Document Title	Glossary
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	55

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R25-11

Document Change History

Date	Release	Changed by	Description
2025-11-27	R25-11	AUTOSAR Release Management	• Replaced abbreviation table by new macros • Changed abbreviations: ASIL (Automotive Safety Integrity Level), ASAM (Association for Standardization of Automation and Measuring Systems), COM (Communication), DET (Default Error Tracer) • Added abbreviations: COMM (COMMunication Manager), CM (Communication Module), DCA (Data Class Adapter), DP (Data Point), HMI (Human Machine Interface), HPC (High performance computer/computing machine), VDP (Vehicle Data Protocol) • Added definitions: Minimum Send Interval, Middleware, Software Cluster (CLassic Platform) • Removed abbreviation: TTCAN (Time Triggered CAN)
2024-11-27	R24-11	AUTOSAR Release Management	• Added terms for Trusted Platform, Adaptive Platform Machine Configuration, Inter-Integrated Circuit, Automotive API (Gateway) • Changed definition for Adaptive Application • Extended abbreviation list

▽

△

2023-11-23	R23-11	AUTOSAR Release Management	• Improved definition of Processed Manifest • Extended abbreviation list
2022-11-24	R22-11	AUTOSAR Release Management	• Added terms for Can XL, Integrity Check Value, Firewall • Improved definitions of Foundation, Software Cluster (Adaptive Platform), Gateway, Use Case
2021-11-25	R21-11	AUTOSAR Release Management	• Added terms for Intermediate PNC coordinator, PN shutdown message, Top-level PNC coordinator, PNC leaf node • Added terms for Logging, Tracing, Profiling • Improved definition of System term
2020-11-30	R20-11	AUTOSAR Release Management	Added new terms: • E2E Protection Alive Counter • E2E Protection Sequence Counter • Vehicle State Manager • Health Indicator • System Health Monitor • Wake-up and sleep on dataline • Foundation • Intrusion Detection System • Onboard Security Event
2019-11-28	R19-11	AUTOSAR Release Management	• Removed FlexRay specific terms • Added new terms: – Secure channel – Abstract Platform – Raw Data Stream – Signal Service Translation • Changed Document Status from Final to published

▽

△

2019-03-29	1.5.1	AUTOSAR Release Management	• Editorial changes
2018-10-31	1.5.0	AUTOSAR Release Management	• Extended abbreviations • Added terms: – AUTOSAR Run-Time Interface – Bus Mirroring – Cluster – Executable Entity Cluster – Execution Order Constraint – Execution Time – LIN Bus Idle – Log and Trace – Logical Execution Time – Mappable Element – Security Event – Synchronization Points – Timed Communication • Changed OSEK references • Incorporated concepts as draft: – AUTOSAR Run-Time Interface – MCAL Multicore Distribution – Transport Layer Security
2018-03-29	1.4.0	AUTOSAR Release Management	• Added terms: – Access Control Policy – Access Control Decision – MetaDataItem – Policy Decision Point (PDP) – Policy Enforcement Point (PEP) – Identity and Access Management (IAM) • Removed terms: – FlexRay Global Time – Meta-Model – MetaDataLength ▽

▽

△

			△ – Model – Multiple Configuration Sets Shipping – Template – Variation Definition Time • Changed terms: – AUTOSAR Definition – AUTOSAR Meta-model – AUTOSAR Model – AUTOSAR Service – AUTOSAR XML description – Link Time Configuration – Manifest – PDU Meta-Data
2017-12-08	1.3.0	AUTOSAR Release Management	• No content changes
2017-10-27	1.2.0	AUTOSAR Release Management	• No content changes
2017-03-31	1.1.0	AUTOSAR Release Management	Added terms: • Adaptability • Adaptive Application • Adaptive Platform Foundation • Adaptive Platform Services • ASIL Decomposition • Audit • AUTOSAR Adaptive Platform • AUTOSAR Runtime for Adaptive Applications • Cascaded Switch • Cascading Failure • Classic Platform • Common Cause Failure ▽

▽

△

			△ • Dependent Failures • Diagnostic Coverage • Diversity • Ethernet Switch Port Groups • Executable • External Port • Failure Mode • Fault Reaction Time • Fault Tolerant Time Interval • Freedom from Interference • Functional Cluster • Functional Safety Concept • Functional Safety Requirement • Host ECU • Host Port • Hypervisor • Independence • Independent Failures • Internal Port • Link State Accumulation • Machine • Manifest • Master Switch • Microcontroller • Performance • Plausibility • Predictabiliy • Proven In Use Argument ▽

▽

△

			△ • Recovery • Safe State • Safety Case • Safety Goal • Safety Measure • Safety Mechanism • Service Discovery • Service Instance • Service Interface • Service Oriented Communication • Service Proxy • Service Skeleton • Slave Switch • Software package • Software Unit • Systematic Fault • Uplink Port • Virtualization Removed terms: • Accreditation Body • Accreditation • Attestation • Conformance Declaration • Conformance Test Agency (CTA) • First party • Implementation Conformance Statement • Interrupt Logic ▽

▽

△

			△ • Partial Model • Surveillance • Third party Changed terms: • Automotive Safety Integrity Levels • Availability • Acceptance Test Suite • Electronic Control Unit • Error • Fail-safe • Fail-silent • Failure Rate • Failure • Fault Tolerance • Fault • FlexRay Bus • FlexRay Cycle • FlexRay L-PDU-Identifier • FlexRay L-SDU-Identifier • FlexRay Matrix • FlexRay Slot Multiplexing • Graceful Degradation • Fail-degraded • Implementation Conformance Class 1 (ICC1) • Implementation Conformance Class 2 (ICC2) • Implementation Conformance Class 3 (ICC3) ▽

▽

△

			△ • Link Time Configuration • Partitioning • Protocol Control Information • Protocol Data Unit • Post-build Time Configuration • Pre-Compile Time Configuration • Probability of Failure • Redundancy • Risk • Safety • Service Data Unit
2016-11-30	1.0.0	AUTOSAR Release Management	• Migration of document to standard • Following terms added: – AUTOSAR Blueprint – Bypassing – Hook – OS Event – Post-build Hooking – Pre-build Hooking – Rapid Prototyping (RP) – Rapid Prototyping Memory Interface – Rapid Prototyping Tool – Reentrancy – Standardized AUTOSAR Blueprint – Standardized Blueprint – Following terms changed: – Asset – Asynchronous Function – AUTOSAR Application Interface – Availability – ECU Abstraction Layer – Feature ▽

▽

△

			△ – Function – Microcontroller Abstraction Layer (MCAL)
2015-07-31	4.2.2	AUTOSAR Release Management	• Following terms changed: – ECU Abstraction Layer – Standardized AUTOSAR Interface – Hook – OS Event – Post-build Hooking – Pre-build Hooking • Following terms removed: – Software Module
2014-10-31	4.2.1	AUTOSAR Release Management	• Following terms changed: – Data Variant Coding – OS-Application – Post-build time configuration – Standardized AUTOSAR Interface
2014-03-31	4.1.3	AUTOSAR Release Management	• Extended Abbreviations • Following terms changed: – Software Component (SW-C)
2013-03-15	4.1.1	AUTOSAR Administration	• Extended Abbreviations • Following terms added: – Application Interface – Asynchronous Functions – AUTOSAR Application Interface – Dynamic PDU – Life Cycle – MetaDataLength – PDU Meta-Data – Pretended Networking – Synchronous Functions

▽

△

2011-12-22	4.0.3	AUTOSAR Administration	• Extended Abbreviations • Following terms added: – Callback – Callout – ECU
2009-12-18	4.0.1	AUTOSAR Administration	• Following terms added: – AUTOSAR Partial Model – Bus Wake-Up – Empty Function
2009-02-02	3.1.4	AUTOSAR Administration	• Following terms added: – Automotive Safety Integrity Levels (ASIL) – Bit Position – Category 1 Interrupt – Category 2 Interrupt – Code Generator – Coordinate – E2E Profile – Error Detection Rate – Failure Rate – ICC1 (Implementation Conformance Class 1) – ICC2 (Implementation Conformance Class 2) – ICC3 (Implementation Conformance Class 3) – Interrupt Frames – Interrupt Handler – Interrupt Logic – Meta-Model – Mode – Model – Network Interface (NWI) – NM Coordination Cluster – NM Coordinator – Rate Conversion ▽

▽

△

			△ – Residual Error Rate – SAE J1939 – Safety Protocol – Software Component Interface (SW-CI) – Synchronize – Variability – Variant – Variation Binding – Variation Binding Time – Variation Definition Time – Variation Point – Formal adaptations • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Following terms added: – Debugging – Implementation Conformance Statement – Document meta information extended – Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Following terms added: – FlexRay – Vendor ID – Callback – Interrupt frames – Interrupt vector table – Accreditation – Accreditation Body – Conformance Test Agency – Assessment – Surveillance – Attestation – (Conformance) Declaration ▽

▽

△

			△ – First party and – Third party – Safety – ECU Configuration – ECU Configuration Description • Legal disclaimer revised
2006-05-16	2.0	AUTOSAR Administration	• removed and added some terms • rework of several descriptions • formal changes
2006-05-16	1.0	AUTOSAR Administration	• Initial release

Disclaimer

Table of Contents

References

[1] ISO/IEC 9646-1:1994 - Information technology - Open Systems Interconnection - Conformance testing methodology and framework
https://www.iso.org

[2] ISO 17356-3: Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS)

[3] ITEA Project 00009 EAST-EEA Embedded Electronic Architecture - Glossary Version 6.1

[4] IEEE 1471-2000: IEEE Recommended Practice for Architectural Description for Software- Intensive Systems

[5] ISO 26262-1:2018 Part 1: Vocabulary
https://www.iso.org

[6] IEEE 1517-1999: IEEE Standard for Information Technology – Software Life Cycle Processes – Reuse Processes

[7] Explanation of Automotive API
AUTOSAR_AP_EXP_AutomotiveAPI

[8] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[9] Virtual Functional Bus
AUTOSAR_CP_TR_VFB

[10] 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

[11] Specification of AUTOSAR Run-Time Interface
AUTOSAR_CP_SWS_ARTI

[12] CiA 610-1 version 1.0.0 (DSP) - CAN XL specifications and test plans - Part 1: Data link layer and physical coding sub-layer requirements
http://www.can-cia.org

[13] CiA 611-1 version 1.0.0 (DSP) - CAN XL higher layer functions - Part 1: Definition of service data unit types
http://www.can-cia.org

[14] Lehrbuch Grundlagen der Informatik

[15] ISO/IEC 61511 Part 1 Information technology – Software life cycle process, First Edition

[16] ISO 7498 – Information processing systems – Open Systems Interconnection – Basic Reference Model
https://www.iso.org

ISO/IEC 7498-1:1994

[17] Binding Specification Version 1.4.1

[18] ISO/IEC 2382 Part 1 Information technology – Vocabulary – Fundamental Terms, Third Edition

[19] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[20] Specification of Operating System
AUTOSAR_CP_SWS_OS

[21] Specification of Diagnostic Event Manager
AUTOSAR_CP_SWS_DiagnosticEventManager

[22] ISO 26262-6:2018 Part 6: Product development at the software level
https://www.iso.org

[23] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[24] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

[25] Specification of Timing Extensions for Classic Platform
AUTOSAR_CP_TPS_TimingExtensions

[26] ISO 15765-2 – Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part2: Network layer services

[27] ISO 14229-1 – Unified diagnostic services (UDS) – Part 1: Specification and requirements (Release 2006-12)
https://www.iso.org

[28] AUTOSAR Feature Model Exchange Format
AUTOSAR_FO_TPS_FeatureModelExchangeFormat

[29] ISO 17458-1:2013, Road vehicles - FlexRay communication system
https://www.iso.org

[30] Specification of FlexRay Driver
AUTOSAR_CP_SWS_FlexRayDriver

[31] Specification of FlexRay Interface
AUTOSAR_CP_SWS_FlexRayInterface

[32] System Template
AUTOSAR_CP_TPS_SystemTemplate

[33] ISO 17356-1: Road vehicles – Open interface for embedded automotive applications – Part 1: General structure and terms, definitions and abbreviated terms

[34] Specification of ECU Resource Template

AUTOSAR_CP_TPS_ECUResourceTemplate

[35] Translation/Adaptation from VDI Lexikon Informatik und Kommunikationstechnik

[36] Specification of Health Monitoring
AUTOSAR_FO_ASWS_HealthMonitoring

[37] ISO/IEC 2382 Part 20 Information technology – Vocabulary – System Development, First Edition

