Document Title	Timing Analysis
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	645
Document Classification	Auxiliary

Document Version	1.0.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Description
2014-03-17	1.0.0	AUTOSAR Release Management	Initial version

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

References

[1] Methodology
AUTOSAR_TR_Methodology

[2] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions

[3] Software Process Engineering Meta-Model Specification
http://www.omg.org/spec/SPEM/2.0/

[4] Embedded Systems Development, from Functional Models to Implementations

[5] Unified Modeling Language: Superstructure, Version 2.0, OMG Available Specification, ptc/05-07-04
http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf

[6] System Modeling Language (SysML)

[7] UML Profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE)
http://www.omg.org/spec/MARTE/1.1/

[8] Architecture Analysis and Design Language (AADL) AS-5506A

[9] EAST-ADL - Model Domain Specification
http://www.east-adl.info/Specification.html

[10] TIMMO-2-USE
http://www.timmo-2-use.org/pdf/T2UBrochure.pdf

[11] Tool Support for the Analysis of TADL2 Timing Constraints using TimeSquare
http://hal.inria.fr/docs/00/85/06/73/PDF/paper.pdf

[12] Guide to Multi-Core Systems
AUTOSAR_EXP_MultiCoreGuide

[13] Specification of Operating System
AUTOSAR_SWS_OS

[14] Scheduling algorithms for multiprogramming in a hard real-time environment
http://cn.el.yuntech.edu.tw/course/95/real_time_os/present paper/Scheduling Algorithms for Multiprogramming in a Hard-.pdf

[15] Analysing real-time communications: Controller Area Network (CAN)
http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=342710

[16] Pushing the limits of CAN-Scheduling frames with offsets provides a major performance
http://www.loria.fr/ nnavet/publi/erts2008_offsets.pdf

[17] Probabilistic response time bound for CAN messages with arbitrary deadlines
http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=6176662

1 Introduction

This document represents recommended methods and practices for timing analysis and design within the AUTOSAR development process.

1.1 Objective

During the development of AUTOSAR based systems, a common technical approach for timing analysis is needed. This document describes all major steps of timing analysis needed from the definition and validation of functional timing requirements to the verification of timing requirements on component and system level. Basis for the described methods are AUTOSAR Methodology [1] and AUTOSAR timing extensions [2].

1.2 Overview

The AUTOSAR timing analysis methodology is divided in following parts:

- Decomposition of timing requirements and levels
- Timing analysis on the ECU level
- Timing analysis on the network level
- Timing properties and methods for timing analysis

For each part, a proposed methodology is presented based on a number of typical real world use-cases. A complete overview of all use-cases is given in section 1.7 on page 10.

A future version of this document will also address end-to-end timing analysis (interface between ECU and network level) and demonstrate the suggested methodology by giving examples.

1.3 Motivation

The increasing number of functions, complexity in E/E Architectures and the resulting requirements on ECUs and communication networks imply increasing requirements on the development process. A central part of the development process is the design of robust and extendible ECUs and network architectures.

In the development of ECUs complexity is introduced through the integration of multiple SW-Cs (constituting various functions) executed in schedulable tasks. The design and verification of the task schedules becomes difficult due to their dependencies on shared resources such as processing cores and memory.

On the network level heterogeneous network types such as CAN, LIN, FlexRay, MOST and Ethernet are in use. This makes it hard to ensure robustness, especially when routing between protocols over a gateway takes place. The design of an efficient and robust network architecture and configuration is increasingly difficult. This creates the need for a systematic approach.

These aspects must be addressed in the E/E development process together with additional requirements regarding quality, testability, ability to perform diagnostic services and so on. The overall goal is to achieve sufficient reliability and performance at a cost optimum under the requirement of scalability over several vehicle classes. In order to enable integration of additional functions over the life-cycle of a vehicle, the extensibility of an E/E architecture is of high importance.

To make optimal technical decisions during the development of E/E architectures and their components it is necessary to have suitable criteria to decide how to implement a function.

One of the most important criteria in the development of current E/E architectures is timing. Many functions are time critical due to their safety requirements. Other functions have certain timing requirements in order to guarantee a high quality (customer) function. These functions often have certain latency and jitter constrains. For distributed functions these constraints are constituted of several segments where ECU and network are the two main parts. In order to specify and analyze these timing requirements functional timing chains are important. These are described in more detail in Chapter 2.

1.4 Scope

This document describes how to implement timing analysis during the development of E/E systems. Similar to [1], this does not include a complete process description but rather a set of practical methods to define timing requirements and how to ensure that these requirements are met. As stated in [1], the methodology is designed to cover the needs of various AUTOSAR stakeholders:

- Organizations: Methodology is modeled in a modular format to allow organizations to tailor it and combine the methodology within their own internal processes, while identifying points where they interact with other organizations.
- Engineers: Methodology is scoped to allow engineers of various roles quickly find AUTOSAR information that is relevant to their specific needs.
- Tool Vendors: Methodology provides a common language to share among all AUTOSAR members and a common expectation of what capabilities tools should support.

The following topics are addressed:

- Definition of appropriate timing analysis methods including related timing properties for all stages of an AUTOSAR development process without the exchange of company confidential information.
- Definition of requirements for timing analysis methods enabling implementation of appropriate tools.
- Documentation of relevant experience in the area of timing analysis (Network and ECU/software) with relevant use-cases.
- Structuring of timing properties and related methods with regard to use-cases.
- Timing as enabler for exchange on a functional level between OEM and tier1.

Delimitation:

- Contents of this document is complementary, and not overlapping, to the contents of the AUTOSAR timing extensions [2]
- Definition of meta models to document timing attributes (e.g. AUTOSAR TIMEX)
- Definition of timing behavior for specific SW-Cs or functions in AUTOSAR.

1.5 Acronyms and Abbreviations

Abbreviation	*Meaning*
BSW	Basic Software
CAN	Controller Area Network
COM	Communication module
ECU	Electrical Control Unit

ID	Identifier
I/O	Input/Output
LIN	Local Interconnect Network
PDU	Protocol Data Unit
RE	Runnable Entities
RTE	Runtime Environment
SW-C	Software Component
TD	Timing Description
TIMEX	AUTOSAR Timing Extensions [2]
UML	Unified Modeling Language
WCET	Worst case execution time
WCRT	Worst case response time
VFB	Virtual Functional Bus

Table 1.1: Glossary of Terms

1.6 Glossary of Terms

Term	*Synonym*	*Definition*
Event-triggered Frame	Sporadic Frame	A frame that is sent on an event triggered by the application independent from a communication schedule. The event-triggered sending is limited by a debounce time which specifies the shortest allowed temporal distance between two send events.
Execution Time		The execution time is the total time that the function needs to be assigned the resource in order to complete.
Frame		Information Package on CAN and FlexRay. A commonly used synonym is “message”.
Information Packages		Smallest send able information unit on a resource (e.g. frame).
Interrupt Load		The load of the CPU for servicing interrupts.
Load	Utilization	The load is the total share of time that a resource is used.
Period		The time period between two time-triggered send events of the same frame (e.g. 100 ms).
Response Time	Latency	The response time is the time between the activation of a function and its termination as defined in TIMEX.
Stuff bit		In CAN frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity.
System Parameter		A quantity influencing the timing behavior of the system.
Task	Technique	A number of steps to accomplish a specific goal.
Timing Constraint		A timing constraint may have two different interpretation alternatives. On the one hand, it may define a restriction for the timing behavior of the system (e.g. minimum (maximum) latency bound for a certain event sequence). In this case, a timing constraint is a requirement which the system must fulfill. On the other hand, a timing constraint may define a guarantee for the timing behavior of the system. In this case, the system developer guarantees that the system has a certain behavior with respect to timing (e.g. a timing event is guaranteed to occur periodically with a certain maximum variation). Compare AUTOSAR Timing Extension [2]

Timing Method	Technique	Defines an ordered number of steps to derive particular timing related work products (e.g. timing property, timing model)
Timing Model		A timing model collects all relevant timing information in one single place, typically tool-based. The model can be used to describe the timing behavior or it can be used to generate timing related configuration files.
Timing Property		A timing property defines the state or value of a timing relevant aspect within the system (e.g. the execution time bounds for a `RunnableEntity` or the priority of a task). Thus, a property does not represent a constraint for the system, but a somehow gathered (e.g. measured, estimated or determined) or defined attribute of the system.
Use-case	Scenario	Typical problem, broken down into tasks
Worst case		The term “worst case” denotes an upper bound on any value a certain property can take during run-time. This is usually different from and may never be smaller than the maximum value observed in the actual system. Typically worst-case values are derived using static analyses based on models of the system.
Work Product		See SPEM [3].

Table 1.2: Glossary of Terms

1.7 Use-cases

In order to show the proposed usage of timing analysis methodology a number of real-world use-cases are included in the document.

The use-cases are divided into the categories using the same structure as the chapters:

- Timing analysis on the ECU level
- Timing analysis on the network level

Section	Use-case	Page
3.3	ECU use-case “Collect Timing Information of a SW-C”	32
3.4	ECU use-case “Select an ECU Supplier”	34
3.5	ECU use-case “Validate Timing after SW-C integration”	34
3.2	ECU use-case “Create Timing Model of the entire ECU”	30
3.6	ECU use-case “Validation of Timing”	37
3.7	ECU use-case “Debug Timing”	39
3.8	ECU use-case “Optimize Timing for an series ECU”	40
3.9	ECU use-case “Optimize Scheduling”	43
3.10	ECU use-case “Optimize Code”	46
3.11	ECU use-case “Verify Timing Model(s)”	47
3.12	ECU use-case “Compare Timing Properties”	49
4.2	NW use-case “Integration of a Distributed Function”	52
4.3	NW use-case “Design of the new developed Network”	55
4.4	NW use-case “Remapping an existing Function”	58

Table 1.3: List of all use-cases in this document

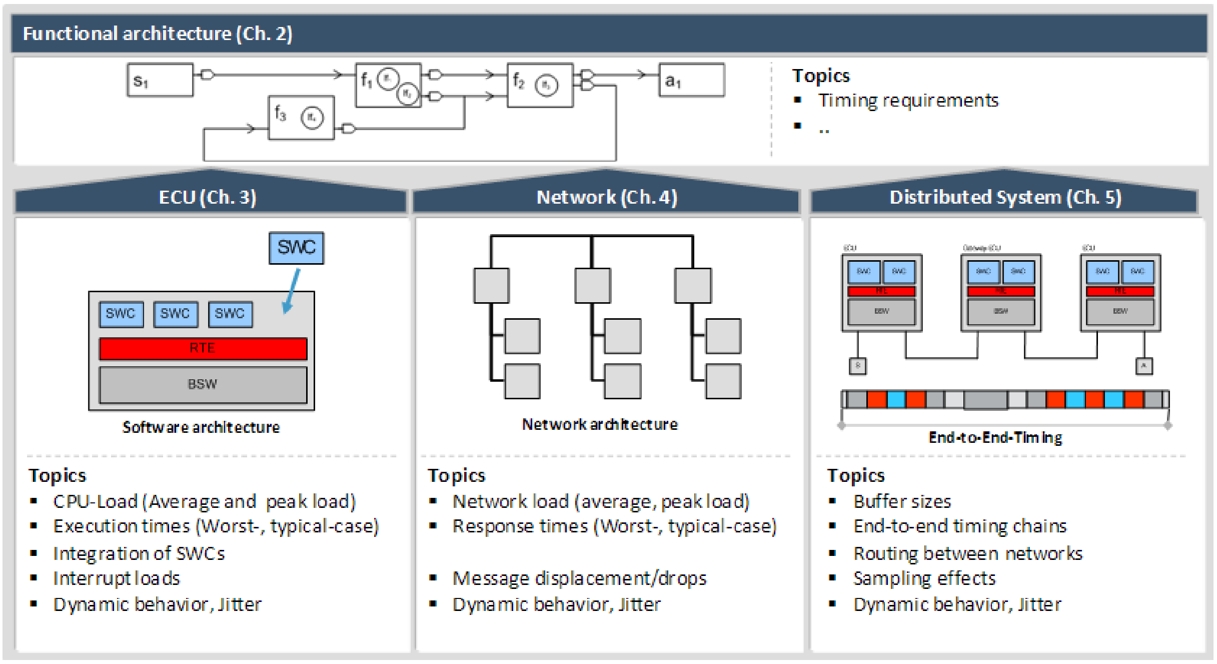

Figure 1.1: Overview of aspects for timing analysis

1.8 Document Structure and Chapter Overview

This section contains an overview of the docment and the chapter contents. Figure 1.1 illustrates the different aspects for timing analysis and indicate the chapters in which these will be addressed.

Chapter 1 “Introduction” contains the objective, motivation, scope of the document abbreviations and glossary of terms. Additionally, a list of the use-cases is contained in section 1.7.

Chapter 2 “Decomposition of Timing Requirements” contains a short introduction of the challenge of breaking down functional timing requirements from an abstract user’s view to the implementation view of AUTOSAR timing extensions. The problem definition and different approaches and concepts for methodological solutions are introduced.

Chapter 3 “Timing Analysis for SW-Integration on ECU Level” contains use-cases for applying timing analysis at ECU level. The chapter covers several use-cases with different levels of abstraction covering the complete development workflow of an ECU ranging from a supplier nomination up to timing optimization. For every use-case the corresponding methods and timing properties are linked. The chapter is addressed mainly to ECU architects and integrators for software components (SW-C).

Chapter 4 “Timing Analysis for Networks” contains use-cases for applying timing analysis at network level, covering scenarios such as extension of an existing network, design of a new network or redesign/reconfiguration of existing network architectures. These use-cases are split into smaller tasks. For each of these tasks the necessary timing properties and the corresponding timing methods are presented. These are used to validate the timing and performance constraints typical for the corresponding use-case. The chapter is addressed mainly to system architects and network integrators.

Chapter 5 “Properties and Methods for Timing Analysis” covers the timing properties and the methods derived from the use-cases and the tasks specified in chapter 3 “Timing Analysis for SW-Integration on ECU Level” and 4 “Timing Analysis for Networks”. The timing methods describe how to solve the tasks derived from the use-cases of the ECU, network or both domains. Every single method is presented in detail including its classification, description, relation to use-cases, requirements, timing properties, inputs, boundary conditions and its implementation. Some of the methods deliver timing parameters as output which can be evaluated by means of timing constraints to check the fulfillment of the timing requirement. Every single timing property is characterized by its classification, description, relation to use-cases, requirements, timing methods, format, (valid) range and implementation. The methods can be grouped in three main groups: simulation, analytical calculation and measurement; whereas the properties can be separated in two main groups: latency-like and bandwidth-like. An overview of the relation between the single methods and the single timing properties respectively is given, but also the interaction between the two is outlined.

2 Decomposition of Timing Requirements

Decomposition of timing requirements is a primary concern for the design and analysis of a real-time system. Actually, at the beginning of the system design process, timing requirements are expressed at the level of the customer functionality identified in the specification. The development of the customer functionality requires its decomposition into small and manageable components. This decomposition activity called architecting implies also a decomposition of timing requirements attached to the decomposed functionality. First sections of this chapter introduce basic concepts of real-time architectures and their properties. Then, after giving an overview of the proposed approach for timing requirements decomposition, a focus on dedicated methodologies and their associated languages is done.

2.1 Basic Concepts of Real Time Architectures

The architecture is the result of early design decisions that are necessary before a group of stakeholders can collaboratively build a system. An architecture defines the constituents (such as components, subsystems, ECUs, functions, runnables, compilation units ...) and the relevant relations (such as “calls”, “sends data to”, “synchronizes with”, “uses”, “depends on” ...) among them.

In addition to the above-mentioned structural aspects, a real-time architecture shall provide means to fulfill timing requirements. Like for the system’s constituents, real-time architecting consists in decomposing timing requirements and identifying relationships (such as refinement and traceability) among them. In fact, the timing requirement decomposition is a consequence of the structural decomposition where timing requirements are segmented upon the decomposed units.

However, while structural decomposition could be driven by functional concerns, input/output data flows, and/or provided/required services, timing decomposition is a more complex task to achieve. Indeed, correct timing requirement decompositions shall be locally and globally feasible. Locally the subcomponent timing properties shall fulfill its assigned timing requirement segment. Real-time software architecture design aims at finding a functional decomposition and a platform configuration which timing properties allows fulfilling local and global timing requirements.

Timing properties are highly dependent on the underlying software and hardware platform resources. Moreover, access to shared platform resources by the decomposed units introduces some overhead (like blocking times or interferences ...). Timing properties will depend on:

- The chosen *placement* (e.g. allocation of functions/components on ECUs);
- The chosen *partitioning* (e.g. grouping of runnables on tasks);
- The chosen *scheduling* (e.g. tasks priority assignment);
- The chosen *concurrency model* (e.g. shared resources access protocol).

In order to assess these architectural choices with regard to timing requirements, timing analysis is necessary. Analysis methods, techniques and associated timing properties used for such an assessment can depend on the kind of real-time architecture under consideration (e.g. time-triggered or event-triggered architecture). Chapter 5 details this aspect. Timing analysis can be introduced at the system level as a prediction instrument for the refinement of systems functions toward their implementation [4]. Early timing analysis requires assumptions or requirements on the implementation platform resources, but constitutes a sound guide for timing requirements decomposition and refinement.

2.2 Some Basic Timing Properties

When dealing with timing aspect in a real-time architecture, we can basically refer to two main timing properties which are:

- *Execution or transmission times*;
- *Response times*.

This section gives a first introduction of these terms. A more detailed description and classification of these notions is provided in Chapter 5.

2.2.1 Execution and Transmission Times

The execution time of a schedulable entity (function, runnable, task) on a computing resource (e.g. ECU) is the duration taken by the schedulable entity to complete its execution in a continuous way without any consideration of other schedulable entities that are sharing the same computing resource (no suspension/preemption).

Similarly, the transmission time of a signal/message/frame on a communication resource (e.g. bus, network) is the duration taken by the signal/message/frame to transit from its source to its destination without any consideration of other signals/messages/frames transiting on the same communication resource.

An execution/transmission time is a quantitative property that can be characterized with the following qualifiers:

- A *statistical qualifier* (worst, best, mean/average) representing the bounds of execution/transmission time. This bound could be the upper bound which corresponds to the worst-case execution/transmission time (WCET/WCTT), the lower bound corresponding to the best-case execution/transmission time (BCET/BCTT), or the average-case execution/transmission time (ACET/ACTT) which could be useful for performance analysis. Among these three qualifiers, the WCET is the most commonly used for timing properties verification/validation of real-time systems.

- A *source* (estimated, measured, calculated (static analysis)) denoting the way an execution/transmission time is obtained. The precision of an execution time is highly dependent on its source. For instance, input data used for measurements triggers specific branches of the function/program which impacts the measured execution time value. For that reason, measurements can only provide average execution time or a distribution of execution times. To obtain execution time upper bound, static analysis techniques are employed (abstract interpretation, model checking ...).
- An *accuracy factor*. The accuracy of the evaluated WCET/WCTT depends on many factors among which the level of details of the software (instruction level) as well as the level of details of the execution/communication resource (like cache mechanisms). This latter could provide elements of unpredictability like branch prediction mechanisms that could affect the WCET analysis by making it more complex to achieve and too pessimistic. In order to avoid over dimensioning of execution platforms, and in order to allow accurate response time analysis (see the following subsection) WCET/WCTT analysis shall provide safe but accurate WCETs/WCTTs.

Sometimes, a WCET/WCTT can be a requirement to satisfy, especially at the very low levels of abstraction once the ECUs, network and deployment are fixed. However, in the very upper levels of abstraction, timing requirements usually refer to end-to-end response time bounds defined in the following subsection.

