated that contains four values: none, confidentiality, integtity, and both (namely confidentiality and integrity). The attributes *confidentiality level indicator* (cLevelIndicator) and *integrity level indicator* (iLevelIndicator) provide a quantitative indication of the confidentiality and integrity, respectively, of the communication. With reference to Figure 4.7, a cLevelIndicator equal to 128 means that the computation complexity necessary to break the communication confidentiality

should not smaller than $\mathcal{O}(2^{128})$. This requirement can be fulfilled by using the AES-128 cipher, for example. Analogously, an iLevelIndicator equal to 64 means that the computation complexity necessary to break the communication integrity (i.e., to find a collision) should not be smaller than $\mathcal{O}(2^{64})$. According to AUTOSAR Secure On-Board Communication, this requirement can be fulfilled by using 64-bit MACs, or larger.

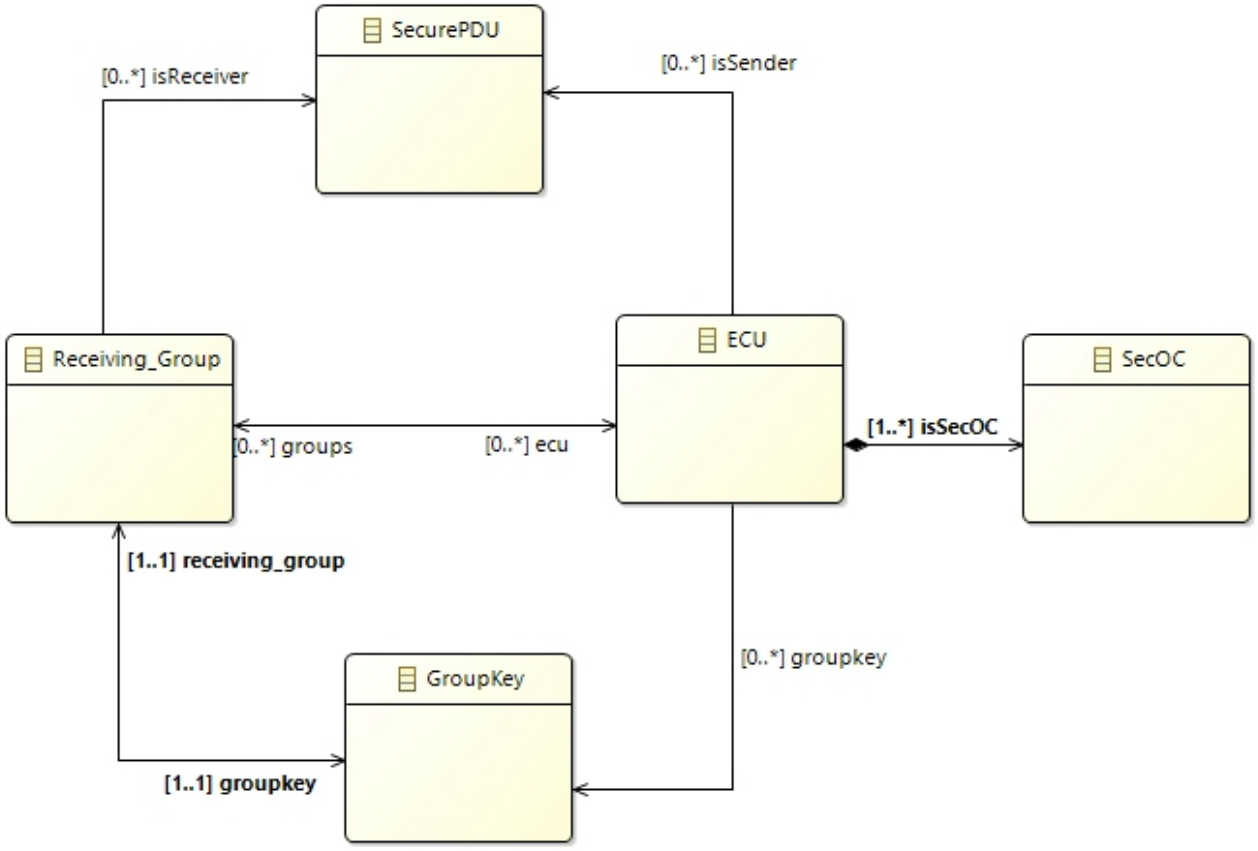

Figure 4.8: The modeling concepts for the representation of secure communication

Figure 4.8 models the SAFURE's secure in-vehicle communication. Let ECU ε be the the sender of PDU μ and $\Sigma_{\varepsilon,\mu}$ the non-empty set of receiving ECUs. This set is partitioned into r non-overlapping, non-empty *receiving groups*, being $\Gamma_{\varepsilon,\mu,i}$ the i-th one, such that $\Sigma_{\varepsilon,\mu} = \bigcup_{i=1}^{r} \Gamma_{\varepsilon,\mu,i}$. For each receiving group $\Gamma_{\varepsilon,\mu,i}$, the sender ε shares a secret cryptographic *group key* $K_{\varepsilon,\mu,i}$ with it, namely, with every member of the group. Of course, there exist no two groups that have the same value for the respective group keys. More formally, $\forall K_{\varepsilon',\mu',i}, K_{\varepsilon'',\mu'',j}, K_{\varepsilon',\mu',i} \equiv K_{\varepsilon'',\mu'',j}$ *iff* $\varepsilon' \equiv \varepsilon'' \wedge \mu' \equiv \mu'' \wedge i \equiv j$. The sender ε uses the key to secure the PDU μ, so obtaining a secured PDU before broadcasting it into $\Gamma_{\varepsilon,\mu,i}$. Each member of the receiving group uses the key to unsecure the secured PDU.
The actual securing/unsecuring operations consist in applying specific cryptographic services implementing the security requirements (see Figure 5.1). In AUTOSAR architecture, the cryptographic services are provided by the Secure On-board Communication object possibly exploiting an HSM, if present on board. Notice that a secured PDU is generalization of SecuredIPDU in the AUTOSAR's parlance (see Figure 2.71).
Intuitively, the rationale behind the grouping is to guarantee secure communication requirements, make the whole system more resilient to internal attacks while keeping its ability to fulfil performance and safety requirements. As to performance, grouping receivers allows us to broadcast just one instance of the message secured by means of the group key instead of as many instances as the receivers, each secured by means of the corresponding receiver's private key. It follows that by doing so we can

both reduce the computation overhead on the sending side and, more importantly, the communication overhead on the bus.
As to safety, we aim to prevent that an ECU ε', whose task *critically* depends on message μ, may share a cryptographic key with an ECU ε'' having a "low" resulting trust level (see Figure 4.7). Actually, this ECU displays a high risk of being compromised. If this is the case, if ε'' shares a key with ε', than the adversary would be able to "forge" secured PDUs for ε' with consequent negative effects in terms of safety. Of course, mapping algorithms must be devised that properly allocate components/executables to ECUs according to their trust level and group them in order to guarantee the required performance-security-safety trade-off.
As a final remark, a GroupKey can be considered an instance of EVITA's SymmetricKey, which is a KeyObject which in turn is a Cryptographic Object (see Figure 2.31).

Chapter 5

Architecture patterns

The architecture patterns that are considered useful for the representation of Mixed-critical CPS with safety and security requirements are:

- HSM: the concept of a Hardware Security Module as defined in Evita and later implemented and sold by several vendors.
- Separation kernel: as described in the MILS recommendations and summarized in the previous chapters

5.1 HSM

The concept of a High Security Module is defined in Evita and now implemented by many chip manufacturers.

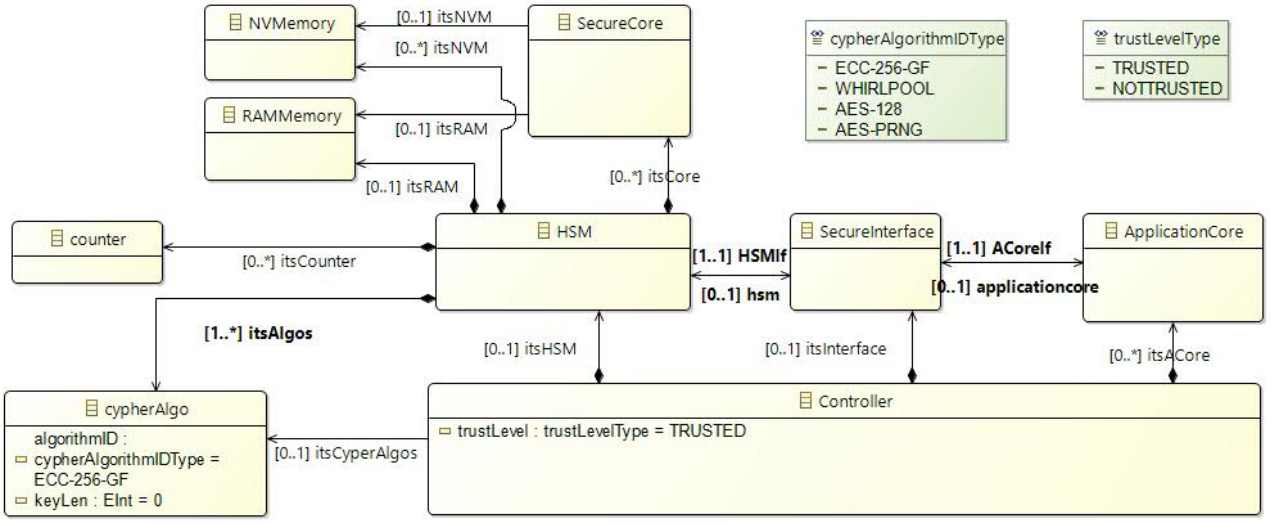

Figure 5.1: The modeling concepts for the representation of the platform elements for security.

With reference to Figure 5.1, at the architecture level, we consider a ECU as a device capable of securing/unsecuring PDUs and thus able to perform adequate cryptographic algorithms (CryptoAlgo). Each algorithm is identified by its type (belonging to a predefined set) and the length of the key that is used for encoding. With reference to Figure 5.1, an ECU runs a number of cryptographic algorithms either on the ECU itself or provided by a dedicated cryptographic (co-)processor called Hardware Security Module (HSM). The HSM is further characterized in terms of the HW resources it contains

(RAM and non-volatile, or NV memory). We assume that an ECU is equipped with one HSM only. With reference to Figure 2.30, we may model the cryptographic algorithms CryptoAlgo in terms of EVITA's cryptographic services (CRS).

5.2 Separation kernel

The separation kernel (SK) architecture pattern is required for the representation of kernels that can execute applications, with possibly a (real-time) OS in protected partitions. The separation kernel enforces the system configuration upon all their communication and resource requests in a non-bypassable way, while not inspecting or protecting what happens within the partition itself. For example, if a partition is authorized to communicate over a network and to use the HTTP protocol, the SK will not protect the application against infection by a virus introduced into the HTTP payload. The Separation Kernel is one of the components of the MILS Trusted OS [4].

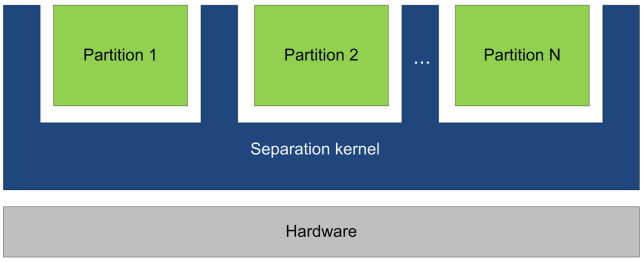

Figure 5.2: The generic structure of a separation kernel

5.2.1 Overview

A separation kernel is a component that enforces a resource allocation policy and an access control policy on its exported resources (partition, resources allocated to a partition, communication objects). Communication objects allow for controlled information flow between partitions. A separation kernel may have an explicit or an implicit information flow policy on its partitions (access control policy together with resource allocation policy). A separation kernel typically exploits mechanisms provided by the hardware to provide the separation between partitions in a MILS core. Examples are

- A resource allocation policy might consist in the allocation of a certain amount of time, for example 20 milliseconds periodically every 100 milliseconds, of the CPU resource to a given partition.
- An access control policy might assign communication object C as writeable to partition A and readable to partition B, defining an implicit information flow policy from A to B.
- An explicit information flow policy for a separation kernel could consist of the specification that only partition P via whatever interface may send information to partition Q.

5.2.2 Partitions

A partition is a component that serves to encapsulate application(s) and/or data. Thus, the content of a partition is application(s) and possibly other data. A partition is a unit of separation with respect to

- resource allocation in the space and time domains,
- an access control policy and an information flow policy in the space domain.

In a MILS system, partitions are created and maintained by the MILS core (see definitions below) based on security policies defined for a given use-case.

5.2.3 Services

Classical Approach

In some of the early work such as [36, 150, 14] a strong emphasis on the implementation of information flow and its absence has been taken.
The only tasks assigned to a MILS separation kernel are the partitioning of processes and failure containment. Consequently, we can represent the safety and security requirements for a separation kernel by four simple foundational policies:

- **Data Isolation**: Information in a partition is accessible only by code running in that partition. Private data remains private.
- **Control of Information Flow**: Information flow among partitions is from an authenticated source to authenticated recipients. The source of information is authenticated to the recipient. Information goes only where intended.
- **Resource Sanitization**: Usage of the microprocessor and other hardware, such as networking hardware, cannot be used as covert channels to leak information.
- **Fault Isolation**: A failure in one partition is prohibited from cascading to any other partition. Failure detection, containment, and recovery are performed locally [14].

Description of functionality grouped according to where separation is made (space/time)

In the following paragraphs, the approach taken in [149] with the comments versus previous section will be presented.
Separation in space: Applications can be hosted in different partitions. Partitions get assigned memory resources (i.e. space). In this way, the separation kernel enforces its configuration: that is, access control on partition content, per-partition provision of physical memory space and I/O memory space. By confining applications into partitions, the separation kernel enforces that these applications can affect neither applications in other partitions nor the separation kernel itself.
Separation in time: Applications can be hosted in different partitions. Partitions get assigned CPU time (i.e. time windows). In this way, the separation kernel enforces its configuration: that is the allocation of a predefined amount of the CPUs' time to partitions. Several partitions can share the same time window. On a partition switch CPUs will be reused. The separation kernel enforces that no residual information is in CPU registers or memory caches according to the configuration. The separation kernel may assign a priority to every subject to allow priority based scheduling within one time window or it may delegate the schedule within the time window to an OS in execution within the partition.
Provision and management of communication objects: Applications hosted in different partitions can get assigned a set of communication objects under control of the separation kernel. A communication object is an object exposed to one or multiple partitions with access rights as defined in the configuration data, thus allowing communication between partitions.
Separation kernel self-protection and accuracy of security functionality: Separation kernel self-protection and accuracy of functionality supports reaching and keeping a safe and secure state of the MILS system. The separation kernel statically assigns automatic invocations of error handling functions to recover from or respond to error conditions.
Again, this characterization is isomorphic to the characterization of Section 5.1.3.1. Like 5.1.3.1, it is optimized to be stand-alone and concrete. It splits up the data into "separation in time" and "separation in space". The "resource sanitization" is subsumed under "separation in time". "Control

of information flow" is represented by "provision and management of communication objects". "Fault isolation" is subsumed under "separation in space" and "self-protection".

5.2.4 Virtualization services on top of separation kernels

Virtualization is not a necessary part of separation kernels. However, because many separation kernel deployments provide support for virtualization services, the concept is described here. A virtual machine (VM) consists of software that imitates a physical hardware machine. The virtual machine will for example give the illusion of a physical CPU and physical memory to an operating system that is running in it. An operating system running in a virtualization environment is called "guest". A virtual machine monitor (VMM), also called a "host" or "hypervisor" is the software managing virtual machines.

5.2.5 Modeling

The metamodel of Figure 5.3 shows the main required concepts.

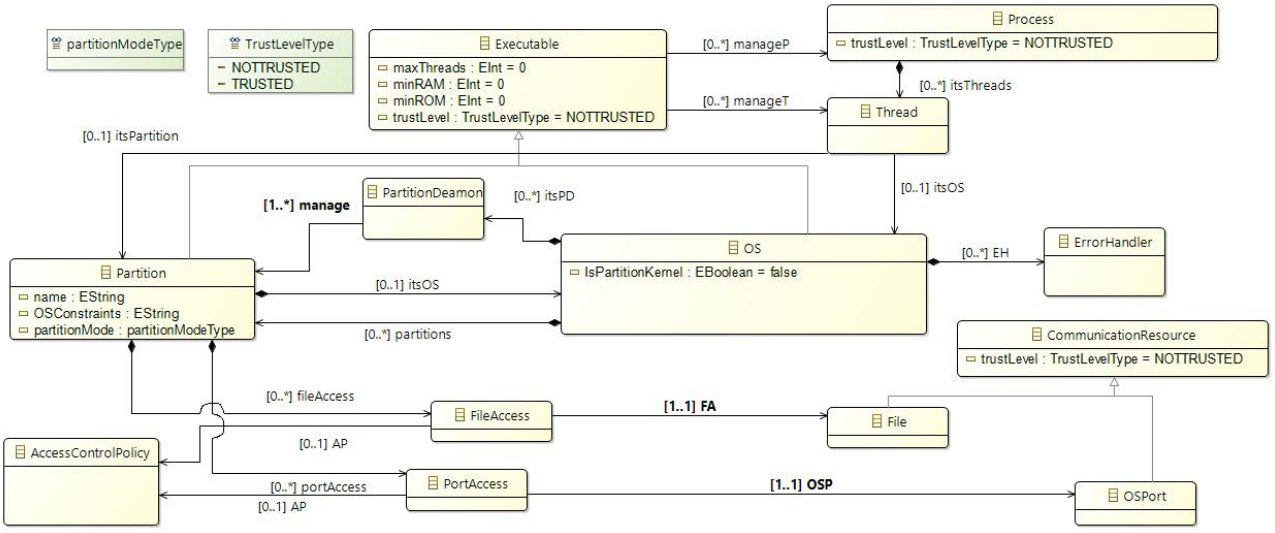

Figure 5.3: The modeling concepts for the representation of Hypervisors

The metamodel allows the representation of simple and hypervisor type OS, by denoting each of them with the simple metaclass OS. An OS can be defined as hypervisor: If this is the case, then the OS may execute a set of Partitions (bottom composition). A Partition may in turn execute an internal OS supporting an application. An OCL constraint needs to be defined to ensure that only hypervisor OS contain partitions. OS and Partitions are derived from the generic class Executable that provides a name, and an indication of the maximum number of supported threads, as well as the minimum amount of RAM/ROM that is required by the OS or the Partition. In additions, relations connect the OS and partition to the communication resources (files and ports) that are created (composition) by the OS and used by the partitions for communication. Please note that Executables have an attribute defining their trust level and the same is true for the communication resources.
A basic set of consistency checks apply to these model entities.

- An OS that is NOTTRUSTED cannot have TRUSTED partitions
- An OS that is NOTTRUSTED cannot have TRUSTED files or ports
- If an OS is trusted but employs a scheduler without time or functional protection, then if is managed at least one NOTTRUSTED partition, then all of its partitions are NOTTRUSTED.

The definition of trust levels associated with executables and communication resources also enable taint analysis once the logical resources are allocated to the platform elements and their implementation. For example, a communication port that is trusted but mapped onto an OS port that is not trusted will become not trusted.
An executable can also be associated to a set of cores, meaning that the partition or the OS manages the execution of threads on those cores. Also, an executable may be characterized by a set of managed processes and threads and a scheduling algorithm. Finally, an additional set of relations connect the executable (OS, Partition) with the Hardware resources assigned to it (RAM and ROM memory, IO devices).