[38] NIST: Secure Hash Standard (SHS)
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[39] IEEE 1588-2019: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems

[40] UM10204 - I2C-bus specification and user manual
https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[41] ISO 17356-4: Road vehicles – Open interface for embedded automotive applications – Part 4: OSEK/VDX Communication (COM)

[42] ISO 17987:2016 (all parts), Road vehicles – Local Interconnect Network (LIN)
https://www.iso.org

[43] Specification of Communication Management
AUTOSAR_AP_SWS_CommunicationManagement

[44] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[45] DIN 40041 Ausgabe:1990-12 Zuverlässigkeit; Begriffe

[46] IEEE Standard for a High-Performance Serial Bus
http://www.1394ta.org/Technology/Specifications/specifications.htm

[47] Specification of Network Management Interface
AUTOSAR_CP_SWS_NetworkManagementInterface

[48] ISO/IEC 27000:2018 Information technology - Security techniques - Information security management systems - Overview and vocabulary
https://www.iso.org

[49] ISO/IEC 2382 Part 15 Information technology – Vocabulary – Programming Languages, First Edition

[50] ISO/IEC 2382 Part 1 Information technology – Vocabulary – Security, Second Edition

[51] Specification of Update and Configuration Management
AUTOSAR_AP_SWS_UpdateAndConfigurationManagement

[52] Specification of Software Cluster Connection module
AUTOSAR_CP_SWS_SoftwareClusterConnection

[53] XML Specification of Application Interfaces
AUTOSAR_CP_MOD_AISpecification

[54] IEEE Standard for System, Software, and Hardware Verification and Validation

1 Introduction

This document constitutes the comprehensive glossary of AUTOSAR. It contains definitions of all major abbreviations, acronyms and terms used within AUTOSAR. Please note that the document is not intended to be exhaustive. It is important to consult the glossaries of the various working group documents, as these contain additional specific terms within their respective domains.

2 Acronyms, Abbreviations (and Initialisms)

2.1 Explanation

An *Abbreviation* is a shortened form of a word or phrase, like "Temp." for "Temperature", whereas an *Acronym* is a specific type of abbreviation formed from the initial letters of a phrase and pronounced as a single word like "AUTOSAR", which stands for "AUTomotive Open System Architecture". An *Initialism* is similar to an Acronym but is pronounced letter by letter, like "ECU" for "Electronic Control Unit", and there are hybrid forms of an Initialism and an Acronym, like "VLAN" for "Virtual LAN. However, in both colloquial and technical contexts, the terms Abbreviation, Acronym and Initialism are used interchangeably, and that is also the case in this document.

Acronym/ Abbreviation	Description
A	
AA	Adaptive Application
ABC	Abstract Base Class
ABI	Application Binary Interface
ADC	Analog Digital Converter
AES	Advanced Encryption Standard
AMM	Application Mode Management
AP	AUTOSAR Adaptive Platform
API	Application Programming Interface
ARA	AUTOSAR Runtime for Adaptive Applications
ARP	Address Resolution Protocol
ARTI	AUTOSAR Run-Time Interface
ARXML	AUTOSAR XML
ASAM	Association for Standardization of Automation and Measuring Systems
ASD	Abstract System Description
ASIL	Automotive Safety Integrity Levels
ASW	Application SoftWare
ATP	AUTOSAR Template Profile
ATS	Acceptance Test Suite
AUTOSAR	AUTomotive Open System Architecture
B	
BFx	Bitfield functions for fixed point
BSW	Basic Software
BIST	Built-In Self Tests
BSWM	BSW Mode manager

▽

△

Acronym/ Abbreviation	Description
BSWMD	Basic Software Module Description
C	
CAN	Controller Area Network
CC	Communication Controller
CCF	Common Cause Failure
CD	Complex Driver
CP	AUTOSAR Classic Platform
COM	Communication
COMM	COMmunication Manager
CM	Communication Module
CRC	Cyclic Redundancy Check
D	
DAC	Digital to Analog Converter
DCA	Data Class Adapter
DDS	Data Distribution Service
DEM	Diagnostic Event Manager
DES	Data Encryption Standard
DET	Default Error Tracer
DEXT	Diagnostic EXTract
DHCP	Dynamic Host Configuration Protocol
DIO	Digital Input/Output
DLC	Data Length Code
DM	Diagnostic Manager
DoIP	Diagnostics over Internet Protocol
DoS	Denial of Service
DP	Data Point
DSL	Domain Specific Languages
DTC	Diagnostic Trouble Code
DTD	Document Type Definition
E	
E2E	End-to-End
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECU	Electronic Control Unit
EDDSA	Edwards-Curve Digital Signature Algorithm

▽

△

Acronym/ Abbreviation	Description
EEC	Executable Entity Cluster
EEPROM	Electrically Erasable Programmable Read-Only Memory
EM	Execution Management
EOC	Execution Order Constraint
EOCEERG	Execution Order Constraint Executable Entity Reference Groupe
F	
FC	Functional Cluster
FIFO	First In First Out
FIBEX	Field Bus Exchange Format
FO	AUTOSAR Foundation
FOTA	Firmware-Over-The-Air
FPU	Floating Point Unit
FQDN	Fully-Qualified Domain Name
FW	FireWire (by Apple, aka IEEE 1394)
G	
GCM	Galios/Counter Modester
GENIVI	GENeva In-Vehicle Infotainment
GPT	General Purpose Timer
GSM	Global System for Mobile Communication
H	
HMAC	Hash-based Message Authentication Code
HMI	Human Machine Interface
HPC	High performance computer / computing machine
HTMSS	Hardware Test Management Startup and Shutdown
I	
I-PDU	Interaction Layer Protocol Data Unit, see: I-PDU
ICC	Implementation Conformance Class, see: Implementation Conformance Class 1 ff.
ICMP	Internet Control Message Protocol
ICOM	Intelligent COMmunication controller
ICU	Input Capture Unit
ICV	Integrity Check Value
ID	IDentifier
IDL	Interface Description Language
IEC	International Electrotechnical Commission
IFI	Interpolation Floating point
IFx	Interpolation Fixed point

▽

△

Acronym/ Abbreviation	Description
IO	Input/Output
IP	Internet Protocol
ISO	International Standardization Organization
ISR	Interrupt Service Routine
IVC	In-Vehicle Communication
J	
JSON	JavaScript Object Notation
L	
LAN	Local Area Network
LBAP	Language Binding for the Adaptive Platform AP
L-PDU	Link Layer Protocol Data Units
L-SDU	Link Layer Service Data Units
LET	Logical Execution Time
LIFO	Last In First Out
LIN	Local Interconnected Network
LT	Log and Trace
LSB	Least Significant Bit
M	
MAC	Media Access Control
MAC	Message Authentication Code
mC	Microcontroller
MC	Measurement and Calibration
MCAL	Microcontroller Abstraction Layer
MCBD	Multicore BSW Distribution
MCU	MicroController Unit
MD	Message Digest
MDIO	Management Data Input/Output
ME	Mappable Element
MFl	Mathematical Floating point
MFx	Mathematical Fixed point
MIPS	Million Instructions Per Second
MMD	MDIO Manageable Device
MMU	Memory Management Unit
MMI	Man Machine Interface
MOST	Media Oriented Systems Transport
mP	MicroProcessor
MPU	Memory Protection Unit

▽

△

Acronym/ Abbreviation	Description
MSB	Most Significant Bit
MSTP	Microcontroller Specific Test Package
MTU	Maximum Transmission Unit
N	
NM	Network Management
N-PDU	Protocol Data Unit of the Network layer (transport protocol)
N-SDU	Service Data Unit of the Network layer (transport protocol)
NvM	Non-Volatile Memory
NVRAM	Non-Volatile Random Access Memory
O	
OBD	On-Board Diagnostic
OEM	Original Equipment Manufacturer
OIL	ISO 17356-6 (OSEK/VDX Implementation Language)
OS	Operating System
OSEK	Open Systems and their interfaces for Automotive Electronics[1]
P	
PCI	Protocol Control Information
PDEP	Profile of Data Exchange Point
PDU	Protocol Data Unit
PHM	Platform Health Management
PKCS	Public Key Cryptography Standards
PNC	Partial Network Cluster
POSIX	Portable Operating System Interface
PS	Product Supplier
PSK	Least Significant Bit
PTP	Precision Time Protocol
PWM	Pulse Width Modulation
R	
RAM	Random Access Memory
RDG	Runnable Dependency Graph
REST	Representational State Transfer
RfC	Request for Change
ROM	Read-Only Memory
RP	Rapid Prototyping

▽

[1]The original German phrase for this acronym is: **O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug

△

Acronym/ Abbreviation	Description
RPC	Remote Procedure Call
RSA	Cryptographic approach according to Rivest, Shamir, and Adleman
RTE	Runtime Environment, see: RTE Event
S	
SAE	Society of Automotive Engineers, see: SAE J1939
SD	Service Discovery
SDG	Special Data Group
SDU	Service Data Unit
SEooC	Safety Element out of Context
SDV	Software-Defined Vehicle
SHA	Secure Hash Algorithm
SHM	System Health Monitoring
SHWA	Safe Hardware Acceleration
SIL	Safety Integrity Level
SL-LET	System Level Logical Execution Time
SM	State Management
SOA	Service-Oriented Architecture
SOC	Service-Oriented Communication
SOME/IP	Scalable Service-Oriented Middleware over IP
SOVD	Service-Oriented Vehicle Diagnostics
SP	Synchronization Points
SPEM	Software and Systems Process Engineering Meta-model
SPI	Serial Peripheral Interface
SST	Signal Service Translation
SWC/SW-C	Software Component
SWCT/SWC-T	SWC-Template
SYST/SYS-T	System Template
T	
TC	Timed Communication
TC	Test Case
TCP	Transmission Control Protocol
TD	Timing Description
TIMEX	TIMing EXTensions
TLS	Transport Layer Security
TLV	Tag Length Value
TP	Transport Protocol
TSN	Time Sensitive Network

▽

△

Acronym/ Abbreviation	Description
TSY	Time Synchronization
TTL	Time To Live
TTP	Time Triggered Protocol
U	
UDP	User (Universal) Datagram Protocol
UDS	Unified Diagnostic Services
UDPNM	UDP Network Management
UML	Unified Modeling Language
USB	Universal Serial Bus
UTF	Universal coded character set Transformation Format
UUID	Universally Unique Identifier
V	
V2X	Vehicle-To-Everything
VDP	Vehicle Data Protocol
VFB	Virtual Functional Bus
VISS	Vehicle Information Service Specification
VLAN	Virtual Local Area Network
VMM	Vehicle Mode Management
VSA	Variable Size Array
VSM	Vehicle State Manager
W	
WCET	Worst Case Execution Time
WCRT	Worst Case Response Time
X	
XCP	Universal Calibration Protocol

Table 2.1: Acronyms and Abbreviations

3 How to read this document

The title of the sub-chapters is identical to the term to be defined.