2.2.2 Response Time

The response time of a schedulable entity (function, runnable, task, ...) is the duration time taken by the schedulable entity to complete its execution. Unlike for execution time, the response time takes into account other schedulable entities that are sharing the same execution/communication resource. Hence, the response time of a schedulable entity comprises its execution time and additional terms induced by the concurrent access to shared resources (blocking times, jitters...). See Chapter 5 for more details.

An end-to-end response time is a response time in which several schedulable entities are involved. These schedulable entities form a chain. First schedulable entity of the chain is called the *source* schedulable entity and the last one is called the *sink* schedulable entity. The end-to-end response time is the elapsed time until the sink schedulable entity of the chain terminates its execution.

Like an execution time, a response time is a quantitative property that can be characterized with the following qualifiers:

- A *statistical qualifier* (worst, best and mean/average). The worst-case response time (WCRT) is the upper bound usually computed by timing analyses to assess timing requirements fulfillment.
- A *source* (estimated, measured, calculated) denoting the way a response time is obtained.

- An *accuracy factor*. The accuracy of a WCRT is highly dependent on the accuracy of the WCETs of the executable entities that are involved in the chain.

2.3 Overview

Mastering timing requirements is one key success factor for development and integration of state of the art automotive E/E-systems. Timing requirements shall be carried out continuously during the complex development process of a vehicle, and further shall be reused and exchanged for the re-use of functions or components to other vehicle projects: timing requirements have to be described systematically and carefully. The required level of detail can vary from timing constraints for high level customer related features at the vehicle level, over timing requirements for the control of a power amplifier for a particular actuator, to ECU-internal timing for data synchronicity of software functions on a multi-core microcontroller at the operational level.

As illustrated on Figure 2.1, the development process follows the well-known V-model, which describes a systematic and staggered top-down approach from system specifications to system integration. On the left branch process steps of specification are described, implementing decomposition from an entire E/E-system to single components. The base of the V describes implementation and associated test procedures. Following the right branch of the V testing and integration procedures up to vehicle system integration can be read in bottom up order.

According to these basic steps of an automotive OEM development process, requirements shall be traceable in any process step. This means that timing requirements shall be identifiable and traceable from a requirements specification via a supplier's performance specification to a test and integration documentation (protocols). As far as E/E-processes are concerned this means that timing requirements shall resist the process transformation between two companies like OEM an tier1-supplier and further down to tier2 and 3 suppliers. This can only be achieved by using a standardized system of description and methodology, referencing the model artifacts that are generally exchanged between development partners.

The AUTOSAR Timing Extensions (TIMEX) [2] based on the AUTOSAR System Template, represents the standardized format for exchange of a system description within an AUTOSAR compliant software development process. In addition TIMEX is an optional component which does not imply changes in the AUTOSAR System Template. The concept of the observable event, which occurs or can be observed in a referenced modeling artifact e.g. a RTE-port, allows specifying observation points and sequences of events in causal order (event chains) with additional timing constraints on them. The TIMEX concept is assumed to meet all use-cases of describing temporal behavior in an AUTOSAR system by means of timing requirements.

Unfortunately the OEM development process does not start with AUTOSAR. AUTOSAR only represents an implementation view for some software components, but not a view on higher level functional concepts that can comprise non software functions. Actually at the very beginning of the process, requirements are described in nat-

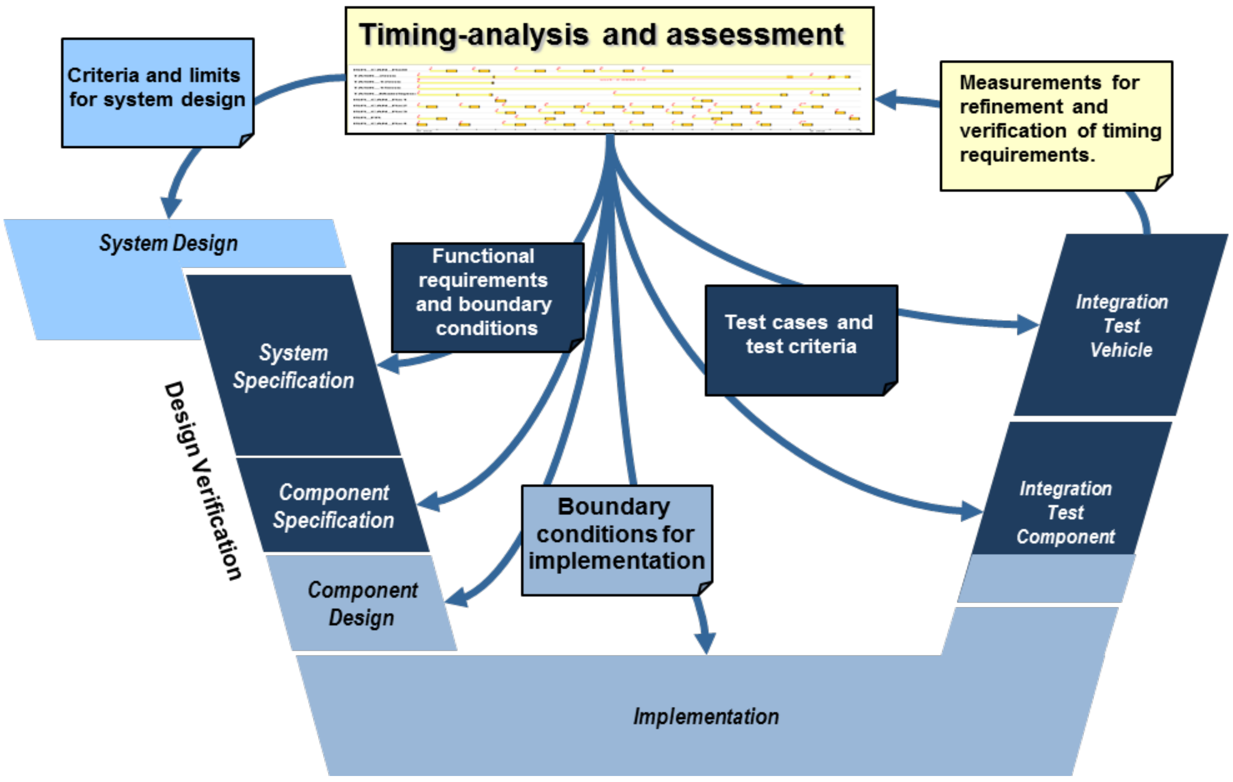
Timing-analysis and assessment
Criteria and limits for system design
Measurements for refinement and verification of timing requirements.
System Design
Functional requirements and boundary conditions
Test cases and test criteria
Integration Test Vehicle
System Specification
Design Verification
Component Specification
Boundary conditions for implementation
Integration Test Component
Component Design
Implementation

Figure 2.1: Application of timing analysis in a development process according to the V-model

ural language. These requirements have to be “formalized” in a non-natural language in order to assess them and allow their decomposition. The assessment of timing requirements must be done as earlier as possible in the development process. To enable this at system/functional level, a system/functional modeling language is needed. This language must provide concepts for functions design modeling and must also provide a formal way to capture and decompose timing requirements during the functional design. Several approaches based on Architecture Description Languages (ADLs) could be used to fill the gap between requirements specification in natural language and the implementation phase modeled in AUTOSAR. We can cite UML-based [5] Architecture Description Languages: SysML [6] (UML specialization for System Modeling) and MARTE [7] (UML specialization for Modeling and Analysis of Real-Time end Embedded systems). Other approaches that are more domain specific like AADL [8] for aerospace or EAST-ADL [9] for automotive also exist. The choice of the appropriate system/functional level modeling language depends on the internal OEMs’ processes. However, there are some general timing related criteria that are important to consider:

- A support for hierarchical timing requirements process;
- The ease of mapping the decomposed timing requirements to AUTOSAR TIMEX model artifacts that constitutes today the exchange format between the OEM and its suppliers.

In this chapter an approach based on all these ideas and concepts is drawn which shall give orientation to implement a hierarchical timing requirements process in the

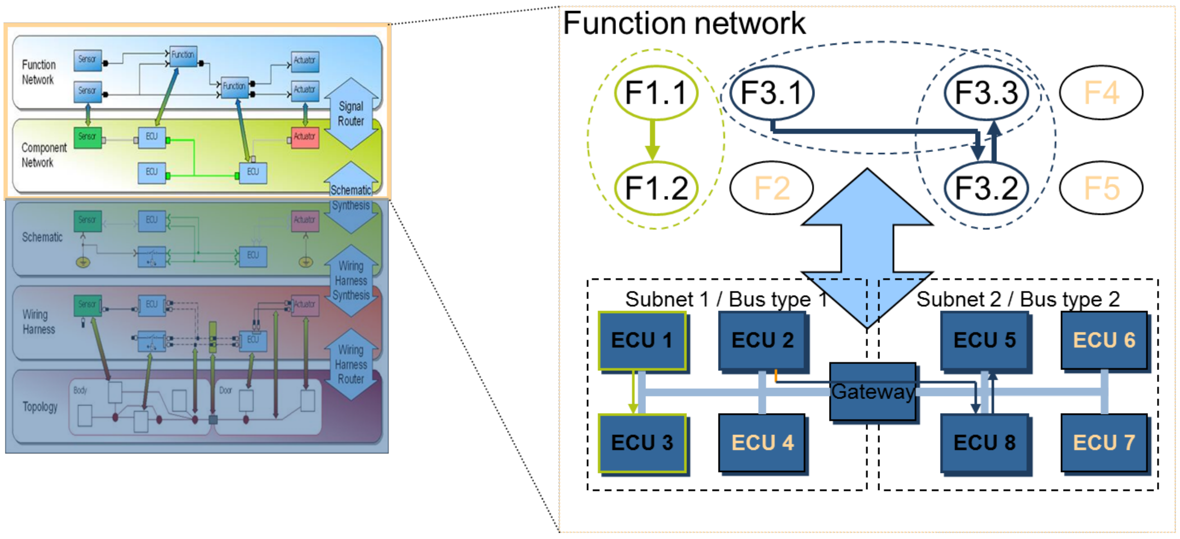

Figure 2.2: Mapping of a function network to a component network

own organization and also, in the end, enables the exchange of AUTOSAR TIMEX compliant model artifacts.

2.4 Hierarchical Timing Description

During the early design phase of an automotive development process the architecture discussion is about high level customer related functions. These functions can be detailed in functional “cause and effect” or “activity” chains, which from a temporal view can be budgeted - justified by customer’s experience. The functional quality and thus technical effort dedicated to the customer’s experience is a business decision of a company.

One example is the reaction time from pressing a button to a reaction, which varies between simply switching (rear window heating) and controlling a motion (e.g. seat or mirror adjustment). The other example is a powertrain or chassis control function which can cause inconveniences like bucking during shifting or braking, and which would not contribute to positive press reviews of a premium vehicle.

From methodological and technical view timing analysis is a tool to assure the desired temporal behavior during the mapping of a functional network to a component network as depicted on Figure 2.2.

Once the major timing budgets for customer related functions is defined and a distribution of functional parts to hardware components is done [1], a more detailed temporal view of a networking architecture can be made. This allows a first assessment of the feasibility of the function distribution in terms of performance and timing. This process

[1] In an AUTOSAR development process a software component (SW-C) is defined with a scope local to the hardware component it is mapped on. It contains a functional contribution to the vehicle function with a system wide scope.

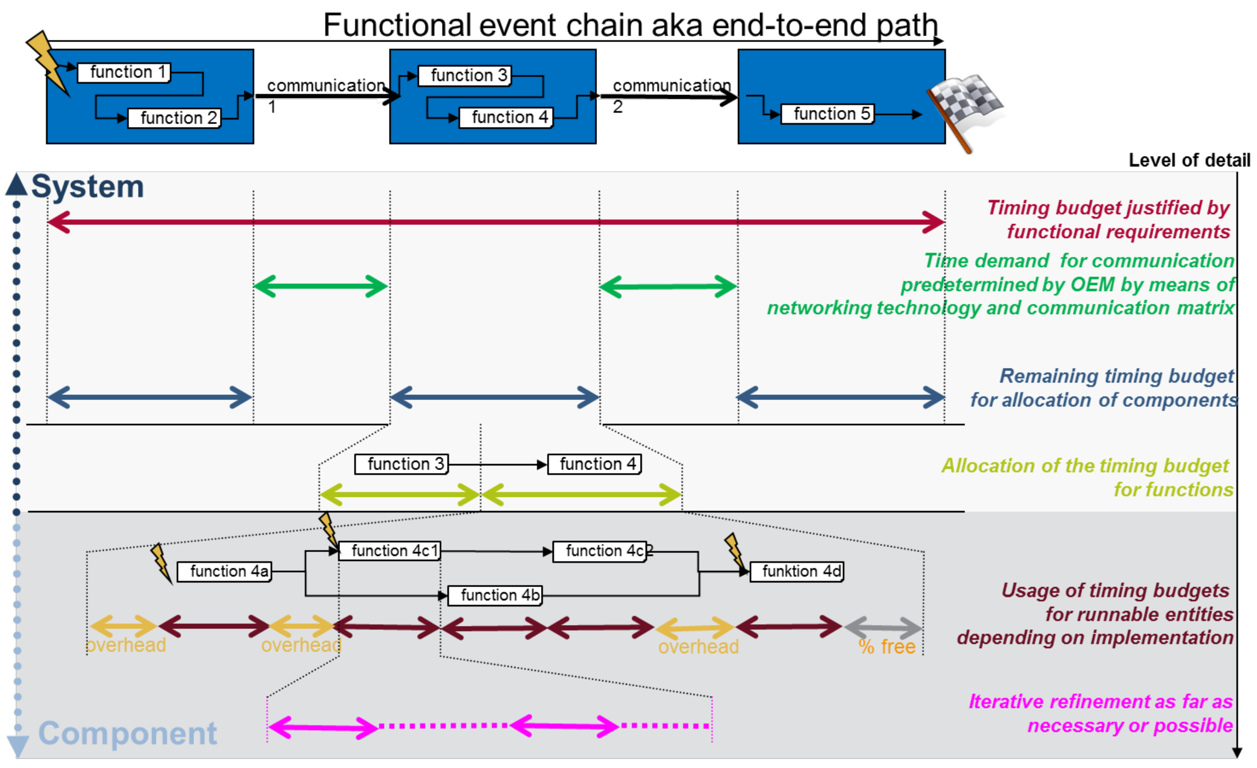
Functional event chain aka end-to-end path
function 1
function 2
communication 1
function 3
function 4
communication 2
function 5
Level of detail
System
Component
Timing budget justified by functional requirements
Time demand for communication predetermined by OEM by means of networking technology and communication matrix
Remaining timing budget for allocation of components
Allocation of the timing budget for functions
Usage of timing budgets for runnable entities depending on implementation
Iterative refinement as far as necessary or possible
function 4a
function 4c1
function 4c2
function 4b
funktion 4d
overhead
overhead
overhead
% free

Figure 2.3: Iterative and hierarchical top down budgeting of timing requirements corresponding to response times

can iteratively be refined during further process steps to have more precise analysis results.

For further understanding, it can be assumed that each function in Figure 2.3 is contained in the compositional scope of an AUTOSAR SW-C, where it is represented as an AUTOSAR runnable entity, shortly often named “Runnable”. Other mapping strategies can also be considered. Regardless of the chosen strategy, the mapping is usually constrained by the functional design choices made at the functional level for timing requirements assessment. For instance, a feasibility test founded on the computation of the utilization (load) of each hardware resource (ECUs, buses), is based on a given allocation of functions on hardware resources. This allocation must be taken into account for the mapping of functions to AUTOSAR SW-Cs in order to avoid the mapping of two functions that are allocated on distinct ECUs on the same AUTOSAR SW-C.

Moreover, in many cases timing demands of physical processes, e.g. the start-up and transient oscillation behavior of electrical actuators, consume more than a few µ s and thus have to be considered carefully.

In a first step the overall timing budget can be split in component-internal and networking parts. As soon as the whole network communication and the type of network are known, the WCRT-analysis of a network can quantify the worst case timing demand for network communication. As shown in the picture above, this divides the overall timing budget in networking budgets and timing budgets for allocation in components (usually ECUs).

This can be enough for an OEM if the development and integration of the component is entirely done by a supplier. In practice a more detailed view considering the timing behavior of a basic software stack and the functions itself is required. Likewise functional relations are more complex, which induces a more complex analysis.

During further analysis steps the end to end timing path or chain of functions can be refined following the concepts of Figure 2.3.

In the following section we introduce methodologies that provide support for the general process described.

2.5 Methodologies for Timing Requirements Decomposition

As previously stated, the AUTOSAR methodology covers the implementation phase of the process of E/E systems development. However, timing requirements are introduced at the very beginning of the development cycle in the form of textual descriptions by OEMs. An extension of the AUTOSAR methodology is then needed to cover the system/functional architecture design phases where the first functional decompositions and timing requirements decomposition must occur. In fact, one of the most challenging activities in the development of systems is determining a system's dimensions in early phases of the development - and the most difficult one is the phase before transitioning from the functional domain to the hard and software domain.

Primarily, two questions must be answered. Firstly, how much bandwidth shall the networks provide in order to ensure proper and timely transmission of data between electronic control units; and secondly, how much processing performance is required on an electronic control unit to process the received data and to execute the corresponding functions. As a matter of fact, these questions can only be completely answered when the system is implemented, including a mapping of signals to network frames and first implementations of functions that are executed on the electronic control units. The reason for this is that one needs to know how much bits per second have to be transmitted and how much instructions shall be executed.

An important aspect that impacts the decisions taken during the task of specifying system dimensions is timing. Especially, information about data transmission frequencies, execution rates of functions, as well as tolerated latencies and required response times provide a framework for performing a first approximation of network and ECU dimensions. This framework allows to continuously refining the system dimensions during system development when more details about the system's implementation are becoming available. The basic idea is to abstract from operational parameters obtained during the implementation phase, like for example measured or simulated execution times of functions, and use them on higher levels of abstraction respectively earlier development phases. And, for new functions as a workaround for missing execution time, an activity called Time Budgeting allows the specification of so called time budgets to functions.

The remainder of this section defines the levels that will be considered for timing requirements decomposition. Then, some generic methodological guidelines will be given for conducting timing requirements refinement between these levels.

2.5.1 Functional and software architectures modeling levels/views

Prior to the AUTOSAR software architecture levels, we can consider two functional architecture modeling levels defined in [9] that are of interest for timing requirements:

- The *Functional Analysis level* which is centered on a logical representation of the system functional units to be developed. Typically based on the inputs of automatic control engineering, system design at this level refines the vehicle level system feature specification by identifying the individual functional units necessary for system boundary (e.g., sensing and actuating functions for the interaction with target physical plant) and internal computation (e.g. feedback control functions for regulating the dynamics of target physical plant). The design focuses on the abstract functional logic, while abstracting any SW/HW based implementation details. Through an analysis level system model, such abstract functional units are defined and linked to the corresponding specifications of requirements (which are either satisfied or emergent) as well as the corresponding verification and validation cases.
- The *Function Design level* provides a logical representation of the system functional units that are now structured for their realizations through computer hardware and software. It refines the analysis level model by capturing the bindings of system functions to I/O devices, basic software, operating systems, communication systems, memories and processing units, and other hardware devices. Again, through a design level system model, the system functions, together with the expected software and hardware resources for their realizations, are defined and linked to the corresponding specifications of requirements (which are either satisfied or emergent) as well as the corresponding verification and validation cases. Moreover, the creation of an explicit design level system model promotes efficient and reusable architectures, i.e. sets of (structured) HW/SW components and their interfaces, hardware architecture, for different functions. The architecture must satisfy the constraints of a particular development project in automotive series production.