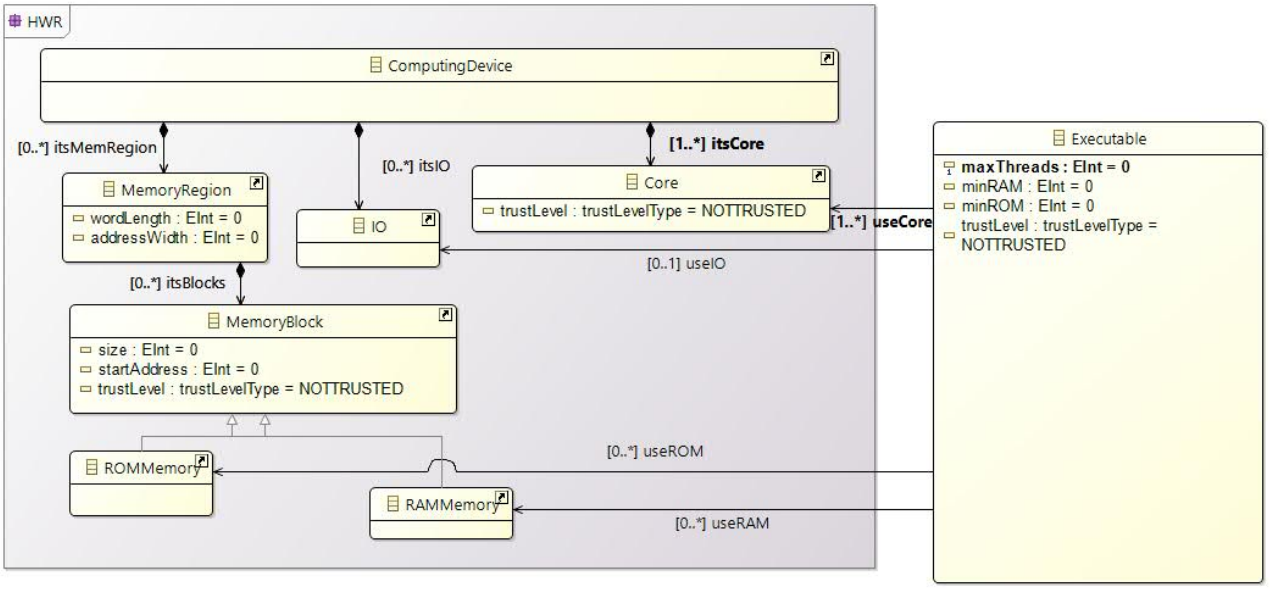

Figure 5.4: The modeling concepts for the representation of hardware resources

Figure 5.4 shows the main modeling concepts for the representation of hardware resources (package HWR). The HW resources of interest are memory regions (partitioned is RAM or ROM), IO devices and computing cores. For the purposes of SAFURE, each HW resource is associated with a trust level attribute (of the previously defined type trustLevelType).
Once a logical resource is associated with a hardware resource, the logical resource trust level will be combined with the trust level of the associated resource (hosting or executing it) and the minimum trust level of the pair will be associated to the logical element as a result of the mapping. Therefore, if, for example, a trusted port is mapped to a NOTTRUSTED memory section, then it should be considered as NOTTRUSTED.
Figure 5.5 shows the representation of schedulers. Each scheduler, of type priority or time-based is characterized by a criticality level and a type. The criticality level is expressed by the scheduler capability of supporting isolation in the domain of memory (functional isolation), time (time isolation) or both.
One of the usages of this model is to be able to execute the following analysis on application level and how the hypervisor is used by these applications:

- Covert channel analysis of hypervisor configuration for particular deployment. They can be timing (e.g. there are side effects on schedule) and the underlying HW platform [95].
- Taint checking/analysis: Wikipedia says: “The concept behind taint checking is that any variable that can be modified by an outside user (for example a variable set by a field in a web form) poses a potential security risk. If that variable is used in an expression that sets a second variable,

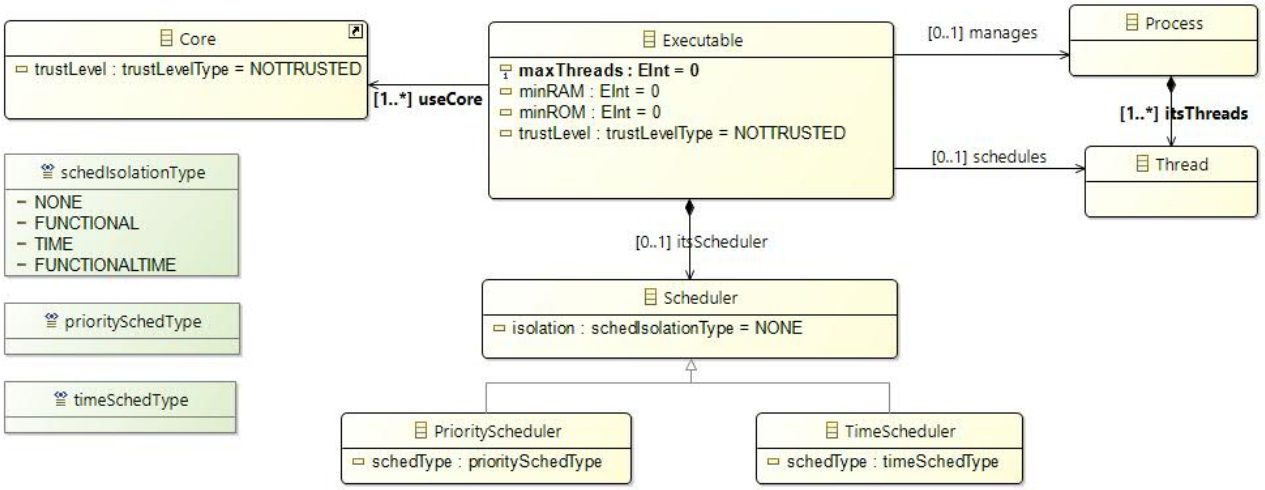

Figure 5.5: The modeling concepts for the representation of schedulers

that second variable is now also suspicious. The taint checking tool proceeds variable by variable until it has a complete list of all variables which are potentially influenced by outside input. If any of these variables is used to execute dangerous commands (such as direct commands to a SQL database or the host computer operating system), the taint checker warns that the program is using a potentially dangerous tainted variable. The computer programmer can then redesign the program to erect a safe wall around the dangerous input."

- Data Coupling and Control Coupling analysis: on resources exported by hypervisor and data exchange by applications.
  - Data coupling -the dependence of a software component on data not exclusively under the control of that software component.
  - Control coupling- the manner or degree by which one software component influences the execution of another software component.

Chapter 6

Concrete Modeling Concepts

The purpose of this chapter is to provide a set of proposals on how to encode the recommended modeling concepts emerging from the gap analysis and abstract modeling stages onto UML/SysML or possible AUTOSAR extensions.
With respect to AUTOSAR, the proposed extension has the main purpose of possibly inspiring future revisions of the standard by the participating members, given that AUTOSAR can only be extended through a formal process controlled by a restricted set of members. On the other side, SysML and UML are meant to be very generic but also easily extensible through a set of controlled features, which ultimately consist in the definition of a suitable profile (or a set of profiles) containing a set of stereotypes with their properties and possible constraints.
In most cases, the UML extension (and examples of its applicability) will be the proposed output of this section. However, the UML extensions will be implemented in Rhapsody, which allows AUTOSAR modeling on top of UML (meaning that the AUTOSAR elements are themselves provided in Rhapsody as stereotypes in a UML profile). The Rhapsody implementation will give hints on a possible (actual) AUTOSAR use.

6.1 General concepts

The two general concepts with applicability to all the domains of Safure that have been identified in the analysis are

- the definition of input-output dependencies at the architecture level;
- the definition of the mapping of functional to platform elements (especially with respect to communication resources and partitions in protection kernel deployments).

The following section will provide a sample extension that allows the representation of input-output dependencies.

6.1.1 Input-Output dependencies in AUTOSAR

There are two possible ways to provide an implementation in AUTOSAR of the required concept. Both of them are illustrated with reference to a Rhapsody implementation. The sample Rhapsody model that is used as a reference scenario is shown in Figure 6.1. The figure shows a sample application component with two sender ports (on the right of the component diagram and also on the model tree), labelled as `sender_1` and `sender_2`. The Sender ports are typed by the data interfaces `if_DATA1` and `if_DATA2`, with their data elements, as shown in the model tree. The component also has two receiver ports (on the left of the diagram and on the model tree), `receiver_1` and `receiver_2`, typed by the same two data interfaces.
The component has an internal behavior model that consists of only one runnable, `Run1`. The runnable reads from both input ports and writes to both output ports, as indicated by the dataReadAccess and

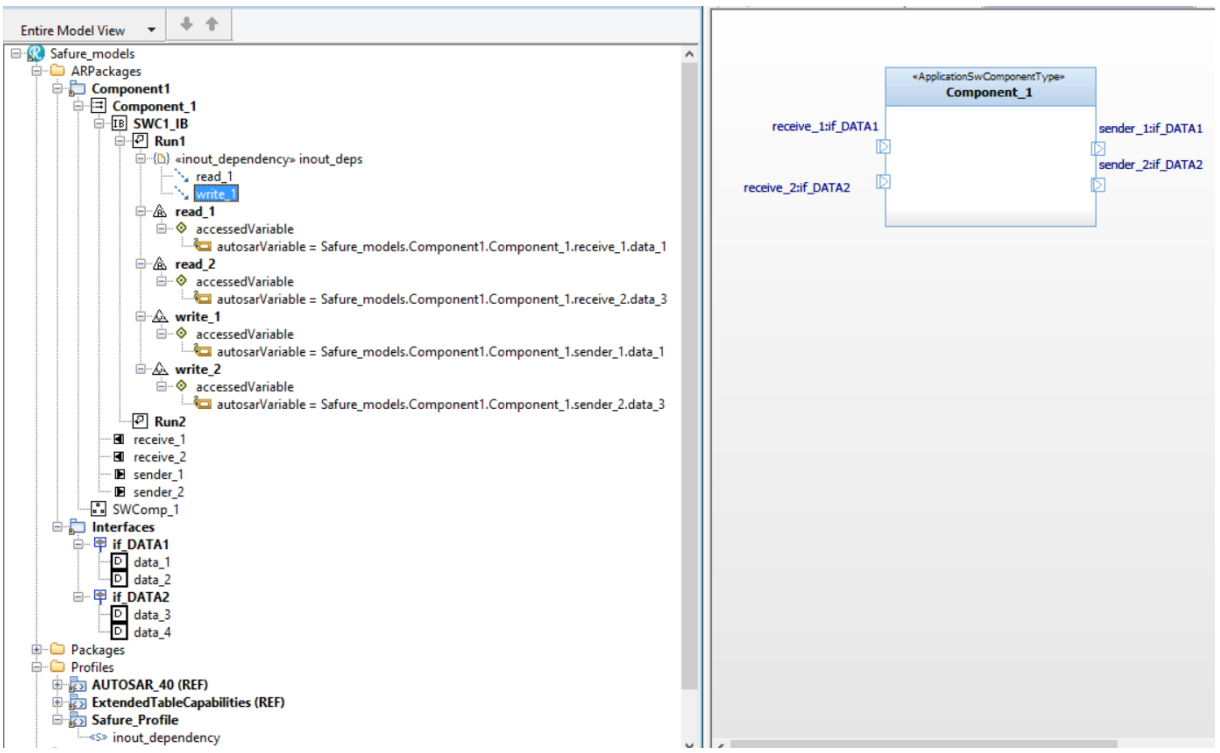

Figure 6.1: Expressing dependencies in AUTOSAR-Rhapsody by means of stereotyped constraints

dataWriteAccess specifications. However, suppose we want to indicate that the value written for the data item `data_1` on the port `sender_1` only depends from the data item `data_1` that is read from the port `receive_1`, and not the value that is read from `receive_2`.
Figure 6.1 shows the first proposed solution. The solution consists in the creation of a stereotype «inout_dependency»that applies to a UML constraint. The dependencies are specified as stereotyped constraints that are added to the runnable specification. Each constraint is connected by dependencies to the dataReadAccess and dataWriteAccess elements that are related by the functional dependency. This is a direct representation of the abstract model and allows for better independency of the additional specification.
However, a more compact implementation is also possible. In this case, the stereotype applies to a dependency (as in Figure 6.2). The dependency is added to the dataWriteAccess element and connects it to all the dataReadAccesses it depends upon.

6.1.2 Input-Output dependencies in UML/SysML

In UML and SysML there is formally no need for any additional stereotype or element. The behavior dependencies can be described using any of the UML/SysML behavioral diagrams (activity or interaction) and/or the associated behavior elements (actions). However, there are several considerations that are worth discussing.
First, in SysML there is no clear way to associate a data item from a Flow port to a behavior specification in a behavior diagram. In Rhapsody, the correspondence between the flow ports and the activity diagram actions is represented by activity parameters with the same name as the flow port name, but this is clearly an informal solution that is not accurate.
The second possible issue with the use of behavior diagrams to define correspondences between write and read actions by using component operations is the need to add a significant number of modeling objects and possibly a new set of diagrams to represent a dependency that could be defined very shortly at the level of the architecture models.
In this case, a solution similar to the one proposed for AUTOSAR can be used, by using dependencies

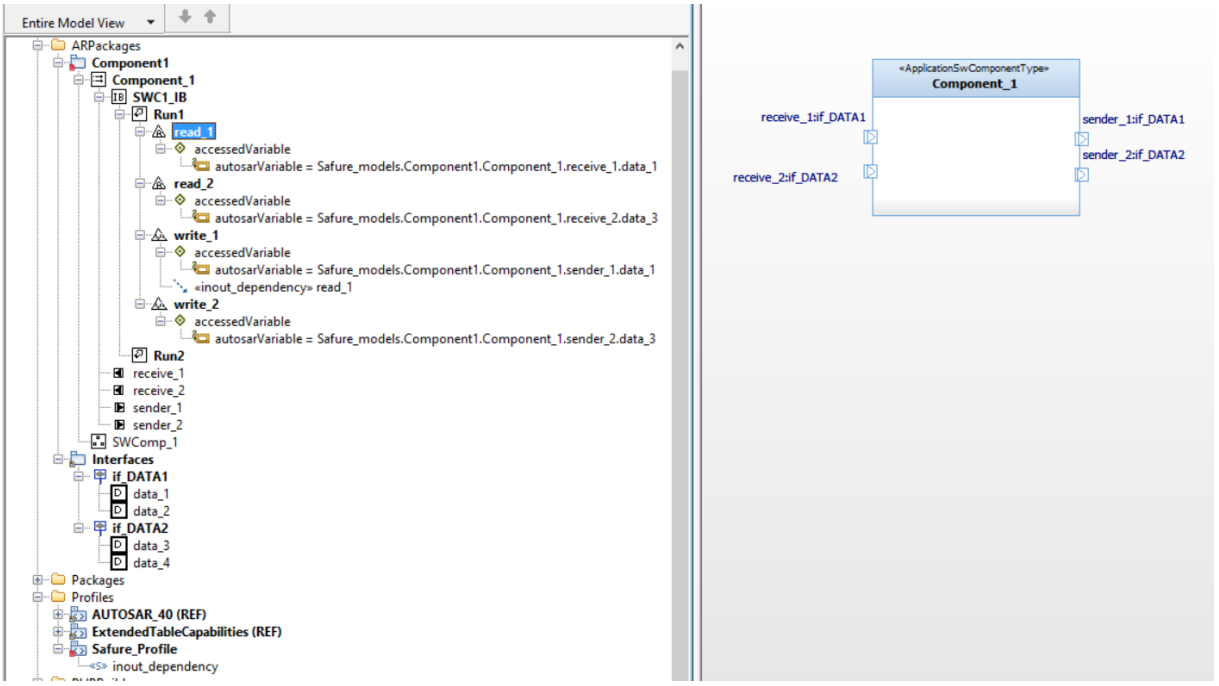

Figure 6.2: Expressing dependencies in AUTOSAR-Rhapsody by means of stereotyped dependencies

connecting operations to flow ports to indicate read and write accesses and constraints or dependencies to indicate input-output dependencies.

6.2 Time

Before introducing the description of the concrete implementation of the time modeling extensions, it is important to state that it is currently very difficult or even impossible to define an actual implementation for the separation of timing concept onto assumptions and assertions, as described in Section 4.3 and illustrated in Figure 4.4. This is because this separation would strongly impact the current structure of the AUTOSAR timing features and any existing UML extension (such as MARTE) by requiring a reworking of almost all of the existing concepts and their organization.
To describe the possible implementation of (some of) the timing extensions in AUTOSAR, we created a simple example outlining two simple configurations for timing constraints at the component level and at the Vfb level (the level describing the integration of components). The first timing example is shown in Figure 6.3. The model contains the specification of a single AUTOSAR component `SWC1`, with a single runnable `Runnable0`, activated by a trigger event.
The timing constraints that apply to the execution of our component model are shown in Figure 6.4, in the section SwcTiming/timing_constraint1. The runnable `Runnable0` is activated by a single trigger with a sporadic activation pattern, and a deadline constraint is enforced on it, as a LatencyTimingContraint `Run1Dline`. Latency constraints require to be applied to a timing chain (`Run1Chain`), that in the case of our example spans from the activation to the termination of the runnable. In practice, Rhapsody also allows a simpler definition, in which the Latency constraint is applied to the Timing event signalling the runnable termination. In the model of Figure 6.4 the deadline constraint is expressed in the standard form, with a specification of a maximum latency.

6.2.1 Deadline criticality specifications

The introduction of the deadline criticality modes and the specification of the system reaction to deadline misses can be introduced by the definition of two additional enumerated types and one

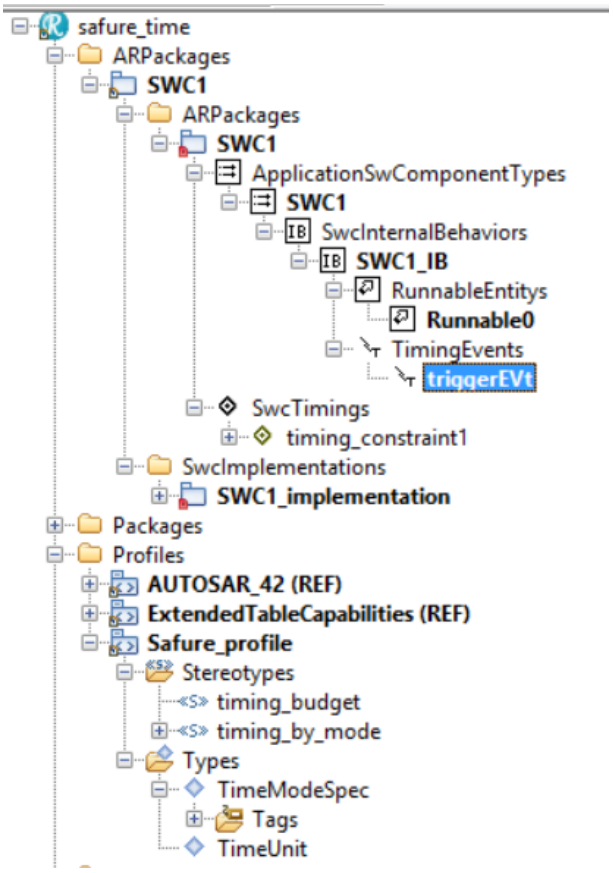

Figure 6.3: An AUTOSAR model example to illustrate the implementation of timing extensions.

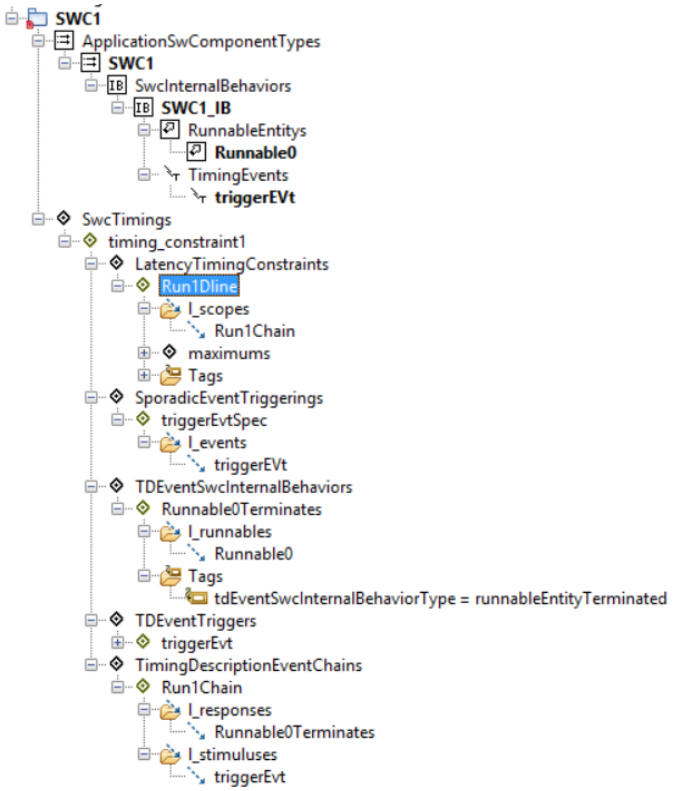

Figure 6.4: An AUTOSAR model example with deadline specifications.

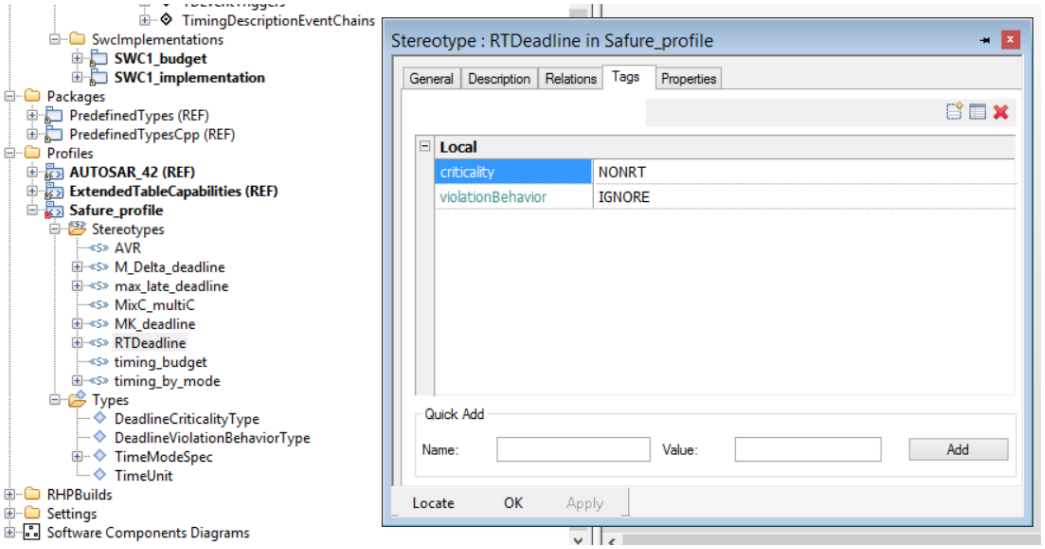

Figure 6.5: The definition of types and stereotypes for deadline criticality and missed deadline behavior.

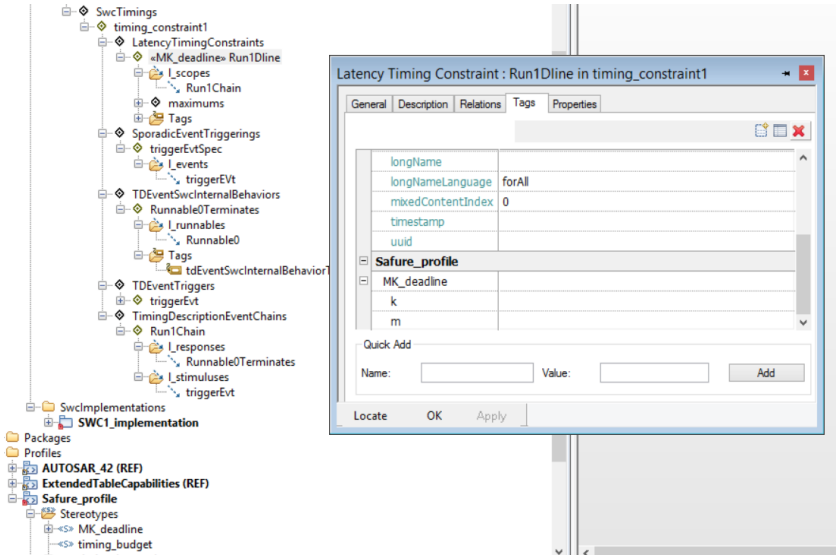

Figure 6.6: An example with the m-k deadline specification.

additional stereotyped concept in our profile. The two types indicate the values for the criticality modes and the reaction)IGNORE or MISS). The stereotype, defined as RTDeadline, as shown in Figure 6.5 applies to constraints (including time constraints) and comments and provides for two tags with the indication of the criticality and the behavior with respect to misses.