3.1 <Term> Template

Definition	<term to be defined>
Initiator	<Functional Cluster which responsible for the term>
Further Explanations	<further explanation of the definition>
Comment	<comment or hints>
Example	<example of the term>
Reference	<reference of definition>

4 Definitions

4.1 Abstract Platform

Definition	A Software Platform defined by AUTOSAR for the definition of functional level communications independent of Classic / Adaptive / Offboard deployments.
Initiator	System Design
Further Explanations	An Abstract Platform exists only in the methodological and modeling sense and is not a physical deployable platform. In the context of AUTOSAR Classic and AUTOSAR Adaptive functional level communications means VFB level communications.
Comment	–
Example	–
Reference	–

4.2 Acceptance Test Suite

Definition	A test case description used in the context of Acceptance Testing
Initiator	Acceptance Testing
Further Explanations	ISO 9646 distinguishes between Abstract Test Suites and Executable Test Suites. For AUTOSAR the earlier relates to the Acceptance Test Specifications, whereas the latter to the test implementations or Acceptance Test Suites.
Comment	–
Example	–
Reference	[1], Parts 1,2 and 4

4.3 Access Control Decision

Definition	The Access Control Decision is a Boolean value indicating if the requested operation is permitted or not. It is based on the identity of the caller and the Access Control Policy
Initiator	Security
Further Explanations	In the case of Identity and Access Management, the 'caller' is an Adaptive Applications.
Comment	–
Example	–
Reference	–

4.4 Access Control Policy

Definition	Access Control Policies are bound to the targets of calls (i.e. `Service` interfaces) and are used to express what `Identity Information` is necessary to access those interfaces.
Initiator	Security
Further Explanations	Policies can be provided through `Configuration` / modeling or by statically pre-programming them into the PDP.
Comment	–
Example	–
Reference	–

4.5 Adaptability

Definition	Adaptability is the ability of a system to adjust itself to changed circumstances in its environment in order to continue to provide the intended `Functionality`.
Initiator	Safety
Further Explanations	One should distinguish between changes in the environment of the system/vehicle ("run-time adaptability") and changes in the development environment where software `Architecture` (like AUTOSAR) is used ("design-time predictability").
Comment	–
Example	–
Reference	• Antonio Carlos Schneider Beck, Carlos Arthur Lang Lisboa, Luigi Carro (eds.), Adaptable Embedded Systems, Springer Science & Business Media, 27 Nov 2012 • Twan Basten, Roelof Hamberg, Frans Reckers, Jacques Verriet, Model-Based Design of Adaptive Embedded Systems, Springer Science & Business Media, 15 Mar 2013

4.6 Adaptive Application

Definition	Software that follows the Adaptive AUTOSAR specifications and therefore can be deployed onto an Adaptive Platform instance. It consists of its implementation, operational data (e.g. map data) and its `Meta-data` given by the Application Design Model. In order to be deployable on different Adaptive Platforms, it only uses ARA programming interfaces including POSIX profile PSE51 APIs.
Initiator	Execution Management
Further Explanations	`Adaptive Applications` are generally more coarse grain than SW-Cs of the `Classic Platform`. They use exclusively Adaptive Platform APIs, and may offer and use services. They are implemented by one or several executables, byte code or libraries with defined entry points and may comprise multiple parts (e.g. libraries, data files).
Comment	The goal of Adaptive Platform is to achieve portability of `Adaptive Applications` among different implementations of the Adaptive Platform at least on source-code level, potentially also on object-code level.
Example	–
Reference	–

4.7 Adaptive Platform Foundation

Definition	Part of an Adaptive Platform implementation, which provides standardized platform `Functionality` to Applications via software interfaces (APIs).
Initiator	Software and `Architecture`
Further Explanations	The Adaptive Platform Foundation includes of core system functionalities such as OS, Execution Manager, Communication Management and Persistency.
Comment	The goal of Adaptive Platform is to achieve portability of `Adaptive Applications` among different implementations of the Adaptive Platform at least on source-code level, potentially also on object-code level.
Example	–
Reference	–

4.8 Adaptive Platform Services

Definition	Standard platform services that is provided by an application which is part of AUTOSAR platform implementation.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.9 Adaptive Platform Machine Configuration

Definition	Configuration model in the AUTOSAR Adaptive Platform that defines the target `Configuration` content whose scope is local to a specific target `Machine`.
Initiator	WG-MT
Further Explanations	The Adaptive Platform Machine Configuration is uploaded with executable code to a target `Machine` and supports the `Integration` of `Executables` onto the machine.
Comment	–
Example	–
Reference	–

4.10 Application Programming Interface

Definition	An Application Programming Interface (API) is the prescribed method of a specific software part by which a programmer writing a program can make requests to that software part.
Initiator	WG-MT
Further Explanations	–
Comment	–

▽

△

Example	[2] (OSEK/VDX Operating System)
Reference	–

4.11 Application Software Component

Definition	An Application Software Component is a specific `Software Component` which realizes a defined `Functionality` on application level and runs on the AUTOSAR infrastructure. It communicates only through the AUTOSAR Runtime Environment (RTE).
Initiator	Software and `Architecture`
Further Explanations	Application Software Components are located "above" the AUTOSAR Runtime Environment.
Comment	–
Example	–
Reference	–

4.12 Architecture

Definition	The fundamental organization of a system embodied in its components, their static and dynamic relationships to each other, and to the environment, and the principles guiding its design and evolution.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	"Static and dynamic" added to definition of [3] .
Example	–
Reference	[4], [3]

4.13 Artifact

Definition	This is a Work Product definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts. At a high level, an artifact is represented as a single conceptual file.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.14 ASIL Decomposition

Definition	See ISO-26262 ([5]) ID 1.7
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.7

4.15 Asserted Property

Definition	A property or quality of a design entity (e.g. SW component or system) is asserted, if the design entity guarantees that this property or quality is fulfilled.
Initiator	Software and `Architecture`
Further Explanations	A property or quality of a design unit can be asserted by the design unit itself or in combination with another design unit.
Comment	–
Example	If the worst case `Execution Time` of a `Task` (w.r.t. a certain CPU etc.) is asserted to be 3 ms, the execution time of this task will under any circumstances be less than or equal to 3 ms.
Reference	Compare `Required Property`

4.16 Assessment

Definition	See ISO-26262 ([5]), ID 1.4
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.4

4.17 Asset

Definition	An item that has been designed for use in multiple contexts.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–

▽

△

Example	An asset can be design, specifications, source code, documentation, test suits, manual procedures, etc. From a security perspective anything that has a value to any of the stakeholders such as critical data (information, software) and critical functions, that could potentially be subject to attacks and possibly, but not necessarily, motivates countermeasures.
Reference	[6], [3]

4.18 Asynchronous Communication

Definition	Asynchronous communication does not block the sending software entity. The sending software entity continues its operation without getting a response from the communication partner(s).
Initiator	Communication
Further Explanations	There could be an acknowledgement by the communication system about the sending of the information. A later response to the sending software entity is possible.
Comment	–
Example	–
Reference	–

4.19 Asynchronous Function

Definition	A `Function` is called asynchronous if the described `Functionality` is not guaranteed to be completed the moment the `Function` returns to the caller.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.20 Atomic Software Component

Definition	Non-composed Software-Component.
Initiator	Software and `Architecture`
Further Explanations	An Atomic `Software Component` might access HW or not, therefore not all Atomic SW-Cs are relocatable.
Comment	–
Example	Application Software-Component, Complex Driver
Reference	–

4.21 Audit

Definition	See ISO-26262 ([5]), ID 1.5
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.5

4.22 Authenticity

Definition	The property that data originated from its purported source.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	NIST SP 800-38B under Authenticity

4.23 Automotive API

Definition	The Automotive API is an interface that allows data-centric communication with the vehicle. It defines how other systems can access selected vehicle data securely and independently of the in-vehicle representation using a standardized interface across vehicle types and manufacturers.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	See [7]

4.24 Automotive API Gateway

Definition	The Automotive API Gateway is the `Functional Cluster` that offers the `Automotive API` and can thus connect clients to vehicle internal data.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	See [7]

4.25 Automotive Safety Integrity Levels

Definition	See ISO-26262 ([5]), ID 1.7
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.7

4.26 AUTOSAR Adaptive Platform

Definition	An adaptive computing platform standardized by AUTOSAR. In a narrow term, it refers to its specification. In a broad term, it may refer to an instance of Adaptive Platform implementation.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.27 AUTOSAR Application Interface

Definition	A set of `Blueprints` which are standardized by AUTOSAR and which can be used for creating `AUTOSAR Interfaces` of an `Adaptive Application`. `AUTOSAR Interfaces` that are derived from `Standardized AUTOSAR Blueprints` are `Standardized AUTOSAR Interfaces`.
Initiator	Application Interfaces
Further Explanations	–
Comment	–
Example	–
Reference	

4.28 AUTOSAR Authoring Tool

Definition	An AUTOSAR Tool used to create and modify `AUTOSAR XML descriptions`.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	System Description Editor
Reference	–

4.29 AUTOSAR Blueprint

Definition	An AUTOSAR Blueprint is a `Blueprint` for an AUTOSAR element. It also includes that it is specified within the AUTOSAR project which attributes are mandatory to be specified for the blueprint of a specific class of AUTOSAR element types as well as how to derive an AUTOSAR object from that blueprint.
Initiator	Application Interfaces
Further Explanations	The AUTOSAR meta-model supports the pre-definition of model elements taken as the basis for further modeling. These pre-definitions are called blueprints. [TPS_STDT_00002]
Comment	–
Example	–
Reference	[8]

4.30 AUTOSAR Converter Tool

Definition	An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	ECU Flattener
Reference	–

4.31 AUTOSAR Definition

Definition	This is the definition of parameters which can have values. One could say that the parameter values are instances of the definitions. But in the meta-model hierarchy of AUTOSAR, definitions are also instances of the meta-model and therefore considered as a description.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.32 AUTOSAR Interface

Definition	The AUTOSAR Interface of a software component refers to the collection of all Ports (`Port`) of that component through which it interacts with other components.
Initiator	Software and `Architecture`
Further Explanations	–

▽

△

Comment	Note that an AUTOSAR Interface is different from a Port Interface. The latter characterizes one specific Port of a component.
Example	–
Reference	See AUTOSAR Specification of [9] Chapter 4 *“Communication on the VFB”*.

4.33 AUTOSAR Meta-model

Definition	The AUTOSAR meta-model is a UML2.0 model that defines the language for describing AUTOSAR systems and related artifacts.
Initiator	Methodology and Templates
Further Explanations	–
Comment	The AUTOSAR XML Schema is derived from the AUTOSAR meta-model.
Example	–
Reference	[10]

4.34 AUTOSAR Model

Definition	This is an instance of the AUTOSAR Meta-model. The AUTOSAR Model represents aspects suitable to the intended use according to the AUTOSAR methodology.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.35 AUTOSAR Partial Model

Definition	In AUTOSAR, the possible Partitioning of models is marked in the meta-model by <<atp Splitable>>. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.36 AUTOSAR Processor Tool

Definition	An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	RTE Generator
Reference	–

4.37 AUTOSAR Runtime for Adaptive Applications

Definition	A set of standard application interfaces provided by `Functional Clusters`, which belong to either `Adaptive Platform Foundation` or `Adaptive Platform Services`.
Initiator	General
Further Explanations	–
Comment	–
Example	–
Reference	–

4.38 AUTOSAR Run-Time Interface (ARTI)

Definition	The AUTOSAR Run-Time Interface shall define an interface between build tools and `Debugging`/`Tracing` tools.
Initiator	Software and `Architecture`
Further Explanations	The interface shall ease and speed up the `Debugging`, `Tracing` and `Verification` of `System` behavior as well as round-trip engineering.
Comment	
Example	–
Reference	AUTOSAR Specification of ARTI ([11])

4.39 AUTOSAR Service

Definition	The term `Service` is used in the layered software architecture to denote the highest layer of the `AUTOSAR Software` architecture that interacts with the application. The term service is used in the meaning defined by the service-oriented architecture. This meaning has the strongest relation to the usage of the term service on the `AUTOSAR Adaptive Platform`. The term service is also used to describe the handling of diagnostic services, e.g. UDS service ReadDataByIdentifier, for the communication between a diagnostic tester and a diagnostic stack on an (AUTOSAR) `Electronic Control Unit` (ECU).
Initiator	Software and `Architecture`

▽

△

Further Explanations	–
Comment	–
Example	–
Reference	–

4.40 AUTOSAR Software

Definition	Application Software or Basic/Platform Software developed according to the AUTOSAR Standard.
Initiator	–
Further Explanations	–
Comment	–
Example	–
Reference	–

4.41 AUTOSAR Tool

Definition	This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an `AUTOSAR Authoring Tool`, an `AUTOSAR Converter Tool`, an `AUTOSAR Processor Tool` or as a combination of those.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.42 AUTOSAR XML description