The AUTOSAR methodology (see [1] for a general introduction) provides several well defined process steps, and furthermore artifacts that are provided or needed by these steps. Figure 2.4 provides a simplified overview of the AUTOSAR methodology, using the Software & Systems Process Engineering Metamodel notation (SPEM) [3], focusing on the process phases which are of interest for the use of the timing extensions. These represented steps and artifacts are grouped by boundaries in the five following views:

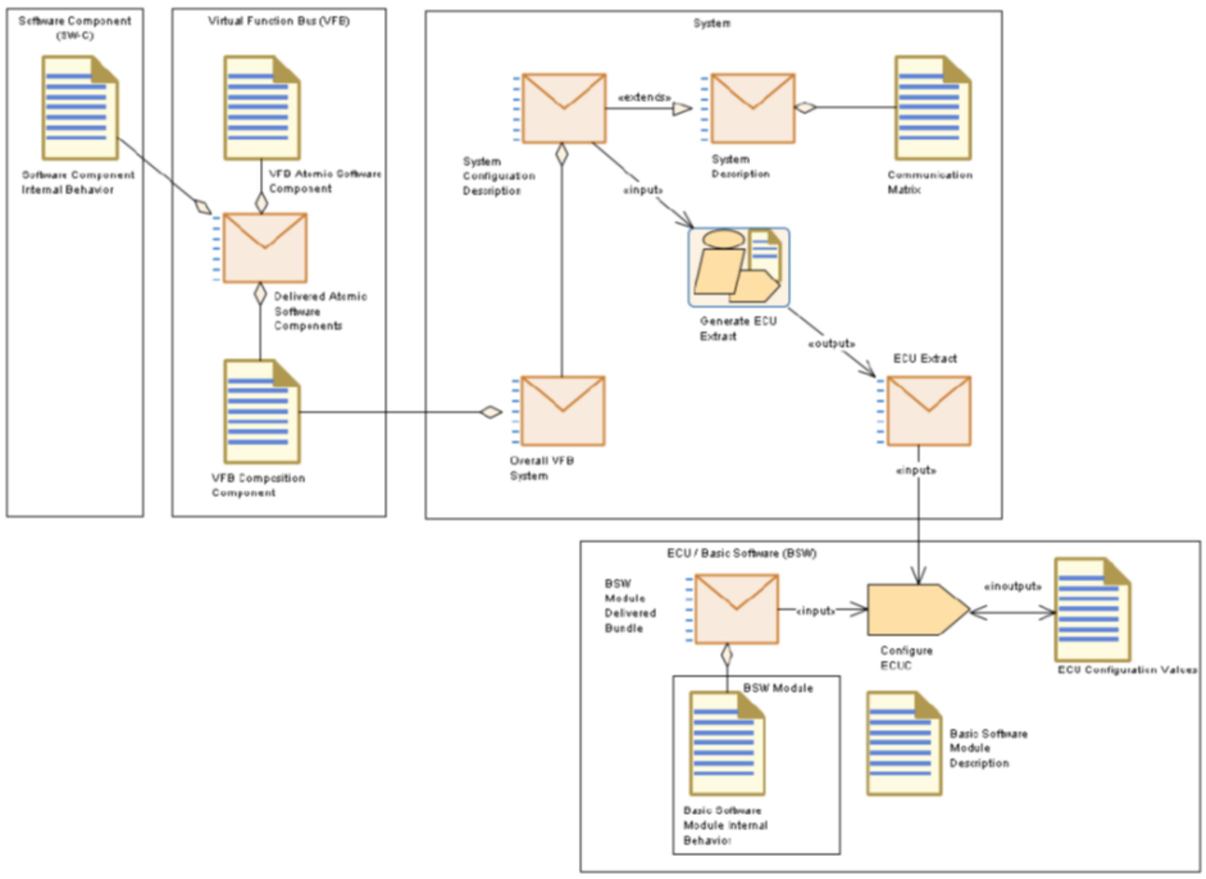
Software Component (SW-C)
Software Component Internal Behavior
Virtual Function Bus (VFB)
VFB Atomic Software Component
Delivered Atomic Software Components
VFB Composition Component
System
System Configuration Description
«extends»
System Description
Communication Matrix
«input»
Generate ECU Extract
«output»
ECU Extract
Overall VFB System
«input»
ECU / Basic Software (BSW)
BSW Module Delivered Bundle
«input»
Configure ECUC
«inoutput»
ECU Configuration Values
BSW Module
Basic Software Module Internal Behavior
Basic Software Module Description

Figure 2.4: SPEM Process model from AUTOSAR Methodology for system design process

- *VfbTiming* This view deals with timing information related to the interaction of SwComponentTypes at VFB level.
- *SwcTiming* This view deals with timing information related to the SwcInternalBehavior of AtomicSwComponentTypes.
- *SystemTiming* This view deals with timing information related to a System, utilizing information about topology, software deployment, and signal mapping.
- *BswModuleTiming* This view deals with timing information related to the BswInternalBehavior of a single BswModuleDescription.
- *EcuTiming* This view deals with timing information related to the EcucValueCollection, particularly with the EcucModuleConfigurationValues.

Further details of these timing views are given in [2].

For each of these views a special focus of timing specification can be applied, depending on the availability of necessary information, the role a certain artifact is playing and the development phase, which is associated with the view.

2.5.2 Guidelines for timing requirements decomposition

The Generic Methodology Pattern (GMP) developed in the TIMMO-2-USE project [10] is an example of a process that defines generic steps for timing requirements refinement. Theoretically, those generic steps are applicable at every level defined in the previous section (including the AUTOSAR levels). Basically, at each abstraction level, GMP takes as input timing requirements and after a sequence of steps gives as output refined timing requirements. GMP defines six main steps. Some of them have been merged in the following short description:

- *Step1 - Create Solution*: describes the definition of the architecture without any timing information. This step can consist in a refinement of an already existing architecture coming from the upper level. Timing requirements shall guide the creation or revision of a solution.
- *Step2 - Attach Timing Requirements to Solution*: describes the formulation of timing requirements in terms of the current architecture. This can imply a transformation of timing requirements coming from the previous level, in order to be compliant with the timing model of the current level of abstraction. For instance in the AUTOSAR SwcTiming view a timing requirement can be modeled with a timing constraint attached to events or event chains.
- *Step 3 - Create, Analyze and Verify Timing Model*: describes the definition of a formalized model for the calculation of specific timing properties of the current architecture. In this step relevant timing analysis methods can be applied to verify timing requirements against calculated timing properties (e.g. maximal load for a bus). If timing requirements are not verified by timing properties resulting from the analysis, the previous tasks shall be iterated until a satisfactory solution is found.
- *Step 4 - Specify and Validate Timing Requirements*: describes the identification of mandatory timing properties and their promotion to timing requirements for the next level.

Chapter 5 gives for each use-cases described in Chapter 3 and Chapter 4 timing properties and methods of interest to insure correct timing requirements decomposition.

2.6 Languages for Timing Requirements Specification

The steps described in the previous section require one or more modeling languages to be used with. The AUTOSAR methodology is based on the AUTOSAR language and its timing extensions. AUTOSAR is the language for the software implementation levels but not applicable at the functional levels (analysis and design). Therefore, in order to insure a complete mode-based approach for timing requirements decomposition, a complementary modeling language for functional levels has to be used. EAST-ADL2 [9] and its timing extension TADL2 [11] allow functional levels specification with precise timing models. Moreover, TADL2 and AUTOSAR Timing extensions are sharing the same base concepts which may facilitate the translation of timing requirements from the functional level to the AUTOSAR level (where timing requirements are expressed with TIMEX).

Therefore, EAST-ADL / TADL is briefly presented as an example of modeling language for the support of the functional levels of a methodology for timing requirements decomposition.

2.6.1 EAST-ADL / TADL

EAST-ADL is an Architecture Description Language (ADL) for automotive embedded systems, developed in several European research projects. It is designed to complement AUTOSAR with descriptions at higher level of abstractions. Aspects covered by EAST-ADL include vehicle features, functions, requirements, variability, software components, hardware components and communication.

TADL2 (Timing Augmented Description Language) language concepts can be used in specific steps of the GMP methodology to describe timing information. TADL2 allows the specification of timing constraints that may express the following timing properties/requirements:

- Execution time (Worst-case, Best-case, Simulated, Measured)
- End-to-end Latency
- Sampling Rates
- Time Budget
- Response Time
- Communication Delay
- Slack
- Repetition pattern
- Synchronization
- ...

TADL2 base concepts are quite equivalent to those of AUTOSAR TIMEX presented in the following section.

2.6.2 Basic concepts of AUTOSAR TIMEX

According to [2], the primary purpose of the timing extensions is to support constructing embedded real-time systems that satisfy given timing requirements and to perform timing analysis/validations of those systems once they have been built up.

The AUTOSAR Timing Extensions provide a timing model as specification basis for a contract based development process, in which the development is carried out by different organizations in different locations and time frames. The constraints entered in the early phase of the project (when corresponding solutions are not developed yet) shall be seen as extra-functional requirements agreed between the development partners.

In such way the timing specification supports a top-down design methodology. However, due to the fact that a pure top-down design is not feasible in most of the cases (e.g. because of legacy code), the timing specification allows the bottom-up design methodology as well.

The resulting overall specification (AUTOSAR Model and Timing Extensions) shall enable the analysis of a system's timing behavior and the validation of the analysis results against timing constraints. Thus, timing properties required for the analysis must be contained in the timing augmented system model (such as the priority of a task, the activation behavior of an interrupt, the sender timing of a PDU and frame etc.). Such timing properties can be found all across AUTOSAR. For example the System Template provides means to configure and specify the timing behavior of the communication stack. Furthermore the execution time of ExecutableEntities can be specified. In addition, the overall specification must provide means to describe timing constraints. A timing constraint defines a restriction for the timing behavior of the system (e.g. bounding the maximum latency from sensor sampling to actuator access).

Timing constraints are added to the system model using the AUTOSAR Timing Extensions. Constraints, together with the result of timing analysis, are considered during the validation of a system's timing behavior, when a nominal/actual value comparison is performed.

The AUTOSAR Timing Extensions provide some basic means to describe and specify timing information: timing descriptions, expressed by events and event chains, and timing constraints that are imposed on these events and event chains. Both means, timing descriptions and timing constraints, are organized in timing views for specific purposes. By and large, the purposes of the Timing Extensions are twofold. The first purpose is to provide timing requirements that guide the construction of systems which eventually shall satisfy those timing requirements. And the second purpose is to provide sufficient timing information to analyze and validate the temporal behavior of a system.

The following subsection describes the main concepts defined in the AUTOSAR Timing Extensions.

2.6.2.1 TIMEX Artifacts

Events refer to locations in systems at which the occurrences of Events are observed. The AUTOSAR Specification of Timing Extensions defines a set of predefined Event types for such observable locations. Those Event types are used in different timing views each corresponding to one of the AUTOSAR views: Virtual Function Bus (VFB) Timing and View; Software Component (SW-C) Timing and View; System Timing and View; Basic Software (BSW) Module Timing and View; as well as ECU Timing and View. In particular, one uses these Events to specify the reading and writing of data from and to specific ports of SW-Cs, calling of services and receiving their responses (VFB Timing); sending and receiving data via networks and through communication stacks (System Timing); activating, starting and terminating executable entities (SW-C Timing and BSW Module Timing); and last but not least calling BSW services and receiving their responses (ECU Timing and BSW Module Timing).

Event Chains specify a causal relationship between Events and their temporal occurrences. The notion of Event Chain enables one to specify the relationship between two Events, for example when an Event A occurs then the Event B occurs, or in other words, the Event B occurs if and only if the Event A occurred before. In the context of an Event Chain the Event A plays the role of the stimulus and the Event B plays the role of the response. Event Chains can be composed of existing Event Chains and decomposed into further Event Chains - in both cases the Event Chains play the role of Event Chain segments.

Timing Constraints imposed on Events. The notion of Event is used to describe that in a system specific Events occur and also at which locations in this system the occurrences are observed. In addition, an Event Triggering Constraint imposes a constraint on the occurrences of an Event, which means that the Event Triggering Constraint specifies the way an Event occurs in the temporal space. The AUTOSAR Specification of Timing Extensions provides means to specify periodic and sporadic Event occurrences, as well as Event occurrences that follow a specific pattern (burst, concrete, and arbitrary pattern).

Timing Constraints imposed on Event Chains. Triggering constraints impose Timing Constraints on Events and their occurrences; the latency and synchronization Timing Constraints impose constraints on Event Chains. In the former case, a constraint is used to specify a reaction and age, for example if a stimulus Event occurs then the corresponding response Event shall occur not later than a given amount of time. And in the latter case, the constraint is used to specify that stimuli or response Events must occur within a given time interval (tolerance) to be said to occur simultaneous and synchronous respectively.

Additional Timing Constraints. In addition to the Timing Constraints that are imposed on Events and Event Chains, the AUTOSAR Timing Extensions provide Timing Constraints which are imposed on Executable Entities, namely the Execution Order Constraint and Execution Time Constraint.

2.7 Conclusions

To apply timing analysis in a comprehensive and holistic way several needs have to be fulfilled:

- All basic terms shall be unified. This means a term like WCRT has the same meaning and comprehensive understanding all over the industry.
- The structure of describing timing aspects shall be unified. For this need AUTOSAR TIMEX delivers an appropriate approach for the implementation driven perspective of AUTOSAR. It fails in higher levels of abstraction, because as soon as no AUTOSAR means like Software Components and Runnable exist, there is no meaning.
- The methodological approach for introducing timing analysis in a timing aware development process shall not be reduced to the definition of TIMEX artifacts referring to AUTOSAR system template artifacts. Additionally information of higher abstraction levels in earlier design phases shall be transferred to AUTOSAR modeling without losing exactness. This requires reference points valid within all phases and levels of abstraction.
- The methodology shall meet the needs of large scale organizations. This means the methodology shall be applicable tailor-made to the processes ruling a particular large scale organization.

3 Timing Analysis for SW-Integration on ECU Level

This chapter outlines use-cases relevant for software integration into a single ECU with respect to timing issues. Network related aspects are covered by chapter 4 and have only an indirect impact on the timing on the ECU level. On the ECU level, the scheduling of tasks and interrupts together with the execution times of the various code fragments define the timing behavior of the overall software for this specific ECU. Depending on the scheduling and the execution times, given deadlines are met or missed. The use-cases in this chapter help to solve problems or tasks which are related to scheduling and/or execution times.

Although speaking of "ECU-level", it is important to bear in mind a single ECU can come with multiple processors each of which comes with its own scheduling. Even multiple cores on one processor are seen more and more often [12] [13]. However, the principles in this chapter still remain valid and can be reflected on each "scheduling entity" (=core).

Typical terms used in this chapter are:

- Execution Time (e.g.: CET, BCET, WCET..), see section 2.1 and 5.2.
- CPU-Load , see section 5.2.
- Interrupt Load, see section 5.2.
- Response Time, see section 5.2.
- Latency, see section 5.2.

3.1 Summary of Use-cases

This chapter describes the use-cases listed in Table 3.1. Figure 3.1 gives an overview.

Section	Use-case	Page
3.2	ECU use-case "Create Timing Model of the entire ECU"	30
3.3	ECU use-case "Collect Timing Information of a SW-C"	32
3.4	ECU use-case "Select an ECU Supplier"	34
3.5	ECU use-case "Validate Timing after SW-C integration"	34
3.6	ECU use-case "Validation of Timing"	37
3.7	ECU use-case "Debug Timing"	39
3.8	ECU use-case "Optimize Timing for an series ECU"	40
3.9	ECU use-case "Optimize Scheduling"	43
3.10	ECU use-case "Optimize Code"	46
3.11	ECU use-case "Verify Timing Model(s)"	47
3.12	ECU use-case "Compare Timing Properties"	49

Table 3.1: List of ECU specific use-cases

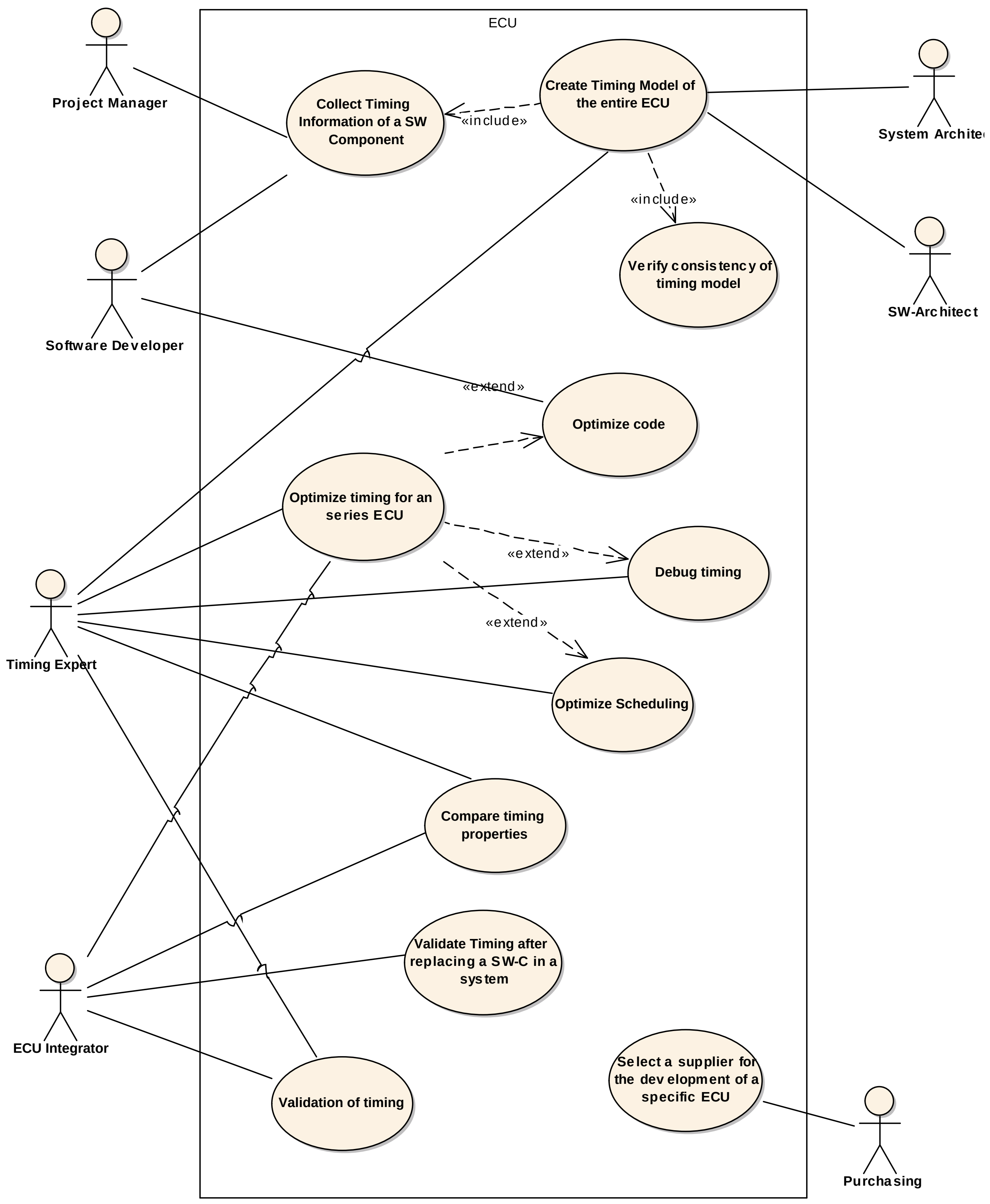
ECU
Project Manager
Software Developer
Timing Expert
ECU Integrator
System Archite
SW-Architect
Purchasing
Collect Timing Information of a SW Component
Create Timing Model of the entire ECU
«include»
«include»
Verify consistency of timing model
«extend»
Optimize code
Optimize timing for an series ECU
«extend»
Debug timing
«extend»
Optimize Scheduling
Compare timing properties
Validate Timing after replacing a SW-C in a system
Select a supplier for the development of a specific ECU
Validation of timing

Figure 3.1: Use-case Diagram: Timing Analysis for ECU

3.1.1 Assumptions

If not otherwise stated the following assumptions hold true for all use-cases described in this chapter:

1. The ECU Extract for a specific ECU is available including the ECU Extract content for System Timing.
2. The VFB View (SW-C Template, hierarchy of SW-Cs) of all SW-Cs mapped onto the specific ECU is available.
3. SW-C descriptions are available
4. The interaction takes place between one OEM and one tier1 supplier
5. All SW-Cs including C source code and object files are available.
6. All required BSW Modules are available including C source code, object files and ECU configuration. (Only valid for use-case “ECU use-case “Validate Timing after SW-C integration”” described in section 3.5 on page 34.)
7. RTE can be generated
8. The contents of this chapter deal solely with the subject matter timing analysis. The assumption made is that any “system” subject to timing analysis is valid from the functional point of view.