6.2.2 Timing overload specifications

The first modeling extension consists in the definition of an m-k deadline model in AUTOSAR that requires that at least m deadlines are not missed in any set of consecutive k activations. This could be simply obtained by defining a stereotype «MK_deadline»that applies to the latency constraint. The stereotype is defined with two tag values, `k` and `m`, both of type integers. An example of its use is in Figure 6.6 showing the specification of the stereotype and its application to the constraint `Run1Dline`. Unfortunately, in Rhapsody the UML element that is specialized for the definition of AUTOSAR latency constraints is a simple comment (not a constraint as would be expected). Hence, in order to be applicable to deadline constraints, the stereotype is specified to be applicable to comment elements. Similarly, stereotypes for the other deadline overload control models are defined, creating the

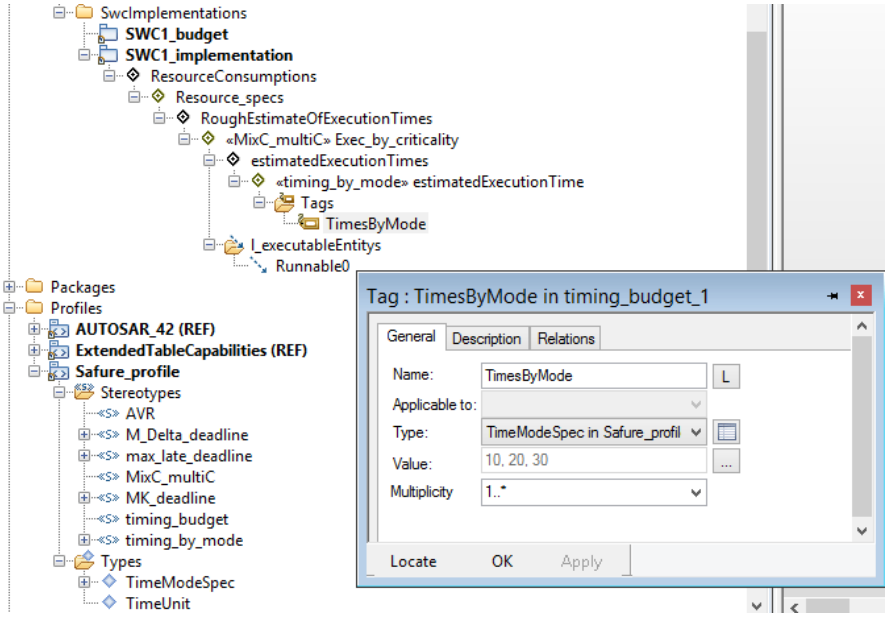

Figure 6.7: An example with multiple execution times for a mixed-critical specification.

«M_Delta_deadline»stereotype (for indicating that at most m instances can be missed in any interval of lenght Δt), with three tags, `m`, of type integer, `delta`, of type integer and `unit` indicating the time unit that is used to express delta. Finally, the «Max_late_deadline» allows to specify the maximum lateness that is allowed with respect to the deadline.

6.2.3 Timing execution specifications

Time execution specifications refer to the possibility of specifying multiple execution time specifications for different modes of execution. This specification (an example in Figure 6.7) requires a dedicated stereotype for associating a set of execution times to a set of mode specifications. The stereotype is defined as «TimeModeSpec». Its application to one of the component runnables requires that a SwcImplementation section is added (Rhapsody does not allow for execution time assumptions or budgets and only allows to define execution times as part of a possible implementation option). Inside the SwcImplementation section, a ResourceConsumption section is created and then, in the example of the Figure 6.7, a RoughEstimateOfExecutionTime element. This entity will define our execution time specification and is stereotyped according to the nature of the mode index that defines the multiplicity of possible execution times. In the case of the figure, a stereotype «MixC_multiC»indicates a definition for multiple criticality levels. Other options are «AVR»or «Multiframe». The execution time estimate is stereotyped as «timing_by_mode»to indicate that it will include multiple definitions for a set of integer values.

6.2.4 Timing budget specifications

The specification of a timing budget is similar to the previous case, with a different timing specification section and a RoughEstimateOfExecutionTime now stereotyped as «timing_budget». An example is shown in Figure 6.8.

6.3 Security

With respect to security, our initial sample implementation is dedicated to the elements identified in Section 4.4 and represented in Figure 4.7. These extensions relate to the possibility of adding a trust specification to a component or a runnable, and a security requirement to a communication at the

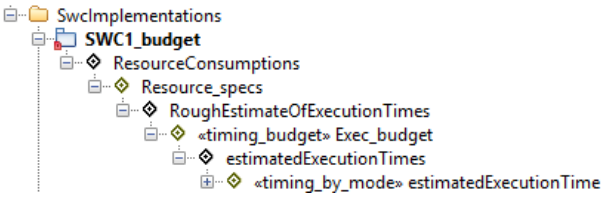

Figure 6.8: AUTOSAR model example with deadline specification

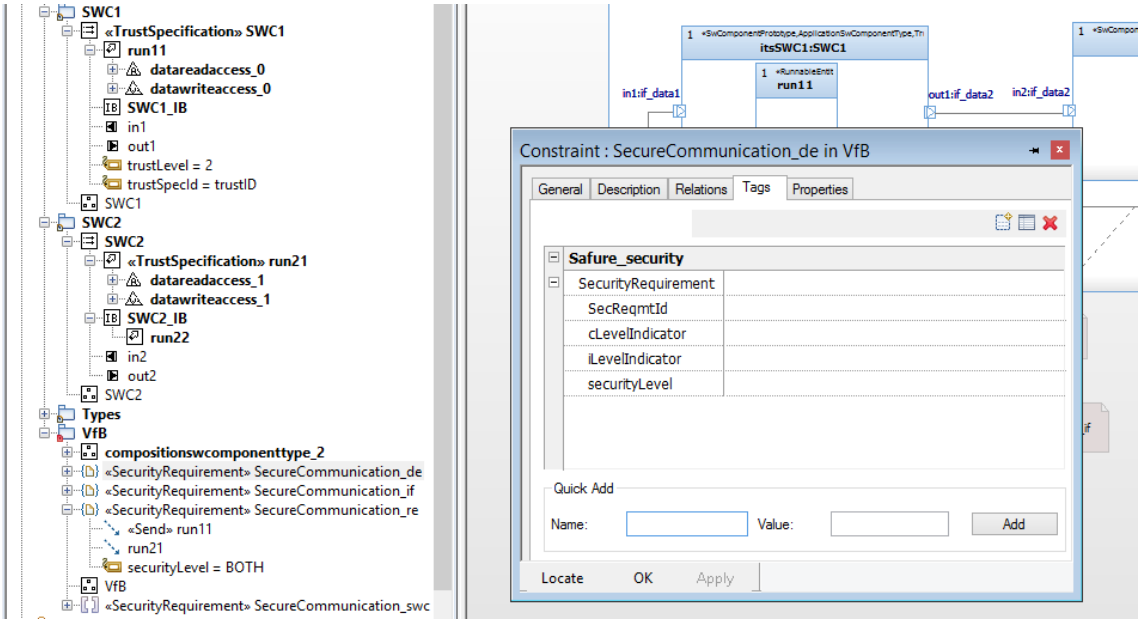

Figure 6.9: Definition of a stereotype for security requirements.

functional level. The security requirement can be applied in many ways. It can apply to an entire data interface (all the elements in it), to a single data item within an interface, to all the communications outgoing from a component, or to an interaction between a sender and a receiver runnable (possibly further qualified by the communication interface or data item).

The implementation in Rhapsody first requires the definition of stereotypes for trust specifications and security requirements. Following the model of Figure 4.7, we defined a stereotype `SecurityRequirement` (as in Figure 6.9), with the corresponding properties. Similarly, a stereotype `TrustSpecification` is defined (Figure 6.10).

The sample model showing the applicability of the proposed extensions is represented in Figure 6.11. It consists of two components with two runnable accessing as reader and writer a pair of sender and receiver ports. The ports are typed by a data interface with a data item.

In our example, a `TrustSpecification` is applied to the component `SWC1` and the runnable `run21` (on the left of Figure 6.9). The security requirements that are added to the communication could be realized in two ways. One way is to stereotype a requirement (for possible application to SysML true requirements); the other is to stereotype a constraint. Requirements or constraints that are stereotyped as `SecurityRequirement` can be added to the communication elements (such as a pair of sender and receiver runnables as in Figure 6.11) using dependencies. In the case of a pair of sender and receiver runnables, the (single) sender needs to be identified by stereotyping the dependency with the sender runnable.

In Figure 6.11, `SecureCommunication_if`, `SecureCommunication_de`, `SecureCommunication_swc` and `SecureCommunication_re` represent the security requirement applied to an entire data interface, to a single data item within an interface, to all the communications outgoing from a component, and to an interaction between a sender and a receiver runnable, respectively.

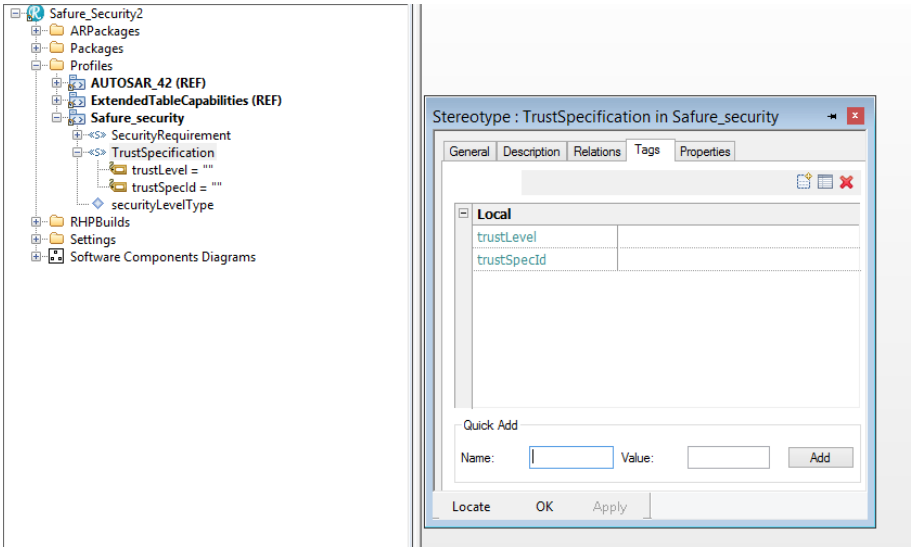

Figure 6.10: Definition of a stereotype for trust specifications.

Secure communication

Following the concept of secure communication expressed in Section 4.4, Figure 4.8 shows an abstract model representing the concept of Receiving Group. An example model on Rhapsody to show the application of the concepts includes five software components, a sender and four receivers. The receiving software components belong to two different receiving groups. A diagram representation is shown in Figure 6.12.

In the figure, SWC1 is the sender software component, SWC2, SWC3, SWC4 and SWC5 are the receivers. SWC2 and SWC3 belong to the same receiving group, SWC4 and SWC5 belong to a different group. In this representation, we assume that each software component is allocated to a different ECU. To provide an implementation of the concept of Receiving Group, we first need to define a new stereotype, as shown in Figure 6.13. The Receiving group stereotype applies to the base concept of constraint and includes two properties: a group ID and a group key.

This new stereotype is then connected with relationships to the SWC in the group and when those are allocated to ECUs will in turn be in a relationship with ECU instances, meaning that the group key will need to be managed (and stored) by those ECUs. Figure 6.14 shows the topology diagram of our sample system. The group constraints have been added to show the group representation.

As already stated, software components are mapped to different ECU (ECU Instances). Each ECU Instance has a CAN Communication Controller, a dedicated hardware device by means of which hosts are sending frames to and receiving frames from the communication medium (the CAN Bus in our case). The CAN Bus is modelled using the CanCluster element; a specialization of the abstract element CommunicationCluster, which is the main element to describe the topological connection of communicating ECUs. A cluster describes the set of all the ECUs, linked by a communication medium of arbitrary topology. The nodes within the cluster share the same communication protocol. A CommunicationCluster aggregates one or more physical channel.

Bus like CAN and LIN have exactly one PhysicalChannel. A physical channel is the transmission medium that is used to send and receive information between communicating ECUs. An ECU is part of a cluster if it contains at least one controller that is connected to at least one channel of the cluster. A physical channel has one or more CommunicationConnector elements, which are references to the ECU Instance to which the physical channel is connected. A physical channel has an arbitrary number of Pdu triggerings. A Pdu triggering describes on which channel the IPdu is transmitted and refers to the iPdu (a secured iPdu in our example) for which the transmission is initiated by the trigger. Figure 6.15 shows the topology diagram implementation on Rhapsody.

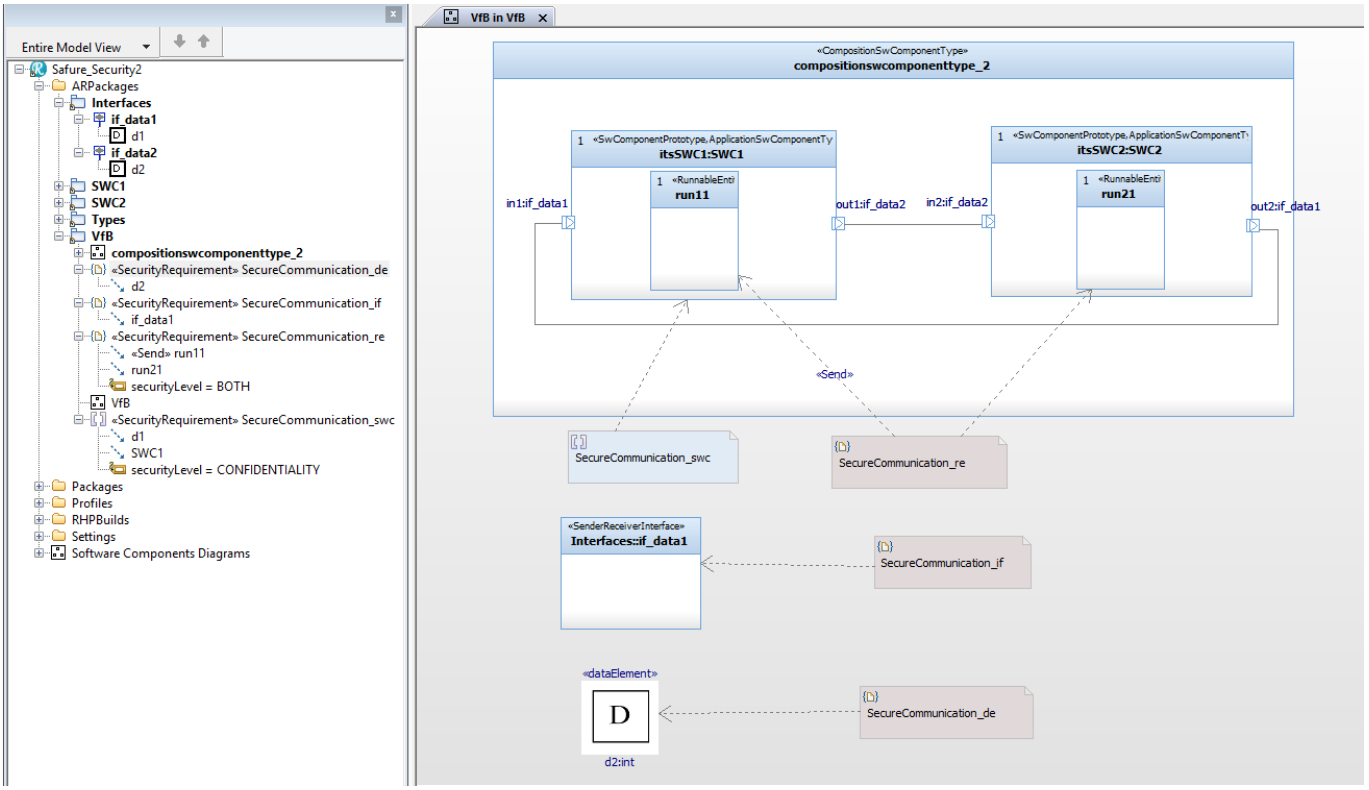

Figure 6.11: AUTOSAR model with security requirements and constraints

6.3.1 Safety

With respect to safety, the implementation is dedicated to the elements identified in Section 4.2 and represented in Figure 4.3. The proposed extension relates to the possibility of adding the concepts of Intentional Fault, Fault, Error, Failure and Hazard, and how they are related to components.
The implementation in Rhapsody requires first the definition of stereotypes for Fault, Intentional Fault, Error, Failure and Hazard. Such definitions, with the corresponding properties, are shown in Figure 6.16. All these stereotypes are realized by extending comments and requirements and are applicable to all design elements (even if those for which an application has meaning are HW components, SW components and runnables. The stereotypes are defined with the corresponding attributes, as specified in the abstract model chapter.
A sample model is shown in Figure 6.17. It consists of two components (SWC1 and SWC2), each of them having a runnable entity inside.
In the picture, Faults and Hazards have been applied to, respectively, comments and requirements. These are then applied to the runnable entity Runnable11 and the software component SWC1.

6.3.2 Patterns

For the patterns identified in section 5 we provide a possible implementation as a reference model with the possible use of stereotypes.

HSM

An implementation on Rhapsody (UML and AUTOSAR) of the HSM architecture pattern introduced in section 5.5 and its metamodel is shown in Figure 5.1. As the HSM is a hardware module, the implementation on Rhapsody is illustrated by means of an ECU Diagram (the AUTOSAR standard diagram for the representation of HW features), and is shown in Figure 6.18.

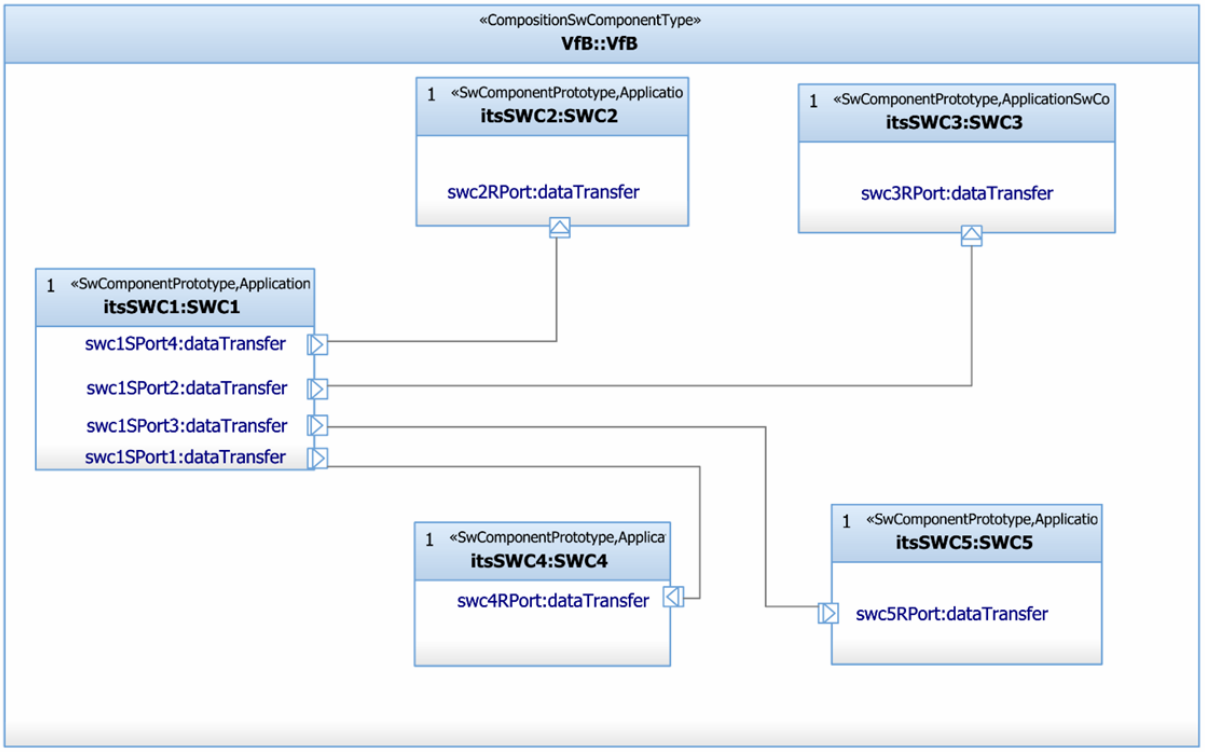

Figure 6.12: An AUTOSAR example for communication security.

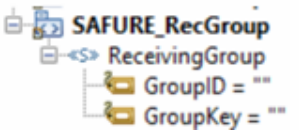

Figure 6.13: Stereotypes for communication security

Both UML and AUTOSAR are extremely vague in the representation of HW elements, and this is easily an issue in the representation of embedded systems as outlined, for example in [112]. UML has standard elements for the representation of computing nodes. AUTOSAR goes a little bit further and allows for the definition of communication interfaces, communication channels and generic HW elements that can be used for the description of any HW feature that is needed, such as memory, pins or any other device. The reason for this lack of features is that both UML and AUTOSAR focus on the description of the allocation of computing units (packages in UML, SWC in AUTOSAR), implemented as software object code on nodes, and possibly the resulting allocation of communication data to networks (in AUTOSAR). Further details on the HW representation have been overlooked until today.
Hence, in our models, all the elements described in the HSM metamodel (as shown in figure 5.1) are implemented by means of Hardware Elements (HwElement). These represent the ability to describe generic hardware elements of any kind (including pins, memories, crossbar switches, arbiters) on an instance level. A HwElement has connections (one or more) with other HwElements (hwElement-Connection), additionally it may have nested hardware elements (nestedElement), representing an association to establish hierarchies of hardware elements. An hardware element can be target of a nested element association only once. The concepts for cypher algorithm and trust level, described in Figure 5.1, have been implemented using two stereotypes, shown in Figure 6.19.
A complete view of the implementation for the HSM is shown in Figure 6.20.