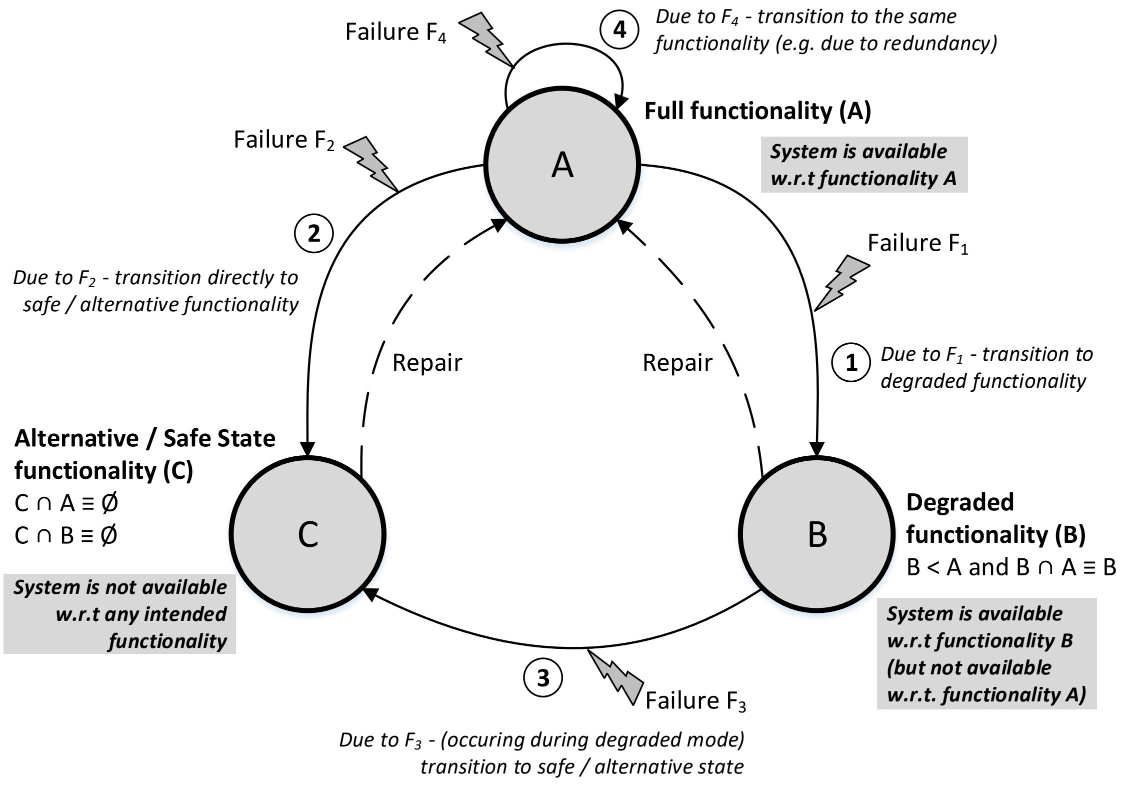
Definition	In AUTOSAR this is a serialized AUTOSAR model. In fact an AUTOSAR XML description is the XML representation of an `AUTOSAR Model`. The AUTOSAR XML description can consist of several files. Each individual file represents an `AUTOSAR Partial Model` and must validate successfully against the `AUTOSAR XML Schema`.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.43 AUTOSAR XML Schema

Definition	The AUTOSAR XML Schema is an XML language definition for exchanging (AUTOSAR Model) and descriptions.
Initiator	Methodology and Templates
Further Explanations	The AUTOSAR XML Schema is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR Meta-model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.
Comment	–
Example	–
Reference	–

4.44 Availability

Definition	1. Availability is the ability of the system to perform a Function A completely according to its specification. 2. The ratio of the total time the system is performing a function A (according to 1) during a given interval to the length of the interval. Alternative: The probability that the system is performing the function A at a specified time t. 3. In a degraded Mode the system has the ability to perform a subset B of A if full A is not available. In this case, the Functionality B is available.
Initiator	Safety
Further Explanations	see Figure 4.1 for more details.
Comment	1. Degraded modes are covered by this definition (see example)
Example	1. Power Steering: if the support Function fails it is not available while the steering as a base function has full availability. 2. From a security perspective availability is an attribute that ensures correct and timely access upon demand by an authorized entity.
Reference	See [5], ID 1.8


Failure F_4
4
Due to F_4 - transition to the same functionality (e.g. due to redundancy)
Full functionality (A)
A
System is available w.r.t functionality A
Failure F_2
2
Due to F_2 - transition directly to safe / alternative functionality
Failure F_1
1
Due to F_1 - transition to degraded functionality
Repair
Repair
Alternative / Safe State functionality (C)
C ∩ A ≡ Ø
C ∩ B ≡ Ø
C
System is not available w.r.t any intended functionality
B
Degraded functionality (B)
B < A and B ∩ A ≡ B
System is available w.r.t functionality B (but not available w.r.t. functionality A)
3
Failure F_3
Due to F_3 - (occuring during degraded mode) transition to safe / alternative state

NOTES:
1. This diagram assumes the system is already in full intended functionality mode and neglects the start-up, shut-down, reset, etc. transitions which lead to this state.
2. The repair transitions are conceptual transitions for ilustration purposes.

Figure 4.1: Explanation of Availability

4.45 Basic Software

Definition	The Basic Software (BSW) provides the infrastructural (schematic dependent and schematic independent) functionalities of an `Electronic Control Unit`. It consists of `Integration Code` and `Standard Software`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	`Microcontroller Abstraction Layer` (MCAL), `AUTOSAR Service`, Communication Layer
Reference	–

4.46 Basic Software Module

Definition	A collection of software files (code and description) that define a certain basic software `Functionality` present on an `Electronic Control Unit`.
Initiator	Software and `Architecture`
Further Explanations	`Standard Software` may be composed of several software modules that are developed independently. A software module may consist of `Integration Code` and/ or `Standard Software`.
Comment	–
Example	A Digital IO Driver, `Complex Driver`, OS are examples of Basic Software Modules.
Reference	–

4.47 Bit Position

Definition	In AUTOSAR the bit position N within an `I-PDU` denotes the bit I, with I = N modulo 8, within the byte J, with J = N / 8. The byte J and bit position I is interpreted in accordance to the definition in ISO 17356-4 (OSEK/VDX Communication).
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.48 Blueprint

Definition	This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta-model resp. types, this process is not an instantiation.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	Standardized Blueprint and `AUTOSAR Blueprint`.
Reference	–

4.49 Bulk Data

Definition	"Bulk Data" is a set of data such big in size, that standard mechanisms used to handle smaller data sets become inconvenient. This implies that bulk data in a software system are modeled, stored, accessed and transported by different mechanisms than smaller data sets.
Initiator	Software and `Architecture`
Further Explanations	Bulk data are typically handled by adding a level of abstraction (e.g. files) which separates the containment of the data from the internal structure.

▽

△

Comment	The critical size, above which data must be regarded as bulk data depends on the technical infrastructure (e.g. bus system) and the considered `Use Case` (transport, storage etc.).
Example	Data on a persistent medium which has a capacity of a few kBytes (e.g. EEPROM) can be directly accessed via memory addresses, address offsets can be mapped to symbols of a programming language: No bulk data mechanisms are needed. For media with bigger capacity this becomes inconvenient or even impossible, so that a file system is used: The data are treated as bulk data.
Reference	–

4.50 Bus Mirroring

Definition	Forwarding information from an internal vehicle bus to an external bus, e.g. the diagnostic `Connector`.
Initiator	Communication
Further Explanations	Bus Mirroring is used to make internal buses accessible to external testers such that internal buses can be debugged in case of errors.
Comment	Because the external (or intermediary) buses typically do not have sufficient bandwidth to transport all information from an internal bus, filters have to be applied to select the frames that are relevant to the analysis.
Example	An `Electronic Control Unit` does not go to sleep in a vehicle. The engineer wants to check the NM traffic to check for irregular behavior, e.g. concerning partial networking information.
Reference	–

4.51 Bus Wake-Up

Definition	A bus wake-up is caused by a specific wake pulse on the bus defined within the specification of the dedicated communication standard (e.g. CAN, LIN, FR). A bus wake-up initiates that the transceiver and controller leave their energy saving `Mode` and enter normal mode to start bus communication again.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.52 Bypassing

Definition	The experimental incorporation of new `Functionality` within an ECU image.
Initiator	Runtime Environment
Further Explanations	Bypassing involves the incorporation of new `Functionality` or to replace existing functionality to an existing ECU image without requiring that the image be rebuilt.

▽

△

Comment	Bypassing can be either "internal" where the new/ replacement Functionality is present on the ECU image or "external" where an Rapid Prototyping Tool provides the functionality out with the ECU.
Example	An RP tool intercepts the output of a bypassed RunnableEntity via the RP Memory Interface and replaces the value with the bypass result. Subsequent RunnableEntitys then process the bypass value rather than the original result.
Reference	–

4.53 Calibration

Definition	Calibration is the adjustment of parameters of Software Components realizing the control functionality (namely parameters of AUTOSAR SW-Cs, ECU abstraction or Complex Driver.
Initiator	Software and Architecture
Further Explanations	Only those software modules can be calibrated, which are above RTE and ECU Abstraction and CDD. Calibration is always done at post-build time. Used techniques to set Calibration Data include end-of-line programming, garage programming and adaptive calibration (e.g. in the case of anti-pinch protection for power window).
Comment	–
Example	The Calibration of the engine control will take into account the production differences of the individual motor this System will control.
Reference	–

4.54 Call Point

Definition	A point in a Software Component where the SWC enforce an execution entity (Entry Point) in another SWC.
Initiator	Software and Architecture
Further Explanations	–
Comment	–
Example	Request Service Send Information
Reference	–

4.55 Callback

Definition	Functionality that is defined by an AUTOSAR module so that lower-level modules (i.e. lower in the Layered Software Architecture) can provide Notification as required (e.g. when certain events occur or asynchronous processing completes).
Initiator	Software and Architecture

▽

△

Further Explanations	In AUTOSAR, modules usually provide a register mechanism for callback functions which is set through `Configuration`. A module provides callbacks so that other modules can initiate its processing while the module calls `Callbacks` to execute functionality that could not be specified by AUTOSAR, i.e. `Integration Code`.
Comment	–
Example	(from the viewpoint of a particular SWS): The module being specified (Msws) should be informed about an `Event` in another module (Mexternal). In this example, Msws calls Mexternal to perform some processing and can only resume when Mexternal completes. Upon completion, Mexternal calls Msws's `Callback Function`. That is, the called module (Mexternal) CALLS the calling module (Msws) BACK when complete ==> a `Callback`.
Reference	–

4.56 Callout

Definition	`Function` stubs that the system designer can replace with code to add functionality to a module which could not be specified by AUTOSAR.
Initiator	Software and `Architecture`
Further Explanations	A module calls callouts to execute functionality that could not be specified by AUTOSAR, i.e. `Integration Code` while the module provides `Callbacks` so that other modules can initiate its processing. Callouts can be separated into two classes: 1) callouts that provide mandatory functionality and thus serve as a `Hardware Abstraction Layer` 2) callouts that provide optional functionality
Comment	–
Example	In the EcuM: For class 1): EcuM_EnableWakeupSources For class 2): The Init Lists (EcuM_AL_DriverInitZero)
Reference	–

4.57 CAN XL

Definition	CAN XL specifies a protocol and physical layer with higher `Data` rate and clearer separation of concerns, that builds on CAN 2.0/FD and features tunneling of Ethernet frames.
Initiator	Communication
Further Explanations	Besides supporting tunneled legacy CAN 2.0 / CAN FD communication between any type of `Electronic Control Unit` (ECU) it shall be also possible to directly tunnel IEEE 802.3 Ethernet frames for e.g. participation of IP communication. This leads to a vehicle wide possible communication with same semantic used everywhere regardless physical connection (CAN XL / Ethernet) or communication paradigm (Signal- and `Service`-based communication).
Comment	CAN XL will help bridge the gap between current CAN implementations and current 100 Mbit Ethernet solutions. On the same `Network` segment, both CAN 2.0/FD/XL and Ethernet traffic can coexist. Baudrate is not fixed to 10 Mbit like at 10BASE-T1S but can be adjusted flexible up to 20 Mbit
Example	Tunneling Ethernet frames to CAN XL connected ECUs.
Reference	[12, CiA610-1], [13, CiA-611-1]

4.58 Cascaded Switch

Definition	A Cascaded Switch is an Ethernet switch that exists of at least two Ethernet switches: a `Master Switch` and a `Slave Switch`. The Master Switch and the Slave Switch are connected by `Uplink Ports`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	Request Service Send Information
Reference	–

4.59 Cascading Failure

Definition	See ISO-26262 ([5]), ID 1.13
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.13

4.60 Category 1 Interrupt

Definition	Category 1 (Cat1) Interrupts are supported by the OS but their code is only allowed to call a very small subset of OS functions. Furthermore they can bypass the OS. The code of Category 1 Interrupts depends (normally) on the used compiler and `Microcontroller`. Category 1 Interrupts are not allowed to use the ISR() macro. Category 1 Interrupts need to implement/ establish their own `Interrupt Frame`. Nevertheless they have to be configured in order to be included in the `Interrupt Vector Table`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.61 Category 2 Interrupt

Definition	Category 2 (Cat2) Interrupts are supported by the OS and their code can call a subset of OS functions. The definition of the Cat2 Interrupt must use the ISR() macro in order to be recognized by the OS. The `Interrupt Frame` of a Category 2 Interrupt is managed by the OS.
Initiator	Software and `Architecture`