Different phases/use-cases in the development of a vehicle system shall be considered, which are described in the following subsections.

3.2 ECU use-case “Create Timing Model of the entire ECU”

This section describes how to generate a timing model for a complete ECU. The difficulties to describe the use-case in are unique manner justified due the fact that Since the OEM and the Tier1 work at different levels of granularity and during different phases in the development process, their views on this use-case also differ. Nevertheless, some basic assumptions are valid for all levels of granularity and all development phases.

As a matter of fact, the creation of a timing model of the entire ECU is one of the important steps to gain a complete system understanding. All other use cases can be seen as somehow connected use-cases, since the existence of a timing model is a precondition in order to execute the steps in other use-cases.

A timing model of an ECU collects all timing data such as timing requirements, timing measurements and also timing relevant configuration data (such as RTE or BSW configuration).

Depending on the development phase, the timing model can be based mainly on assumptions and requirements (requirement timing model) or mainly based on measurements and exiting configuration information. Ideally, both views are accessible in one model.

3.2.1 Characteristic Information

Goal In Context:	Collect all relevant timing information for an selected ECU
Brief Description:	Collect all relevant timing information for a ECU and create a timing model of the entire ECU
Scope:	ECU
Level:	Process
Precondition:	Knowledge about basic functionality of the ECU and basic understanding about the functional requirements of the ECU and the application domain
Success End Condition:	Valid timing information
Failed End Condition:	n.a.
Primary Actor:	ECU responsible person (This might be the project leader or the leading software developer.)
Trigger Event:	Request for timing information during the development process. These can occur at every time during the development.
End Event:	n.a.

Table 3.2: Characteristic Information of ECU UC "Create Timing Model of the entire ECU"

3.2.2 Main Scenario

1. The ECU responsible person collects all available timing data for the specific ECU.
2. Checking of the collected data.
3. Add the retrieved timing data to timing model.
4. The use-case ends with ECU timing model. The timing information will be usable for further work.

3.2.3 Alternative Scenario

Due the different levels of granularity and different phases different scenario extensions possible. In concrete cases the timing expert must choose the matching scenario.

3.2.4 Related Information

Performance Target:	Timing model is usable for next integration level
Precondition:	
Frequency:	On request
Super-use-case:	n.a.
Sub-use-case(s):	n.a.

Secondary Actor(s):	A: System Architect, A: Timing Expert, I: Project Manager (requests deliverables)

Table 3.3: Related Information for ECU UC "Create Timing Model of the entire ECU"

3.2.5 Related methods and properties

In short, in order to create a timing all methods in section 5.3 and all properties in section 5.2 described in the chapter Properties and Methods for Timing Analysis are required to build a timing model of the entire ECU.

Furthermore a appropriate tool chain is required. Such a tool chain must be able to import and export the artifacts generated from different tools during the complete development cycle.

3.3 ECU use-case "Collect Timing Information of a SW-C"

3.3.1 Characteristic Information

Goal In Context:	Collect all relevant timing information an selected SW-C
Brief Description:	Collect all relevant timing information for a SW-C
Scope:	SW-C for a specific target
Level:	Process
Precondition:	Knowledge about basic functionality of the SW-C
Success End Condition:	Valid timing information
Failed End Condition:	n.a.
Primary Actor:	SW-C responsible person (This might be the project leader or the leading software developer.)
Trigger Event:	Request for timing information during the development process. These can occur at every time during the development.
End Event:	n.a.

Table 3.4: Characteristic Information of ECU UC "Collect Timing Information of a SW Component"

3.3.2 Main Scenario

1. The use-case begins when the responsible SW-C person begins the collection of timing information which is usually triggered by an ECU-Integrator request.
2. The SW-C responsible person collects all available timing data for the specific SW-C and collects them in a timing model for SW-C scope.
    - Some estimation about previous and similar project, methods, see section 5.3

- Runtime measurements on runnable level and below, methods, such as Processor-In-The-Loop Simulation (PIL) or Static Worst Case Execution Time Analysis, see section 5.3
- Timing requirements for this SW-C based on functional requirements, for instance
  - Trigger events
  - Latencies
  - Jitters
  - Execution orders
  - Relations to safety-relevant requirements
- Methods: see section 5.3

3. Add retrieved timing data to timing model.
4. The use-case ends with SW-C timing information. The timing information will be usable for SWC integration in the whole system.

3.3.3 Alternative #1 Scenario

At step #2 of the main scenario the sub-steps can be carried out in arbitrary order or might be skipped. The justification for skipping can be missing information at this specific phase in time.

3.3.4 Related Information

Performance Target:	Timing model is usable for next integration level
Precondition:	
Frequency:	On request
Super-use-case:	ECU UC Create Timing Model of the entire ECU see section 3.2
Sub-use-case(s):	n.a.
Secondary Actor(s):	A: System Architect, A: Timing Expert, I: Project Manager (requests deliverables)

Table 3.5: Related Information for ECU UC "Collect Timing Information of a SW Component"

3.3.5 Related methods and properties

- Methods
  - Tracing, see section 5.3

  - PIL, see section 5.3
  - Scheduling Analysis, see section 5.3
  - SchedulingSimulation, see section 5.3
  - Static Worst Case Execution Time Analysis, see section 5.3
- Properties
  - Resource Load, see section 5.2
  - Execution Time, see section 5.2
  - Response Time, see section 5.2
  - Interrupt Load, see section 5.2

3.4 ECU use-case “Select an ECU Supplier”

The following use-cases does not match completely into the AUTOSAR methodology, but it is quite important anyway.

During the order phase for a new ECU some performance key indicators are used to evaluate metrics in order to decide the ECU design (e.g. µC type, memory, frequency). Additionally a supplier fulfilling the overall requirements must be selected.

At this stage timing experts from OEM and Tier1 must work together using some of the use-cases described in this document sketching a rough ECU architecture (regarding hardware and software) with the purpose to show the overall feasibility.

Typically an initial timing model will be available after this work was finished.

3.4.1 Related methods and properties

- Methods
  - Load , see section 5.3
- Properties
  - “Resource Load” , see section 5.2

3.5 ECU use-case “Validate Timing after SW-C integration”

One can also suggest the title “Build up a system using existing, in the sense of most suitable from the timing perspective, SW-Cs”. In this case the objective is that this system satisfies a given time constraint, for example “from sensor to actuator”.

3.5.1 Characteristic Information

Goal In Context:	[Validate timing after a SW-C has been replaced in an existing system]
Brief Description:	In a given/already existing system one of the SW-Cs is replaced by a new version. The new version may consist of 1) the same number of RE as the previous version (but different implementations), or 2) a different number of REs than the previous version (fewer or more REs). From a timing analysis point of view it must be ensured that the new version still satisfies the given timing constraints. This requires to conduct 1) a response time analysis, and/or 2) a scheduling analysis which indicates that the given timing constraint is satisfied
Scope:	ECU
Level:	Activity or Task
Precondition:	1. Definition of relevant timing constraints, which should be satisfied. 2. The SW-C/s has/have been mapped to a specific ECU which is the one subject to timing analysis. 3. The SW-C has been integrated from a structural point of view, which means that the "REs" have been mapped to the corresponding tasks and properly positioned within the tasks. Work product: ECU Configuration including OS configuration (Task Model and Task Parameters) and RTE configuration (RTE Event to Task Mapping) 4. All port interfaces are valid/compatible and all ports have been connected with the corresponding ports of the SW-Cs the SW-C subject to integration is exchanging data with. Work product: SW-C Description, System Description 5. System/ECU timing model of the system is available [StKu: Shall one state the granularity of the timing model here?] Work product: ECU Timing, System Timing
Success End Condition:	Timing analysis indicates that the given timing constraint defined in the precondition is satisfied (in all conditions). Timing analysis is ok: E.g. System Description is updated, Latency Timing Constraint with reaction semantics – Reaction Time Constraint. $\rightarrow$ Timing Guarantee. What is about [timing] measurements?
Failed End Condition:	Neither response time analysis nor scheduling analysis indicate that the given timing constraint are satisfied. Timing measurements indicate that the timing constraint are violated more times than accepted by the customer.
Primary Actor:	ECU Integrator
Trigger Event:	SW-C Package including the SW-C Description, SW-C Implementations, VFB Timing and SW-C Timing are becoming available ... are received from SW-C supplier.
End Event:	System is released.

Table 3.6: Characteristic Information of ECU UC "Validate Timing after SW-C integration"

3.5.2 Main Scenario

1. The use-case begins when the actor receives the SW-C package together with a change order from the Change Control Board (CCB).
2. The actor performs the structural integration.
3. The actor replaces the timing model of the current version of the SW-C by the timing model of the new version of the SW-C [VFB or/and SW-C Timing] in the system's timing model.
4. In essence, this is simply referencing an event chain in the new SW-C's timing model from within the timing model of the system the new SW-C is "integrated".
5. The actor determines the differences between the previous and new SW-C. What does this tell the integrator?
6. The actor conducts the timing analysis
7. The actor reviews the result of the timing analysis and concludes that the given timing constraint is satisfied.
8. The actor marks the work products as valid, namely the timing model and the analysis report.
9. The use-case ends.

3.5.3 Alternative #1 Scenario

No scenario extensions identified.

3.5.4 Related Information

Performance Target:	No performance key indicators identified for this use-case.
Precondition:	No constraints that may apply during the course of the use-case are identified.
Frequency:	Whenever a new version of a SW-C that is part of a system is becoming available and the decision has been taken to update the SW-C in the existing system.
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	A: Quality Manager, S: Timing Expert/Analyst, I: [System] Project Manager, Architect

Table 3.7: Related Information for ECU UC "Validate Timing after SW-C integration"

3.6 ECU use-case “Validation of Timing”

3.6.1 Characteristic Information

Goal In Context:	Validate the timing of a defined system
Brief Description:	Validate the timing to ensure the schedulability of a system and that all given timing constraints are satisfied. The validation of the timing can be conducted via various timing analysis methodologies e.g. response time analysis and/or schedulability analysis depending on the nature of the timing constraints and/or expected level of confidence. If an implementation of the system exists the timing can be also measured and validated on basis of timing traces. Validate the timing by conducting a scheduling analysis.
Scope:	ECU
Level:	Activity
Precondition:	At least one event chain with a stimulus specifying the receipt of a signal and a response specifying the transmission of a signal on a connected bus/network and a latency timing constraint is imposed on this event chain. The mapping of runnable entities to task is executed and RTE and OS configuration are available.
Success End Condition:	The schedulability analysis and the tracing confirm the given timing constraints are satisfied.
Failed End Condition:	The schedulability analysis and the tracing show the given timing constraints are violated.
Primary Actor:	SW-C responsible person (This might be the project leader or the leading software developer.), Timing-Expert on ECU Level, ECU Integrator
Trigger Event:	The value of the latency timing constraint changes. The RTE and/or OS Configuration changes.The primary actor decides to validate the timing of an existing system e.g.: • The architect validates the timing of a system on a specific hardware • The integrator validates the timing of a system after an integration step • The software engineer performs measurements on the real system to validate the timing
End Event:	System is released.

Table 3.8: Characteristic Information of ECU UC “Validation of timing”

3.6.2 Main Scenario

1. The use-case begins With the trigger event
2. The actor conducts the timing analysis with: 1) Response Time Analysis, 2) Scheduling Analysis, or 3) Measurement. → sub-use-cases
3. The actor reviews the result of the timing analysis and concludes that the given timing constraint is satisfied.

4. The actor marks the work products as valid, namely the timing model and the analysis report.
5. The use-case ends.

3.6.3 Alternative #1 Scenario

No scenario extensions identified.

3.6.4 Related Information

Performance Target:	No performance key indicators identified for this use-case.
Precondition:	No constraints that may apply during the course of the use-case are identified.
Frequency:	Whenever the decision has been taken to validate the timing of the existing system.
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	A: Quality Manager, S: Timing Expert/Analyst, I: [System] Project Manager, Architect

Table 3.9: Related Information for ECU UC "Validation of timing"

3.6.5 Related methods and properties

- Methods
  - Tracing, see section 5.3
  - PIL, see section 5.3
  - Scheduling Analysis, see section 5.3
  - Scheduling Simulation, see section 5.3
- Properties
  - Resource Load, see section 5.2
  - Execution Time, see section 5.2
  - Response-Time, see section 5.2
  - Interrupt-Load, see section 5.2

3.7 ECU use-case “Debug Timing”

Whenever an ECU shows unexpected behavior like sporadic system crashes or data inconsistencies, a timing issue could be the cause of the problem. Tracking the problem down with conventional debug methods can be very painful and time consuming. This is also true even if a certain problem is very obviously related to timing.
Before any problem can be *solved*, it has to be understood. This is what timing debugging is about: understanding a timing problem that is present on a real ECU. Once the problem is understood, the solution finding and solving follows, see section 3.8 “ECU use-case “Optimize Timing for an series ECU”” on page 40.

3.7.1 Characteristic Information

Goal In Context:	Understand a (timing) problem and isolate the cause of the problem.
Brief Description:	Using dedicated timing debugging methods (see chapter 5), debug a problem and find out, if it is a timing problem. If so, track down the cause of the problem so that it is completely understood. This makes solving the problem possible in a next step.
Scope:	ECU
Level:	Activity
Precondition:	A running system
Success End Condition:	Problem understood, cause of the problem isolated. Artifacts: documentation describing the problem, e.g. schedule traces
Failed End Condition:	• problem not understood or • problem is not caused by faulty timing or • problem is not reproducible or based on the data of previous occurrences not sufficiently analyzable.
Primary Actor:	Timing Expert on ECU-level
Trigger Event:	• Sporadic system crashes • Sporadic data inconsistencies • Unexpected overload scenarios • Unexpected delays/jitters • ...
End Event:	All happy

Table 3.10: Characteristic Information of ECU UC “Debug Timing”

3.7.2 Main Scenario

1. The use-case begins when the main-actor is confronted with a timing problem or a problem that could be caused by timing on a real ECU.
2. If the problem is reproducible, use the real system for timing debugging. If not, try to make it reproducible. When this implies a modification of the system, reflect the changes in all later steps on the results (are all assumptions still true even

with the modification?). If the problem is still not reproducible, try using the data of previous occurrences

3. Debugging using dedicated timing analysing methods see chapter 5
4. Isolate the problem
5. If the cause is trivial, fix it and test it. If not, hand the results over to a use-case finding a more sophisticated solution, e.g. ECU UC08 Optimize Timing for an series ECU.
6. The use-case ends.

3.7.3 Alternative #1 Scenario

No scenario extensions identified.

3.7.4 Related Information

Performance Target:	n.a.
Precondition:	n.a.
Frequency:	Whenever a not trivial problem is detected in the ECU.
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	S: Software developer

Table 3.11: Related Information for ECU UC "Debug Timing"

3.7.5 Related methods and properties

- Methods
  - Extract Timing Traces, see section 5.3
  - Evaluate Timing Traces, see section 5.3
- Properties
  - "Resource Load", see section 5.2
  - Interrupt Load, see section 5.2

3.8 ECU use-case "Optimize Timing for an series ECU"

The main idea behind this use-case is to optimize the timing behavior of a working ECU. Sometimes the resource consumption is higher than expected or it is required to integrate further SW-C into the ECU.

3.8.1 Characteristic Information

Goal In Context:	Remove timing violations (optimize resource consumption, data consistency, reduce jitter,..) or minimize resource consumption
Brief Description:	Based on timing requirements, while taking all timing constraints into account the overall timing architecture for an ECU is optimized
Scope:	ECU
Level:	Activity
Precondition:	A running system and/or ideally a useful system description (timing-model)
Success End Condition:	Found a better solution which fulfill all timing and resources requirements (even with additional functionality if applicable). Artifacts: • (New Schedule • Updated Timing model) and/or • (optimized code • New memory layout • New code generator options • New compiler options)
Failed End Condition:	No solution found
Primary Actor:	Timing expert on ECU-level, ECU integrator
Trigger Event:	Presence of timing violation, resource bottlenecks or the need to add further functionality which does not fit into the current version of the ECU SW
End Event:	All happy

Table 3.12: Characteristic Information of ECU UC “Optimize Timing for an series ECU”

3.8.2 Main Scenario

1. The use-case begins when the main-actor becomes aware of timing violations or the need to add more functionality into an already heavily loaded system.
2. Analyze the current system (validate the timing of the system, see ECU use-case “Debug Timing”) and find hot-spots. These are situations in the schedule, where either timing requirements or resource consumption constraints are violated already or would be if more load was added.
3. Definition of the optimization goal(s) on a per hot-spot basis.
4. Analysis of available options in order to relax the hot-spots. These options can include modification of the scheduling configuration (including the runnable to task mapping, the runnable sequence/order inside tasks, the allocation of task to different cores, the partitioning of tasks into smaller entities for load balancing, the change of priorities/offsets/recurrences of task) and/or code optimization (including the re-mapping of data to memory). For each option, continue with the corresponding sub-use-case.

5. The actor performs a trade-off analysis to weight the different possibilities for the optimization of the timing and its impact on the system
6. The actor decides for a modification and changes the timing-model/the code of the system
7. The actor validates the timing of the ECU
8. Verification against optimization goal
9. The use-case ends.

3.8.3 Alternative #1 Scenario

No scenario extensions identified.

3.8.4 Related Information

Performance Target:	Different performance key indicators possible: • load balancing (distribute load on time axis, load balancing over different cores) • minimize systematically response times, jitters etc. • reduce number of preemptions (and thus reduce OS overhead) • reduce number of migration (and thus reduce migration overhead) • reduce resource consumption (inter-core communication, memory (buffer sizes), load) • reduce number of scheduling interrupts • reduce waiting times See also chapter metrics 5
Precondition:	n.a.
Frequency:	Whenever a timing violation is detected in the ECU, an additional functionality is added/expected or existing functionality is modified
Super-use-case:	n.a.
Sub-use-case(s):	ECU use-case "Optimize Scheduling" and ECU use-case "Optimize Code"
Secondary Actor(s):	• S: Software developer • S: SW-Architect

Table 3.13: Related Information for ECU UC "Optimize Timing for an series ECU"

3.8.5 Related methods and properties

- Methods

- Tracing, see section 5.3
- PIL, see section 5.3
- Scheduling Analysis, see section 5.3
- Scheduling Simulation, see section 5.3

• Properties

- Resource Load, see section 5.2
- Execution-TIME, see section 5.2
- Response-TIME, see section 5.2
- Interrupt-Load, see section 5.2

3.9 ECU use-case “Optimize Scheduling”

The main idea behind this use-case is the optimization of an existing schedule of a working ECU with a defined goal such as “remove local overload” or “reduce response time of task xyz”.