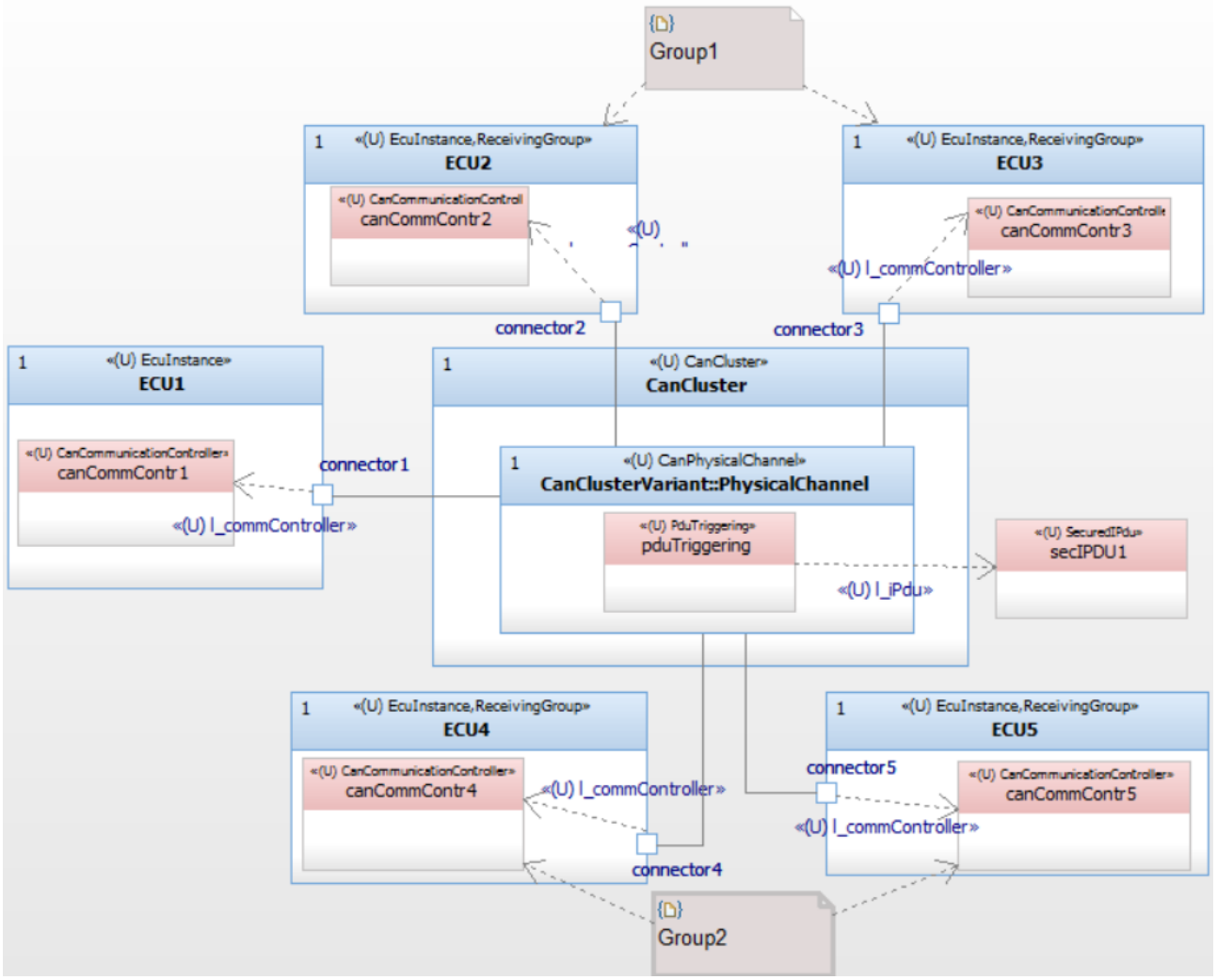

Figure 6.14: An AUTOSAR example of a communication system with a network configuration.

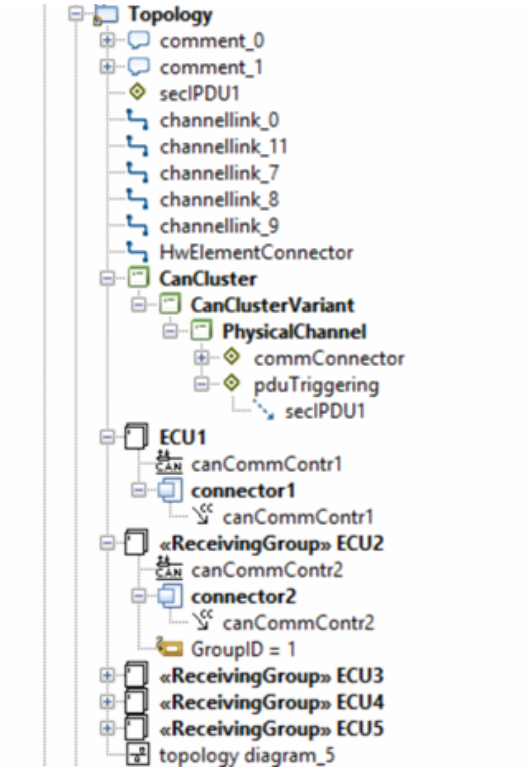

Figure 6.15: The model for the implementiation of the communication system.

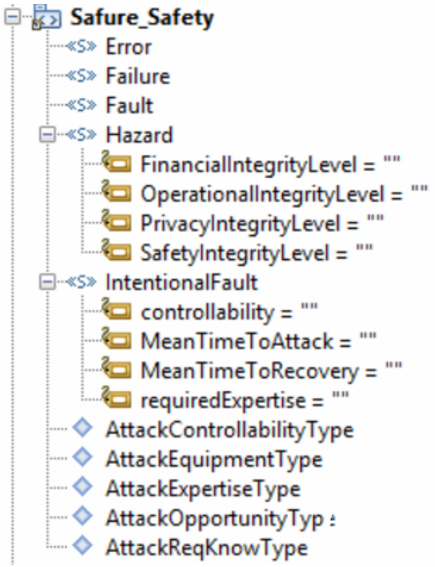

Figure 6.16: Rhapsody stereotypes for intentional faults.

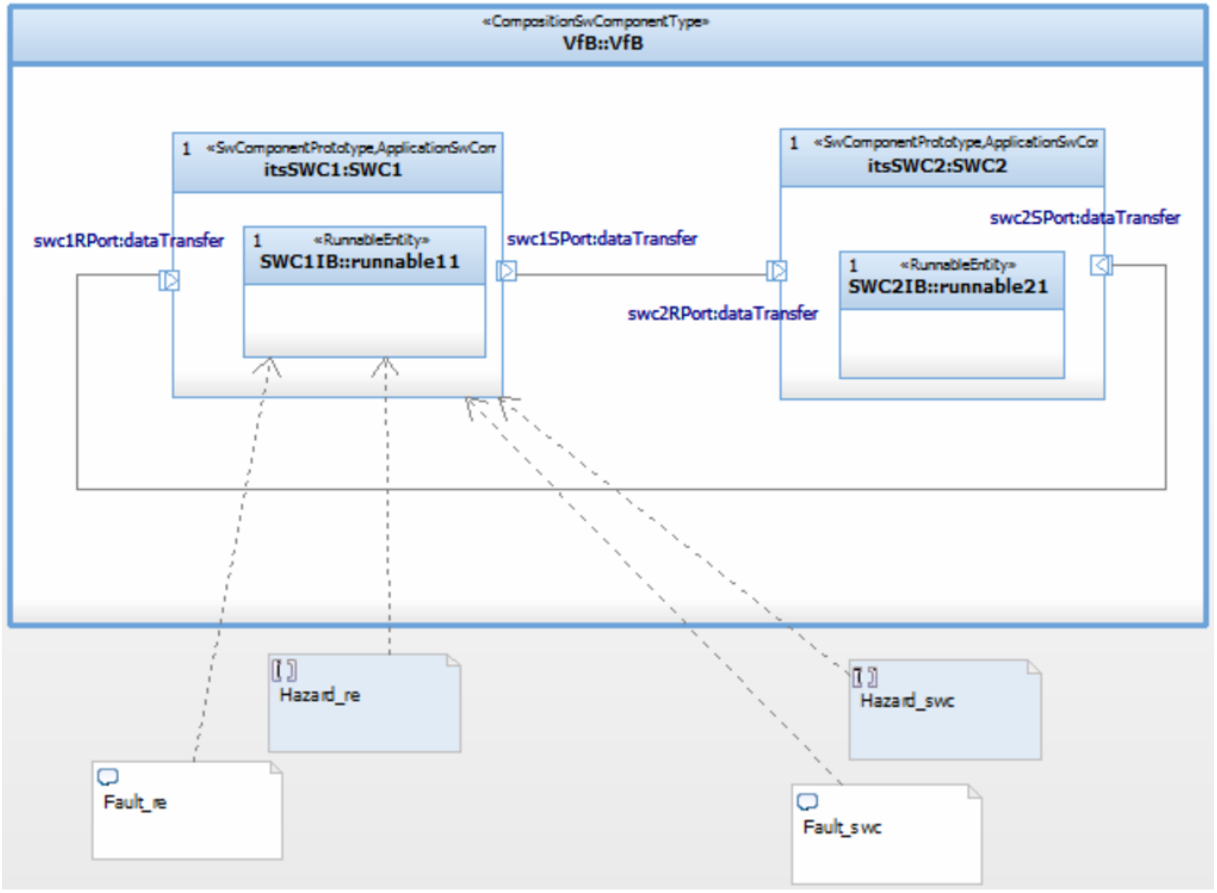

Figure 6.17: AUTOSAR model with security requirements and constraints

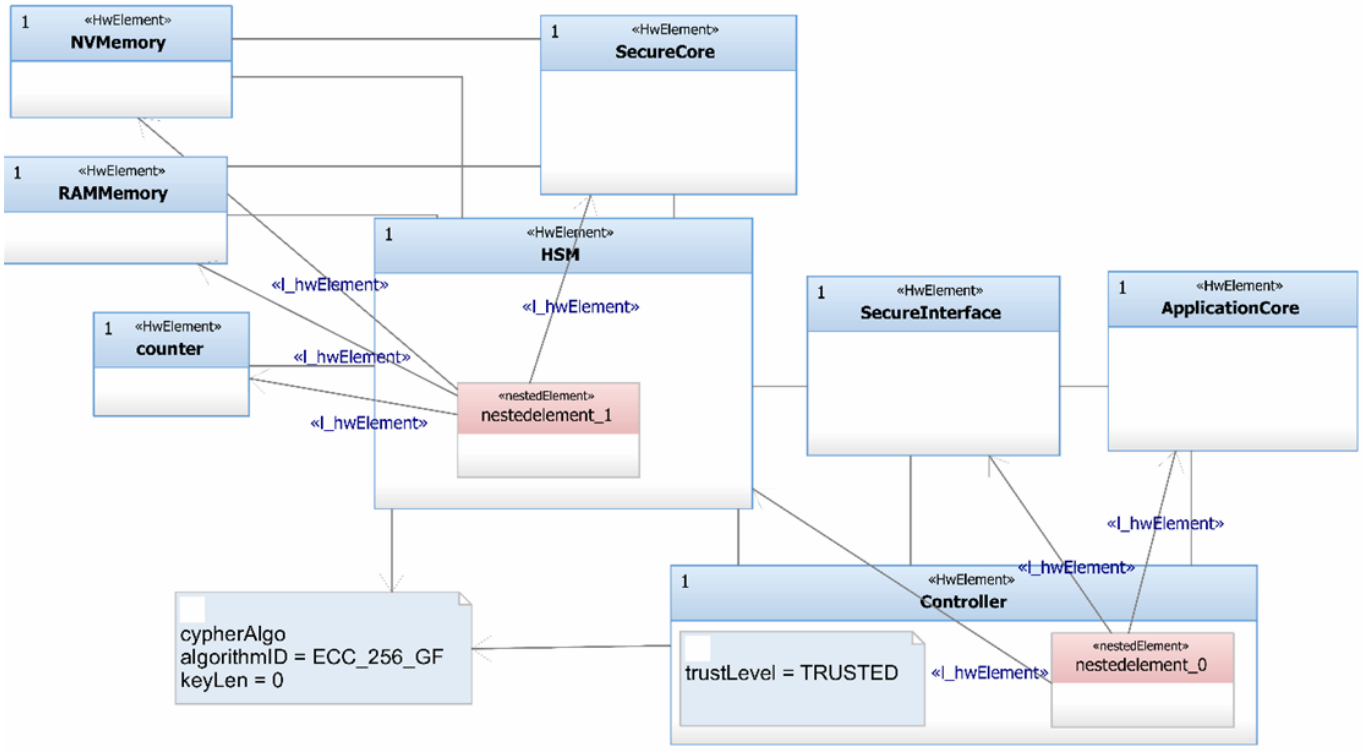

Figure 6.18: AUTOSAR model for an HSM pattern

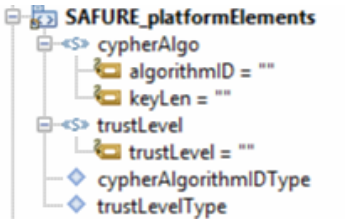

Figure 6.19: Stereotypes for the definition of the HSM pattern.

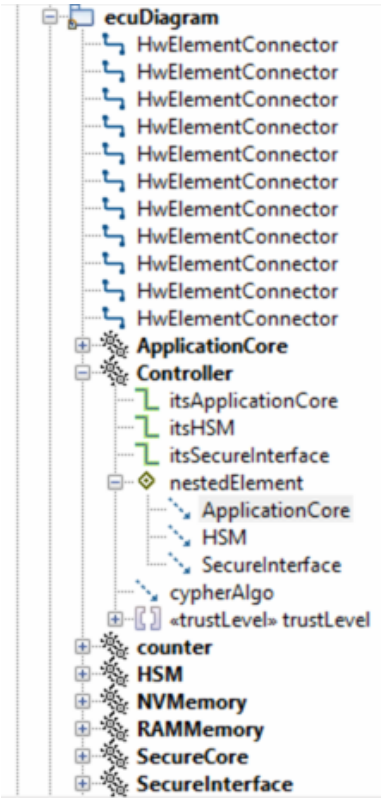

Figure 6.20: AUTOSAR model with HSM elements

Chapter 7

Use of Modeling Extensions for the Analysis and Synthesis of Safety and Security Properties and Mechanisms

The purpose of this chapter is to provide a general description of how the models presented in the previous chapters can be used for the purpose of the analysis of the properties of mixed-critical systems and the synthesis of components that preserve some of their required properties by construction. The chapter contains an initial section that details the overall methodology and some possible implementation options. The next two sections are dedicated to the analysis and synthesis methods respectively. As for the other chapters, each section is further divided in subsections addressing the separate topics of time, safety and security.

7.1 Methodology for Analysis and Synthesis

The proposed methodology for the analysis and synthesis of the Safure models, including the modeling extensions proposed in this document is consistent with the typical analysis and synthesis flow recommended in AUTOSAR and UML.
In both cases, the modeling language is defined in such a way that the modeling or design tool can be effectively isolated from the analysis tools (often very specialized and restricted to a very specific class of users). This is because the types of analysis that can possibly be performed on models is hardly predictable in advance and analysis and synthesis tools may require specific competences that are not necessarily under the control of a company developing a UML or AUTOSAR authoring tool. Also, this separation of concerns help fostering more competition in the market, letting tool vendors compete for each technology niche or opportunity.
Of course, for this to work, a standard interchange format or operation connection needs to be made available by the tool vendors or even the modeling language standard itself. In the case of UML, the OMG has developed the standard XMI interchange format. XMI is an XML language implementation, and most UML tools allow exporting the model in XMI. Unfortunately, XMI allows for too many variation points in its structure, which makes the interoperability of different designer tools quite hard. This is not a problem for a dedicated connection with an analysis tool, upon condition that the analysis tool is capable to adapt to the different XMI dialects implemented by tools.
AUTOSAR interoperability is much easier since AUTOSAR has its own XML interchange format, called ARXML with a much more controlled structure, ensuring an unambiguous interpretation of the modeling elements and their attributes. However, because of the tight control on the ARXML format, any language extensions defined, for example, in Rhapsody, are not exported in the ARXML output and need to be obtained in some other way.
Finally, most modeling tools also provide an API for querying the model structure using a standard programming language (such as Java or C++) and a connection to some server port. In the case of this

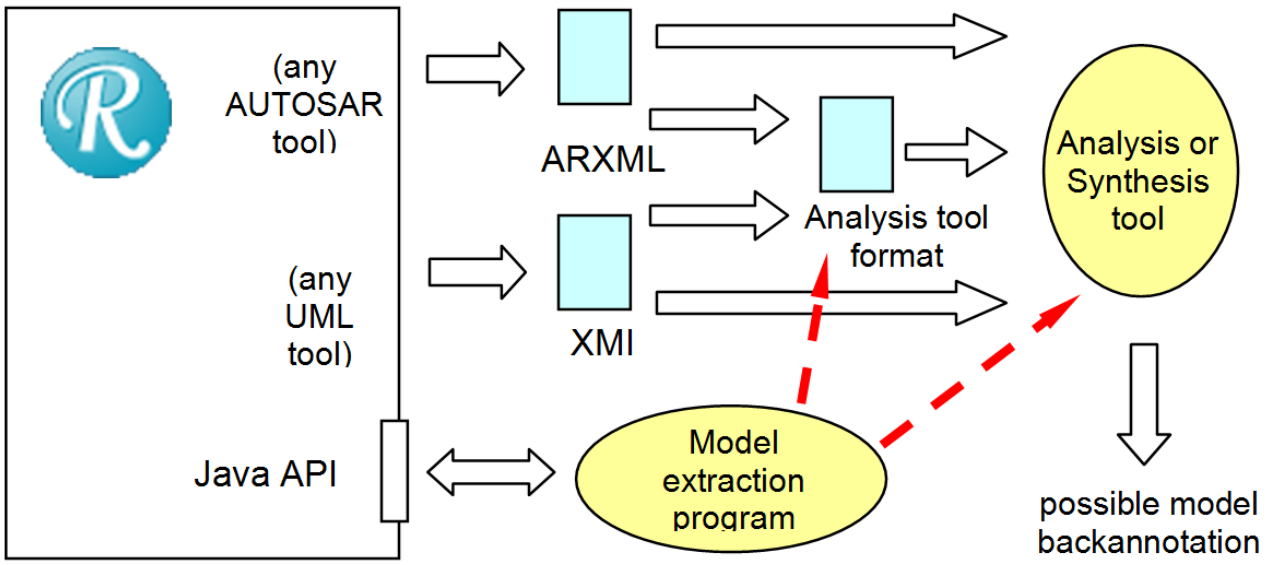

Figure 7.1: The general steps and output formats for the analysis of a (AUTOSAR or UML) model or its processing for synthesis.

project we outline the possible connection options when using Rhapsody, with a set of possible variation points in the processing flow for several alternatives. However, what is described for Rhapsody can be easily extended to other modeling tools that offer similar capabilities.
As shown in Figure 7.1, the designer tool (in our case Rhapsody) allows for multiple ways to obtain model information (ARXML, XMI or through a program-level API). The analysis and synthesis tool can read the information directly from those formats, or through a plug-in (external or part of the analysis tool itself) that reads the model information and stores in some intermediated format. Another option is when the analysis tool has its own format (possibly different from XMI and ARXML) and a conversion (typically implemented using a stylesheet or XML parsing) is necessary to bridge between the two formats.
At the end of the analysis process, the results can be returned to the models through a process of backannotation (not shown in the Figure).

7.2 Analysis of AUTOSAR or UML/SysML models

The following steps introduce some analysis options from Rhapsody models, as representative of UML or AUTOSAR models.

7.2.1 Time

We provide two examples of timing analysis. One very simple example at the node level, using the extensions that are provided to define m-k deadlines; the other for the analysis of Ethernet frames.

Analysis of m-k task deadlines

The first example of timing analysis is the simple extraction of a model with m-k deadline constraints. The model is exported in ARXML and XMI. the two files and parsed and the final task configuration is analyzed using the m-k analysis algorithm described as part of WP3.

Model Transformation to support AUTOSAR SOME/IP Socket Adapter Behavior in Worst-Case Timing Analysis of Ethernet

Symtavision is developing a timing and buffer analysis for Ethernet in WP3, which will be integrated into the timing analysis tool SymTA/S in WP5. This analysis uses a generic modeling of Ethernet systems using the following modeling elements:

- Ethernet Message, consisting of one or more (e.g. for UDP messages) Ethernet frame
  - with a Period, Jitter, minimum distance (PJd) event model for "internal" triggering
- "External" Triggering of Ethernet frames by other model elements (e.g. tasks)
  - This will not be used (because multiple trigger sources are presently not supported by the worst-case analysis in SymTA/S)
- Payload Data Unit (PDU) with payload size fields, modeling the actual payload
  - PDUs are mapped to Ethernet frames
- Dynamic frame assembly of PDUs to Ethernet frames
  - Modeled via Polling at PDU, activated via GenericElements (which are configured using the PDU cyclic timings)

In automotive networks, AUTOSAR SOME/IP is often used in order to package Ethernet messages. Next to service discovery mechanisms (for setting up the communication dynamically), it offers capabilities to package individual PDUs into Ethernet frames based on PDU-specific configuration. This way, SOME/IP controls the packaging (i.e. which PDUs are grouped together) and triggering (i.e. when is a group being sent out) of Ethernet frames. In order to analyze such systems in SymTA/S, we perform a model transformation that maps the "complex" packaging/trigger behaviors supported by SOME/IP to the "generic" modeling elements of SymTA/S. This transformation should be conservative, i.e. never loose triggerings. For this, the effective Trigger-Period (ETP) is calculated for every PDU using SOME/IP properties TRIGGER_ALWAYS or PDU with TRIGGER_NEVER and timeout. The ETP is the period, by which a PDU is effectively transmitted.