▽

△

Further Explanations	–
Comment	–

Example	ISR(timer1)
	{
	/* here is the code which handles timer1 interrupts */
	...
	}

Reference	–

4.62 Causality of Transmission

Definition	Transmit order of PDUs with the same identifier (instances of `Protocol Data Units`, PDUs) from a source `Network` is preserved in the destination network.
Initiator	Communication
Further Explanations	Transmission of `Protocol Data Unit` (PDU) with the same identifier has a particular temporal order in a given source `Network`. After routing over a `Gateway` the temporal order of transmission of PDUs in a destination network may be changed. Only in case that the temporal order is the same, causality is given. Otherwise causality is violated. Causality can be in contradiction to prioritization of PDUs.
Comment	–
Example	–
Reference	–

4.63 Classic Platform

Definition	A `Software Platform` defined by AUTOSAR for deeply embedded systems and Application Software with high demands regarding predictability, `Safety` and responsivness.
Initiator	General
Further Explanations	–
Comment	–
Example	–
Reference	–

4.64 Client

Definition	Software entity which uses services of a `Server`.
Initiator	Software and `Architecture`
Further Explanations	The client and the `Server` might be located on one `Client-Server Communication` or distributed on different calculation units (e.g. `Electronic Control Unit` (ECU), external diagnostic tester).
Comment	Adapted from [14]

▽

△

Example	–
Reference	[14]

4.65 Client-Server Communication

Definition	A specific form of communication in a possibly distributed system in which software entities act as `Clients`, `Servers`, or both, where 1...n clients are requesting services via a specific protocol from typically one Server.
Initiator	Communication
Further Explanations	Client-server communication can be realized by synchronous or asynchronous communication. • `Client` takes initiative: requesting that the `Server` performs a `Service`, e.g. client triggers action within server (server does not start action on its own) • Client is after service request blocked / non-blocked • Client expects response from server: `Data Flow` (and control flow, if blocked) One example for 1 client to n server communication (currently not supported) is a functional request by diagnosis. This has to be treated as a specific exception.
Comment	–
Example	Internet (TCP/IP)
Reference	–

4.66 Client-Server Interface

Definition	The Client-Server interface is a special kind of `Port Interface` used for the case of `Client-Server Communication`. The Client-Server interface defines the operations that are provided by the `Server` and that can be used by the `Client`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	AUTOSAR Specification of `Virtual Functional Bus` ([9])

4.67 Cluster Signal

Definition	A cluster signal represents the aggregating `System Signal` on one specific communication cluster. Cluster signals can be defined independently of frames. This allows a development methodology where the signals are defined first, and are assigned to frames in a later stage.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	–

4.68 Code Generator

Definition	The Code Generator consumes complete and correctly formed XML for a `Basic Software Module` (BSW) and generates code and data that configures the module.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.69 Code Variant Coding

Definition	Adaptation of SW by selection of functional alternatives according to external requirements
Initiator	Software and `Architecture`
Further Explanations	Code Variant Coding might influences RTE (Runtime Environment) and Basic Software Modules, not only the application software modules. Code Variant Coding is always done at pre-compile time or at link time. Code Variant Coding also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed dependent automatic locking).
Comment	In case of the C language the #if or #ifdef directive can be used for creating code variants. Code Variant Coding is a design time concept.
Example	The same window lifter ECU is used for cars with 2 and 4 doors, however different code segments have to be used in both cases.
Reference	

4.70 Common Cause Failure

Definition	See ISO-26262 ([5]), ID 1.14
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.14

4.71 Communication Attribute

Definition	Communication attributes define, according to the development phase, behavioral as well as implementation aspects of the AUTOSAR communication patterns.
Initiator	Communication
Further Explanations	The exact characteristics of the communication patterns provided by AUTOSAR (`Client-Server Communication` and `Sender-Receiver Communication`) can be specified more precisely by communication attributes.
Comment	See AUTOSAR Specification of [9] Chapter 4 *"Communication on the VFB"*.
Example	–
Reference	AUTOSAR Specification of `Virtual Functional Bus` ([9])

4.72 Complex Driver

Definition	A software entity not standardized by AUTOSAR that can access or be accessed via `AUTOSAR Interfaces` and/or `Basic Software Module` APIs.
Initiator	Software and `Architecture`
Further Explanations	CDD used to be the acronym for Complex Device Driver, but is not limited to drivers.
Comment	–
Example	• Communication stack CDD to support the communication on a bus not supported by AUTOSAR • Reuse of legacy SW • Integration of software with high HW interaction requirements within a standardized AUTOSAR Architecture
Reference	–

4.73 Composition

Definition	An AUTOSAR Composition encapsulates a collaboration of `Software Components`, thereby hiding detail and allowing the creation of higher abstraction levels. Through Delegation `Connectors` a Composition explicitly specifies, which `Ports` of the internal components are visible from the outside. AUTOSAR Compositions are a type of Components, e.g. they can be part of further compositions.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	See `Virtual Functional Bus` Specification, [9] Chapter 4 *"Communication on the VFB"*
Example	–
Reference	AUTOSAR Specification of `Virtual Functional Bus` ([9])

4.74 Compositionality

Definition	Compositionality is given when the behavior of a `Software Component` or subsystem of a `Systems` is independent of the overall system load and `Configuration`.
Initiator	Software and `Architecture`
Further Explanations	Compositionality is an important property of deterministic systems. This property leads to a complete decoupling of systems. Smooth subsystem `Integration` without backlashes is then easily achievable.
Comment	–
Example	A new component or a subsystem can be added to a system without changing the behavior of the original components.
Reference	–

4.75 Conditioned Signal

Definition	The conditioned signal is the internal electrical representation of the `Electrical Signal` within the `Electronic Control Unit` (ECU). It is delivered to the processor and represented in voltage and time (or, in case of logical signals, by high or low level).
Initiator	General
Further Explanations	The `Electrical Signal` usually can not be processed by the peripherals directly, but has to be adopted. This includes amplification and limitation, conversion from a current into a voltage and so on. This conversion is performed by some electronic devices in the ECU and the result of the conversion is called the Conditioned Signal. The description means for the `Conditioned Signal` can also be the same as for `Technical Signals` and Electrical Signals, but limited to electrical voltage.
Comment	–
Example	–
Reference	–

4.76 Confidentiality

Definition	The property that data or information is not made available or disclosed to unauthorized persons or processes.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	NIST SP 800-66 Rev. 1 under Confidentiality from 45 C.F.R., Sec. 164.304

4.77 Configuration

Definition	The arrangement of hardware and/or software elements in a system.
Initiator	Software and Architecture
Further Explanations	A configuration in general takes place before runtime.
Comment	–
Example	–
Reference	[3], [15]

4.78 Confirmation

Definition	Service primitive defined in the ISO/OSI Reference model ([16]). With the 'confirmation' service primitive a service provider informs a service user about the result of a preceding service request of the service user.
Initiator	Software and Architecture
Further Explanations	A confirmation is e.g. a specific notification generated by the underlying layer to inform about a Message Transmission Error.
Comment	–
Example	–
Reference	[17]

4.79 Connector

Definition	A Connector connects Ports of Software Components and represents the flow of information between those ports.
Initiator	Software and Architecture
Further Explanations	–
Comment	For more information see AUTOSAR Specification of VFB ([9])
Example	AssemblyConnector, DelegationConnector
Reference	[9]

4.80 Control Flow

Definition	The directed transmission of information between multiple entities, directly resulting in a state change of the receiving entity.
Initiator	Software and Architecture
Further Explanations	A state change could result in an activation of a schedulable entity.
Comment	–
Example	–
Reference	–

4.81 Coordinate

Definition	To control and harmonize two or more events or operations to act in an organized and predictable way.
Initiator	Communication
Further Explanations	–
Comment	–
Example	Two NM Channels can be coordinated to synchronize different stages of `Network` sleep.
Reference	

4.82 Data

Definition	A reinterpretable representation of information in a formalized manner suitable for communication, interpretation or processing.
Initiator	General
Further Explanations	–
Comment	–
Example	`Flag`, `Notification`, etc.
Reference	[18]

4.83 Data Element

Definition	Data elements are declared within the context of a `Sender-Receiver Interface`. They serve as the data units that are exchanged between sender and receiver.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	[19]

4.84 Data Flow

Definition	The directed transmission of `Data` between multiple entities. The transmissioned data are not directly related to a state change at the receiver side.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Asynchronous communication.
Reference	–

4.85 Data Variant Coding

Definition	Adaptation of SW by setup of certain characteristic `Data` according to external requirements.
Initiator	Software and `Architecture`
Further Explanations	Data Variant Coding might influence RTE (Runtime Environment) and `Basic Software Modules`, not only the application software modules (Multiple `Configuration` parameter sets are needed). `Variant Coding` also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed dependent automatic locking). Used techniques to select variants include end-of-line programming and garage programming.
Comment	The major difference with `Calibration` is that this later doesn't aim to adapt the SW `Functionality` itself but only aims to adjust the characteristic data of the SW to the HW/SW environment. Characteristic `Data` in the source code of a software `Function` have a significant impact on the functionality of the software.
Example	- Steering wheel controller adaptation to the left or right side can be done with `Variant Coding`. (Selection of the `Configuration`.) - Country related adaptation of MMI with respect to speed and/or temperature unit (km/h vs. mph, ℃ vs. F).
Reference	

4.86 Deadline

Definition	The point in time when an execution of an entity must be finished.
Initiator	Software and `Architecture`
Further Explanations	A deadline is calculated dependent on its local reference system.
Comment	–
Example	–
Reference	[20]

4.87 Debugging

Definition	Debugging is the process of gathering information in case of a software problem. The information is used to analyze the software problem.
Initiator	Software and `Architecture`
Further Explanations	To analyze and later fix a software problem, in many cases more information than the one provided by the software API is necessary. This can be for example the state of internal variables of the software or a trace of the communication. The information can be collected by different means, e.g. an emulator or a `Tracing` tool for the communication bus.
Comment	–
Example	–
Reference	–

4.88 Dependability

Definition	Dependability is defined as the trustworthiness of a computer system such that reliance can justifiable be placed on the `Service` it delivers.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	[3]

4.89 Dependent Failure

Definition	See ISO-26262 ([5]), ID 1.22
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.22

4.90 Diagnostic Coverage

Definition	See ISO-26262 ([5]), ID 1.25
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.25

4.91 Diagnostic Event

Definition	A diagnostic event defines the atomic unit that can be handled by the DEM module. The status of a diagnostic event represents the result of a monitor. The DEM receives the result of a monitor from a `Software Component` (SW-C) via the RTE or other `Basic Software Modules`.
Initiator	Diagnostics
Further Explanations	–
Comment	For definition of 'monitor' see chapter [21] Chapter 7.3.5 *"Diagnostic monitor definition"* in Specification of DEM.
Example	–

▽

△

Reference	[21]

4.92 Diversity

Definition	See ISO-26262 ([5]), ID 1.28
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.28

4.93 Dynamic PDU

Definition	`Protocol Data Unit` with dynamic identifier.
Initiator	Communication
Further Explanations	Dynamic PDUs are PDUs where the <bus> identifier (e.g. CAN ID) is dynamically assigned (transmission) or evaluated (reception) at run time.
Comment	AUTOSAR supports two types of dynamic PDUs in CanIf: CanIf_SetDynamicTxId (only transmission), and PDUs with Meta-data (reception and transmission).
Example	PDU with variable source address, encoded in the CAN ID, e.g. ISO15765 NormalFixed.
Reference	–

4.94 Dynamic Routing

Definition	The routing of signals or `Protocol Data Units` in a `Gateway` can be changed throughout operation without change of the operation `Mode` of the Gateway.
Initiator	Communication
Further Explanations	Dynamic routing requires the change of routing tables during operation. It is not intended to use dynamic routing in the `Gateway`.
Comment	–
Example	–
Reference	[3]

4.95 E2E Profile

Definition	A functional and complete description of a specific communication stack in terms of data structures, `Services`, behavioral state-machines, `Error` handling. E2E Profiles are defined in AUTOSAR E2E Library. An E2E Profile is configurable by runtime parameters. A specific set of runtime parameters is called E2E profile `Variant`. In order to reach interoperability, the application developers should use the E2E profile variants defined in the E2E library.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.96 E2E Protection Alive Counter