3.9.1 Characteristic Information

Goal In Context:	Fulfill predefined optimization goal
Brief Description:	Find a modified schedule configuration which fulfills the goal without causing new timing violations or violates resource constraints
Scope:	ECU
Level:	Activity
Precondition:	A running system and/or ideally a useful system description (timing-model)
Success End Condition:	Found a modified schedule configuration which fulfills the goal without causing new timing violations. Artifacts: • New Schedule, better than the original schedules with respect to a specific metric, see chapter 5.2 • Updated Timing model
Failed End Condition:	No solution found
Primary Actor:	Timing expert on ECU-level
Trigger Event:	Some timing violation or other needs for timing optimization
End Event:	All happy

Table 3.14: Characteristic Information of ECU UC “Optimize Scheduling”

3.9.2 Main Scenario

1. The use-case begins when the main-actor is confronted with a certain optimization goal regarding the scheduling

2. Analysis of available options e.g. modification of the runnable to task mapping, the runnable sequence/order inside tasks, the allocation of task to different cores, the partitioning of tasks into smaller entities for load balancing, the change of priorities/offsets/recurrences of task
3. The actor performs a trade-off analysis to weight the different possibilities for the optimization of the schedule and its impact on the system
4. The actor decides for a solution and modifies the timing-model/code of the system.
5. The actor validates the timing of the ECU by conducting response time analysis, scheduling analysis or measurements
6. Verification against optimization goal
7. The use-case ends.

3.9.3 Alternative #1 Scenario

No scenario extensions identified.

3.9.4 Related Information

Performance Target:	Different performance key indicators possible: • load balancing (distribute load on time axis, load balancing over different cores) • minimize systematically response times, jitters etc. • reduce number of preemptions (and thus reduce OS overhead) • reduce number of migration (and thus reduce migration overhead) • reduce resource consumption (inter-core communication, memory (buffer sizes), load) • reduce number of scheduling interrupts • reduce waiting times See also 5.2
Precondition:	n.a.
Frequency:	Whenever a timing violation is detected in the ECU, an additional functionality is added/expected or existing functionality is modified
Super-use-case:	ECU use-case "Optimize Timing for an series ECU"
Sub-use-case(s):	• Conduct Response Timing Analysis • Conduct Scheduling Analysis • Conduct Measurements
Secondary Actor(s):	• S: Software developer • S: SW-Architect

Table 3.15: Related Information for ECU UC "Optimize Scheduling"

3.9.5 Related methods and properties

- Methods
  - Tracing, see section 5.3
  - PILTracing, see section 5.3
  - Scheduling AnalysisTracing, see section 5.3
  - Scheduling SimulationTracing, see section 5.3
- Properties
  - Resource Load, see section 5.2
  - Execution Time, see section 5.2
  - Response Time, see section 5.2

3.10 ECU use-case “Optimize Code”

Since the code and the deployment of code has a huge impact on timing, different optimization activities can be performed. The scope of the optimization can be different (memory, run-time, safety, re-usability, easy to understand, etc.), however in the scope of this document, the optimization scope is limited to timing effects. But it has to take into account, that such timing optimization influence other aspects of the system, such as memory and reusability and that such optimization is constrained by safety or security aspects

3.10.1 Characteristic Information

Goal In Context:	Optimize the code with respect to timing. Typically: minimize the WCET, the average execution time or both.
Brief Description:	Based on timing requirements optimize the overall timing architecture for an ECU
Scope:	ECU
Level:	Task
Precondition:	Code available (ideally compilable, linkable and executable on the target platform)
Success End Condition:	Found a better code which respect to timing. Possible artifacts: • Optimized code • New memory layout • New code generator options • New compiler options
Failed End Condition:	No solution found
Primary Actor:	Software developer
Trigger Event:	Need for timing optimization
End Event:	All happy

Table 3.16: Characteristic Information of ECU UC “Optimize Code”

3.10.2 Main Scenario

1. The use-case begins when the main-actor determines to optimize a certain code fragment (a task, an interrupt, a runnable, a function or part of a function)
2. Definition optimization goal
3. Analysis of available options
4. Modification
5. Test
6. Verification against optimization goal
7. The use-case ends.

3.10.3 Alternative #1 Scenario

No scenario extensions identified.

3.10.4 Related Information

Performance Target:	Methods:	• measurement/tracing • static code analysis • review (including output generated by compiler) • mapping of symbols to memory • ...
Precondition:	n.a.	
Frequency:	Whenever a timing optimization in the ECU is needed.	
Super-use-case:	ECU use-case "Optimize Timing for an series ECU"	
Sub-use-case(s):	n.a.	
Secondary Actor(s):	• S: Timing expert • S: SW-architect	

Table 3.17: Related Information for ECU UC "Optimize Code"

3.10.5 Related methods and properties

- Methods
  - Tracing, see section 5.3
  - PIL, see section 5.3
- Properties
  - Resource Load, see section 5.2
  - Execution-TIME, see section 5.2
  - Response-TIME, see section 5.2
  - Interrupt-Load, see section 5.2
  - Code-Metrics, see section 5.2

3.11 ECU use-case "Verify Timing Model(s)"

Any model based design or verification process must undergo a model check to make sure, the model represents reality with respect to the relevant properties.
Example 1: a perfect static code analysis tool for WCET calculation on code level can easily produce wrong results (false positives!) when not configured correctly.
Example 2: a perfect static scheduling analysis tool for WCRT calculation on ECU

level can easily produce wrong results (false positives!) when the real ECU suffers an operating system bug.

In both examples, the models and the model based algorithms might be absolutely correct but still they produce false positive results. In other words: a software might pass model based verification and still show drastic timing defects.

3.11.1 Characteristic Information

Goal In Context:	Verify that a model based method reflects the real system with respect to the relevant properties.
Brief Description:	Based on methods which profile the behavior of the real ECU/the real code, the results of the model based approach gets cross-checked.
Scope:	ECU or code
Level:	Task
Precondition:	A running system or executable code or code fragments
Success End Condition:	A comparison of measured/traced timing metrics with the timing metrics provided by the model based approach shows the model based approach generates plausible results.
Failed End Condition:	Measurement/tracing uncovered timing behavior beyond the worst case results proclaimed by the model based approach.
Primary Actor:	Timing expert on ECU-level or code-level
Trigger Event:	Model based approaches are used and the real system is (or becomes) available
End Event:	All happy

Table 3.18: Characteristic Information of ECU UC "Verify Timing Model(s)"

3.11.2 Main Scenario

1. The use-case begins when model based approaches are or become available and the real system is or becomes available. Typically, this will happen in either of two set-ups: for an existing system, model based approaches are added or model based approaches are used in an early development phase and the (real) system becomes available.
2. Produce the metrics with the model based approach, e.g. WCET or WCRT
3. Measure or trace the comparable metrics with the real system, e.g. max. CET or max. RT
4. Compare the model based results with the measured or traced results. All the "worst case" results produced by the model based approaches must be "worse" than the observed results. For comparing timing properties, see also ECU use-case "Compare Timing Properties".
5. The use-case ends. However, this approach cannot guarantee the correctness of the model, because the test vectors the measurements were based on were not

covering a case which would have uncovered a problem with the model. But at least it provides an additional and very important check.

3.11.3 Alternative #1 Scenario

No scenario extensions identified.

3.11.4 Related Information

Performance Target:	n.a.
Precondition:	n.a.
Frequency:	Once model based approaches are added while the real system is available or when the real system becomes available and model based approaches are used already. Afterwards, the cross check should be done again at least for major software releases of the system's software.
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	Software developer

Table 3.19: Related Information for ECU UC "Verify Timing Model(s)"

3.11.5 Related methods and properties

- Methods
  - Tracing, see section 5.3
  - PIL, see section 5.3
- Properties
  - n.a.

3.12 ECU use-case "Compare Timing Properties"

Compare two sets of timing properties. These might be

- obtained by different timing analysis techniques, e.g. simulation and measurement,
- obtained by analyzing different versions of the ECU software or
- related to different constraint types, e.g. "requirement" and "guarantee". See chapter 5 "Properties and Methods for Timing Analysis" on page 61 for details.

3.12.1 Characteristic Information

Goal In Context:	Compare timing properties obtained by means of response time analysis and/or scheduling analysis with measured time values.
Brief Description:	Unlike measurement the validation of timing can be conducted early in the design phase by response time analysis and/or scheduling analysis based on legacy software components and/or budgets. In order to validate a) the execution times of individual software components and b) the resulting timing properties like response times gathered early in the design phase the timing properties have to be compared with measured time values.
Scope:	ECU
Level:	Implementation / Integration / Validation
Precondition:	A timing model of the ECU exists (See ECU use-case "Create Timing Model of the entire ECU") and an implementation of the system exists
Success End Condition:	The timing properties of the timing model are compared to measured values • the elements of the timing model which differ from the measured values are updated or complemented with the measured values • a new timing model with the measured values is generated
Failed End Condition:	The timing properties cannot be compared
Primary Actor:	Timing expert on ECU-level, ECU integrator
Trigger Event:	The primary actor decides to compare timing properties
End Event:	System is released.

Table 3.20: Characteristic Information of ECU UC "Compare Timing Properties"

3.12.2 Main Scenario

1. 1. The use-case begins when the actor decides to compare timing properties.
2. The actor measures the timing properties of the system
3. The actor compares timing properties obtained by response time analysis and/or scheduling analysis with measured values
   - The actor generates a timing model out of the measured values and compares the timing models with each other
   - The actor compares selected elements of the timing model with the measured values
4. The actor reviews the result of the comparison and chooses one of the following actions
   - the elements of the timing model which differ from the measured values are updated or complemented with the measured values
   - a new timing model with the measured values is generated

   See also section 5.3]

5. The use-case ends.

3.12.3 Alternative #1 Scenario

No scenario extensions identified.

3.12.4 Related Information

Performance Target:	No performance key indicators identified for this use-case.
Precondition:	Target code exists which can be measured
Frequency:	Whenever the decision has been taken to compare timing properties
Super-use-case:	n.a.
Sub-use-case(s):	• Conduct Response Timing Analysis • Conduct Scheduling Analysis (e.g. find a schedule for a system) • Conduct Measurements • Create Timing Model
Secondary Actor(s):	• A: Quality • S: Timing Expert/Analyst • I: [System] Project Manager, Architect

Table 3.21: Related Information for ECU UC "Compare Timing Properties"

3.12.5 Related methods and properties

- Methods
  - Tracing, see section 5.3
  - PIL, see section 5.3
  - Scheduling Analysis, see section 5.3
  - Scheduling Simulation, see section 5.3
- Properties
  - n.a.

4 Timing Analysis for Networks

This chapter outlines use-cases relevant for network communication. ECU related aspects were covered by chapter 3 and have only an indirect impact on the timing on the ECU level.

On the network level, the so called communication matrix, which contains the PDUs/frames with their specific parameters (e.g. size of the communication signals, IDs, transmission pattern), together with the communication protocols (e.g. CAN, LIN, FlexRay) define the timing behavior on each individual communication network.

Depending on the amount of traffic to be transmitted on the network and on the communication paradigm, a network configuration does or does not satisfy given performance constraints, such as maximum latency for a PDU/frame or maximum load on the bus.

The use-cases described in what follows will highlight some problems and solutions related to the design of communication networks.

Typical terms used in this chapter are:

- Load, see section 4.2, 4.3, 4.4 and 5.2
- Latency, see section 5.2
- Response Time, see section 5.2

4.1 Summary of Use-cases

This chapter describes the use-cases listed in Table 4.1.

Section	Use-case	Page
4.2	NW use-case “Integration of a Distributed Function”	52
4.3	NW use-case “Design of the new developed Network”	55
4.4	NW use-case “Remapping an existing Function”	58

Table 4.1: List of network specific use-cases

4.2 NW use-case “Integration of a Distributed Function”

This use-case focuses on integrating a distributed function into a networked architecture.

Goal In Context:	Feasible integration of a new function into an existing networked architecture.

Brief Description:	Considering an existing E/E automotive architecture consisting of several ECUs connected via several buses, it is required to integrate the communication demands of the new functionality into network such that the legacy and additional communication entirely fulfills the performance constraints. The buses implement different communication protocols (e.g. CAN, LIN, Flexray, etc.). The communication on each bus is specified by a communication matrix containing the PDUs/frames with their protocol specific parameters and the communication behavior (timing parameters).
Scope:	System, System Timing
Level:	Activity
Precondition:	For the new communication following properties are defined: • The size of the communication signals (SW-C Template / GenericStructureTemplate). • The transmitter and receiver nodes / system mapping • The PDU/Frame timing/triggering • Required bandwidth Additionally, for the communication on the networked is defined a set of performance constraints: • Maximum busload on each bus • Maximum latency for each PDU/Frame Furthermore, a specification of the communication paradigm for the existing bus controllers is defined, e.g. the CAN controller sends CAN-frames with different identifiers via a queue (priority ordered or FIFO), while different instances of the same frame are sending via a register (always send the newest frame instance). It is assumed that the current network configuration satisfies the performance constraints.
Success End Condition:	The new communication was completely defined and the performance constraints are satisfied.
Failed End Condition:	The new communication cannot be defined without violating at least one performance constraint.
Primary Actor:	System Architect / Network Architect
Trigger Event:	New vehicle function
End Event:	Update of the communication matrix.

Table 4.2: Characteristic Information of NW UC "Integration of a distributed function"

4.2.1 Main Scenario

For the sake of clarity following notations are used: The existing networked architecture consists of several ECUs (ECU1, ECU2, a.s.o) connected via multiple communication buses (denoted Bus1, Bus2, a.s.o) and one or multiple gateways. The new distributed function that has to be integrated into the existing network is denoted as F. F consists of multiple Software Components (SW-Cs) which can be mapped on one or multiple ECUs. Each SW-C has it own communication interfaces through which it sends or receive information, i.e. communication signals packet in PDUs/frames.

1. The network architect maps the new communication required by F to the existing PDUs/frames according to the timing information and the transmitter/receiver relation.

2. Depending on the sender/receiver relation it might be necessary to additionally route PDUs/frames on several buses. This happens when SW-Cs of F are mapped to several ECUs which are connected to different buses, e.g. on ECU1 on Bus1 and on ECU2 on Bus2.

3. Analysis 1: The bus load analysis describes the average use of the bus bandwidth. The bus load analysis has to consider the additional traffic generated by the communication required by F. The bus load analyis has to be applied for each bus on which ECUs accommodate SW-Cs of F. The bus load analysis requires the data size and the average timing of the PDUs/frames. The output of the analysis is the timing property GENERIC PROPERTY Load. The bus load property is used to initially approve the traffic on each communication bus. The present value of the timing property load obtained for every single bus is compared to the maximum acceptable load on that bus. For typical constraints for the bus load see section 4.2.3.

4. Analysis 2: In order to approve the communication after integrating F into the existing networked architecture, latency constraints have to be also verified on each bus for all PDUs/frames of the legacy and of the new traffic. The latency analysis computes the timing properties of the PDUs/frames under the resource sharing protocol. The results of the analysis are timing properties such as response times of the PDUs/frames GENERIC PROPERTY Latency / Response Time (or SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN) specific for CAN buses), the jitter of the PDUs/frames, or the blocking times due to arbitration. The values of the timing properties are compared to the specified constraints. For typical constraints on the PDUs/frames response times see section 4.2.3.

5. Analysis 3: In case that the PDUs/frames associated to F are routed by one or more gateways, the routing times are relevant for the end-to-end timing. The routing time analysis of the routed PDUs/frames provides the delay values due to routing engine. These usually consist of buffering delay and arbitration delay. The results of the routing time analysis are the routing response times, the blocking times due to buffering and arbitration, or the memory requirements for buffering. The values obtained for these properties are compared to the specified constraints. For typical constraints for routing times see section 4.2.3.

4.2.2 Alternative #1 Scenario

At step #1 of the main scenario, if the new communication exceeds the size of the unused space in the existing PDUs/frames, new PDUs/frames are defined according to the timing properties of the signals. The impact of the new traffic on the existing communication has to be minimized. The methodology continues with Step 2 in the Main Scenario.

4.2.3 Performance/Timing Constraints

The maximum load on each bus shall not exceed a certain bound, for example 60%.

For each frame/PDU, the worst-case response time shall not exceed the cycle time of the frame.

Routing times in gateways have to be short. Typically, for each frame/PDU the routing time shall not exceed for example 10% of the cycle time of the frame.

4.2.4 Related Information

Performance Target:	Bus load, Response Times, Routing Times
Precondition:	n.a.
Frequency:	Regular
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	Network Architect: Support/Approve Timing Expert: Support ECU Integrator: Informed

Table 4.3: Related Information of NW UC "Integration of a distributed function"

4.3 NW use-case "Design of the new developed Network"

Goal In Context:	Design and feasible integration of a (domain specific) network into existing automotive platform architecture. Possible variants: • Complete new design of the (on-board network) (total automotive network); • A replacement of an old partial network by a new partial network maybe under use of unaltered legacy ECUs (beside the network connectors). This network is connected to the residual on-board network by a gateway.
Brief Description:	Regarding an existing E/E automotive architecture consisting of several ECUs connected via several legacy networks, it is required to design and to integrate a new designed network. The new designed network shall be connected to the residual on-board network via a gateway. Therefore the intra-communication within the new network and the inter-communication between different networks have to considered. Further, this new network shall be stable extensible in a-priori predictable way, i.e. it shall be possible to analyze the network with respect to all present and future communication constraints. The new network implements a communication protocols (e.g. CAN, LIN, Flexray, etc.) and possesses sufficient bandwidth to cover all communication requirements. The communication on the network is specified by a communication matrix containing the PDUs/frames/packages with their protocol specific parameters and the communication behavior (timing parameters).
Scope:	System, System Timing
Level:	Activity

Precondition:	For the new communication following properties are defined: • The size of the communication signals (SW-C Template / GenericStructureTemplate). • The transmitter and receiver nodes / system mapping • The PDU/frame/package timing/triggering • Required bandwidth • The residual on-board network including gateways and communication matrix Additionally, a set of performance constraints is defined for the communication on the network: • Maximum load on each network • Maximum latency for each PDU/frame/package Furthermore, a specification of the communication paradigm for the existing network controllers is defined, e.g. the CAN controller sends PDUs/frames with different identifiers via a queue (priority ordered or FIFO), while different instances of the same PDU/frame are sending via a register (always send the newest PDU instance). It is assumed that the current (residual) on-board network configuration satisfies the performance constraints.
Success End Condition:	The communication on the new (partial) network was completely defined and the performance constraints of the on-board network are satisfied.
Failed End Condition:	The new communication cannot be defined without violating at least one performance constraint of the on-board network.
Primary Actor:	System Architect / Network Architect
Trigger Event:	New vehicle functions
End Event:	Initial definition for the new partial network and update of the residual on-board network of the communication matrix.

Table 4.4: Characteristic Information of NW UC "Design of the new developed network"

4.3.1 Main Scenario

1. The network architect chooses an appropriate network technology to fulfill the communication requirements of the new functions. The consequences for the residual system have to be considered because many ECUs should not be altered if possible.
2. The network architect defines and /or designs the connection point (s) to the residual on-board network (via gateways).
3. The network architect connects the ECUs to the new network and partitions the functions onto these ECUs.
4. The network architect maps the new traffic according to the timing information and the transmitter/receiver relation.
5. Depending on the sender/receiver relation it might be necessary to additionally route PDUs/frame on several networks and gateways.