- ETP of PDU with TRIGGER_ALWAYS = CyclicTiming
- ETP of PDU with TRIGGER_NEVER = $\left\lceil \frac{Timeout}{CyclicTime} \right\rceil CyclicTime$

Timeout at frame-level (SocketConnection) is considered as timeout at every PDU without an individual timeout. With this, PDUs are mapped into separate Ethernet frame based on the following rules:

- Rule1: PDU has TRIGGER_ALWAYS; in this case a new frame shall be created.
- Rule2: For every PDU with TRIGGER_NEVER and a timeout smaller than or equal to the min. ETP; a new frame is created.
- Rule3: Every PDU with TRIGGER_NEVER (with and without timeout) is duplicated and attached to every created Ethernet frame
- Rule4: If CyclicTiming < Timeout of a PDU, the PDU may be contained more than once in a frame (PDU without Timeout has an infinite Timeout).
  - Duplication occurs if no ETP $\leq$ CyclicTiming exists.
  - Duplication count = $\left\lceil \frac{\min(EffectiveTriggerPeriod)}{CyclicTime} \right\rceil$

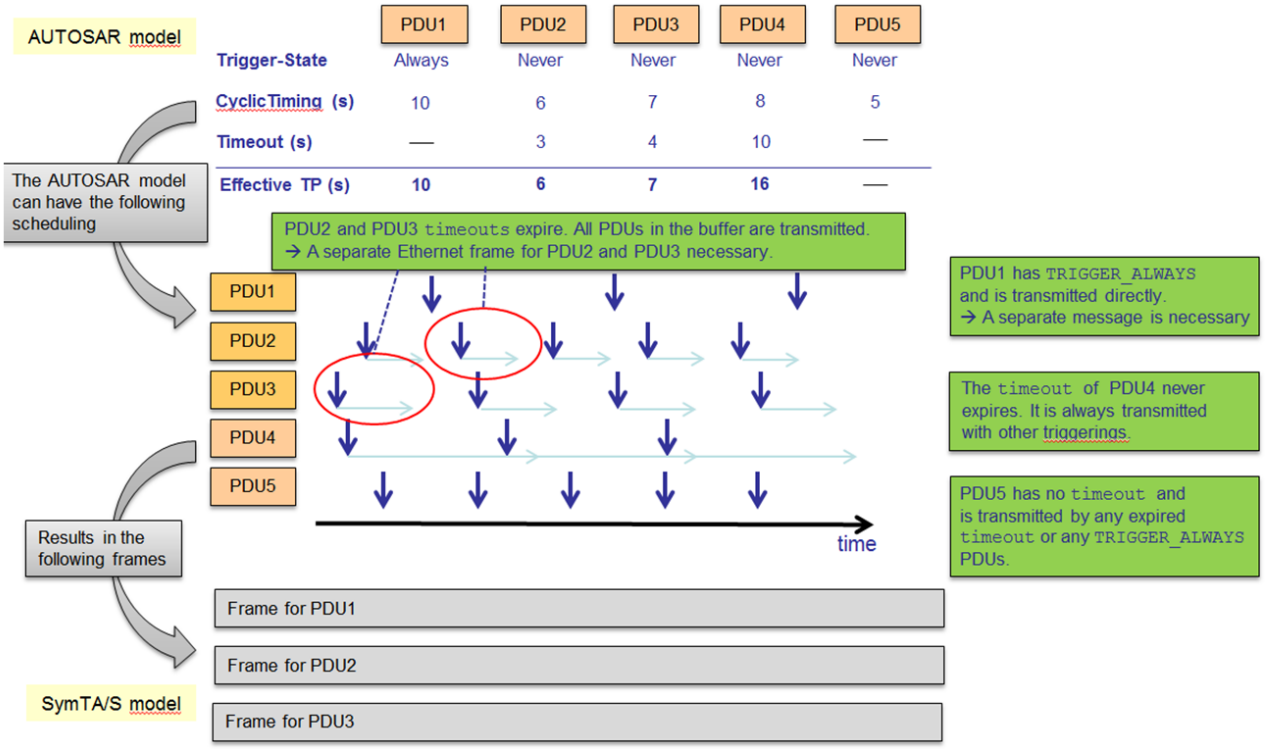

Figure 7.2: Example AUTOSAR to SymTA/S Model Transformation.

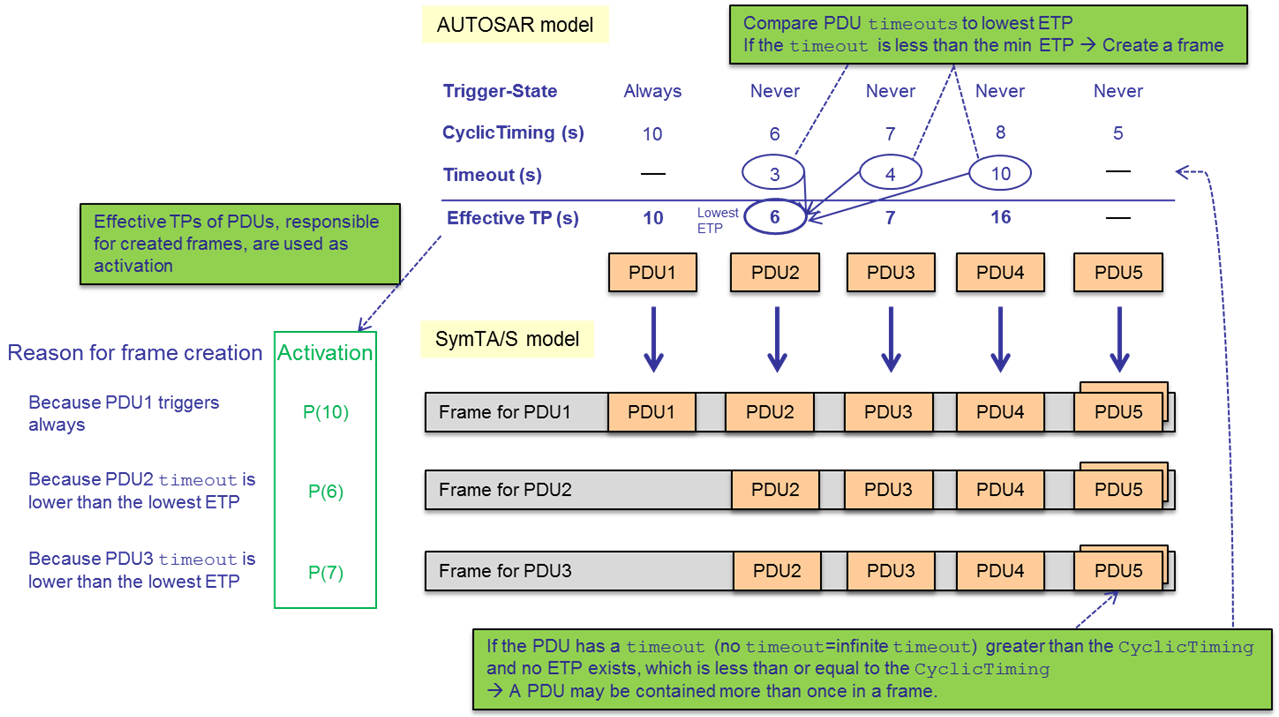

Figure 7.3: Model Transformation Effects.

Figure 7.2 shows and explains an example transformation, which transforms 5 SOME/IP PDUs into 3 Ethernet Frames. Each of the frames contains multiple PDUs. It can be seen that several PDUs appear in multiple frames, because they could appear in each of these frames due to the dynamic packaging of SOME/IP.

Figure 7.3 shows the effects of the model transformation. It illustrates that there are separate Frames necessary for PDU1, PDU2, PDU3, as they could be triggered individually from each other. The timeout of PDU4 is so large that it never triggers individually, but is included in one of the other frames. PDU5 has no timeout and hence is included in the other frames. All frames contain PDU2-5, because they could contain them. Only the frame triggered by PDU1 contains PDU1, as it cannot be transferred in the other frames. This duplication of PDUs into frames does not lead to an increased load in system distribution analysis, because Polling is used to only include these PDUs when they have actually been activated. For the worst-case analysis, however, this is a pessimistic overestimation due to lack of polling support in worst-case analysis.

7.2.2 Safety and Security

Analysis of data dependencies from AUTOSAR models for security and safety

The standard AUTOSAR definition of security annotations does not consider the problem of dependencies between data that traverse components and communication links. For example, if integrity is requested for data sent on the communication link input to the Brake actuation sub-system of Figure 7.4, all the communication links traversed from the sensors that originated the data to the Brake sub-system must be protected; otherwise, the security constraint cannot be satisfied. We introduce the concept of *data secure flow*. Data secure flow is verified if, for every possible execution, data sent on a communication always satisfy the security annotation written in the model.
Dependencies between data in an AUTOSAR model, can be studied using approaches for checking secure information flow in programs [53]. In particular, we use an approach based on abstract interpretation [49], a static analysis technique for the automatic extraction of information about the possible executions of computer programs. Abstract interpretation has been used in [31] to analyse secure information flow in a simple imperative language. We extend the technique to cover the analysis of AUTOSAR models. The main points of the approach are:

- an abstract interpreter executes the functional units of software components on abstract domains that abstract from real values and consider only data dependency levels. A fixpoint iterative analysis computes the dependency between data written/read at ports of the software components.
- since the analysis computes all possible dependencies for any real execution of the functional units, the lowest security level of data sent on a communication is counted.
- data secure flow property is satisfied if the security level computed by the analysis for data sent on a communication always satisfy the security annotations in the model.

The complexity of the analysis can be reduced by using static program analysis techniques [115], which analyze the source code without executing the program. Static analysis techniques are applied for enforcing information flow security in programs in several works, the reader can refer to [133] for a survey. One of the advantages of our approach is that, being based on abstract interpretation, the analysis can be fully automated. Moreover, the analysis scales up, since AUTOSAR software components are analysed separately.
Figure 7.4 shows an example of an AUTOSAR annotated model for a simplified active safety application that makes use of information coming from sensory input devices, such as lidars, radars and cameras, and from the GPS system in order to sense the surrounding environment and detect the roadmarks and objects (vehicles, pedestrians) on and around the street. Position information coming from the GPS, together with object and road position information coming from sensors are forwarded to several navigation and active safety functions, including Path planning, Lane keeping and Lane Departure warning. These functions, in turn, produce commands for the actuation systems (steering, throttle and brakes).

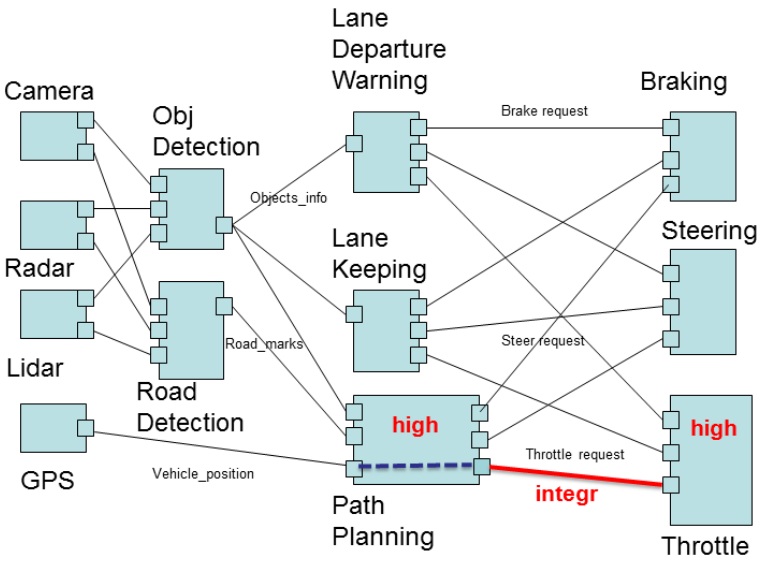

Figure 7.4: An example of security annotated model .

For semplicity, we assume that 1) Throttle and PathPlanning software components are assigned high trust level; 2) the other components are assigned a low level. Moreover, we assume that 1) The Throttle request signal is annotated with data integrity security requirement; 2) the other communication links have no security requirements.
Finally, we assume send/receive data communications between components; and dependencies between data read/written at ports of PathPlanning are shown as dotted lines internal to the component in the figure.
We introduce the following definitions:

- *Data security* Every signal is assigned a pair $\langle$`trust level`, `security requirement`$\rangle$ that characterizes its degree of security. As data flow through the components and the communication links, its *data security* is updated.
- *Data secure flow property.* When the system is in operation, the *data security* of a data value sent on a link must have a trust level not lower than the level of the receiver component and security requirement not lower than the requirement of the link, both defined at design time through the security annotations in the AUTOSAR model.

With reference to Figure 7.4, the trust level of the data received by Throttle component must be high. Throttle receives data directly by PathPlanning, a high trust level component, which in turn, receives data GPS, which has a low trust level. Therefore, the *Data secure flow* requirement is violated. Assume GPS has a high trust level. The data sent on the Throttle request link must satisfy integrity requirements. The dependency between the data read/written at ports in PathPlanning, makes the property false, because such data depends on the Vehicle_position signal, which does not have any security requirement.
The problem is that security annotations are local to single components and links, and the *data secure flow* property may or may not be satisfied due to data dependencies in the model.

We propose a static analysis approach to check the *data secure flow* property in security annotated models. For the purpose of the analysis, we define an ordering between security values with the meaning that if security value s precedes security value s', then s is in "lower in security degree" to s'. Fundamental to the analysis are the structure of the system model (components, runnables, ports and links), the ordering relation mentioned above, and the dependencies between data in the model.

Data dependencies in an AUTOSAR model The basic idea is modelling ports as variables, and modelling runnables as functions in the programming language. In particular,

- for send-receive data communications, reading a data item from some port is equivalent to reading a variable; writing a data item to some port is equivalent to writing a variable.
- for client-server communications, the client request is equivalent to a function call, that corresponds to the invocation of the runnable implementing the requested service.
- for updates of variables that trigger the execution of runnables, the write of the variables corresponds to the invocation of the functions implementing the runnables. If a variable corresponds to a port of a send/receive data communication, a write of the variable may trigger a runnable local to the sender SWC or a runnable at the receiver SWC.

The analysis we present is based on an abstract interpretation technique [49]:

- the standard operational semantics of the programming language is enhanced to include information useful for the analysis.
- abstract domains are identified and abstract semantics rules are defined that execute the program on abstract domains.
- the abstract rules compute the flow of information in the program

The rules take into account explicit and implicit flow of information. Let v_{pi} and v_{pj} be variables correspondent to p_i and p_j ports.
An example of explicit information flow is:

$$x := v_{pi}; \quad v_{pj} := x + 3;$$

An example of implicit information flow is:

$$\texttt{if } (v_{pi} == -1) \texttt{ then } v_{pj} := 0 \texttt{ else } v_{pj} := 1$$

In both the examples above, the value of variable v_{pj} depends on the value of variable v_{pi}. Data written to port p_j depends on data read at port p_i.
For functions, information flows through function parameters and return. An example of explicit information flow is:

$$x := v_{pi}; \qquad f(x);$$

The call to a function without parameters and return, can be the cause of an information flow, in case of implicit flow:

$$\texttt{if } (v_{pi} == -1) \texttt{ then } f();$$

Function f is invoked depending on the value read at port p_i.

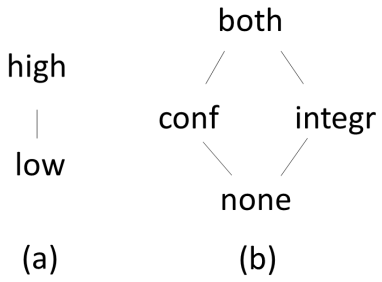

Figure 7.5: Order relation on (a) trust levels (b) security requirements

Data secure flow property verification Given an AUTOSAR model, we use the following notations and definitions:

- $C = \{c_1, c_2, \cdots, c_k\}$ is the set of SWCs.
- $P = \{p_1, \cdots, p_n\}$ is the set of ports of the SWCs.
- $L = \{l_1, \cdots, l_m\}$ is the set of links. A link denotes a connection between two ports. The link $l = (p_i, p_j)$ connects the port p_i to the port p_j, with p_i output port of the sender SWC and p_j input port of the receiver SWC.
- `trustlevel`(c) is the trust level assigned to the software component c.
- `securityrequirement`(l) is the security requirement assigned to link l.

In addition we use the following definitions and functions:

- Given a port p, $cmp(p)$ is the component to which the port belongs.
- Given a port p, $Deps(p)$ is the set of ports on which the data written at port p depends.

Definition 1 (Trust level)**.** *Let $A = \{low, high\}$ be the set of trust levels, ordered by $low \sqsubset high$, where $\sqsubset$ is the lower between levels, see Figure 7.5 (a). Let* glb *denote the greatest lower bound, and* lub *the least upper bound between levels: it is $glb(low, high) = low$ and $lub(low, high) = high$.*

Definition 2 (Security requirement)**.** *Let $B = \{conf, integr, both, none\}$ be the set of security requirements of links, partially ordered by $\sqsubset$, as shown in Figure 7.5 (b). Let* glb *denote the greatest lower bound, and* lub *the least upper bound between levels. $(B, \sqsubset)$ is a lattice (i.e., every pair of elements of B has both a greatest lower bound and a least upper bound). For example, $glb(integr, conf) = none$; $lub(integr, conf) = both$*

Note that *conf* and *integr* are not ordered, because one is not "lower in security degree" to the other.

The method In the analysis, we compute the lowest trust level and the lowest security requirement of data sent on a link l ($\langle$`trust level`, `security requirement`$\rangle$), with the algorithm shown in Figure 7.6. The algorithm records in δ_l and μ_l such levels ($\langle \delta_l, \mu_l \rangle$).
Assume $l = (p_i, p_j)$. The data sent to the link are data written at port p_i.
First the algorithm sets δ_l equal to the greatest trust level and μ_l equal to the greatest security requirement. Then for each port p on which data sent on the link l depends ($p \in Deps(p_i)$) , δ_l is

Given a link $l = (p_i, p_j) \in L$,
1. $\langle \delta_l, \mu_l \rangle = \langle high, both \rangle$
2. $\forall p \in Deps(p_i)$
   $\delta_l = glb(\delta_l, \texttt{trustlevel}(cmp(p)))$
3. $\forall l' = (q, q'), \mid q, q' \in Deps(p)$
   $\mu_l = glb(\mu_l, \texttt{securityrequirement}(l'))$

Figure 7.6: Algorithm for data security of link l.

updated to consider the trust level of the SWC to which the port p belongs: the trust level δ_l is set equal to the greatest lower bound between the current value and the trust level of the SWC to which port p belongs. Finally, for each link l' in the model traversed by data sent on link l (source and destination ports of l' belong to $Deps(p_i)$), μ_l is updated: the security requirement μ_l is set equal to the greatest lower bound between the current value and the security requirement of the link l'. Note that, at each step δ_l can only be downgraded; analogously, μ_l can only be downgraded.
An AUTOSAR model satisfies data secure flow if for each communication link, 1) the trust level of destination component of the link is not greater than the trust lowest trust level of data sent on the link, and 2) the security requirement of the communication link is not greater than the lowest security requirement of data sent on the link.

Definition 3 (Data secure flow property)**.** *Given an AUTOSAR model with security annotations, the model satisfies the data secure flow property if, for each link $l = (p_i, p_j) \in L$, with $\langle \delta_l, \mu_l \rangle$ the data security of data sent at l:*

$$\texttt{trustlevel}(cmp(p_j)) \sqsubseteq \delta_l$$
$$\wedge\ \texttt{securityrequirement}(l) \sqsubseteq \mu_l$$

Dependencies between ports of an AUTOSAR model A port p_j does not depend on port p_i if data sent at p_j are independent from data received at p_i.
We formally define port dependencies as follows.

Definition 4 (Port dependencies)**.** *Let us consider an AUTOSAR model. A port p_j does not depend on the port p_i if: for each pair of values v_1, v_2 at p_i, with $v_1 \neq v_2$, it is: $p_j(p_1, \cdots, p_{i-1}, v_1, in_{i+1}, \cdots, p_n) = p_j(p_1, \cdots, p_{i-1}, v_2, in_{i+1}, \cdots, p_n)$ for each possible execution, where $p_j(p_1, \cdots, p_{i-1}, x, p_{i+1}, \cdots, p_n)$ is the value written on port p_j when x is read from input port p_i .*

In the analysis, we define an AUTOSAR model as a tuple $A = (R, Var, C, \Theta)$, that consists of a set R of runnables, a set Var of variables, a set C of connections (links), a set Θ of levels. In particular

- $Var = VP \cup VIR \cup VG$ is a set that consists of the set VP of port variables, the set VIR of inter-runnable variables, and the set VG of global variables of SWCs.
- R is the set of all runnables. A runnable $r \in R$ is a function $r : Var \rightarrow Var$.
- $C = \{(src, dst) \mid src \subseteq VP \wedge dst \subseteq VP\}$ is a set of connections between port variables (src is the source and dst is the destination port of the connection).
- $\Theta = \{\theta_1, \cdots, \theta_n\}$ is the set of levels, one level for each port. The level of port p_i is called θ_i.

Data dependency levels We assume the following set $\Theta = \{\theta_1, \theta_2, \cdots, \theta_n\}$ of dependency levels, one for each port. We consider the powerset $\Sigma = 2^{\Theta}$, i.e. the set of all subset of Θ, ordered by subset inclusion. $(\Sigma, \subseteq)$ is a complete lattice (every pair of elements of Σ has both a greatest lower bound, *glb*, and a least upper bound, *lub*). The *lub* is given by the union ($\cup$) and the *glb* is given by the intersection of subsets ($\cap$). Given $X \subseteq Y$, $X \cup Y = Y$ and $X \cap Y = X$. The analysis operates over

levels in Σ. The singleton set $\{\theta_i\}$ denotes a dependency from input port p_i. The set $\{\theta_i, \theta_j\}$ denotes dependency on both input ports p_i and p_j. The minimum of Σ is the empty set.

Analysis of a runnable In the following we describe the basic concepts of the analysis.
A program is a sequence q of commands. Let m be a memory that contains all the variables accessed by the program.
The execution of a program is a transition system obtained by executing q starting from the initial memory, by applying the rules of the operational semantics of the language. As an example, the rule for a simple expression consisting of variable `x` is:

$$\textbf{Expr}_{var} \quad \frac{}{\langle x, m\rangle \longrightarrow_{expr} m(x)}$$

The semantics of the expression x is the value of x in memory m.
The rule for the assignment `x := e` is the following, where e is an expression, and $m[k/x]$ the memory m, where the variable x is assigned the new value k:

$$\textbf{Ass} \quad \frac{\langle e, m\rangle \longrightarrow_{expr} k}{\langle x{:=}e, m\rangle \longrightarrow m[k/x]}$$

If k is the evaluation of the expression e in memory m, the semantics of $x{:=}e$ changes the memory by assigning value k to variable x.
The operational semantics of the language is extended to convey the set of data dependency levels during the execution.

- Each value is annotated with the set of dependencies (both implicit and explicit);
  Data become pairs (k, τ), where k is the value and τ is the dependency level.
- and each command is executed under an environment σ that represents the *lub* of the dependencies of the open implicit flows.
  The notation $(x{:=}e)^{\sigma}$, represents the execution of the assignment under an environment σ.