Definition	An Alive Counter is a counter which is incremented in every transmission request. The counter value is checked at the receiver side, whether it changes at all, but not if the counter was incremented correctly.
Initiator	Safety
Further Explanations	With an alive counter, the receiver monitors if new values arrive, this means it checks if the sender is still alive
Comment	–
Example	[22] Annex D.2.4 EXAMPLE 3: Communication protocols can contain information such as identifiers for communication objects, keep-alive messages, alive counters, sequence numbers, error detection codes and error-correcting codes.
Reference	–

4.97 E2E Protection Sequence Counter

Definition	A Sequence Counter is a counter which is incremented in every transmission request. The value of the counter value is checked at the receiver side for each message, whether it was incremented correctly.
Initiator	Safety
Further Explanations	A Sequence Counter is used to check whether messages got lost or a message is repeated.
Comment	–
Example	[22] Annex D.2.4 EXAMPLE 3: Communication protocols can contain information such as identifiers for communication objects, keep-alive messages, alive counters, sequence numbers, error detection codes and error-correcting codes.
Reference	–

4.98 ECU Abstraction Layer

Definition	The Electronic Control Unit (ECU) Abstraction Layer is located above the Microcontroller Abstraction Layer and abstracts from the ECU schematic. It is implemented for a specific ECU and offers an API for access to peripherals and devices regardless of their location (onchip/offchip) and their connection to the Microcontroller (Port pins, type of interface). Task: Make higher software layers independent of the ECU hardware layout.
Initiator	Software and Architecture
Further Explanations	The ECU Abstraction Layer consists of the following parts: • I/O Hardware Abstraction • Communication Hardware Abstraction • Memory Hardware Abstraction • Crypto Hardware Abstraction • Onboard Device Abstraction Properties: • Implementation: μC independent, ECU hardware dependent • Upper Interface (API): μC and ECU hardware independent, dependent on signal type
Comment	–
Example	–
Reference	See Layered Software Architecture ([23])

4.99 ECU Configuration

Definition	Activity of integrating and configuring one ECU's software.
Initiator	Methodology and Templates
Further Explanations	Further Explanations: ECU Configuration denotes the activity when one ECU's software is set up for a specific usage inside the Electronic Control Unit. In AUTOSAR the ECU Configuration activity is divided into "Pre-compile time", "Link time" and "Post-build time" configuration.
Comment	–
Example	–
Reference	ECU Configuration Description, Pre-Compile Time Configuration, Link Time Configuration, Post-build Time Configuration.

4.100 ECU Configuration Description

Definition	Output of the ECU Configuration activity containing the values of Configuration parameters and references.
Initiator	Methodology and Templates
Further Explanations	ECU Configuration Description holds the Configuration parameter values and references to other module's configurations which have been defined in the ECU Configuration activity.

▽

△

Comment	ECU Configuration Description may contain the whole ECU Configuration information or only the parts relevant for a specific Configuration step (e.g. Pre-Compile Time Configuration).
Example	–
Reference	ECU Configuration Description (see [24]), Pre-Compile Time Configuration, Link Time Configuration, Post-build Time Configuration.

4.101 ECU HW

Definition	A container term for any combination of AP Machines and CP ECUs. In general terms, an ECU-HW is typically abstracted/independent from physical or virtual realization and does not convey any HW-Housing i.e. multiple physical ECU-HWs or a physical ECU-HW providing multiple virtual ECU-HWs (unless otherwise stated).
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.102 ECU Instance

Definition	An ECU Instance represents a single instantiation of a Classic Platform stack, that may run on physical ECU-HW (without Hypervisor) or on virtual ECU-HW (with hypervisor).
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.103 Electrical Signal

Definition	The electrical signal is the electrical representation of Technical Signals. Electrical signals can only be represented in voltage, current and time
Initiator	General
Further Explanations	When a sensor processes the Technical Signal it is converted into an Electrical Signal. The information can be provided in the current, the voltage or in the timely change of the signal (e.g. a pulse width modulation).
Comment	To describe the Electrical Signal the same means as for the Technical Signal can be used, limited to electrical current and voltage.
Example	–

▽

△

Reference	–

4.104 Electronic Control Unit (ECU)

Definition	Embedded computer `System` consisting of at least one CPU and corresponding peripherals which are placed in one housing.
Initiator	General
Further Explanations	An ECU is typified by a connection to one or more in-vehicle networks, sensors and actuators.
Comment	–
Example	–
Reference	–

4.105 Empty Function

Definition	Any C function defined by an AUTOSAR specification which does not implement or alter behavior required to accomplish the assigned functional responsibility.
Initiator	Software and `Architecture`
Further Explanations	As such an empty function in the context of AUTOSAR can still have code but this code shall not impact the state `Machine` other than `Error` reporting. Auxiliary code like validating arguments to report to the DET does not constitute functional behavior because without the code and proper calling this code would still fulfill its architectural responsibility.
Comment	–
Example	–
Reference	–

4.106 Entry Point

Definition	A point in a `Software Component` where an execution entity of the SW-C begins.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	• `Service` of the `Server` in `Client-Server Communication` • Reaction after receive Information (`Notification`)
Reference	–

4.107 Error

Definition	See ISO-26262 ([5]), ID 1.36
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.36

4.108 Error Detection Rate

Definition	Ratio between detected lost/faulty words/symbols/blocks, divided by the total number of symbols/words/blocks sent.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.109 Ethernet Switch Port Groups

Definition	Ethernet Switch Port groups are Ethernet switch `Ports` of an Ethernet switch which are grouped to so called `Port` groups. Ethernet Switch Port groups are only relevant for the host `Electronic Control Unit` (ECU). Ethernet Switch Port Groups are derived from the model per VLAN and per PNC. The host port is participating in all port groups. A Ethernet Switch Port Group could be a mix of internal and external ports.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.110 Event

Definition	State change of a hardware and/or software entity.
Initiator	Software and `Architecture`
Further Explanations	See `OS Event`, `RTE Event`, `Diagnostic Event` and `Event Message (SOME IP)`
Comment	–
Example	–
Reference	–

4.111 Event Message (SOME/IP)

Definition	Event - a message sent by an Electronic Control Unit (ECU) implementing a Service Instance to an ECU using this service instance (Publish/Subscribe).
Initiator	Communication
Further Explanations	Eventgroup - a logical grouping of 1 or more events. An eventgroup is part of a Service.
Comment	–
Example	–
Reference	–

4.112 Executable

Definition	Part of an application which consists of either a file containing executable code with a defined entry point and suitable for the platform instance as the target of deployment (deployment time) or software code which is ready to be integrated for a specific platform.
Initiator	Execution Management
Further Explanations	In POSIX systems, an executable is typically running within a single Process. Therefore, intra-executable communication is different from inter-executable communication and should therefore be considered during design time of an executable.
Comment	–
Example	–
Reference	–

4.113 Executable Entity Cluster

Definition	A set of Executable Entities (EEs) and a reference to a set of Execution Order Constraints (EOCs) between these EEs. The Executable Entity Cluster is formed for the purpose of Mapping EEs to Logical Execution Time (LET) intervals and to tasks. Several EECs may be mapped to the same LET interval.
Initiator	Software and Architecture
Further Explanations	An Executable Entity Cluster (EEC) groups EEs from any Software Component. EEs with different triggers/periods can be part of the same EEC, if the triggers/periods are harmonic (i.e. that all periods are integer multiples of the smallest period). The EEC can reference a Logical Execution Time (LET) interval specification. See Figure 4.2 for an example.
Comment	The set of EOCs is cycle-free.
Example	–
Reference	–

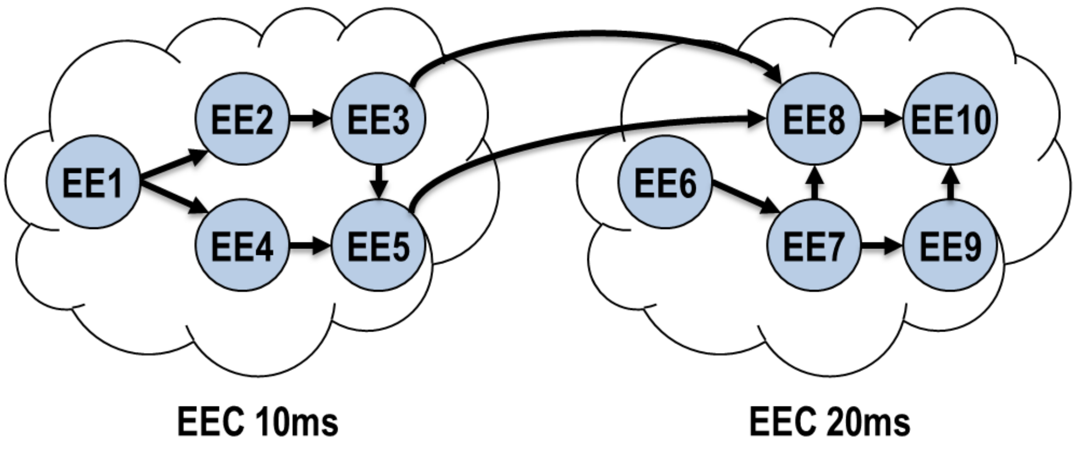


Figure 4.2: Executable Entity Cluster Example

4.114 Execution Order Constraint

Definition	Defines the execution order between instances of Executable Entities (EEs) within the same LET interval.
Initiator	Software and `Architecture`
Further Explanations	Example in Figure 4.3 shows an EE 2 that is marked as successor/directSuccessor of an EE 1, if the execution order of the instances of the respective Executable Entities is 12 (2 runs after 1) within the LET interval.
Comment	–
Example	–
Reference	[25]

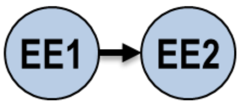


Figure 4.3: Example of an Execution Order Constraint

4.115 Execution Time

Definition	The time during which a program is actually executing, or more precisely during which a certain thread of execution is active.
Initiator	Software and `Architecture`

▽

△

Further Explanations	The execution time of software is the time during which the CPU is executing its instructions. The time the CPU spends on `Task` switches or on the execution of other pieces of software is not considered here. See also: `Response Time`, `Worst Case Execution Time`, `Worst Case Response Time`.
Comment	–
Example	–
Reference	–

4.116 Exit Point

Definition	A point in a `Software Component` where an execution entity of the SW-C ends.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Return point.
Reference	–

4.117 External Port

Definition	External Ports are ports of an automotive Ethernet switch used to communicate over an Ethernet physical connection with other `Electronic Control Units` (ECUs) (e.g. 100BASE-TX, 100BASE-T1).
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.118 Fail-operational

Definition	Property of a system or `Functional Unit`. Describes the ability of a system or `Functional Unit` to continue normal operation at its output interfaces despite the presence of hardware or software faults.
Initiator	Safety
Further Explanations	–
Comment	1. Typically, a fail-operational system or `Functional Unit` has no `Safe State`. 2. `Safety` means are not regarded as a part of the normal functionality respectively operation.
Example	Braking system
Reference	–

4.119 Fail-safe

Definition	Property of a system or `Functional Unit`. In case of a fault the system or `Functional Unit` transits to a `Safe State`.
Initiator	Safety
Further Explanations	Fail safe systems needs to have a `Safe State`. Note: not all the systems have a safe state.
Comment	–
Example	–
Reference	See also note of [5], ID 1.137

4.120 Fail-silent

Definition	Fail-silent is a property of a system in which no output is produced in the presence of a fault. In automotive domain, fail-silent systems are usually only used if the next hierarchical system level provides a safe-state.
Initiator	Safety
Further Explanations	Fail-silent is a special case of the fail-safe property.
Comment	–
Example	The fail-silent property can be used to avoid that "babbling idiots" disturb the overall communication.
Reference	–

4.121 Failure Mode

Definition	See ISO-26262 ([5]), ID 1.40
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.40

4.122 Failure

Definition	See ISO-26262 ([5]), ID 1.39
Initiator	Safety
Further Explanations	Termination is a reduction in, or loss of, ability of an element or an item to perform a `Function` as required. There is a difference between "to perform a function as required" (stronger definition, use-oriented) and "to perform a function as specified", so a `Failure` can result from an incorrect specification.