6. Analysis 1: Load analysis determines the average and the maximum use of the network bandwidth and the input buffer of the ECUs. The load analysis must consider the total traffic on the new partial network and on the legacy on-board network as well. The analysis requires the data size and the timing of the PDUs/frames. The output of the analysis is the timing property load GENERIC PROPERTY Load. The timing property load is used to initially approve the chosen function mapping and architecture and if the new infrastructure is sufficient to cover the communication requirements in general. The present value of the timing property load for every single network is compared to the maximum acceptable load for this network.
7. Analysis 2: A detailed latency analysis of all PDUs/frames/packages and every communication relations on the networks is necessary. The method yields the timing properties response time GENERIC PROPERTY Latency / Response Time, jitter, blocking time, etc. Every communication relation has to fulfill its corresponding latency requirement.
8. Analysis 3: In order to consider the ECU influence and the total communication the event chain / the routing time analysis of the PDUs/frames/package has to be considered. This leads to the new timing properties: routing response time SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN), blocking time, buffer requirements GENERIC PROPERTY Load.
9. Optimization of the design of the new network subject to the requirement to reduce resource needs, to increase system stability and robustness and to allow easily future extensions.

4.3.2 Alternative #1 Scenario

n.a.

4.3.3 Performance/Timing Constraints

The maximum load on each bus shall not exceed a certain bound, for example 60%.

For each frame/PDU, the worst-case response time shall not exceed the cycle time of the frame.

Routing times in gateways have to be short. Typically, for each frame/PDU the routing time shall not exceed 10% of the cycle time of the frame.

4.3.4 Related Information

Performance Target:	Bus load, Response Times, Routing Times
Precondition:	n.a.

Frequency:	Regular
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	Network Architect: Support/Approve Timing Expert: Support ECU Integrator: Support

Table 4.5: Related Information of NW UC "Design of the new developed network"

4.4 NW use-case "Remapping an existing Function"

This use-case focuses on remapping an existing function onto the existing networked architecture. For the moment, the use-case does not consider the Ethernet and its point-to-point communication.

Goal In Context:	Validate the communication on the network after reconsidering the mapping of an existing function on the E/E architecture.
Brief Description:	Assuming an E/E automotive architecture that contains ECUs connected via one or more busses, it is required to remap an existing function to a new resource within the network. The busses may implement different communication protocols (e.g. CAN, LIN, Flexray). The communication on each bus is specified by a communication matrix containing the PDUs/frames with their protocol specific parameters and the communication behavior (timing parameters).
Scope:	System, System Timing
Level:	Activity
Precondition:	The set of bus signals received or transmitted by the function to be remapped is known and included in the communication matrix. Additionally, for the communication on the network is defined a set of performance constraints: • Maximum busload on each bus • Maximum latency for each communication frame. Furthermore, the specification of the communication paradigm for the existing bus controllers is available. For example, the CAN controller sends CAN message frames with different identifiers via a queue (priority ordered or FIFO), while different instances of the same frame are sent via a register (always send the newest instance of the frame). It is assumed that the current network configuration satisfies the performance constraints.
Success End Condition:	The communication on the network after function remapping fulfills the performance constraints.
Failed End Condition:	The communication on the network after function remapping cannot be defined without violating at least one performance constraint.
Primary Actor:	System Architect / Network Architect
Trigger Event:	Modification of the car architecture, integration of several distributed functions on a resource.
End Event:	Update of the communication matrix.

Table 4.6: Characteristic Information of NW UC "Remapping an existing function"

4.4.1 Main Scenario

For the sake of clarity following notations are used: the function to be remapped is denoted as F. This function is currently mapped onto ECU1 at Bus1. The resource that will host F after remapping is denoted ECU2, which is connected to Bus2.

1. The network architect identifies the PDUs/frames on Bus1 required by F. These must be transmitted on Bus2 after remapping F to ECU2.
2. The PDUs/frames transmitted by F and additionally required by other nodes at Bus1 must be routed from Bus2 to Bus 1 after remapping F to ECU2. The PDUs/frames received by F and additionally received by other nodes at Bus1 must be routed to Bus2 after remapping F to ECU2. In case that a specific PDU/frame is not required by other nodes at Bus1, one may decide to remove the routing of this PDU/frame to Bus1.
3. The PDU/frames moved or copied to Bus2 should preserve the parameters of the communication protocol defined for Bus1, in order to ensure the function compatibility with the different architecture variants.
4. PDUs/frames required by F at Bus2, that are not originating at Bus2 need to be routed/transmitted to Bus2.
5. Analysis 1: The bus load analysis describes the average use of the bus bandwidth. The analysis has to consider the additional traffic on Bus2 due to mapping of F to ECU2. The analysis requires the data size and the average timing of the PDUs. The output of the analysis is the static bus load GENERIC PROPERTY Load. The bus load property is used to initially approve the traffic on Bus2. Optionally, one can carry out bus load analysis on Bus1 to determine the freed performance slack after remapping F to ECU2. The value of the timing property load obtained for every single bus is compared to the maximum acceptable load on that bus. For typical constraints for the bus load see section 4.4.3.
6. Analysis 2: In order to approve the communication after remapping F to BUS2, the latency constraints of the PDUs/frames on Bus2 must be verified. The latency analysis of the PDUs/frames computes the timing properties of the PDUs/frames under the resource sharing protocol. The results of the analysis are timing properties such as the response times of the PDUs/frames GENERIC PROPERTY Latency / Response Time (or SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN) specific for CAN buses), the jitter of the PDUs/frames, or the blocking times due to arbitration. The values of the timing properties are compared to the specified constraints. For typical constraints on the PDUs/frames response times see section 4.4.3.
7. Analysis 3: In case that the PDUs/frames required at Bus2 are routed by one or more gateways, the routing times are relevant for the end-to-end timing. The routing time analysis of the routed PDUs/frames provides the delay values due to routing engine. These usually consist of buffering delay and arbitration delay. The results of the routing time analysis are the routing response times, the

blocking times due to buffering and arbitration, or the memory requirements for buffering. The values obtained for these properties are compared to the specified constraints. For typical constraints for routing times see section 4.4.3.

4.4.2 Alternative #1 Scenario

n.a.

4.4.3 Performance/Timing Constraints

The maximum load on each bus shall not exceed a certain bound, for example 60%.

For each frame/PDU, the worst-case response time shall not exceed the cycle time of the frame.

Routing times in gateways have to be short. Typically, for each frame/PDU the routing time shall not exceed for example 10% of the cycle time of the frame.

4.4.4 Related Information

Performance Target:	Bus load, Response Times, Routing Times
Precondition:	n.a.
Frequency:	Regular
Super-use-case:	n.a.
Sub-use-case(s):	n.a.
Secondary Actor(s):	Network Architect: Support/Approve Timing Expert: Support ECU Integrator: Informed

Table 4.7: Related Information of NW UC "Remapping an existing function"

5 Properties and Methods for Timing Analysis

5.1 General Introduction

This section describes the general relations between use-cases and tasks (see chapters 3 and 4) on the one hand and timing properties and timing methods on the other hand specified in details in this chapter.

The timing use-cases (e.g. "add a new function to the signal indicator") presented in detail in the former chapters (ECU, network, or end-end examination) can be divided in smaller tasks (e.g. "decompose a timing constraint from ECU A to ECU B in the signal indicator example") . Therefore, a task describes what is required to carry out the complete use-case. To every timing related task, a timing method (e.g. "compute the message latency transmitting the information for signal indicating between function A and B") can be related specifying how to solve this task. The input (e.g. "the communication matrix" or "measured core execution times") for the timing methods arises from the system specification or from observing the real system. Some of the methods deliver timing properties as an output (e.g. "worst case response time of the transmitted message") which can be evaluated by means of timing constraints (e.g. due to functional safety reason) to check the fulfillment of the timing requirement. Thus, the system can be evaluated. Important, but out of scope in this document is the implementation of timing methods and timing properties in tools. The approach and the timing terminology are illustrated in Figure 5.1 and 5.2.

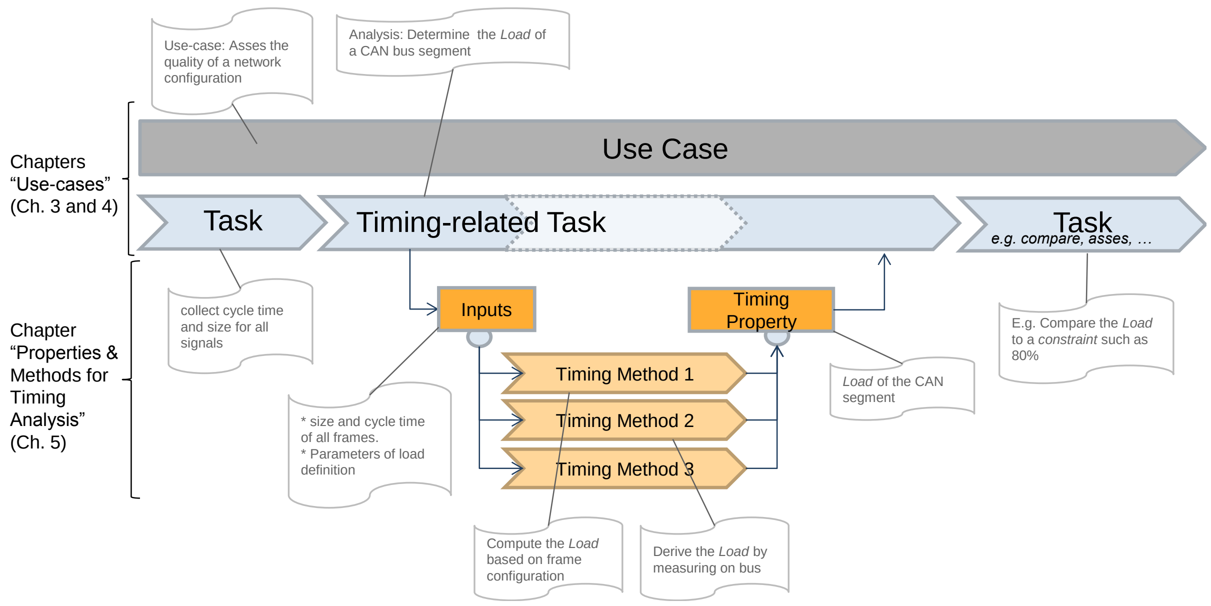
Use-case: Asses the quality of a network configuration
Analysis: Determine the Load of a CAN bus segment
Chapters "Use-cases" (Ch. 3 and 4)
Use Case
Task
Timing-related Task
Task
e.g. compare, asses, …
Chapter "Properties & Methods for Timing Analysis" (Ch. 5)
collect cycle time and size for all signals
Inputs
Timing Property
E.g. Compare the Load to a constraint such as 80%
Timing Method 1
Timing Method 2
Timing Method 3
Load of the CAN segment
* size and cycle time of all frames.
* Parameters of load definition
Compute the Load based on frame configuration
Derive the Load by measuring on bus

Figure 5.1: Illustration of hierarchy between use cases, timing properties, and timing methods (and related sections).

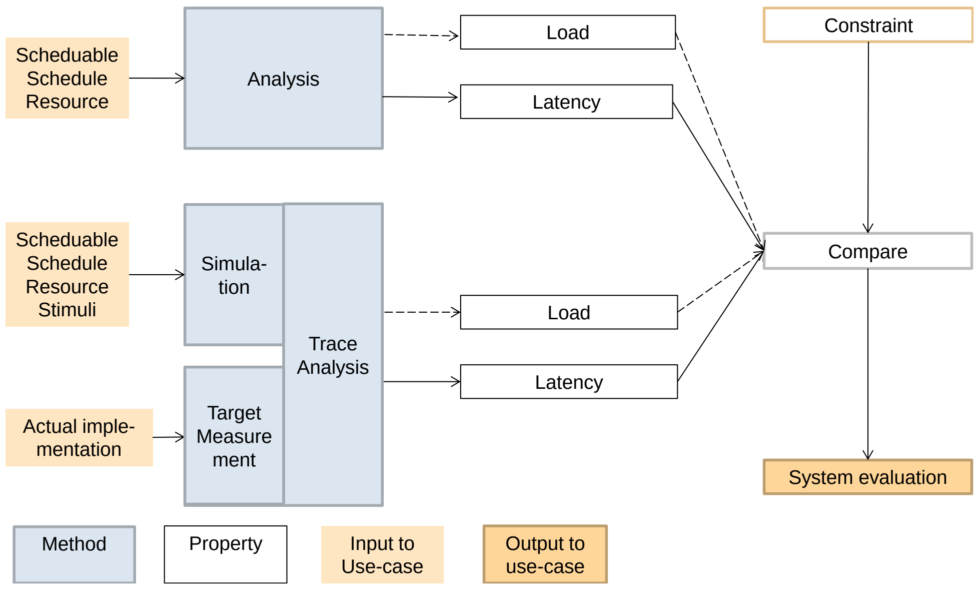

Figure 5.2: The interplay between different timing methods, timing properties and constraints

5.1.1 A Simple Grammar of Timing Properties

In order to avoid repeating similar definition of timing properties and methods in the following sections, this document follows a generic approach. Timing properties are described with supporting placeholders, such as for example “<schedulable>” and “<resource>”. A “<resource>” can be either e.g. a “CPU” or a “CAN bus”, and a “<schedulable>” can be the corresponding e.g. “RunnableEntity", "BswSchedulableEntity" or “frame”.

Not all combinations of such terms lead to relevant/valid definitions. Therefore the actual instances are listed with the definitions. For reasons of practicality, the document however presently does not formalize the placeholder structure into a complete and consistent grammar (but such refinement may be possible in future releases).

5.1.1.1 Resources and Schedulables

<Resources> are needed to execute <schedulables>. They can schedule between several <schedulables> over time, based on an online or offline scheduling scheme. <Resources> have the capability to compute, store, transmit or receive information.

<Resources> can be divided in two categories: <unary resources>, which can execute only one <schedulable> at any given time and <multi resources> which can execute multiple <schedulables> in parallel.

A <schedulable> computes, stores, or transmits information on a <resource>. In order to make progress it must be assigned the <resource> in the scheduling process.

<Resource>	
<Unary Resource>	**Allowed <Schedulable>**
CAN bus segment	CAN frame
Single-Core CPU	Task
FlexRay Segment	FlexRay frame
Ethernet Link	Ethernet message
LIN bus	LIN frame
<Multi Resource>	
Switched Ethernet-Network	Ethernet message
Multi-Core CPU	Task

Table 5.1: Resource Overview

Note: <Multi Resources> are not covered by any of the present definitions in the document.

The timing of a schedulable is defined by its <activate> and its <terminate> events. The <activate> is the moment in time at which the <schedulable> becomes ready to perform its operation, and the <terminate> is the moment in time when it is finished.

A <schedulable> may contain <subschedulables> to differentiate between different operations.

<Schedulable>	**Allowed <Subschedulable>**
Processor Task (equivalent: ISR)	Runnable BSW function
OS-Function	RunnableEntity, BswSchedulableEntity
CAN frame	PDU Signal

Table 5.2: Allowed Schedulable

5.1.1.1.1 Example: Scheduling situation with tasks on a single-core CPU

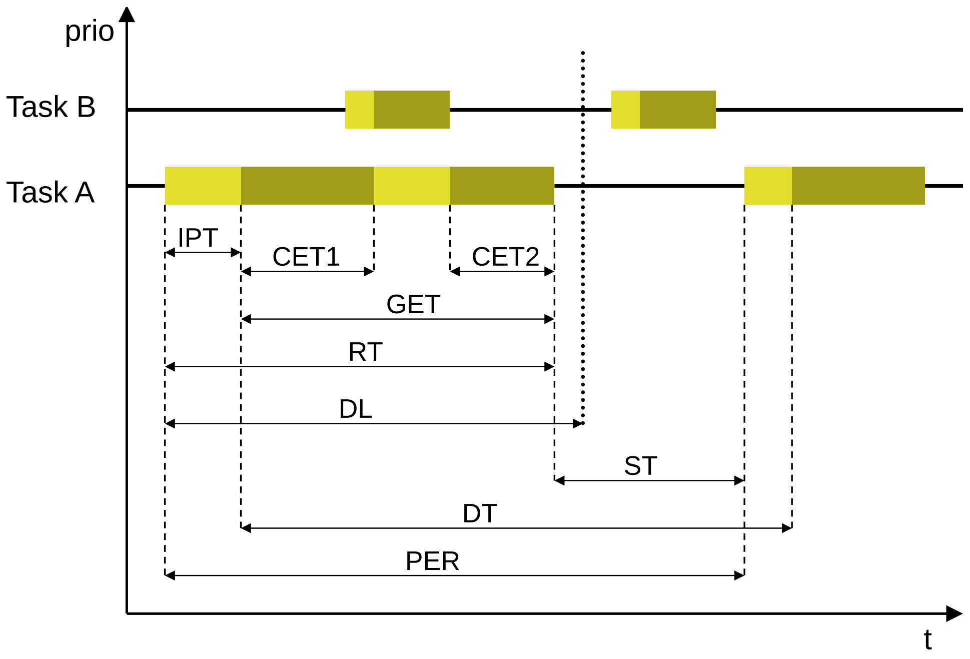

Figure 5.3: Timing properties relevant for tasks scheduled on a single-core CPU

Abbreviation	**Description**
IPT	Initial pending time
CET	Core execution time
GET	Gross execution time
RT	Response time
DL	Deadline
DT	Delta time
PER	Period
ST	Slack time
PRE	Preemption time *(not shown in the figure)*
JIT	Jitter *(not shown in the figure)*
CPU	CPU load *(not shown in the figure)*

Table 5.3: Some important timing parameters (run-time situation on a single-core CPU)

5.1.1.2 Method of Derivation

The different timing properties can be derived with various methods, while not every property can be properly derived with every method (but often approximated). For example, during simulation, the message load can be observed, but it is difficult to derive the real worst-case latency. For the purpose of this document, we differentiate between the following methods:

<TimingMethod>	Explanation
Analysis	Computation or theoretical estimation of the value of the timing property

Simulation	Simulation of a system to determine the temporal development of the value of the timing property
Measurement	Measurement of a target to determine the temporal development of the value of the timing property

Table 5.4: Method of Derivation

5.1.1.3 Statistical Qualifier

Many timing properties can be tailored to different <Statistical Qualifiers>. For example, one may be interested in the average latency of a message in one case and in the maximum latency in another (for example if it is a time critical message). Base to do this is to determine the temporal development of the latency over the time by means of e.g. the simulation and to derive the relevant quantities like the average latency. This can be more generalized to the determination of the temporal development of an arbitrary quantity ("'x-over-Time"') and to derivation of the distribution and its momenta.

For this reason, the following <Statistical Qualifiers> are introduced:

Method	**<Statistical Qualifier>**	**<Statistical Qualifier>** derived quantity
Analysis	Best-Case	
	Worst-Case	
Simulation / Measurement	Distribution / X-over-time	Minimum
		Maximum
		Average

Table 5.5: Different Types of Timing Methods and the resulting Statistical Qualifiers

The x-over-time and the distribution depend on the related timing method, the input parameters and the boundary conditions. In contrast, the analysis approach delivers the timing property as a single value (e.g. worst-case). The (best-)worst-case denotes the state of the system with the (minimum) maximum system requirement, sometimes overestimated by the applied algorithm. However, the (minimum) maximum represents the actual observed value of the timing property here in this context.

5.1.1.4 Constraint Type

Finally, in accordance with the definition in TIMEX, the actual value of the timing property can be interpreted as a requirement (a priori to an analysis) or the worst-case can be regarded as a guarantee for the system specification (a posteriori to an analysis).