For example, the dependencies of a variable expression, is given by le *lub* between the level of the data in the variable and the level of the environment in which the command is executed.
The previously introduced rules become the following, where M is the memory defined on extended values:

$$\textbf{Expr}_{var} \quad \frac{M(x) = (k, \tau)}{\langle x^{\sigma}, M\rangle \longrightarrow_{expr} (k, \sigma \sqcup \tau)}$$

$$\textbf{Ass} \quad \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} v}{\langle (x{:=}e)^{\sigma}, M\rangle \longrightarrow M[v/x]}$$

The abstract semantics, abstracts from actual values and maintains only annotations on data dependencies. let $M^{\sharp}$ be the abstract memory. The abstract semantics for the previous rules is following:

$$\textbf{Expr}_{var} \quad \frac{M^{\sharp}(x) = \tau}{\langle x^{\sigma}, M^{\sharp}\rangle \longrightarrow_{expr} lub(\sigma, \tau)}$$

$$\textbf{Ass} \quad \frac{\langle e^{\sigma}, M^{\sharp}\rangle \longrightarrow_{expr} \tau, \sigma' = lub(\sigma, \tau)}{\langle (x{:=}e)^{\sigma}, M^{\sharp}\rangle \longrightarrow M[\sigma'/x]}$$

A program is executed on the abstract domain starting from the abstract initial memory, and applying the abstract rules. We note that all branches of conditional/iterative instructions are always executed, due to the loss of real data in the abstract semantics.
In previous work [31], it is proved that the transition system built by executing q with the enhanced semantics is the same as the transition system built with the standard semantics, for a simple high level

language. We extended the semantics rules to include indirections, structures, arrays and function calls (we assume Misra-C as programming language of runnables [27]).
In particular, to deal with shared memory and functions calls, a context file is introduced (similarly to the approach in [30]).
Variables are scalar in the following. For structures, a variable is used for each field. For arrays, we abstract from array indexes.

Context file A context file is defined to record information stored in the shared memory of a SWC and information about runnable calls.
The context file maintains:

- for each variable $v \in Var$, the entry $v : \sigma$, where σ is the dependency level of data assigned to v;
- for each runnable $r \in R$, the entry $r(\sigma_1, ..., \sigma_k)\sigma; \sigma'$, where $\sigma_1, ..., \sigma_k$ are the levels for the actual parameter, σ is the level for the result and σ' is the level of the calling environment.

In particular, during the analysis, for each variable, the context file maintains the maximum dependency level of data recorded in the variable.
For each runnable, the context file maintains how the runnable is called in terms of the maximum level of the calling environment, the actual parameters and return. We assume parameters and return of runnables, for generality.
Each port variable is initialized to the level of the port. All other variables are initially assigned the lowest level ($\emptyset$). Runnables are initially assigned the lowest level for calling environment, parameters and return.
Entries in the context file are manged in a special way. Their dependency level never decreases. An assignment to a variable in the context file, updates the dependency level of the variable in the context file to the *lub* between its current level and the level of the assigned expression. Analogously, for runnable entries.
Example. In the following, we will consider one software component of a simple example of an AUTOSAR application [105]. The software component manages three runnable entities, 2 input ports, 4 output ports and 4 inter-runnable variables (irv1, irv2, irv3, irv4). Runnable 1 performs the sum of the value read from the input port in1 and the value from irv3 and writes the result in irv1; runnable 2 performs the subtraction between irv1 and irv2, accumulates the result and writes it on irv4 and on the output port 4; finally Runnable 3 multiplies irv4 and the value read from input port in2 and writes the result in irv2. The data sent to the other output ports (1, 2 and 3) and in irv3 oscillate between 1 and -1 and is not related to the other elements of the component.
The code of runnable r1 in the example is shown in Figure 7.8.
For simplicity, we use in1, in2, and out1, out2, out3, out4 as names of input/output port variables. For example, `RPort_DE1` is called in1 and `PPort_DE1` is called out1.
A runnable is abstractly executed starting from its local memory and the context file. The initial context file and the local memories for the software component are shown in Table 7.1.

Calls to RTE functions We deal with calls to RTE functions in the runnable code as follows.

- Send/Receive data communication ports
  RTE function for writing/reading a port are mapped to read/write of the corresponding variable:
  `Rte_IRead_Run1_RPort_DE1()` is mapped to the read of variable `RPort_DE1`
  `Rte_IWrite_Run1_PPor_DE1(Delay_n)` is mapped to the write of variable of the port: `PPort_DE1 := Delay_n`.
- Updates of variables that trigger runnables
  The calling environment of the triggered runnable is updated in the context file. The runnable is executed into an environment which depends on the level of the variable.

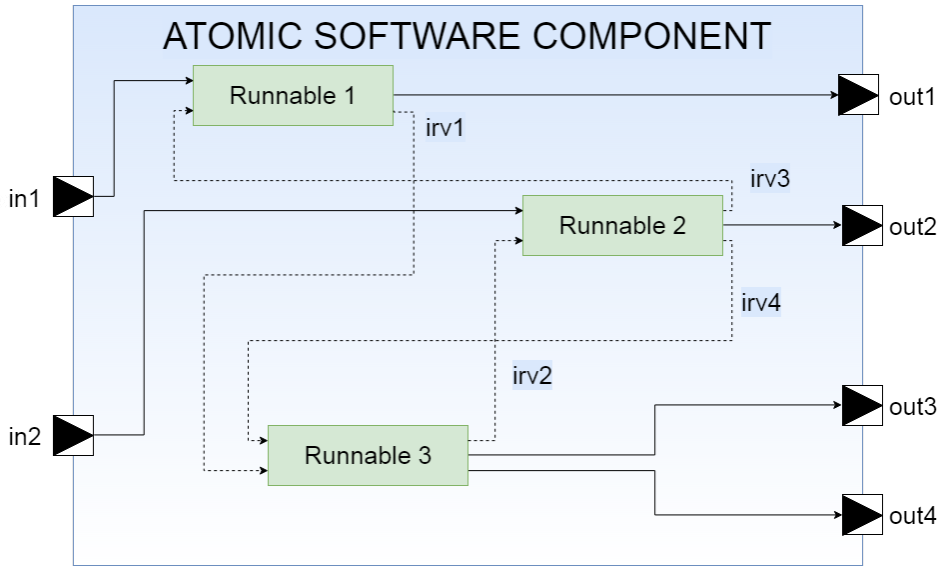

Figure 7.7: A software component and its runnables.

```
FUNC(void) Run1(void) {
int8_T Delay_n;
Delay_n = rtDWork.Delay_DSTATE_a;
if(((int32_T)Rte_IStatus_Run1_RPort_DE1())==0)
   {
     rtB.Add = Rte_IRead_Run1_RPort_DE1()+
    ((real_T) Rte_IrvIRead_Run1_IRV3());
   }

Rte_IrvIWrite_Run1_IRV1(rtB.Add);      (*)
rtDWork.Delay_DSTATE_a = (int8_T)((int32_T)
      (-((int32_T)rtDWork.Delay_DSTATE_a)));
Rte_IWrite_Run1_PPort_DE1(Delay_n);
}
```

Figure 7.8: Code of runnable r1.

- Client/server ports
  RTE function for client/server communication
  This function trigger the runnable that implements the service. The calling environment of the triggered runnable is updated in the context file. The runnable implementing the service is executed into an environment which depends on the level of the environment of the caller.

Analysis of an AUTOSAR model The analysis of an AUTOSAR model is based on an iterative process that performs the abstract execution of all runnables in R, using an abstract context file. In the initial abstract context, the level of each port variable is fixed to the level of the port, and the level of every other variable is set to $\emptyset$. If during the analysis a level in the context file changes, all runnables must be re-executed.
The analysis uses an abstract interpreter, named EXEC, that analyses a single runnable. EXEC performs an abstract execution of the runnable starting from a context file D and producing a newcontext file D'. The state of a runnable is a tuple $\langle \sigma, q, D, M^\sharp \rangle$, where q is the code of the runnable, D is the abstract context file and $M^\sharp$ is the abstract memory and σ is the environment.
The analysis terminates when, starting from a context file, all runnables are executed and the context is not changed.

$D := D^0$
$T := R$
`while`$(T \neq \emptyset)$
  `select` $r \in T$
  $T := T - \{r\}$
  $D' := EXEC(r, D)$
  `if`$(D' \neq D)$
    $D := D'$
    $T := R$

Figure 7.9: Analysis of an AUTOSAR model

The main steps of the iterative analysis are shown in Figure 7.9, where D^0 is the initial context.
A the end of the analysis, the context file records the dependencies for ports of all the software components. The approach is conservative, in the sense that all possible dependencies for any real execution of the runnables are detected. False dependencies are possible, since in the abstract analysis all branches of control instructions are executed, even those that in real execution would have never been executed.
Example In the following, we report results of the application of the method to the example.
The analysis starts from Table 7.1, that reports the inital context file and local memories of runnables. Let us consider the abstract execution of runnable 1. The `if` instruction causes the beginning of an implicit flow. According to the semantics rules, instructions in both branches of an `if` are executed under an environment set to the level of the condition of the `if`. On entering the `if` instruction, the environment is upgraded to the level of the variable (`RPort_DE1`). The implicit flow terminates at instruction (*), which is the first instruction out of the scope of the `if`. At this point, the environment is downgraded to the level before the execution of the conditional instruction.
Table 7.2 reports the state of the memory at the end of the analysis of the three runnables.
With our static analysis we can assert that the output port out4, irv2 and irv4 depend on both the input ports, meanwhile the irv1 only depends on input port in1. With a simple approach where all output variables of a runnable depends on all its input variables irv1 would depend on input port i2 too.

7.3 Synthesis of Components and Mechanisms

7.3.1 Time

The synthesis of mechanisms that are related to time for a mixed-safety critical system will consist of the automatic generation of RTE code and kernel configuration from a model description with components at different criticality levels.
This activity will be part of WP4 and is described in the context of the WP4 deliverables.

7.3.2 Security

Currently, the steps that the developers have to follow in order to specify a communication making use of the AUTOSAR security mechanisms are complex and error prone.
AUTOSAR does not provide any means to specify high level security requirements at the level of the application component models, but rather requires the application developers to directly use the standard services of the CSM. This is in contrast with the AUTOSAR approach for scheduling and communication. Application components are not allowed to use the basic software services for communication over network buses or the operating system services. Rather, high level specifications are provided and, based on them, code is automatically generated by tools to define the threads and

Table 7.1: Initial context file and local memories.

Variables	Dependencies
Context file	
in1	{ in1 }
in2	{ in2 }
rtb.Add	∅
rtdWork.Delay_DSTATE_a	∅
rtdWork.Delay_DSTATE_m	∅
rtdWork.Delay_DSTATE	∅
rtdWork.Integrator_DSTATE	∅
irv1	∅
irv2	∅
irv3	∅
irv4	∅
out1	{ out1 }
out2	{ out2 }
out3	{ out3 }
out4	{ out4 }
Runnable R1	
Delay_n	∅
Runnable R2	
Delay	∅
SubtracterBuffer	∅
Runnable R3	
tmp	∅
*tmp	∅
OutputBuffer	∅

invoke the appropriate network communication services. We propose a similar approach to handle the needs of confidentiality and integrity, allowing the developers to specify a security level (confidentiality and/or integrity) for the communication of the application components (in the AUTOSAR model). In addition, as a demonstrator, we develop a prototype model and code generation tool, which automatically synthesizes the right services to use to achieve the security level specified by the developers.

The AUTOSAR application model consists of a set of application software components that communicate using ports that express client-server relationships (in this case the port is typed by an operation interface) or send-receive data interactions, where the port is typed by a data interface consisting of a set of typed data items. An example of an AUTOSAR communication is shown in figure 7.10, in which two components SW-C1 and SW-C2 exchange data over a pair of send-receive port (indicating a data item that is provided on one side, required on the other). The internal behavior of AUTOSAR components consists of runnables or functional units, represented by a function entry point. Each runnable indicates the port it uses. Internally, the runnable code accesses the ports through a set of standard API for port communication and port service request (denoted as RTE services). A set of events triggering the execution of runnables completes the set of the main modeling entities.

The coder that provides the internal representation of the runnable behavior has the responsibility of using the standard API functions for communication over the ports. This API is independent from the component placement and translated by the RTE code into the actual communication mechanism. An example of RTE function is the API for writing data over a send-receive port:

```
Std_ReturnType  Rte_Write_<port>_<comp>(.. , data, ..)
```

Table 7.2: Final context file and local memories.

Variables	Dependencies
Context file	
in1	{ in1 }
in2	{ in2 }
rtb.Add	{ in1 }
rtdWork.Delay_DSTATE_a	∅
rtdWork.Delay_DSTATE_m	∅
rtdWork.Delay_DSTATE	∅
rtdWork.Integrator_DSTATE	{ in1, in2 }
irv1	{ in1 }
irv2	{ in1, in2 }
irv3	∅
irv4	{ in1, in2 }
out1	{ out1 }
out2	{ out2 }
out3	{ out3 }
out4	{ out4, in1, in2 }
Runnable R1	
Delay_n	∅
Runnable R2	
Delay	∅
SubtracterBuffer	{ in1, in2 }
Runnable R3	
tmp	{ in2 }
*tmp	{ in2 }
OutputBuffer	∅

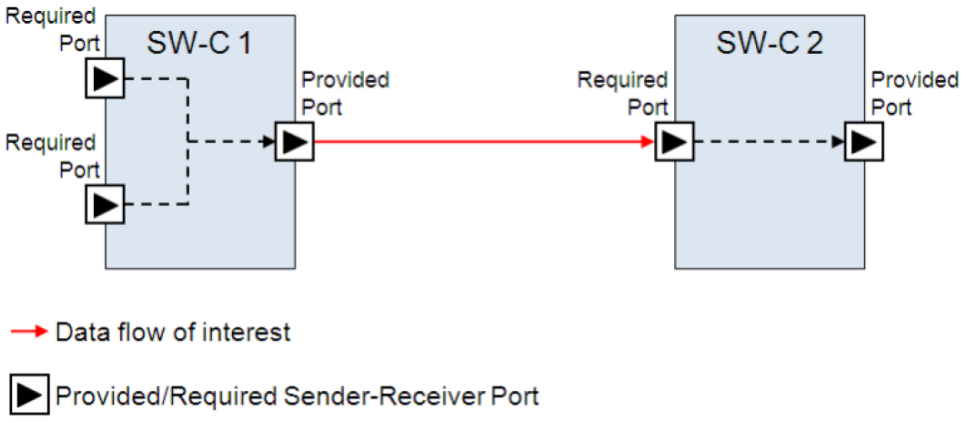

Figure 7.10: An example of AUTOSAR port interaction.

In general, an AUTOSAR software component cannot directly access BSW modules [26]; software components communicate between each other and with BSWs only through the RTE. The RTE implements communication and scheduling, and provides the actual implementation of the RTE port communication primitives (using network services or direct local communication primitives); it has the responsibility of organizing the execution of runnables in threads; and its code uses the services of the Basic Software (BSW), that is, the set of the drivers and the operating system. This model allows software components to be developed independently of the underlying hardware, which means

that they are transferable and reusable.

The AUTOSAR modules for Security

The AUTOSAR standard provides a number of mechanisms and modules that can be used by the software developers to build safe and secure software.

- Secure On-board Communication (SecOC) [22]: the purpose of the SecOC module is to allow the transmission of secured data between two or more peers over a network;
- Crypto Abstraction Library (CAL) [20]: the CAL provides other BSW modules and application software components with cryptographic functionalities. As a library, the CAL is not related to a special layer of the AUTOSAR Layered Software Architecture.
- CSM: the CSM is an AUTOSAR service which provides cryptographic functionalities to other software modules, based on a software library or based on an hardware module (HSM).

The AUTOSAR CSM and CAL specifications define the same cryptographic functionalities, including hash calculation; the generation and verification of message authentication codes; random number generation; encryption and decryption using symmetrical and asymmetrical algorithms; the generation and verification of digital signature and key management operations. The existence of both CSM and CAL, which provide the same (or similar) functionalities is for historical reasons; CAL is a library whereas CSM is a service.

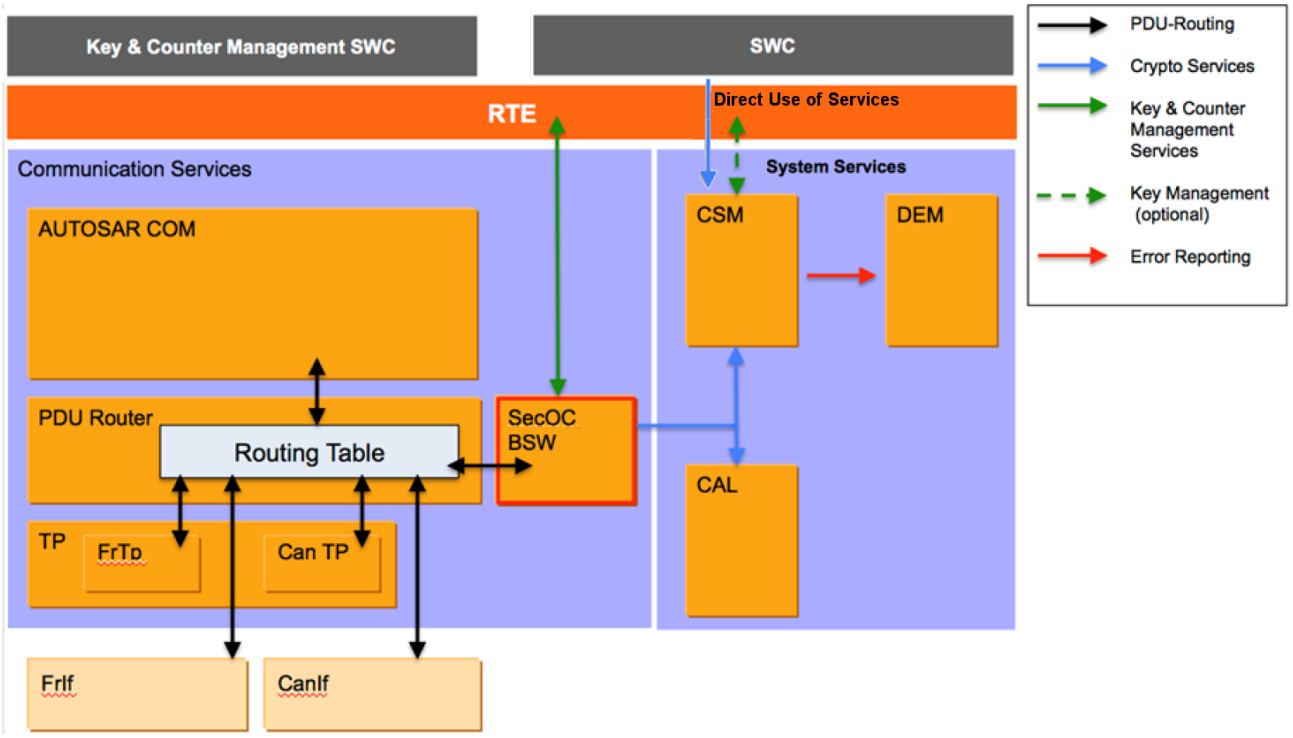

Figure 7.11: The AUTOSAR standard modules for security.

The Crypto Service Manager [17] is an AUTOSAR service, part of the AUTOSAR service layer, as shown in Figure 7.11. The CSM offers a standardized interface to higher software layers to access cryptographic functionalities. The CSM service can be configured to provide the services that are strictly needed and the selection of synchronous or asynchronous processing. The CSM also controls the concurrent, multiple and synchronous/asynchronous access of one or multiple clients to one or more services (i.e. it performs buffering, queuing, arbitration, multiplexing). A sample list of CSM interfaces consists of the following.

Service name	Csm_MacGenerateStart	
Syntax	`Std_ReturnType Csm_MacGenerateStart( Csm_ConfigType cfgId, const Csm_SymKeyType *keyPtr)`	
Parameters (in)	cfgId	Identifier of the CSM module configuration
	keyPtr	Pointer to the key structure, used by the MAC generation algorithm
Return values	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description	The interface is used to initialize the MAC generate service of the CSM module. If the service is busy, the function returns `CSM_E_BUSY`. Otherwise the function stores the given configuration and call the `Cry_<Primitive>Start` of the primitive identified by the cfgId and return the value returned by that function.	

Figure 7.12: Example of the CSM function for the initialization of the Mac Generation service.

- CsmMacGenerate;
- CsmMacVerify;
- CsmSymEncrypt;
- CsmSymDecrypt;
- CsmSymBlockEncrypt;
- CsmSymBlockDecrypt;

The interfaces specified above are implemented using the streaming approach of start, update and finish functions. An example for the CsmMacGenerate interface is shown below.

- Csm_MacGenerateStart();
- Csm_MacGenerateUpdate();
- Csm_MacGenerateFinish();

Table 7.12 shows the detailed description of the CSM service that is supplied for the generation of message authentical codes (MACs).
The services offered by the CSM can be used locally only. If remote access is needed, it is up to the provider to specify, implement and provide some proxy for access to CSM. If the CSM is used remotely (via a proxy), it must be taken into account that this raises security implications: any communication between ECUs is done via not protected communication buses (e.g. CAN). Unencrypted data, not yet signed data, would be transmitted and might become stolen or manipulated.
CSM services use cryptographic algorithms that are implemented using a software library or cryptographic hardware modules - both are out of scope and not specified by AUTOSAR. Note that there is no user management in place, which prevents nonauthorized access to any of the CSM services. This means, that if any access protection is needed, it must be implemented by the application; access protection is not target of the CSM.

AUTOSAR Extensions for Modeling Secure Communications

The AUTOSAR security model defines the mechanisms that should be implemented as part of the BSW layers and mostly disregards the application level, that is, how the designer of an application with security concerns should specify that its communications need to be suitably protected. These requirements should be applicable to the elements of the model, that is to runnables and any port interaction, of sender/receiver or client server type, as well as on events.
The security metamodel defined in Safure focuses on intra-platform security issues whereas, in contrast, SAFURE focuses on the security of in-vehicle communication and its relationship with performance and safety. The main concepts of the SAFURE security modeling at the functional level are shown in Figure 4.7.
SAFURE defines two main concepts at the functional level: the trust level of a functional element and the security requirements of communications between elements. A functional element, either a component or a runnable (an executable function), may be associated to a trust specification which specifies to what extent the element can be trusted to provide the expected function, or service, with respect to attacks targeted to compromise its functionality. A trust specification consists of: i) a trust specification identifier (trustSpecID), which identifies the specification, and ii) a trust level (trustLevel) which provides an indication of the extent to which the element can be trusted. The trustLevel is an attribute of type trustLevelType that corresponds to an integer in the range 1 to 5, being 1 the highest trust level and 5 the basic one.
The security requirement can apply to an entire data interface (all the elements in it), to a single data item within an interface, to all the communications outgoing from a component, or to an interaction between a sender and a receiver runnable (possibly further qualified by the communication interface or data item). A security requirement defines a required amount of trust in terms of a system-specific criterion and a minimum level of an associated quality metric that is necessary to meet one or more trust policies. A secure communication requirement is defined by four attributes:

- a security requirement identifier,
- a security level,
- a confidentiality level indicator (defined as the required key length), and
- an integrity level indicator (defined as the required key length).