▽

△

Comment	–
Example	–
Reference	See [5], ID 1.39

4.123 Failure Rate

Definition	See ISO-26262 ([5]), ID 1.41
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.41

4.124 Fault

Definition	See ISO-26262 ([5]), ID 1.42
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.42

4.125 Fault Detection

Definition	The action of monitoring errors and setting fault states to specific values is called fault detection.
Initiator	Software and `Architecture`
Further Explanations	The different states are called "not detected"/ "present"/ "intermittent or maturing"/... The names of the fault states are following the ISO/SAE norms; however there is a coordination step in between the states of the DTCs (Diagnostic Trouble Code see definition in ISO 15765/ ISO14229) and the states of the faults. The SW-C's Fault Detection is executed decentralized, e.g. each `Software Component` (SW-C) sets the state of a fault according to the defined fault qualification (SW-C Template). Therefore the Fault Detection is implemented in the SW-C (SW-C could be either `Application Software Component` or Basic Software Component). There are exceptions; these will be pointed out individually for each fault. The SW-C's developer will define the conditions (=fault qualification), when these conditions are fulfilled the SW-C notifies a fault to the Diagnostic Memory Management.
Comment	–
Example	–
Reference	[26], [27], [9]

4.126 Fault Reaction

Definition	In case of a `Failure` of a `Software Component` (SW-C) there is a specific action to be carried out. This action is called "Fault Reaction".
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Fault Reactions can be implemented decentralized in the SW-C. There might also be the need of coordinating the fault reactions since there are reactions excluding each other. This will be done by a central fault reaction manager.
Reference	–

4.127 Fault Reaction Time

Definition	See ISO-26262 ([5]), ID 1.44
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.44

4.128 Fault Tolerance

Definition	Ability to deliver the specified functionality in the presence of one or more specified faults.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.129 Fault Tolerant Time Interval

Definition	See ISO-26262 ([5]), ID 1.45
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.45

4.130 Feature

Definition	A Feature is a notable characteristic of a system.
Initiator	General
Further Explanations	AUTOSAR defines and interacts with many entities where the term Feature can be applied (e.g. the AUTOSAR standard itself, its implementations, `Electronic Control Units` (ECUs) built with AUTOSAR, AUTOSAR Authoring Tools, AUTOSAR Feature Model). For each usage the term Feature may be used in a refined way - which is then defined for that specific usage (e.g. [TPS_ FeatureModelExchangeFormat]).
Comment	–
Example	CAN FD support, Automatic windshield wiper, Editing of the FlexRay schedule
Reference	[3], [28]

4.131 Firewall

Definition	Functional element that inspects and filters `Network` traffic based on firewall rules.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	–

4.132 Flag

Definition	A piece of data that can take on one of two values indicating whether a logical condition is true or false.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	`Notification` flag
Reference	–

4.133 FlexRay

Definition	Automotive time-triggered and fault-tolerant `Network` communication protocol that is standardized in the ISO 17458 ([29]) and provides options for deterministic data that arrives in a predictable time frame as well as dynamic `Event`-driven data.
Initiator	Communication
Further Explanations	FlexRay provides a communication cycle with a pre-defined space for static and dynamic data.
Comment	–

▽

△

Example	–
Reference	For FlexRay specifications please see ISO 17458 ([29]). For more details about FlexRay in Classic AUTOSAR please see [30], [31], [32]

4.134 Foundation

Definition	Foundation contains the generic artifacts that are common for `AUTOSAR Adaptive Platform` and AUTOSAR `Classic Platform` to ensure compatibility between: • Classic- and `AUTOSAR Adaptive Platform` • Non-AUTOSAR platforms to AUTOSAR platforms In AUTOSAR the particular realization of those generic artifacts is described in `AUTOSAR Adaptive Platform` and `Classic Platform` and may differ.
Initiator	General
Further Explanations	The role of the Foundation in AUTOSAR is two fold: 1) To group those `Artifacts` which are agnostic of, or outside of, the immediate scope of the Classic and `AUTOSAR Adaptive Platform`. Example: Protocol specification (PRS), AUTOSAR `Abstract Platform` or Non-AUTOSAR platform artifacts. 2) To provide a repository for those artifacts, which have content applicable to both Classic and `AUTOSAR Adaptive Platform`. Example: Requirement Specification (RS), Model (MMOD or MOD), Template Specification (TPS), Adaptive Software Specification (ASWS) artifacts. This depth of this commonality between Classic and `AUTOSAR Adaptive Platform` is typically detailed further in the respective artifacts.
Comment	–
Example	
Reference	–

4.135 Frame

Definition	`Data` unit according to the data link protocol specifying the arrangement and meaning of bits or bit fields in the sequence of transfer across the transfer medium.
Initiator	Communication
Further Explanations	–
Comment	–
Example	A CAN frame consists of up to 8 bytes of payload `Data` and additional protocol specific bits / bit fields (e.g. CAN-Identifier).
Reference	[33]

4.136 Frame PDU

Definition	A PDU that fits into 1 frame instance e.g. it does not need to be fragmented across more than 1 frame for transmission over a `Network`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.137 Freedom from Interference

Definition	See ISO-26262 ([5]), ID 1.49
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.49

4.138 Freshness

Definition	Data Freshness implies that the data is recent and it ensures that replayed messages in a replay attack are detected.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	–

4.139 Function

Definition	1. A `Task`, action or activity that must be accomplished to achieve a desired outcome. 2. A part of programming code that is invoked by other parts of the program to fulfill a desired purpose. 3. In mathematics, a function is an association between two sets of values in which each element of one set has one assigned element in the other set so that any element selected becomes the independent variable and its associated element is the dependent variable.
Initiator	Software and `Architecture`
Further Explanations	–

▽

△

Comment	Due to the different meanings in texts using the term application the appropriate meaning should be explained in detail or referenced.
Example	1. C-Code Function 2. Y=f(x)
Reference	[3]

4.140 Functional Cluster

Definition	A Functional Cluster is a high level logical grouping of functionality.
Initiator	Software and `Architecture`
Further Explanations	Note 1: Functional Clusters implement one or more `Features` or parts of features. Note 2: Functional Clusters are represented in our logical architecture.
Comment	–
Example	–
Reference	–

4.141 Functional Network

Definition	A logical structure of interconnections between defined functional parts of `Features`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.142 Functional Safety Concept

Definition	See ISO-26262 ([5]), ID 1.52
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.52

4.143 Functional Safety Requirement

Definition	See ISO-26262 ([5]), ID 1.53
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.53

4.144 Functional Unit

Definition	An entity of software or hardware, or both, capable of accomplishing a specified purpose.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	ECU, Software Component, ...
Reference	[18]

4.145 Functionality

Definition	Functionality comprises User-visible and User-non-visible functional aspects of a `System`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	
Example	Functionality of a communication system is a user-non-visible aspect.
Reference	–

4.146 Gateway

Definition	A gateway is `Functionality` within a `Classic Platform` ECU that performs a `Frame` or signal `Mapping Function` between two communication systems. Communication system in this context means e.g. a CAN system or one channel of a FlexRay system.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	`Gateway ECU`

4.147 Gateway ECU

Definition	A Gateway ECU is an `Electronic Control Unit` that is connected to two or more communication channels, and performs `Gateway Functionality`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	`Gateway`

4.148 Graceful Degradation

Definition	Graceful Degradation: The system continues to operate in the presence of errors, accepting a partial degradation of `Functionality` or `Performance` during `Recovery` or repair. Found in the literature also as "fail soft".
Initiator	Safety
Further Explanations	–
Comment	`Safety` means are not regarded as a part of the normal `Functionality` respectively operation. Also known as: Fail-reduced, Fail-soft
Example	"Limp home" `Functionality` for `Electronic Control Unit` (ECU) (reduce torque to assure an arrival at home or service station)
Reference	See also: [5]: 3.181 - warning and degradation strategy.

4.149 Hardware Abstraction Layer

Definition	A hardware abstraction layer is a software layer that serves as an abstraction layer between the physical hardware and its software. It allows to access the hardware resources through programming interfaces.
Initiator	–
Further Explanations	In AUTOSAR `Classic Platform` the Hardware Abstraction Layer is realized by the `Microcontroller Abstraction Layer` and `ECU Abstraction Layer`.
Comment	–
Example	–
Reference	Layered Software Architecture

4.150 Hardware Connection

Definition	HW Connections are used to describe the connection of `Hardware Elements` among each other. It defines/characterizes the interrelationship among HW Elements (for abstract modeling). The `Hardware Ports` of HW Elements serve as connection points for this purpose.

▽

△

Initiator	General
Further Explanations	In AUTOSAR are 2 kinds of HW Connections defined: • Assembly HW Connection • Delegation HW Connection
Comment	–
Example	–
Reference	[34]

4.151 Hardware Element

Definition	The HW Element is the main describing element of an Hardware Connection. It provides Hardware Ports for being interconnected among each others. A generic HW Element specifies definitions valid for all specific HW Elements.
Initiator	General
Further Explanations	A HW Element is the piece or a part of the piece to be described with the ECU Resource Template. It uses other elements as primitives: This means HW elements can be nested (through HW Containers, a hierarchical structure of HW Elements). At the lowest level a HW Element only uses primitives
Comment	–
Example	–
Reference	[34]

4.152 Hardware Interrupt

Definition	Interrupt triggered by HW Event.
Initiator	Software and Architecture
Further Explanations	2 sorts of HW events • Processor-intern: events as for example division by zero, arithmetical overflow, non-implemented instruction • Processor-extern: events as for example response of peripheral device (e.g. PWM), Memory Error, Timer
Comment	–
Example	–
Reference	Translation/Adaptation from [35]

4.153 Hardware Port

Definition	The HW port exposes functionality to the exterior of the Hardware Element. HW elements can be connected via a Hardware Connection. It defines a connection Endpoint for the HW Element.
Initiator	Communication

▽

△

Further Explanations	HW elements provide HW ports for being interconnected among each others. Each HW port has a name which is unique within the HW element it is located in.
Comment	–
Example	–
Reference	[34]

4.154 Health Indicator

Definition	Health Indicators (HIs) are an evaluation metric of current system `Performance` with regard to `Safety` requirement
Initiator	Safety
Further Explanations	Health Indicator format: <HI_ID, `Performance`, `Reliability`, SubsystemState>
Comment	–
Example	–
Reference	[36]

4.155 Hook

Definition	An intervention point within ECU software for the exchange of data.
Initiator	Runtime Environment
Further Explanations	–
Comment	–
Example	Hooks used to read `Electronic Control Unit` (ECU) variables and/ or overwrite ECU variables with values generated by `Rapid Prototyping` algorithm.
Reference	–

4.156 Host ECU

Definition	A Host ECU is a `Electronic Control Unit` (ECU) that controls one or more automotive Ethernet switches (e.g. switch on / off the Ethernet switch and its ports, read and write the Ethernet switch `Configuration`). For this purpose the Host ECU is connected to the Ethernet switch over a common control interface (e.g. SPI, MDIO). The Host ECU also take part in the `Network` communication. For this purpose the Host ECU is connected by a data interface (e.g. MII) to a specific Ethernet switch `Port` (host port). It transmits and receives Ethernet frames.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.157 Host Port

Definition	A host port is a port of an automotive Ethernet switch where the data interface (e.g. MII) of the `Host ECU` is connected to. The host port could either be an `Internal Port` or an `External Port`. The host port has a special role from the perspective of the software (see link accumulation and port groups).
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.158 Hypervisor

Definition	Low-level software that provides and manages several virtual machines in one physical machine. Maybe an independent software or contained as an OS functionality.
Initiator	Execution Management
Further Explanations	Shared physical resources are either exclusively assigned to single virtual machine, or accessed through virtual device which is managed by Hypervisor. Various hardware and software mechanisms can support the efficient implementation of virtual devices.
Comment	–
Example	–
Reference	–

4.159 Identity and Access Management (IAM)

Definition	IAM is about managing access rights of `Adaptive Applications` to interfaces of the `Adaptive Platform Foundation` and `Services`.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	–

4.160 Identity Information

Definition	The access control is decided / enforced upon the identity information which represents properties of the Adaptive Applications.
Initiator	Security
Further Explanations	–