<ConstraintType>
Requirement
Guarantee

Table 5.6: ConstraintType

Figure 5.4 sketches the interplay between the value of the timing property (and its development over time and its distribution) and the constraints. The value of the timing property results from the timing method. Some of the Statistical Qualifiers are indicated on the left hand side of the distribution. The guarantee results from the worst case analysis of the timing property of interest whereas the external requirement for this timing property cannot be fulfilled in this case

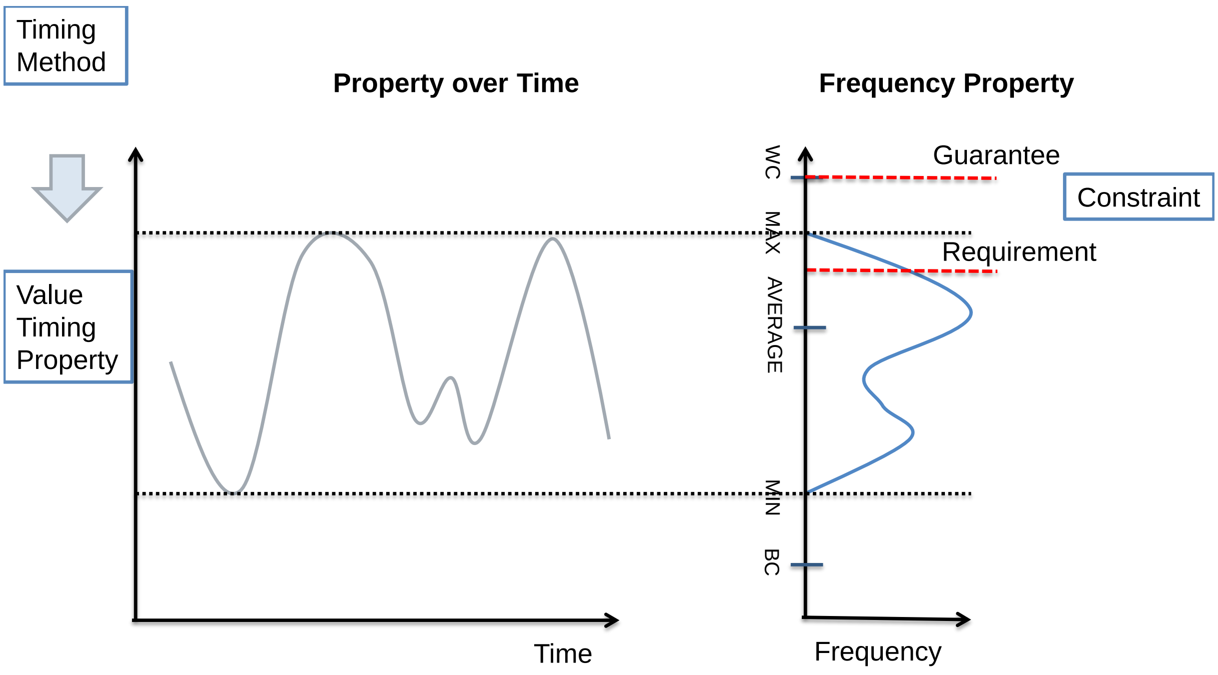

Figure 5.4: The figure illustrates the relation between the timing methods,the timing property, the constraint and qualifiers (see text for more details).

5.2 Definition and Classification of Timing Properties

5.2.1 Classification and Relation of Properties

The properties can be grouped in two main fields: capacitive (<resource> capacity) and latency property (<schedulable> latency). Capacitive properties are the ratio of the capacity requirement by the <schedulables> to the capacity of the <resource>. Latency properties are the delays of <schedulables> due to the schedule (priority schema) on the common used <resource>.

5.2.2 Summary of regarded Timing Properties

NW/ECU	Group	Name
Generic	Load	GENERIC PROPERTY Load
NW	Load	SPECIFIC PROPERTY Bus Load of a CAN Segment
Generic	Latency	GENERIC PROPERTY Latency / Response Time
NW	Latency	SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN)
ECU	Latency	SPECIFIC PROPERTY Worst Case Execution Time

Table 5.7: Overview about the here described Timing Properties

5.2.3 GENERIC PROPERTY Load

5.2.3.1 Scope and Application

Name	Load
Definition	The load is the total share of time that a set of <schedulables> occupies a <single resource>.
Brief Description	If the time for the occupation is calculated it can exceed the available resource time (overload). In the practical realization using simulation or measurement this scenario cannot occur. But, if the input load of all <schedulables> in an overload situation is not buffered the required to transmit information can be lost or overridden.
Application	The property supports the estimation of the resource needs in ECUs and gateways and of the network, respectively.
Assumptions and Preconditions	• The time of the occupation for every individual <schedulable> is known. • The partition for the total communication amount in individual <schedulables> is done.

Table 5.8: Scope and Application

5.2.3.2 Relation

Requirements	-
Process Steps	-

Referencing Use-cases	• NW use-case "Integration of a Distributed Function" on page 52 • NW use-case "Design of the new developed Network" on page 55 • NW use-case "Remapping an existing Function" on page 58 • ECU use-case "Collect Timing Information of a SW-C" on page 32 • ECU use-case "Validate Timing after SW-C integration" on page 34 • ECU use-case "Validation of Timing" on page 37 • ECU use-case "Optimize Scheduling" on page 43
Belonging (Pre) Methods	• GENERIC METHOD Load

Table 5.9: Relation

5.2.3.3 Interface

Notation	$L(t, t_{window}, ...)$	
Possible<Statistical Qualifiers>	All which were mentioned in the introduction	
Parameters	t_{window}	The size of the time interval over which the load is determined. Default value: INF
	t	The end of the time interval over which the load is determined. This parameter is required for load-over-time analysis. Default value: not specified
Range	0 to 100% (0.. infinity for calculation)	

Table 5.10: Interface

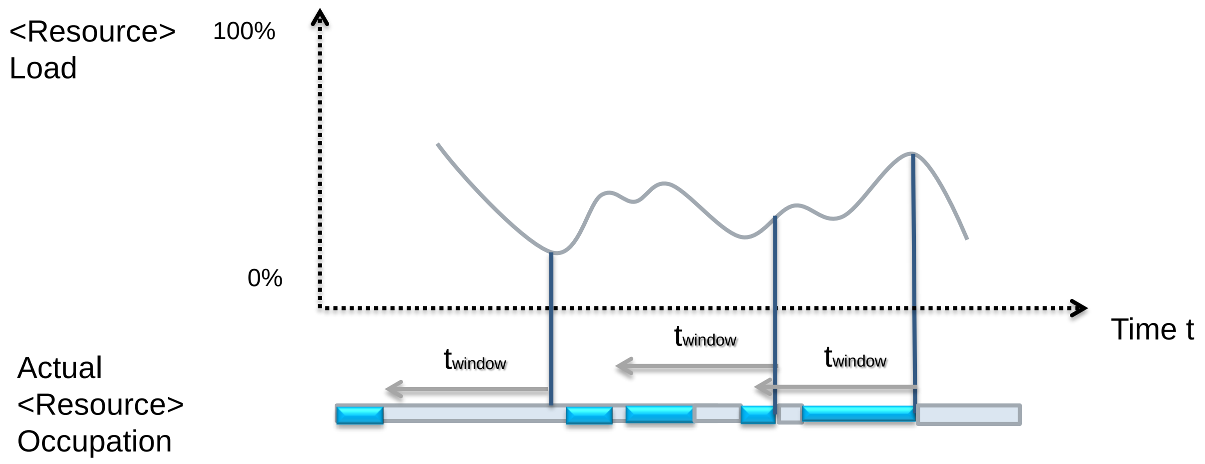

Figure 5.5: Illustration of the relation of the actual occupation and the load over time. The load $L(t, t_{window})$ is the average of the occupation over the interval t_{window} till the point in time t.

5.2.3.4 Expressiveness

The "load" indicates the overall utilization of a given <single resource>. A small load is better for stable operations due to safety and extensibility reasons. However, it shows that the <single resource> is not fully utilized, possibly missing opportunities for cost-optimization.

From perspective of real-time applications and schedulable with timing constraints, the expressiveness of load is limited. A load value below 100% allows deducing the guarantee that eventually every instance of each <schedulable> will be scheduled and executed on the <resource>. However, the completion time of a schedulable may be larger than its period or any given deadline.

Actually, the correlation to the <schedulable's> worst-case response time is small. Depending on the schedule there are examples with high load and small over-all response times and with low (but highly variable) load and high over-all response time. (compare latency, timing property worst-case response/execution time).

In [14], it was shown that given only periodic <schedulables> with deadlines equal to their periods, all <schedulable> will be serviced before their deadline if the load is smaller than 69% (independent <scheduables>). However, in practice, the presence of sporadically activated <schedulables> avoids a direct applicability of this statement.

5.2.4 SPECIFIC PROPERTY Bus Load of a CAN Segment

In order to determine the load of a CAN bus segment the following definitions are used. Summing up all parameters together yields the frame length.

A CAN frame consists of

Payload	0..8 byte (CAN-FD 0..64 byte)
Header	Standard 19 bit Extended 37 bit
Stuff bits	0..19 (Extended 0..25) bit
Footer	25 bit
Inter frame space	3 bit

Table 5.11: Definition parameter for a CAN Segment

Frame	Definition
Periodic Frame	A frame that is activated periodically with period defined by the “cycle time”
Event-Triggered Frame	A frame that is activated sporadically by an external event.
Mixed Frame	A frame that is activated by the passing of the period or an external event. Different concepts on treating the periodic part exist (i.e. resetting of the periodic timer on arrival of sporadic events).

Table 5.12: Definitions of the frame activation for CAN

With this, the following CAN loads are differentiated:

Periodic load	The share of time that the set of periodic frames occupies the bus.
Total load	The share of time that all frame (periodic and event-triggered, including the mixed-triggered frames) occupy the bus.

Table 5.13: Different kinds of Bus Load of a CAN Segment depending on the frame activation

During runtime, the CAN bus and the transmitted frames typically exhibit dynamic behavior:

- frame periods may slightly fluctuate from the specified cycle time (jitter and drift)
- the number of stuff bits depend on the actual payload
- the frame may not always carry the same amount of payload with each transmission

Depending on the selected <Statistical Qualifier> (i.e. average, maximum, ...) the properties of the CAN configuration may need to be interpreted differently due to this dynamism.

5.2.5 GENERIC PROPERTY Latency / Response Time

5.2.5.1 Scope and Application

Name	Latency (Response time)
Definition	The latency is the amount of time between the <activate> and the <terminate> of a <schedulable> instance
Brief Description	The property provides the total time from when a <schedulable> is ready to transmit on/occupy a <resource> until the <resource> is freed from the occupation of the <schedulable>. The response time of a <schedulable> is equal to its execution (or transmission) time in the case where the resource is exclusively available to this <schedulable>. In the presence of multiple <schedulables> that are ready at the same time, the resulting response times are defined by the actual schedule. The term "‘latency"’ is used synonymously with the term "‘response time"’ in this document.
Application	The property supports the estimation of the resource needs and the rescheduling in ECUs and gateways and of the network, respectively.
Assumptions & Preconditions	For each <resource> is known: • The access schema/arbitration strategy like bus protocol or OS scheduling • All occupation of a <resource> is error free, i.e. every utilization by the <schedulable> takes place exactly once. For each individual <schedulable> is known • The priority of the <schedulables> • The brutto transmission/execution time • The triggering/activation schema including any send delay

Table 5.14: Scope and Application

5.2.5.2 Relation

Requirements	-
Process Steps	-
Referencing Use-cases	• NW use-case “Integration of a Distributed Function” on page 52 • NW use-case “Design of the new developed Network” on page 55 • NW use-case “Remapping an existing Function” on page 58
Belonging (Pre) Methods	GENERIC METHOD Latency

Post Methods	The property can be used for computation of the real-time slack (available bandwidth after accommodating all frames specified in the communication matrix).

Table 5.15: Relation

5.2.5.3 Interface

Notation	$T(t, t_{window}, X...)$	
Possible<Statistical Qualifiers>	All in the introduction mentioned	
Parameters	*X*	The information package for which to compute the response time
	t_{window}	The size of the time interval over which the latency is determined. Default value: INF
	t	The beginning or end of the time interval over which the latency is determined. This parameter is required for X-over-time analysis. Default value: not specified
	CAN specific	
	stuff bits	the number of stuff bits to be assumed during analysis.
Range	0 to infinity	

Table 5.16: Interface

5.2.5.4 Expressiveness

A latency of the <schedulable> measures the temporal delay for its utilization of a <single resource>. A small latency is better for stable functional operations due to safety and extensibility reasons. However, it shows that the <resource> is not fully utilized if the latency is too small against the <schedulable> deadline, possibly missing opportunities for cost-optimization. Nevertheless the latency must be smaller than the <schedulable's> deadline, otherwise information loss may occur. If a considerable part of <schedulables> misses their deadlines the <single resource> has not enough capacity or the schedule is not sufficiently good.

Errors during a transmission or an execution of a <schedulable> may lead to a retransmission/re-execution of specific <schedulables> which increases both the load and the latency.

The worst-case of the latency can be derived by model based analysis by methods such as [15]. By this, the property is conservatively computed.

The worst-case of the latency can be approximated by simulation, albeit only optimistically. The related transmission/execution requests and transmission/ execution complete events can be randomly generated and observed. The maximum of the observed values is an optimistic approximation of the worst-case latency.

When the property is derived using different methods (especially simulation/analysis and measurement) the following must be true:

$WC\ Latency_{Analysis}($<Schedulable>$) >= WC\ Latency_{Simulation}($<Schedulable>$)$ and
$WC\ Latency_{Analysis}($<Schedulable>$) >= WC\ Latency_{Measurement}($<Schedulable>$)$

5.2.6 SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN)

The worst-case frame response time is the maximum amount of time between the <activate> and the <terminate> of a <schedulable> instance.

Thus, the Worst-Case Frame Response Time (CAN) is an instance of the Latency property (PROPERTY_02) with the following parameters:

Generic parameter	**Actual value**
<resource>	CAN bus segment
<schedulable>	CAN frame
<activate>	Event TDEventFrame.frameQueuedFor Transmission on sender ECU
<terminate>	Event TDEventFrame.frameTransmittedOnBusbetween network and receiver ECU

Table 5.17: Relation between the general and the CAN specific parameters

5.2.6.1 Scope and Application

Name	Worst-case frame response time
Brief Description	The property provides the total time from when a frame is ready to send until a frame is completely transmitted over a bus.
Goal	The property allows assessing the communication delay of a timing critical message.
Assumptions	It is assumed that all communication on the bus is error free, i.e. every transmission takes place exactly once. It is assumed that of all messages in a network that are ready to send, the CAN bus always selects the one with the lowest CAN-ID for transmission.

Table 5.18: Scope and Application

5.2.6.2 Classification

System	Network
Network	CAN
Classification	

Table 5.19: Classification

5.2.6.3 Relation

Requirements	
Process Steps	
Use-cases NW	
Use-case ECU	-
Super Property	Response time
Sub Property (s)	
Belonging (Pre) Methods	
Post Methods	The property can be used for computation of the real-time slack (available bandwidth after accommodating all frames specified in the communication matrix).

Table 5.20: Relation

5.2.6.4 Interface

Precondition	For each frame on the bus, the following is known: • Frame length including stuff bits • CAN-ID • Optimization of a new CAN-ID For each periodic and for each mixed frame the following is known: • Period • Reference clock (optional) • Offset to reference (optional) For each event triggered and for each mixed frame the following is known: • Event model of external events including minimum arrival time
Output	The worst case response time between when a frame becomes ready to send (i.e. placement of the frame in an output message buffer of the CAN driver) and when it has been completely transmitted over the CAN bus (usually leading to a Tx IRQ on a receiving ECU).
Notation	WCRT(frame)
Parameter	• *frame X*. The frame for which to compute the response time. • *stuff bits,* the number of stuff bits to be assumed during analysis.
Range	0 to infinite
Valid Range	Frame response time must be smaller than frame deadline.

Table 5.21: Interface

5.2.6.5 Validation and Application

Expressiveness

The expression of the response time as defined is limited in some sense:

1. Due to internal buffer structure, some CAN controllers may not be able to always provide the frame with the lowest CAN-ID (highest priority) that is ready to send to the bus arbitration. This can lead to a priority inversion with potentially larger response times than as defined by this property.
2. Errors during the frame transmission may lead to a retransmission of specific messages which increases the bus-load and frame response times.
3. In the case of a large number of non-harmonic time bases, analysis time can grow beyond acceptable times. In this case, some offset relations can be ignored during analysis which may slightly decrease accuracy.
4. It is difficult to measure latency in target setups. While it is easy to identify the transmission complete events by probing the bus, the point in time when a frame becomes ready to send is more difficult (black box measurement). One option is to estimate the time by checking the bus busy time before the transmission complete event. Another option is to combine an ECU internal trace with the network trace using a reference time base.

These constraints are in part relaxed by current research such as [16], [17].

Determination of the Comparability of the Different Methods (Analysis, Simulation, Measurement)

The worst-case response time can be derived by model based analysis by methods such as [15]. By this, the property is conservatively computed.

The worst-case response time can be approximated by simulation, albeit only optimistically. The related send requests and transmission complete events can be randomly generated and observed. The maximum of the observed values is an optimistic approximation of the worst-case response time.

It is difficult to measure frame response time in target setups. While it is easy to identify the transmission complete events by probing the bus, the point in time when a frame becomes ready to send is more difficult. One option is to estimate the time by checking the bus busy time before the transmission complete event. Another option is to combine an ECU internal trace with the network trace using a reference time base.

Verification for Achieving identical Results

When the property is derived using different methods (especially simulation/analysis and measurement) the following must be true:

$$WCRT_{analysis}(frame\ X) >= WCRT_{simulation}(frame\ X)$$
$$WCRT_{analysis}(frame\ X) >= WCRT_{measurement}(frame\ X)$$

5.2.7 SPECIFIC PROPERTY Worst Case Execution Time

5.2.7.1 Scope and Application

Name	Worst Case Execution Time (WCET)
Brief Description	The WCET indicates the maximum time required for a certain computation. In this context a computation can be a runnable, a sub-function or just a sequence of commands.
Goal	This property is a required input information for run time budgeting and the ECUs schedule feasibility.
Assumptions	WCET is deterministic.

Table 5.22: Scope and Application

5.2.7.2 Classification

System	ECU
Applied Network	n/a
Classification	Timing of individual separated computation.

Table 5.23: Classification

5.2.7.3 Relation

Requirements	
Process Steps	
Use-cases NW	n/a
Methodology/Task ECU	The property worst-case execution time (WCET) is related to the methodologies
Use-case ECU	• ECU use-case "Collect Timing Information of a SW-C" • ECU use-case "Validate Timing after SW-C integration" • ECU use-case "Optimize Timing for an series ECU" • ECU use-case "Optimize Scheduling" • ECU use-case "Optimize Code" • ECU use-case "Compare Timing Properties"
Super Property	n/a
Sub Property (s)	

Belonging (Pre) Methods	• ECU_METHOD Static Worst Case Execution Time Analysis • Simulation via target simulator • ECU_METHOD Processor-In-The-Loop Simulation (PIL) • Measurement/Tracing
Post Methods	Worst-Case Response Time (WCRT) analysis

Table 5.24: Relation

5.2.7.4 Interface

Output	The scalar result value is usually stated in micro-, milli- or nanoseconds.
Range	0 to infinity

Table 5.25: Interface

5.2.7.5 Expressiveness

Since the WCET is an indicator for resource consumption usually a predefined value must be reached or derived. To predict and proof the correct software execution the WCET is an important property. In practice it is recommended to use different timing methods to determine the WCET in order to gain the confidence of the result. These methods are static, dynamic and hybrid approaches.

5.3 Definition, Description and Classification of Timing Methods

5.3.1 Classification and Relation of Methods

Roughly, the methods can be grouped in three main fields: simulation, analytical calculation and measurement. Another criterion to distinguish methods is to consider the origin of the data: model-based or measurement-based. This classification is closely related to the moment in which stage of the timing process the method can carry out (in the specification phase or verification phase).