Using the Rhapsody implementation of the AUTOSAR extensions, the security requirements for the communication are realized by stereotyped constraints. Constraints that are stereotyped as SecurityRequirement can be added in the Rhapsody AUTOSAR model to the communication elements (such as a pair of sender and receiver runnables) using dependencies. In the case of a pair of sender and receiver runnables, the (single) sender needs to be identified by stereotyping the dependency with the sender runnable.
Of course constraints can also be added to components, interface definitions and ports, realizing the desired flexibility as in the abstract metamodel. A sample model showing the applicability of the proposed extensions is represented in Figure 7.13. It consists of two components with two runnable accessing as reader and writer a pair of sender and receiver ports. The ports are typed by a data interface with a data item. In our example, a TrustSpecification is applied to the component SWC1 and the runnable run21 (on the left of Figure 7.13).
In Figure 6.10, SecureCommunication_if, SecureCommunication_de, SecureCommunication_swc and SecureCommunication_re represent the security requirement applied to an entire data interface, to a single data item within an interface, to all the communications outgoing from a component, and to an interaction between a sender and a receiver runnable, respectively.

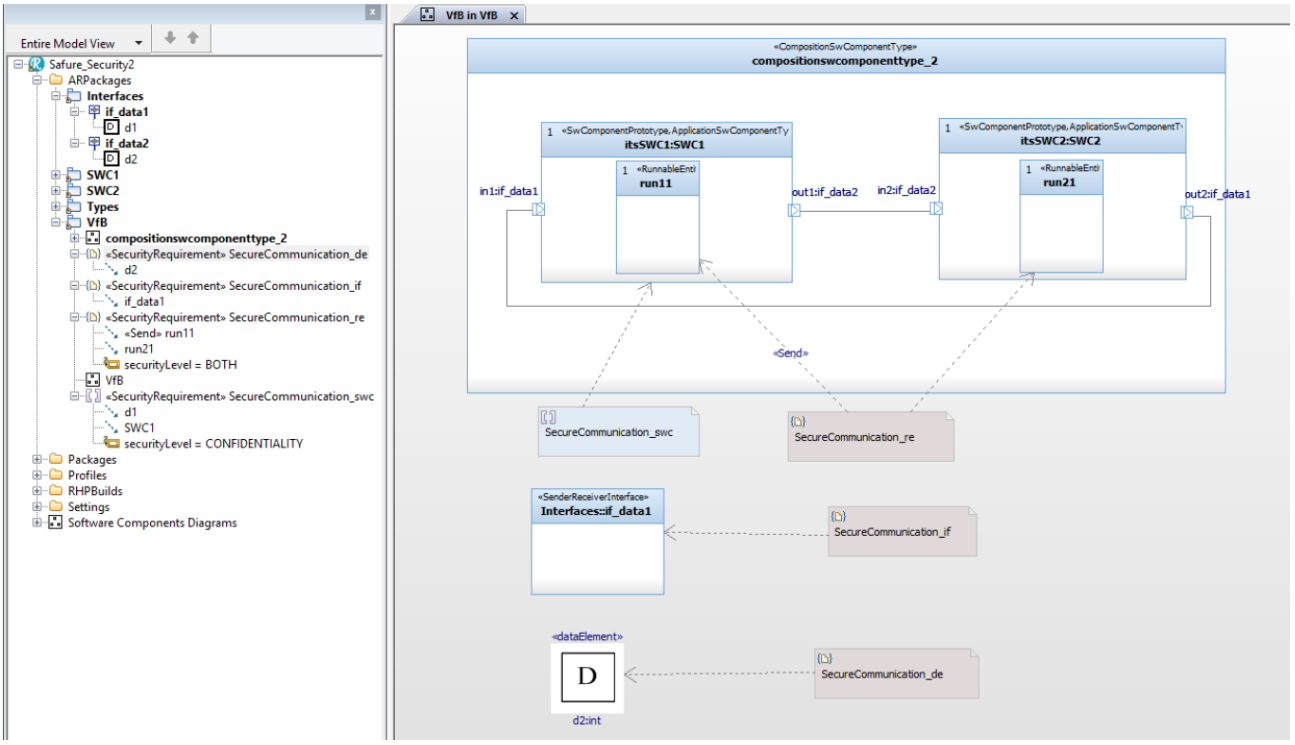

Figure 7.13: An example of use of the Safure extensions.

```
Std_ReturnType
   Csm_SymEncryptStart (Csm_ConfigIdType cfgId,  const  Csm_SymKeyType *keyPtr,
                              const uint8* InitVectorPtr,  uint32 InitVectorLength) {
          Check_parameters;
         //
        Initialize_configuration_structure;
        //
        // call the underlying cryptographic primitive
        return_value = Cry_SymEncryptStart((const void *)csm_encrypt->PrimitiveConfigPtr,
                                    keyPtr, InitVectorPtr, InitVectorLength);
        return return_value;
}
```

Figure 7.14: A sample funtion of CSM library

CSM library implementation

To support our experimental work and test the execution of the generated code, we implemented a CSM library compliant with the AUTOSAR standard. The library has been implemented in the C-language and relies on the OpenSSL library. The services implemented are the symmetrical encryption/decryption and the MAC generation/verification. The CSM supports the processing of a single instance of each service at a time. Each service makes use of a configuration structure, where all the information regarding the current request are stored. When a new instance of a service is created, a configuration structure is allocated. New service requests cannot be served until the previous instance is completed and the structure deallocated. The CSM is based on the streaming approach with start, update and finish functions. Although it is possible to configure synchronous or asynchronous job processing, we implemented only synchronous services, and the interface functions immediately compute the result. An example of implementation of a CSM function is shown in Figure 7.14.

Automatic Generation of components for Security

The definition of an AUTOSAR model with security extensions applied to communication links or to the ports involved in the communication results in the production (by model export) of a model description file in the standard ARXML (XML) format.
In our prototype implementation, developers can insert the security specification by using an AUTOSAR-compliant tool like Rhapsody, and then export the system as ARXML; or insert the security tag directly in the ARXML file.
In the ARXML, the security levels are expressed as a tag in the form of a pair [name; value] within the description field of the two ports involved in the communication and they can assume the following values:

- `SecurityNeeds=INTEG`, which stands for "integrity": in a communication between two entities (A and B), if A sends a message to B, B is able to verify that the received message was not altered by an external entity;
- `SecurityNeeds=CONF`, which stands for "confidentiality": in a communication between two entities (A and B), a third (non-authorized) entity (C), is not able to understands the content of the message exchanged between A and B;
- `SecurityNeeds=BOTH`, which means that both (integrity and confidentiality) are required.

Our tool consists of a Python script that parses the ARXML file and automatically generates the required elements based on the specified security level. The security requirements are fulfilled by using the services provided by the CSM. Client ports and interfaces to the CSM services are generated automatically. The script allows the developers to choose if these elements are added directly within the component which requires a specific security level, or if they are be added within a new purposely generated component (which acts as a proxy or filter). For this purpose, an additional tag add in the description field of the component is defined:

- `NewComponent=TRUE`: the script adds a new component which provides the required security elements. This component acts as a filter: it takes the output data of one component, it applies the required security service to that data, and then it sends out the protected data. This is useful when dealing with legacy components that cannot be changed.
- `NewComponent=FALSE`: (default value) the required security elements are added directly within the component which requires the security service.

We show an example of automatic generation applied to a sample model of an active safety system, as outlined in Figure 7.15.
The system consists of a portion of an active safety subsystem, in which multiple sensor components are feeding information on the environment around the car to subsystems dedicated to the detection of objects and the road profile. Following the sample model of two active safety functions (Lane departure warning and Lane keeping) and a Path planning component, a subset of the typical actuation systems of a car is represented (braking, steering, and throttle).
Confidentiality and integrity requirements are added to the communications between the GPS and the PathPlanning components, and between the latter and all the actuators, by the means of security tags. Other communications are only characterized by integrity requirements.
After the system is modelled using Rhapsody, as shown in Figure 7.16, the ARXML file exported by Rhapsody, is processed by our script, which generates a new ARXML file.
Consider the subsystem composed by GPS, PathPlanning and actuators (components below the red line in Figure 7.15).
If NewComponent=TRUE, the resulting model is shown in Figure 7.17. Security elements (filters) are shown in gray. Filters have client ports to call the CSM functions for integrity (MAC) and confidentiality (symmetric encryption).

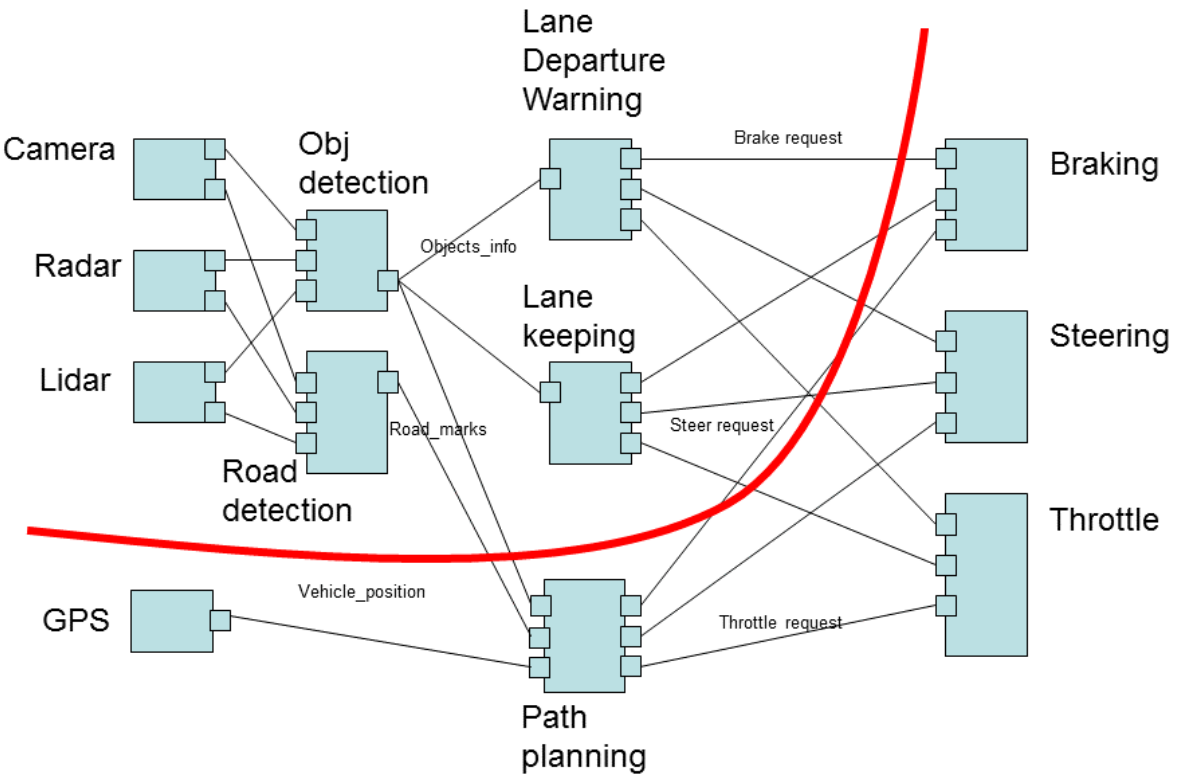

Figure 7.15: Braking system model overview.

If NewComponent=FALSE, the resulting model is shown in Figure 7.18. Client ports are added as shown in the figure enclosed by a rectangle.

Automatic Generation of secure RTE code

Consider the sender-receiver transmission of data elements over the GPS - PathPlanning communication in the sample model.
Depending on the allocation of the components on the system processor, specified in the AUTOSAR model allocation, the implementation of the RTE communication API can be of different types.
In the case of intra-ECU communication, the data transmission is performed by a write on a global variable (shared memory), no encryption is required (we assume the ECU is trusted), and no security elements are added by our script. For inter-ECU communications, the data transmission occurs over a communication bus (in our example, a CAN bus, but this does not affect the generated code), and security statements are added by the script in the prototype implementation of the RTE write.
To show an example of a code generated for the automatic call of the encryption functions, we operate on a configuration in which the GPS and PathPlanning subsystems reside on different ECUs. The example only details the code generation features that are added to satisfy the confidentiality requirements (encryption). The integrity requirements are satisfied by the internal implementation of the AUTOSAR COM function, by invoking the services of the SECOC. The code generated for the runnable operation that writes into the gpsPort in case of NewComponent=FALSE is:

```
Std_ReturnType  Rte_Write_gpsPort_GPS(uint32 datum) {
```

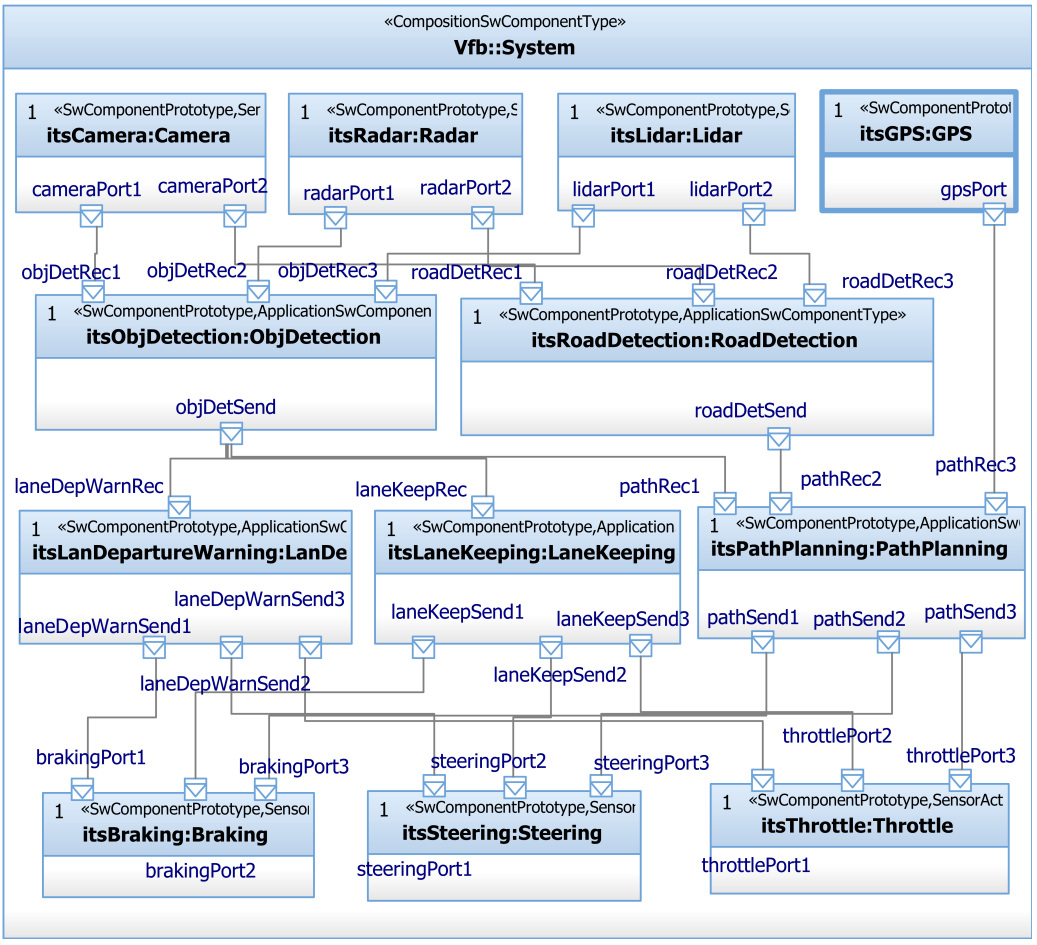

Figure 7.16: Braking system modeled on IBM Rational Rhapsody.

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

the code implementation of the RTE function performs, transparent to the user, the encoding of the data value and then its transmission using the higher level communication functions prescribed by AUTOSAR (in its COM layer).
In the case the user specifies NewComponent= TRUE a filter component is automatically added. In this case the code generated for the RTE Write function performs a simple copy into the buffer variable that is shared with the filter component

```
Std_ReturnType Rte_Write_gpsPort_GPS(uint32 datum) {
...
Rte_RxBuf = datum;
Rte_Write_Port0_GPSFilter(Rte_RxBuf);
...
return RTE_E_OK;
 }
```

the encryption code in this case is added to the communication outgoing from the filter, as shown below for the function `Rte_Write_Port0_GPSFilter(...)`:

Figure 7.17: Braking system after security tags processing (NewComponent=TRUE).

```
Std_ReturnType  Rte_Write_Port0_GPSFilter(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;
 }
```

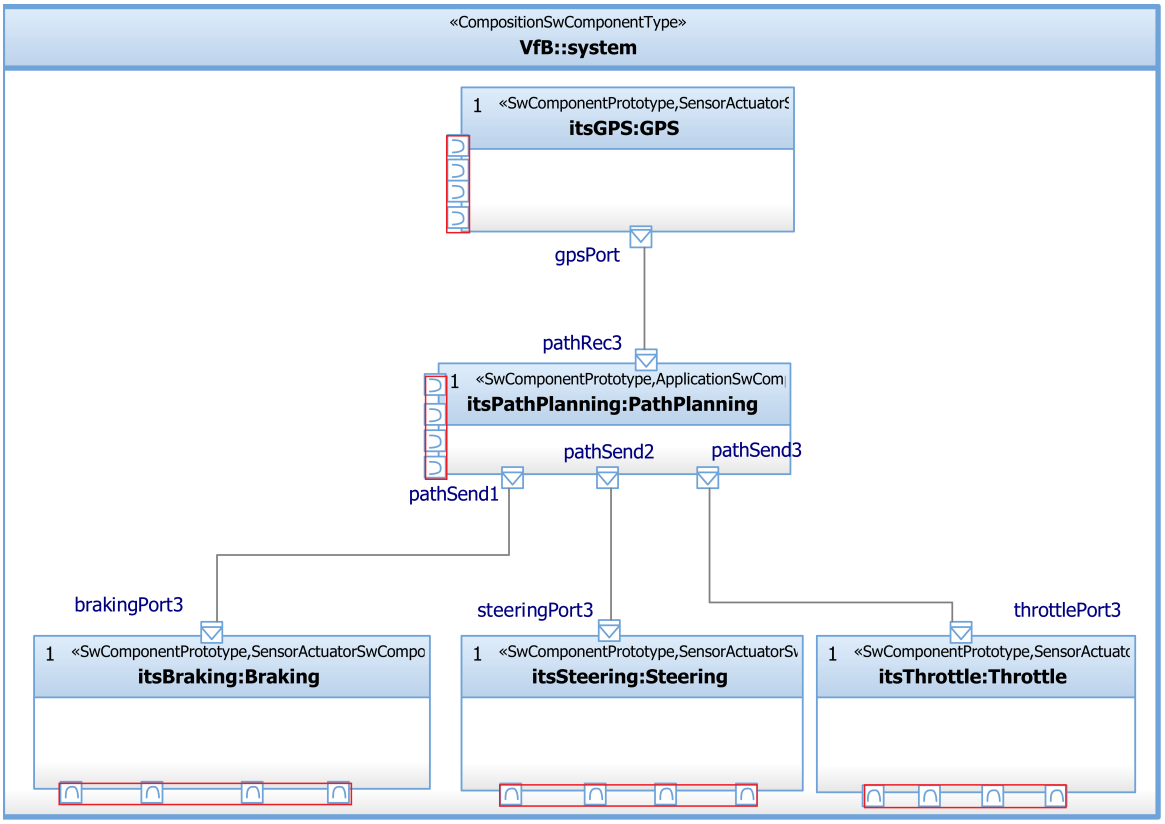

Figure 7.18: Braking system after security tags processing (NewComponent=FALSE).

Chapter 8

Summary and Conclusions

The deliverable provided a set of modeling concepts for the representation of safety, security and time attributes and constraints in mixed-critical systems. The subject is extremely wide and overlaps with a number of efforts, in the academia, in the industry and by standardization bodies. We chose to define the required modeling concepts in an abstract way and then to provide a possible concrete implementation according to two very popular standards: UML/SysML and AUTOSAR.
There are several concepts that surely require further investigation and represent open problems for the modeling and analysis of safety and security- critical systems, and one of them is surely the integration of security faults in traditional safety analysis.
We provided examples of analysis methods that stem from the availability fo the modeling concepts presented in this document. The list of the possible analysis methods is quite far from being exhaustive or complete, but hopefully is enough to convince the reader of the quality and feasibility of the approach.
We hope that the developed concepts and patterns could be the starting point for further investigations and they are surely going to be leveraged in the following WPs of Safure for the development of analysis and synthesis tools as applied to the project case studies.

Chapter 9

List of Abbreviations

ADL	Architecture Description Language
AES	Advanced Encryption Standard
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
BSW module	Basic Software Module
CAN	Controller Area Network
CC	Common Criteria
CPS	Cyber-Physical System
CRC	Cyclic Redundancy Check
DoS	Denial of Service
EC	European Commission
ECU	Electronic Control Unit
E/E	Electrical/Electronic
GCM	Galois/Counter Mode
GMP	Generic Methodology Pattern
HARA	Hazard Analysis and Risk Assessment
HSM	Hardware Security Module
HW	Hardware
MAC	Message Authentication Code
OS	Operating System
PDU	Protocol Data Unit
RBAC	Role Based Access Control
SAHARA	Security-Aware Hazard Analysis and Risk Assessment
SW	Software
SWC	Software Component
TADL	Timing Augmented Description Language
TBD	to be determined
TWCA	Typical Worst-Case Analysis
UML	Unified Modeling Language
VM	Virtual Machine
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle

Bibliography

[1] EAST-ADL Association. http://www.east-adl.info/index.html.

[2] TIMMO2 USE. http://adt.cs.upb.de/timmo-2-use/index.htm.

[3] UML MARTE – The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. http://www.omgmarte.org/.

[4] *Section 3.1 of MILS Architecture (EURO-MILS Report) is adopted to create the concrete MILS architecture of Trusted OS.*, 2015.

[5] Easwaran A. Demand.based Scheduling of mixed-criticality sporadic tasks on one processor. In *Proceedings of the 2012 RTSS*, 2012.

[6] T. F. Abdelzaher and K. G. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 162–171, 1995.

[7] Luca Abeni, Giuseppe Lipari, and Juri Lelli. Constant bandwidth server revisited. volume 11, pages 19–24, New York, NY, USA, January 2015. ACM.

[8] ARINC Industry Activities. *Avionics Full DupleX Switched Ethernet (AFDX)*. 2002.

[9] ARINC Industry Activities. *653-1 Avionics Application Software Standard Interface*. 2003.

[10] Andre Adelsbach, Ulrich Huber, and Ahmad-Reza Sadeghi. Secure software delivery and installation in embedded systems. In *Embedded Security in Cars*, pages 27–49. Springer, 2006.

[11] Sysgo AG. PikeOS Hypervisor, 2015.

[12] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[13] Rajeev Alur and Parthasarathy Madhusudan. Decision problems for timed automata: A survey. In *Formal Methods for the Design of Real-Time Systems*, pages 1–24. Springer, 2004.

[14] Jim Alves-Foss, Scott Harrison, Paul W. Oman, and Carol Taylor. The MILS Architecture for high-assurance embedded systems. In *International Journal of Embedded Systems*, volume 2, no. 3–4, pages 239–247, 2006.

[15] Ross Anderson. On the security of digital tachographs. In *Computer Security—ESORICS 98*, pages 111–125. Springer, 1998.

[16] AUTOSAR. *AUTOSAR Software Component Template: AUTOSAR Release 4.2.2.*

[17] AUTOSAR. *AUTOSAR Specification of Crypto Service Manager: AUTOSAR Release 4.2.2.*

[18] AUTOSAR. *Glossary: AUTOSAR Release 4.2.1.*

[19] AUTOSAR. *Overview of functional safety measures: AUTOSAR Release 4.2.2.*

[20] AUTOSAR. *Specification of Crypto Service Manager: AUTOSAR Release 4.2.2.*

[21] AUTOSAR. *Specification of Safety Extensions: AUTOSAR Release 4.2.1.*

[22] AUTOSAR. *Specification of Security Extensions: AUTOSAR Release 4.2.1.*

[23] AUTOSAR. *Specification of Timing Extensions: AUTOSAR Release 4.2.1.*

[24] AUTOSAR. *Standardization Template: AUTOSAR Release 4.2.2.*

[25] AUTOSAR. *Timing Analysis: AUTOSAR Release 4.2.1.*

[26] AUTOSAR. http://www.autosar.org, a.

[27] AUTOSAR. https://www.autosar.org/fileadmin/files/releases/2-0/software-architecture/rte/standard/autosar_sws_rte.pdf, b.

[28] Avionic Systems Group AS-2D2. TTEthernet (SAE AS6802). http://standards.sae.org/as6802/.

[29] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1), 2004.

[30] Marco Avvenuti, Cinzia Bernardeschi, Nicoletta De Francesco, and Paolo Masci. Jcsi: A tool for checking secure information flow in java card applications. *Journal of Systems and Software*, 85(11):2479âĂŞ2493, 2012.

[31] R. Barbuti, C. Bernardeschi, and N. De Francesco. Abstract interpretation of operational semantics for secure information flow. *Inf. Process. Lett.*, 83(2):101–108, July 2002.

[32] Sanjoy Baruah and Zhishan Guo. Mixed-Criticality Scheduling upon Varying-Speed processors. In *Proceedings of the 2012 RTSS*, 2012.

[33] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security for process-oriented systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 100–109. ACM, 2003.

[34] Matthias Beckert and Rolf Ernst. Designing time partitions for real-time hypervisor with sufficient temporal independence. In *Proc. of Design Automation Conference (DAC)*, Jun 2015.

[35] Matthias Beckert, Moritz Neukirchner, Stefan M. Petters, and Rolf Ernst. Sufficient temporal independence and improved interrupt latencies in a real-time hypervisor. In *Proc. of Design Automation Conference (DAC)*, Jun 2014.

[36] William Beckwith, Carolyn Boettcher, Mark Hama, Jahn Luke, and Tod Reinhart. High Assurance Safe and Secure Distributed Systems and Information Sharing. In *Infotech@Aerospace Conferences, American Institute of Aeronautics and Astronautics*, 2005.

[37] Pierfrancesco Bellini, Riccardo Mattolini, and Paolo Nesi. Temporal logics for real-time system specification. *ACM Computing Surveys (CSUR)*, 32(1):12–42, 2000.

[38] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer, 2004.

[39] G. Bernat, A. Burns, and A. Liamosi. Weakly Hard Real-Time Systems. *Computers, IEEE Transactions on*, 50(4):308–321, 2001.

[40] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of computer programming*, 19(2):87–152, 1992.

[41] Tom Bienmüller, Werner Damm, and Hartmut Wittke. The Statemate verification environment. In *Computer Aided Verification*, pages 561–567, 2000.

[42] Alessandro Biondi, Alessandra Melani, Mauro Marinoni, Marco Di Natale, and Giorgio Buttazzo. Exact interference of adaptive variable-rate tasks under fixed-priority scheduling. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[43] Alessandro Biondi, Marco Di Natale, and Giorgio Buttazzo. Response-time analysis for real-time tasks in engine control applications. In *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, Washington, USA, April 14-16, 2015.

[44] Berardino Carnevale, Francesco Falaschi, Diego Pacini, Gianluca Dini, and Luca Fanucci. A Hardware Accelerator for the IEEE 802.1X-2010 Key Hierarchy in Automotive Applications. In *Proceedings of the 12-th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2015)*, Marrakech, Morocco, November 17–20 2015.

[45] Antonio Cerone and Andrea Maggiolo-Schettini. Time-based expressivity of time Petri nets for system specification. *Theoretical Computer Science*, 216(1):1–53, 1999.

[46] Miguel Leon Chavez, Carlos Hernandez Rosete, and Francisco Rodriguez Henriquez. Achieving confidentiality security service for can. In *Electronics, Communications and Computers, 2005. CONIELECOMP 2005. Proceedings. 15th International Conference on*, pages 166–170. IEEE, 2005.

[47] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[48] MOST Cooperation. *MOST Specification*. 2000.

[49] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 4(2):511–547, 1992.

[50] Common Criteria. Common Criteria for Information Tecnology Security Evaluation. CCMB-2012-09-001, (`www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4.pdf`), 2012.

[51] P. Cuenot, C. Ainhauser, N. Adler, S. Otten, and F. Meurville. Applying Model Based Techniques for Early Safety Evaluation of an Automotive Architecture in Compliance with the ISO 26262 Standard. In *Proceedings of Conference on Embedded Real-Time Software and Systems (ERTS 2014)*, 2014.

[52] Inc. (FIRST) CVSS Special Interest Group (SIG), FIRST.Org. . "Common Vulnerability Scoring System v3.0: Specification Document," (`www.first.org/cvss/specification-document`).

[53] P. J. Denning.. D. E. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 7(20):504–513, 1977.

[54] Roberta Daidone and Gianluca Dini. A performance evaluation method for wsns security. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.

[55] Roberta Daidone, Gianluca Dini, and Giuseppe Anastasi. On evaluating the performance impact of the ieee 802.15. 4 security sub-layer. *Computer Communications*, 47:65–76, 2014.

[56] Roberta Daidone, Gianluca Dini, and Marco Tiloca. On experimentally evaluating the impact of security on ieee 802.15. 4 networks. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[57] Robert I. Davis, Timo Feld, Victor Pollex, and Frank Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *Proc. 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, Berlin, Germany, April 2014.

[58] de Niz D., K. Lakshmanan, and Rajkumar R. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the IEEE RealTime Systems Symposium*, pages 291–300, 2009.

[59] M. Di Natale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *Real-Time Systems Symposium, 1994., Proceedings*, pages 216–227, 1994.

[60] Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES12)*, jun 2012. Invited Paper.

[61] Gianluca Dini and Ida M Savino. Lark: a lightweight authenticated rekeying scheme for clustered wireless sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):41, 2011.

[62] Gianluca Dini and Marco Tiloca. Hiss: A highly scalable scheme for group rekeying. *The Computer Journal*, 56(4):508–525, 2013.

[63] M. Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication. *U.S. Department of Commerce, Information Technology Laboratory (ITL), National Institute of Standards and Technology (NIST), Federal Information Processing Standards Publication*, 2005.

[64] EVITA. Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios. EU FP7 Project no. 224275, "E-safety vehicle intrusion protected applications," (`www.evita-project.org`), 2009.

[65] EVITA. Deliverable D3.1: Security and trust model. EU FP7 Project no. 224275, "E-safety vehicle intrusion protected applications," (`www.evita-project.org`), November 24 2009.

[66] EVITA. Deliverable d3.3: Deliverable D3.3: Secure On-Board Protocols Specification. EU FP7 Project no. 224275, "E-safety vehicle intrusion protected applications," (`www.evita-project.org`), July 2011.

[67] International Organization for Standardization. *ISO 17458*. 2003.

[68] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 53–62, 2014.

[69] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. Modeling time in computing: A taxonomy and a comparative survey. *ACM Computing Surveys (CSUR)*, 42(2):6, 2010.

[70] J.J.G. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Parallel and Distributed Real-Time Systems, 1995. Proceedings of the Third Workshop on*, pages 124–132, 1995.

[71] Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer, and Stephan Flake. Towards the compositional verification of real-time UML designs. *ACM SIGSOFT Software Engineering Notes*, 28(5):38–47, 2003.

[72] Arda Goknil, Julien DeAntoni, Marie-Agnès Peraldi-Frati, and Frédéric Mallet. Tool support for the analysis of TADL2 timing constraints using TimeSquare. In *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*, pages 145–154, 2013.

[73] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *Computers, IEEE Transactions on*, 44(12):1443–1451, 1995.

[74] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proceedings of the International Conference on Embedded Software*, New Delhi, India, October 2014.

[75] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[76] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *Software Engineering, IEEE Transactions on*, 16(4):403–414, 1990.

[77] David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[78] Tobias Hoppe and Jana Dittman. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proceedings of the 2nd workshop on embedded systems security (WESS)*, pages 1–6, 2007.

[79] IEEE Audio Video Bridging Task Group. 802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams. http://www.ieee802.org/1/pages/802.1av.html.

[80] IEEE Time-Sensitive Networking Task Group. 802.1Qci - Per-Stream Filtering and Policing. http://www.ieee802.org/1/pages/802.1ci.html.

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

[82] IEEE Time-Sensitive Networking Task Group. P802.1Qbv (Draft 3.0) - Enhancements for Scheduled Traffic. http://www.ieee802.org/1/pages/802.1bv.html.

[83] ISO26262. *ISO 26262 standard for functional safety of road vehicles*. 2010.

[84] ITL. Fips pub 197: Advanced encryption standard (aes). *U.S. Department of Commerce, Information Technology Laboratory (ITL), National Institute of Standards and Technology (NIST), Federal Information Processing Standards Publication*, 2001.

[85] Axel Jantsch. *Modeling embedded systems and SoCs: concurrency and time in models of computation*. Morgan Kaufmann, 2004.

[86] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2–3):131–146, 2005.

[87] Jan Jonsson and Kang G. Shin. Robust adaptive metrics for deadline assignment in distributed hard real-time systems. *Real-Time Systems*, 23(3):239–271, 2002.

[88] Jan Jürjens. Towards development of secure systems using umlsec. In *Fundamental approaches to software engineering*, pages 187–200. Springer, 2001.

[89] Jan Jürjens. Umlsec: Extending uml for secure systems development. In *UML 2002—The Unified Modeling Language*, pages 412–425. Springer, 2002.

[90] Hwanju Kim, Hyeontaek Lim, Jinkyu Jeong, Heeseung Jo, and Joonwon Lee. Task-aware virtual machine scheduling for i/o performance. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 101–110, New York, NY, USA, 2009. ACM.

[91] John C. Knight. Safety Critical Systems: Challenges and Directions. In *Proceedings of the 24th International Conference on Software Engineering*, pages 547–550, 2002.

[92] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan, and Srivaths Moderator-Ravi. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*, pages 753–760. ACM, 2004.

[93] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[94] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[95] Don Kuzhiyelil and Sergey Tverdyshev. Timing Covert Channel Analysis on Partitioned Systems. In *Escar Europe 2014*, 2014.

[96] Andreas Lang, Jana Dittmann, Stefan Kiltz, and Tobias Hoppe. Future perspectives: The car and its ip-address–a potential safety and security risk assessment. In *Computer Safety, Reliability, and Security*, pages 40–53. Springer, 2007.

[97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[98] Kerstin Lemke, Christof Paar, and Marko Wolf. *Embedded security in cars*. Springer, 2006.

[99] Chung-Wei Lin and Alberto Sangiovanni-Vincentelli. Cyber-security for the controller area network (can) communication protocol. In *Cyber Security (CyberSecurity), 2012 International Conference on*, pages 1–7. IEEE, 2012.

[100] Chung-Wei Lin, Qi Zhu, Congchi Phung, and Alberto Sangiovanni-Vincentelli. Security-aware mapping for can-based real-time distributed automotive systems. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 115–121. IEEE, 2013.

[101] John Lloyd and Jan Jürjens. Security analysis of a biometric authentication system using umlsec and jml. In *Model Driven Engineering Languages and Systems*, pages 77–91. Springer, 2009.

[102] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *âĽł UMLâĽń 2002—The Unified Modeling Language*, pages 426–441. Springer, 2002.

[103] G. Macher, M. Stolz, E. Armengaud, and C. Kreiner. Filling the gap between automotive systems, safety and software engineers. 132/3:142–148, 2015.

[104] Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud, and Christian Kreiner. SAHARA: A Security-Aware Hazard and Risk Analysis Method. In *Proceedings of the Conference Design, Automation and Test in Europe Conference and Exhibition (DATE 2015)*, pages 621–624, 2015.

[105] MathWorks. Generate autosar-compliant code for multiple runnable entities (https://it.mathworks.com/help/ecoder/examples/autosar-code-generation-for-multiple-runnable-entities.html), a.

[106] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[107] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[108] Philip M. Merlin and David J. Farber. Recoverability of communication protocols–Implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, 1976.

[109] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[110] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[111] Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, and Suhaib A. Fahmy. Security Analysis of Automotive Architectures using Probabilistic Model Checking. In *Proceedings of the 52nd Design Automation Conference (DAC 2015)*, 2015.

[112] Marco Di Natale, Alessandro Biondi, Youcheng Sun, and Stefania Botta. Moving from single-core to multicore: Initial findings on a fuel injection case study. In *Proceedings of the International SAE Conference*, Detroit, USA, April 2016.

[113] Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Proc. of Real-Time Systems Symposium (RTSS)*, dec 2013.

[114] Moritz Neukirchner, Tobias Michaels, Philip Axer, Sophie Quinton, and Rolf Ernst. Monitoring arbitrary activation patterns in real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, dec 2012.

[115] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 2005.

[116] Dennis K Nilsson and Ulf E Larson. Simulated attacks on can buses: vehicle virus. In *IASTED International conference on communication systems and networks (AsiaCSN)*, pages 66–72, 2008.

[117] OMG. Unified Modeling Language: Version 2.5, 2015.

[118] Diego Ongaro, Alan L. Cox, and Scott Rixner. Scheduling i/o in virtual machine monitors. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, pages 1–10, New York, NY, USA, 2008. ACM.

[119] Marie-Agnès Peraldi-Frati, Daniel Karlsson, Arne Hamann, Stefan Kuntz, and Johan Nordlander. The TIMMO-2-USE project: Time modeling and analysis to use. In *ERTS2012 International Congres on Embedded Real Time Software and Systems*, 2012.

[120] Adrian Perrig, Ran Canetti, Dawn Song, and J Doug Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS*, volume 1, pages 35–46, 2001.

[121] James L. Peterson. Petri net theory and the modeling of systems. 1981.

[122] Carl A. Petri. Fundamentals of a theory of asynchronous information flow. *Proceedings of IFIP Congress, Amsterdam*, pages 386–390, 1962.

[123] Sophie Quinton, Torsten T. Bone, Julien Hennig, Moritz Neukirchner, Mircea Negrean, and Rolf Ernst. Typical worst case response-time analysis and its use in automotive network design. In *Proc. of DAC*, pages 1–6. ACM, 2014.

[124] Sophie Quinton and Rolf Ernst. Generalized weakly-hard constraints. In *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, pages 96–110. Springer, 2012.

[125] Sophie Quinton, Matthias Hanke, and Rolf Ernst. Formal analysis of sporadic overload in real-time systems. In *DATE*, pages 515–520. IEEE, 2012.

[126] Sophie Quinton, Mircea Negrean, and Rolf Ernst. Formal analysis of sporadic bursts in real-time systems. In *DATE*, pages 767–772, 2013.

[127] Christoph Ainhauser Raphael Trindade, Lukas Bulwahn. Automatically generated safety mechanisms from semi-formal software safety requirements. In *Proceedings of SAFECOMP 2014, Lecture Notes in Computer Science, volume 8666*, pages 278–293, 2014.

[128] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):461–491, 2004.

[129] Baruah S., Bonifaci V., DâĂŹAngelo G., Li H., Marchetti-Spaccamela A., Megow N., and Stougie L. Scheduling real-time mixed-criticality jobs. In *Proceedings of the 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27*, 2010.

[130] Baruah S. and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 147–155, 2008.

[131] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. On the need for extending marte with security concepts. In *International Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, 2011.

[132] Mehrdad Saadatmand and Thomas Leveque. Modeling security aspects in distributed real-time component-based embedded systems. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 437–444. IEEE, 2012.

[133] A. Sabelfeld and A.C Mayers. Language-based information-flow security. *IEEE journal on selected areas in communications*, 21(1), 2003.

[134] SAFE. ITEA 2 Project, "Safe Automotive soFtware architEcture," (`www.safe-project.eu`), 2011-2014.

[135] SAFE. Deliverable D322b: Proposal for extension of metamodel for hardware modeling. ITEA 2 Project, "Safe Automotive soFtware architEcture (SAFE)," (`www.safe-project.eu`), December 2013.

[136] SAFE. Deliverable D331a2: Proposal for extension of metamodel for error failure and propagation analysis. ITEA 2 Project, "Safe Automotive soFtware architEcture (SAFE)," (www.safe-project.eu), December 2013.

[137] SAFE. Deliverable D321d: Proposal for extension of metamodel for software and system modeling. ITEA 2 Project, "Safe Automotive soFtware architEcture (SAFE)," (www.safe-project.eu), November 2014.

[138] M. Saksena and Seongsoo Hong. An engineering approach to decomposing end-to-end delays on a distributed real-time system. In *Parallel and Distributed Real-Time Systems, 1996. Proceedings of the 4th International Workshop on*, pages 244–251, 1996.

[139] M. Saksena and Seongsoo Hong. Resource conscious design of distributed real-time systems: An end-to-end approach. In *Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on*, pages 306–313, 1996.

[140] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, (2):38–47, 1996.

[141] Martin Skoglund, Hans Svensson, Henrik Eriksson, Thomas Arts, Rolf Johansson, and Alex Gerdes. Checking Verification Compliance of Technical Safety Requirements on the AUTOSAR Platform Using Annotated Semi-formal Executable Models. In *Proceedings of SAFECOMP 2014 Workshops: ASCoMS, DECSoS, DEVVARTS, ISSE, ReSA4CI, SASSUR., Lecture Notes in Computer Science, volume 8696*, pages 19–26, 2014.

[142] Winfried Stephan, Solveig Richter, and Markus Muller. Aspects of secure vehicle software flashing. In *Embedded Security in Cars*, pages 17–26. Springer, 2006.

[143] Chris Szilagy and Philip Koopman. A flexible approach to embedded network multicast authentication. 2008.

[144] Chris Szilagyi and Philip Koopman. Low cost multicast authentication via validity voting in time-triggered embedded control networks. In *Proceedings of the 5th Workshop on Embedded Systems Security*, page 10. ACM, 2010.

[145] Christopher Szilagyi and Philip Koopman. Flexible multicast authentication for time-triggered embedded control network applications. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 165–174. IEEE, 2009.

[146] Theoretische Grundlagen der Informatik, UniversitÃđt Hamburg. Petri Nets Tool Database: https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html.

[147] TIMMO-2-USE. *State-of-the-Art Report: Deliverable D9.3*. 2011.

[148] Theodore Tryfonas, E Kiountouzis, and Angeliki Poulymenakou. Embedding security practices in contemporary information systems development approaches. *Information Management & Computer Security*, 9(4):183–197, 2001.

[149] Sergey Tverdyshev, Holger Blasum, and Igor Furgel. Compositional Assurance: EURO-MILS ST/PP for Separation Kernel Based Virtualization. In *ICCC*, 2013.

[150] Gordon M. Uchenik and W. Mark Vanfleet. Multiple independent levels of safety and security: high assurance architecture for MSLS/MLS. In *Military Communications Conference, 2005. MILCOM*, pages 610–614, 2005.

[151] OSEK VDX. OSEK/ VDX Communication Version 3.0.3. http://portal.osek-vdx.org/files/pdf/specs/osekcom303.pdf.

[152] Markus Voelter, Christian Salzmann, and Michael Kircher. Model driven software development in the context of embedded component infrastructures. In *Component-Based Software Development for Embedded Systems*, pages 143–163. Springer, 2005.

[153] Jon Whiteaker, Fabian Schneider, and Renata Teixeira. Explaining packet delays under virtualization. *SIGCOMM Comput. Commun. Rev.*, 41(1):38–44, January 2011.

[154] Marko Wolf, André Weimerskirch, and Christof Paar. Secure in-vehicle communication. In *Embedded Security in Cars*, pages 95–109. Springer, 2006.

[155] Alexander M Wyglinski, Xinming Huang, Taskin Padir, Lifeng Lai, Thomas R Eisenbarth, and Krishna Venkatasubramanian. Security of autonomous systems employing embedded computing and sensors. *Micro, IEEE*, 33(1):80–86, 2013.

[156] Tao Xie and Xiao Qin. Scheduling security-critical real-time applications on clusters. *Computers, IEEE Transactions on*, 55(7):864–879, 2006.

[157] Tao Xie and Xiao Qin. Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters. *Parallel and Distributed Systems, IEEE Transactions on*, 19(5):682–697, 2008.

[158] Wenbo Xu, Zain A. H. Hammadeh, Alexander Kröller, Sophie Quinton, and Rolf Ernst. Improved deadline miss models for real-time systems using typical worst-case analysis. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*, Lund, Sweden, July 2015.

[159] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):123–133, 1997.

[160] Rafael Zalman and Albrecht Mayer. A secure but still safe and low cost automotive communication technique. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–5. ACM, 2014.