5.3.2 Summary of regarded Methods

NW/ECU	Group	Name
Generic	Analysis, Simulation, Measurement	GENERIC METHOD Load
Generic	Analysis, Simulation, Measurement	GENERIC METHOD Latency

ECU	Simulation	ECU_METHOD Static Worst Case Execution Time Analysis
ECU	Simulation	ECU_METHOD Processor-In-The-Loop Simulation (PIL)
ECU	Simulation	ECU_METHOD Discrete-Event-Simulation (DES)

Table 5.26: Summary of regarded Methods

5.3.3 GENERIC METHOD Load

5.3.3.1 Scope and Application

Brief Description	The method yields the load (distribution) over a defined time interval.
Related Development Process Steps	The method can be used at the following development steps ([Internal ref]), however the complete application can be done at the end of every iteration step.
Actor	Timing Analyst
Reasoning	The method supports the estimation of the resource needs in ECUs and gateways and of the network, respectively.

Table 5.27: Scope and Application

5.3.3.2 Detailed Description

5.3.3.2.1 Specific for CAN

5.3.3.2.1.1 Specific for Analysis

For CAN, the formula to calculate the bus load includes a pessimistic/optimistic approach depending on estimation of the stuff-bits for the analysis (see the formula for the stuff bits below, for CAN frames with 29-Bit Identifier there are deviations)

$$t_{frame} = (t_{\text{stuff bits}} + 47 + 8 * payloadlength[Byte]) * \tau_{Bit} \tag{5.1}$$

$$t_{min} = t_{cycle} - t_{tolerance} \tag{5.2}$$

$$L(t_{cycle}, t_{tolerance}, payloadlength) = \sum_{frame} \frac{t_{frame}}{t_{min}} \tag{5.3}$$

Whereas payload length (in Byte) is the length of the data part of the CAN frame, t_{bit} is the time for the transmission of one bit, t_{cycle} is the specified period and $t_{tolerance}$ is the allowed deviation from the period. Therefore t_{min} is the maximum tolerable jittering including drift.

The approach takes the maximum load by subtracting a tolerance time into the consideration. This approach estimates the bus load generated by the periodic messages on a bus during an infinitely long time window (t_{window} is infinity, the present point in time t does not play any role). The time for a frame is maximized due to a conclusion of all possible stuff bits. The event-triggered frames are neglected.

5.3.3.2.1.2 Specific for Simulation/Measurement

The load using the simulation or measurement is given by:

$$L(t_{window}, t) = \sum_{frame} \frac{t_{frame}}{t_{window}} \tag{5.4}$$

(Event triggered frames have to be considered separately.)

5.3.3.3 Relation

Requirements	-
Process Steps	-
Referencing Use-cases	• NW use-case “Integration of a Distributed Function” on page 52 • NW use-case “Design of the new developed Network” on page 55 • NW use-case “Remapping an existing Function” on page 58 • ECU use-case “Collect Timing Information of a SW-C” on page 32 • ECU use-case “Validate Timing after SW-C integration” on page 34 • ECU use-case “Validation of Timing” on page 37 • ECU use-case “Optimize Scheduling” on page 43
(Pre) Timing Property	No output from another property
Belonging Post Timing Property	GENERIC PROPERTY Load

Table 5.28: Relation

5.3.3.4 Interface

Input	• <Schedulables> with their over-all times and their activation pattern (e.g. periodic, sporadic) • Transmission/execution speed of the regarded <single resource> • Model of the spontaneous occurrence of <schedulable> (e.g.event-triggered frames) • Scheduling/priority rules
Boundary condition, Settings and Variants, Precondition	• Environmental states (like driving states)

Table 5.29: Interface

5.3.3.5 Validation and Application

5.3.3.5.1 Effort to their Determination

The timing property can be effortlessly calculated or measured.

5.3.3.5.2 Limitation in Application

At the moment, there is no established treatment for the spontaneous occurrence of <scheduables> (e.g. event-triggered frames). Therefore, a unified model or activation pattern for the spontaneous occurrence has to be applied in order to achieve comparable results between different configurations and analysis/simulation/measurement.

5.3.3.5.2.1 Specific for CAN

A general treatment for the calculation/simulation of stuff-bits is missed. Further, it is unclear if the “bus load” implies the cycle time with the specified value or with worst-case deviation. Due to the finite simulation time the worst-case may not be captured.

5.3.3.6 Implementation

The analysis, the simulation and the measurement should be implemented in the same manner with the in this document defined formula. All boundary condition shall be revealed. Different algorithms can be applied as long as the results are identical under identical conditions. Any approximation shall be signalized and the parameter for the cut-off shall be open.

The property <schedulable> shall be implemented with different deviation from the specified period in case of cyclic activation. Different possibilities for modeling event-triggered activation patterns shall be supported. However, for the analysis, the load takes into the consideration the cyclic events with their periods and the spontaneous events with an event model. E.g. the spontaneous events can be modeled with their debounce times as a “cycle” or with their maximum occurrence rate. Depending of the pessimistic or optimistic approach the calculation can estimate the upper bound with the lower limit of the period or with a specified period for the latter one.

5.3.3.6.0.2 Specific for Simulation

In general, the load is given by the ratio of the used time and the total time. Thus, the method “load simulation” sums up the time for every single <schedulable> with the complete overhead within the pre-defined interval between <activate> and <terminate>. In order to get the “load” the sum is divided by this interval.

Especially, in case of the simulation, one has to pay attention to the partially transmitted <schedulables> at the temporal interval boundaries.

A good choice of the seed and the sample probe is extremely important to achieve relevant results. The configuration space should be equal-covered especially if one considers a peak load over a time interval.

5.3.3.6.0.3 Specific for CAN

For CAN, different assumptions for stuff bits shall be implemented (minimal, average, maximal). Depending on the implemented approach, the calculation shall include a minimum (optimistic approach), an average or a maximum (pessimistic approach) number of stuff-bits. For each frame the following calculation formula for the maximal stuff-bit time shall be used. The average number of stuff-bit time can be derived by dividing by 2.

$$t_{\text{stuff bits}} = \left\lfloor \frac{34 + 8 * payloadlength([Byte])}{4} \right\rfloor * \tau_{Bit} \tag{5.5}$$

5.3.3.7 Determination of the Comparability of the Different Methods

Comparing analysis on one hand and simulation/measurement on the other hand the loads shall be coincident in the long-time limit (under identical boundary conditions). The difference vanishes if all parameters are chosen in the same manner. In general, the simulation and the observation yield an optimistic approximation in the same manner depending on the sample/probe size (measurement/simulation time).

In order to compare the results of different methods (especially simulation/analysis and measurement) a check that all <schedulables> are contained in the output is highly recommended.

5.3.4 GENERIC METHOD Latency

5.3.4.1 Scope and Application

Brief Description	The method yields the latency of <schedulables> when executed on <resources>. Depending on the application level, the generic term "'latency"' represents: • the execution time of a <schedulable> on a <resource> without considering the execution of other <schedulables> on that <resource>, for example of a RunnableEntity or BSWSchedulableEntity on a ECU or the transmission time of a PDU/frame on a network. • the response time of a <schedulable> on a <resource> by considering the execution of other <schedulables> on that <resource> and the corresponding <resource> arbitration policy. For example the response time • the end-to-end time in case of routing <schedulables> via one or multiple <resources>, for example PDUs/frames via one or multiple gateways.
Related Development Process Steps	The method can be applied at every release iteration when a new software implementation is provided.
Actor	Timing Analyst
Reasoning	The method supports the estimation of the resource needs in ECUs and gateways and of the network, respectively.

Table 5.30: Scope and Application

5.3.4.2 Detailed Description

Specific instances of the GENERIC METHOD Latency can be defined for ECUs and networks.

The implementation of the method for deriving timing properties of type latency for networks or ECUs depend on the considered approach, i.e. analysis, simulation or measurement.

5.3.4.3 Classification

System	ECU / Network
Applied Protocol	CAN / FlexRay / OSEK / AUTOSAR etc.
Approach	Analysis / Simulation / Measurement

Table 5.31: Classification

5.3.4.4 Relation

Requirements	Interface input, see Table 5.33.
Process Steps	The method shall be applied during the following process steps: • Verification of an software implementation • Requirement analysis for further development • Resource optimization during development phase
Referencing Use-case	• NW use-case "Integration of a Distributed Function" on page 52 • NW use-case "Design of the new developed Network" on page 55 • NW use-case "Remapping an existing Function" on page 58
(Pre) Property	-
Belonging Post Property	GENERIC PROPERTY Latency / Response Time and SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN).
Super Method	-
Sub Method(s)	-

Table 5.32: Relation

5.3.4.5 Interface

Precondition	Determination of the precondition
Input	The method requires parameters such as: • Implementation of the software, analyzable executable (e.g. elf file), input vectors for stimulation a.s.o. • <Schedulables> (e.g. tasks/frame/PDUs) with their overall times (transmission time, execution time), their activation pattern (e.g. periodic/cyclic, sporadic) and other parameters (e.g. stuff-bits in case of CAN communication) • Scheduling/priority rules (e.g. preemptive, non-preemptive, mixed-preemptive) on the <resource> • Transmission/execution speed of the regarded <resource> (e.g. CAN bus, processor speed) • Model of the spontaneous occurrence of <schedulables> (e.g. event triggered frames) / Approximation of the occurrence of the spontaneous events • Environmental states (e.g. driving states)
Input documents	-
Output	The result of this method are timing properties such as: the execution time ECU_METHOD Static Worst Case Execution Time Analysis), the latency GENERIC PROPERTY Latency / Response Time and SPECIFIC PROPERTY Worst-Case Frame Response Time (CAN).

Table 5.33: Interface

5.3.5 ECU_METHOD Static Worst Case Execution Time Analysis

5.3.5.1 Scope and Application

Name	Static Worst Case Execution Time Analysis
Brief Description	This method is used to calculate the worst case execution time by static analysis. The result is determined by computation.
Actor	Timing Analyst
Related Development Process Steps	The method can be applied at every release iteration when a new software implementation is provided.
Reasoning	The result of this method is the worst case execution time by computation.

Table 5.34: Scope and Application

5.3.5.2 Classification

System	ECU
Applied Network	n/a
Classification	Analysis on compiled code level.

Table 5.35: Classification

5.3.5.3 Relation

Referencing Use-cases	• ECU use-case “Collect Timing Information of a SW-C” • ECU use-case “Validate Timing after SW-C integration” • ECU use-case “Validation of Timing” • ECU use-case “Optimize Timing for an series ECU” • ECU use-case “Optimize Scheduling” • ECU use-case “Optimize Code” • ECU use-case “Verify Timing Model(s)” • ECU use-case “Compare Timing Properties”
(Pre) Property	-
Belonging Post Property	-
Super Method	-
Sub Method(s)	-

Table 5.36: Relation

5.3.5.4 Interface

Precondition	Determination of the precondition

Input	• Compiled software implementation • Symbol information • Annotations for additional constraints (e.g. build options, range of input values, integration/hardware specific constraints)
Input documents	Annotations for additional constraints (e.g. build options, range of input values, integration/hardware specific constraints)
Output	The worst case execution time

Table 5.37: Interface

5.3.5.4.1 Effort to their Determination

5.3.5.4.2 Limitation in Application

Verifying results and cross-checks with the input parameters: The results of the static worst case execution time analysis method should be validated with the results from alternative worst case execution time methods (e.g. ECU_METHOD Processor-In-The-Loop Simulation (PIL)).

Limitations:
For proper static worst case execution time analysis and calculation the target hardware behavior must be known in detail (e.g. access time for different memory areas, caching, and so on). In modern systems the behavior model can be quite complex and therefore limitations regarding the results precision may occur.

5.3.5.5 Implementation

Implementation in tooling:

- Call graph/instruction sequence analysis

Resulting Tool-requirements:

- The analysis tool should be able to analyze either executables or compiled source code.
- The call graph and the instruction sequence shall be reconstructed and analyzed.
- The tool shall evaluate the call graph and instruction sequence in order to determine the worst case execution time.
- It shall be possible to specify further constraints/annotation (e.g. target hardware, physical limitations, and so on) to configure the calculation for real-world properties and needs.

5.3.6 ECU_METHOD Processor-In-The-Loop Simulation (PIL)

5.3.6.1 Scope and Application

Name	Processor-In-The-Loop Simulation (PIL)
Brief Description	This method is used to measure execution times of a specific software system. Therefore the compiled software will be executed on the embedded target processor. In order to be able to execute the software the required run-time environment will be simulated.
Actor	Software Developer, Timing Analyst
Related Development Process Steps	The method can be applied whenever a new software implementation is provided.
Reasoning	This method aims to determine required timing properties like the SPECIFIC PROPERTY Worst Case Execution Time

Table 5.38: Scope and Application

5.3.6.2 Detailed Description

The compiled software under investigation is downloaded to the target hardware. Afterward the simulation platform stimulates and executes the software. During the execution the required run-time is measured. The resulting execution time vector is analyzed to derive the required timing properties. Assuming that the execution path which requires the maximum possible run-time is executed during the simulation the SPECIFIC PROPERTY Worst Case Execution Time can be derived.

5.3.6.3 Classification

System	ECU
Applied Network	-
Classification	Measurement on compiled code level

Table 5.39: Classification

5.3.6.4 Relation

Referencing Use-cases	• ECU use-case “Collect Timing Information of a SW-C” • ECU use-case “Validate Timing after SW-C integration” • ECU use-case “Validation of Timing” • ECU use-case “Optimize Timing for an series ECU” • ECU use-case “Optimize Scheduling” • ECU use-case “Optimize Code” • ECU use-case “Compare Timing Properties”

(Pre) Property	-
Belonging Post Property	-
Super Method	-
Sub Method(s)	-

Table 5.40: Relation

5.3.6.5 Interface

Input	• Analyzable executable (e.g. elf file) • Symbol information • Input vectors for stimulation
Output	• Measured execution time vector • SPECIFIC PROPERTY Worst Case Execution Time

Table 5.41: Interface

5.3.6.5.1 Limitation in Application

Verifying results and cross-checks with the input parameters:
The results of the PIL method should be validated with the results from alternative methods like ECU_METHOD Static Worst Case Execution Time Analysis. In general, the result of the static analysis shall be a conservative upper bound on the worst case execution time of a software piece and thus equal to or larger than the SPECIFIC PROPERTY Worst Case Execution Time measured with the PIL method.

The input stimuli vector which will be used for the PIL needs to stimulate the software in a way that the highest physically possible code coverage is reached. The quality of the input stimuli vector shall be shown in a separate "input stimuli vector acceptance test" which proofs an appropriate coverage.

Limitations:
The accuracy of the result strongly depends on the quality of the input stimuli vector.

5.3.6.6 Implementation

Implementation in tooling /Alternative:

- Execution time tracing on evaluation board
- Execution time tracing on ECU level

Resulting Tool-requirements:

- The tracing solution must have the capability to measure the execution time between defined profiling points. Profiling points define the start and end point for the measurement.

5.3.7 ECU_METHOD Discrete-Event-Simulation (DES)

5.3.7.1 Scope and Application

Name	Discrete-Event-Simulation (DES)
Brief Description	This method is used to simulate the dynamic behavior of the system. It models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system.
Frequency	The method can be applied whenever a timing model of the system is available.
Actor	Software Developer, Timing Analyst, System Integrator
Goal	The results of this method are timing properties of the system.
Examination stage concerning implementation	a-priori

Table 5.42: Scope and Application

5.3.7.2 Detailed Description

5.3.7.3 Classification

System	ECU
Applied Network	-
Classification	Simulation of a timing model

Table 5.43: Classification

5.3.7.4 Relation

Process Steps	The method shall be applied during the following process steps: • Evaluation of the timing behavior of a software architecture • Requirement analysis for further development • Resource optimization during development phase
Referencing Use-case	• ECU use-case “Collect Timing Information of a SW-C” on page 32 • ECU use-case “Validate Timing after SW-C integration” on page 34 • ECU use-case “Create Timing Model of the entire ECU” on page 30 • ECU use-case “Validation of Timing” on page 37 • ECU use-case “Optimize Scheduling” on page 43

(Pre) Property	-
Belonging Post Property	-
Super Method	-
Sub Method(s)	-

Table 5.44: Relation

5.3.7.5 Interface

Precondition	A Timing Model of the system is available
Input	Timing Model
Input documents	-
Output	Timing properties

Table 5.45: Interface

5.3.7.5.1 Limitation in Application

The accuracy of the result strongly depends on the quality of the input model.

5.3.8 ECU_METHOD ECU Measurement and Tracing

5.3.8.1 Scope and Application

Name	ECU Measurement and Tracing
Brief Description	This method is used to measure execution times of a specific software system which is running in the actual ECU.
Actor	Software Integrator, Timing Analyst
Related Development Process Steps	The method can be applied when a new software integration is done.
Reasoning	This method aims to determine required timing properties like the SPECIFIC PROPERTY Worst Case Execution Time

Table 5.46: Scope and Application

5.3.8.2 Detailed Description

The software under investigation is flashed to the ECU and therefore runing in the actual target environment. The ECU might either be part of a Hardware-in-the-loop test block or already integrated into a test car. During the execution the required run-time is measured. The resulting execution time vector is analyzed to derive the required timing properties. Assuming that the execution path which requires the maximum possible run-time is executed during test run the SPECIFIC PROPERTY Worst Case Execution Time can be derived.

5.3.8.3 Classification

System	ECU
Applied Network	-
Classification	Measurement on ECU/integration level

Table 5.47: Classification

5.3.8.4 Relation

Referencing Use-cases	• ECU use-case “Collect Timing Information of a SW-C” • ECU use-case “Validate Timing after SW-C integration” • ECU use-case “Validation of Timing” • ECU use-case “Optimize Timing for an series ECU” • ECU use-case “Optimize Scheduling” • ECU use-case “Optimize Code” • ECU use-case “Compare Timing Properties”
(Pre) Property	-
Belonging Post Property	-
Super Method	-
Sub Method(s)	-

Table 5.48: Relation

5.3.8.5 Interface

Input	• ECU with software to analyse • Input vectors/test szenarios for stimulation
Output	• Measured execution time vector • Maximum execution time which might correspond the SPECIFIC PROPERTY Worst Case Execution Time

Table 5.49: Interface

5.3.8.5.1 Limitation in Application

Verifying results and cross-checks with the input parameters:
The results of the ECU_METHOD ECU Measurement and Tracing should be validated with the results from alternative methods like ECU_METHOD Static Worst Case Execution Time Analysis and ECU_METHOD Processor-In-The-Loop Simulation (PIL).

The test vectors which will be used for the ECU_METHOD ECU Measurement and Tracing need to stimulate the ECU software in a way that the highest physically possible run-time is reached.

Limitations:
The accuracy of the result strongly depends on the quality of the test vectors.

5.3.8.6 Implementation

Implementation in tooling:

- Execution time tracing on ECU level

Resulting Tool-requirements:

- The tracing solution must have the capability to measure the execution time between defined profiling points. Profiling points define the start and end point for the measurement.

A History of Constraints and Specification Items

A.1 Constraint History of this Document related to AUTOSAR R4.1.3

A.1.1 Changed Constraints in R4.1.3

No constraints were changed in this release.

A.1.2 Added Constraints in R4.1.3

No constraints were added in this release.

A.1.3 Deleted Constraints in R4.1.3

No constraints were deleted in this release.

A.2 Specification Items History of this Document related to AUTOSAR R4.1.3

A.2.1 Changed Specification Items in R4.1.3

No specification items were changed in this release.

A.2.2 Added Specification Items in R4.1.3

No specification items were added in this release.

A.2.3 Deleted Specification Items in R4.1.3

No specification items were deleted in this release.

List of Figures

List of Tables