▽

△

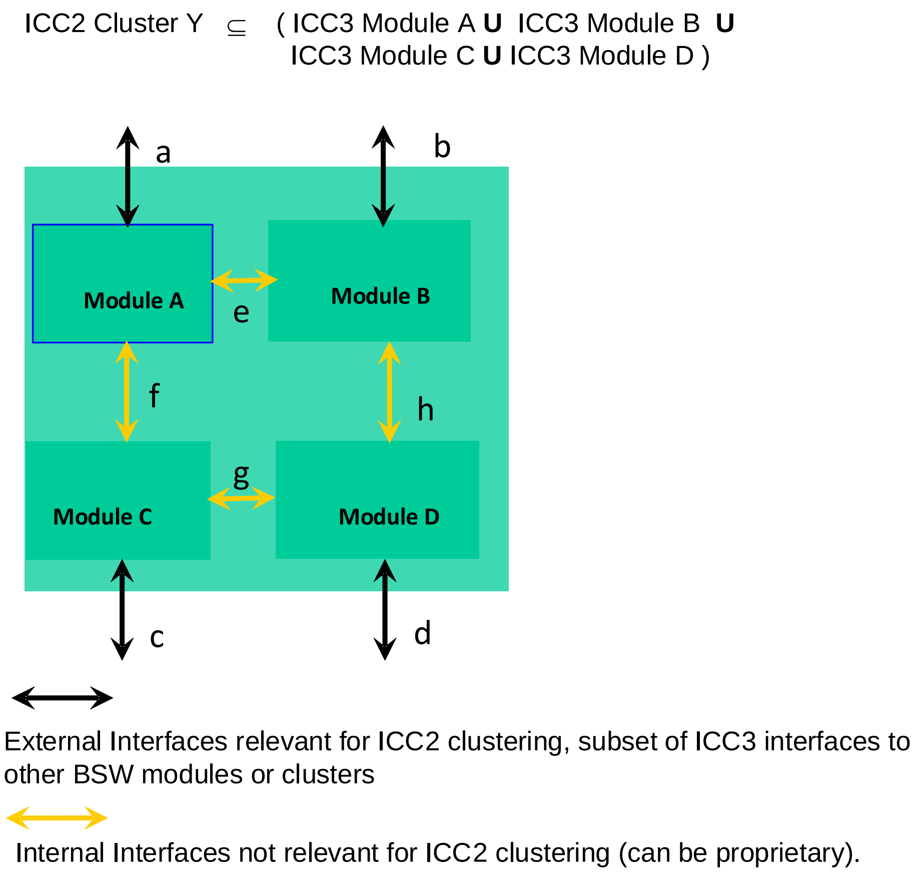
Comment	–
Example	An example for identity information are Capabilities.
Reference	–

4.161 Implementation Conformance Class 1 (ICC1)

Definition	An ICC1 cluster offers a `Software Component Interface` and/or `Network Interface`. The Software Component Interface and Network Interface of an ICC1 cluster provide the functional behavior as specified in the AUTOSAR specifications on ICC3 level.
Initiator	Software and `Architecture`
Further Explanations	In an ICC1 cluster the basic software is regarded as a black box. It allows legacy platforms to migrate to AUTOSAR: - to be integrated into an AUTOSAR `Network` - to support `Software Components` (SW-C). The features of an ICC1 cluster can be a subset of the ICC3 features (e.g. FlexRay not used). This has to be indicated in the Implementation Conformance Statement (ICS). The `Functionality` represented in AUTOSAR by the RTE must be a part of any ICC1 cluster that provides an SW-CI. Typically an ICC1 cluster - is not structured into Basic Software (BSW) modules (ICC3) or BSW module clusters (ICC2) - has a proprietary internal structure and might consist of legacy/proprietary or highly optimized code. An ICC1 cluster shall provide an interface to the boot loader. ICC1 shall support SWC compatible configuration for SW-CI and AUTOSAR `Network` compatible `Configuration` for NWI.
Comment	Up to Release 4.0 the boot loader architecture is not standardized in AUTOSAR. Therefore the term ICC1 is not applicable to the boot loader architecture itself.
Example	–
Reference	–

4.162 Implementation Conformance Class 2 (ICC2)

Definition	ICC2 clusters logically related ICC3 Basic Software (BSW) modules (2...N modules). The number of Cluster Features in an ICC2 cluster is a subset of the union of the number of features of the clustered ICC3 modules.
Initiator	Software and `Architecture`
Further Explanations	Each ICC2 cluster presents a subset of the clustered ICC3 module's interfaces. ICC2 cluster provides the functional behavior as specified in the AUTOSAR specifications on ICC3 level. ICC2 cluster have a proprietary internal structure and might consist of proprietary or highly optimized code. ICC2 shall support AUTOSAR ECU `Configuration` description as an input for the Cluster Configuration It shall be possible to combine ICC2 Clusters and ICC3 Modules in a `Basic Software` (BSW) Architecture. Application interface Conformance (above RTE, `Software Component Interface`) and Bus Conformance (`Network Interface`) must be testable for a BSW which contain one or more ICC2 clusters.
Comment	–
Example	See Figure 4.4 for an example.
Reference	–


ICC2 Cluster Y ⊆ (ICC3 Module A **U** ICC3 Module B **U** ICC3 Module C **U** ICC3 Module D)
a
b
Module A
e
Module B
f
h
g
Module C
Module D
c
d

External Interfaces relevant for ICC2 clustering, subset of ICC3 interfaces to other BSW modules or clusters

Internal Interfaces not relevant for ICC2 clustering (can be proprietary).

Figure 4.4: ICC2 Cluster example

4.163 Implementation Conformance Class 3 (ICC3)

Definition	For ICC3 the AUTOSAR BSW consists of BSW modules as defined in the `Basic Software Module` List, including the RTE. ICC3 is the highest level of granularity.
Initiator	Software and `Architecture`
Further Explanations	All `Basic Software Modules` as defined in the CP SWS BSWGeneral including the RTE, must comply with the defined interfaces and `Functionality` as specified in their respective software specification document (SWS).
Comment	–
Example	–
Reference	–

4.164 Independence

Definition	See ISO-26262 ([5]), ID 1.61
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.61

4.165 Independent Failures

Definition	See ISO-26262 ([5]), ID 1.62
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.62

4.166 Indication

Definition	`Service` primitive defined in the ISO/OSI Reference Model ([16]). With the service primitive ’indication’ a service provider informs a service user about the occurrence of either an internal `Event` or a service request issued by another service user.
Initiator	Software and `Architecture`
Further Explanations	An indication is e.g. a specific `Notification` generated by the underlying layer to inform about a Message Reception `Error`.
Comment	–
Example	–
Reference	–

4.167 Integration

Definition	The progressive assembling of system components into the whole system.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	[37]

4.168 Integration Code

Definition	Code that the Integrator needs to add to an AUTOSAR System, to adapt non-standardized functionalities. Examples are `Callouts` of the ECU State Manager and `Callbacks` of various other `Basic Software Modules`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.169 Integrity

Definition	The property that data or information have not been altered or destroyed in an unauthorized manner.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	NIST SP 800-66 Rev. 1 under Integrity from 45 C.F.R., Sec. 164.304

4.170 Integrity Check Value

Definition	The result of a cryptographic `Function` used to ensure that unauthorized modifications of a message are detected.
Initiator	Global Time `Synchronization`
Further Explanations	–
Comment	See [38]
Example	The value might be the result of the keyed hash `Function` HMAC-SHA256.
Reference	[39]

4.171 Inter-Integrated Circuit I2C

Definition	I2C (Inter-Integrated Circuit) is a 2-wire serial data bus. It was developed by Philips Semiconductors (now NXP Semiconductors). I2C is a simply structured bus system and is widely used in automotive industry.
Initiator	Semiconductors
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	See [40]

4.172 Intermediate PNC Coordinator

Definition	An intermediate PNC (Partial Network Cluster) coordinator is located beneath a Top-level PNC Coordinator or a further intermediate PNC coordinator and above a PNC Leaf Node or a further intermediate PNC coordinator within a PNC network. The intermediate PNC coordinator forwards received PNC requests and own PNC requests to its Top-level PNC Coordinator. An intermediate PNC coordinator processes a received PN shutdown message immediately, forwards it to its subordinated ECUs (either a further intermediate PNC coordinator or a PNC leaf nodes), releases the indicated PNCs and resets the PN reset timer(s) for those PNCs.
Initiator	Partial Networking
Further Explanations	–
Comment	An intermediate PNC coordinator always immediately processes received PN shutdown messages
Example	–
Reference	–

4.173 Internal Port

Definition	Internal Ports are ports of an automotive Ethernet switch used for local communication (Host ECU or Cascaded Switch)
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.174 Interrupt Frame

Definition	An interrupt frame is the code which handles the entering/leaving of (C written) Interrupt Service Routines. This code is Microcontroller specific and often written in assembly language. Interrupt frames are typically generated by the OS generation tool.
Initiator	Software and Architecture
Further Explanations	–
Comment	–
Example	–
Reference	

4.175 Interrupt Handler

Definition	In the case of a Category 2 interrupt, the ISR is synonymous with Interrupt Handler. In the case of Category 1 interrupt the Interrupt Handler is the `Function` called by the `Hardware Interrupt` vector. In both cases the Interrupt Handler is the user code that is normally a part of the `Basic Software Module`. So the Interrupt Handler is a user level piece of code.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.176 Interrupt Service Routine

Definition	A software routine called in case of an `Interrupt`.
Initiator	Software and `Architecture`
Further Explanations	ISRs have normally higher priority than normal processes and can only be suspended by another ISR which presents a higher priority than the one running.
Comment	–
Example	–
Reference	[35]

4.177 Interrupt Vector Table

Definition	An interrupt vector table is a table of interrupt vectors that associates the `Interrupt Service Routine` with the corresponding interrupt request (typically by an array of jumps or similar mechanisms).
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.178 Interrupt

Definition	`Event` that enforces the processor to change its state. This interruption causes the normal sequence of instructions to be stopped. Once an interrupt occurred, the running software entity is suspended and an `Interrupt Service Routine` (the one dedicated to this interrupt) is called.
Initiator	Software and `Architecture`

▽

△

Further Explanations	Two sorts of interrupts exist: Hardware Interrupts and Software Interrupts.
Comment	–
Example	–
Reference	Translation/Adaptation from [35]

4.179 Intrusion Detection System

Definition	A system that is responsible for detecting, recording and reporting Security events.
Initiator	Security
Further Explanations	An AUTOSAR compliant IDS consist of Security sensors, Intrusion Detection System Manager (IdsM) and Intrusion Detection System Reporter (Idsr).
Comment	–
Example	–
Reference	–

4.180 Invalid Flag

Definition	For a signal in a Protocol Data Unit (PDU) an optional invalid flag can be added to the PDU payload layout. This Flag indicates the validity of other signals in the payload. In case the invalid flag of a signal is set to true in a PDU instance, the respective signal in the payload of the PDU instance does not contain a valid signal value.
Initiator	Communication
Further Explanations	This mechanism may be used in Gateways to indicate that parts of a PDU do not contain valid data.
Comment	–
Example	–
Reference	–

4.181 Invalid Value of Signal

Definition	For a signal in a Protocol Data Unit (PDU) an optional invalid value can be defined.
Initiator	Communication
Further Explanations	The invalid value is element of the signal value range that can be represented and transported by the signal. The invalid value is the value that is used in all situations where the receiver should be notified that the value in a signal is not valid.
Comment	–
Example	In case a PDU for a destination network of a Gateway is composed from two PDUs of two different source Networks, the Failure to receive one PDU can be indicated as invalid values in the respective signals of the transmitted PDU in the destination Network.
Reference	–

4.182 I-PDU

Definition	Interaction Layer Protocol Data Unit Collection of messages for transfer between nodes in a Network. At the sending node the Interaction Layer (IL) is responsible for packing messages into an I-PDU and then sending it to the Data Link Layer (DLL) for transmission. At the receiving node the DLL passes each I-PDU to the IL which then unpacks the messages sending their contents to the application.
Initiator	Communication
Further Explanations	–
Comment	–
Example	ISO 17356-4 (OSEK/VDX Communication) specifies an Interaction Layer and works on I-PDUs
Reference	[41]

4.183 Life Cycle

Definition	The course of development/evolutionary stages of a model element during its life time.
Initiator	Methodology and Templates
Further Explanations	A life cycle consists of a set of life cycle states. A life cycle state can be attached to an element in parallel to its version information. A typical life cycle is {valid, obsolete} and means that a valid element is up to date when first introduced but is substituted later by a new one and therefore gets the life cycle state "obsolete".
Comment	–
Example	–
Reference	–

4.184 LIN Bus Idle

Definition	Bus Idle is defined as no transition between recessive and dominant bit values on the LIN bus.
Initiator	Communication
Further Explanations	LIN slave nodes observe the LIN line for bus idle state. After a specific duration of bus idle (bus idle timeout), slave nodes enter bus sleep Mode.
Comment	Synonym "LIN Bus Inactivity"
Example	–
Reference	[42]

4.185 Link State Accumulation

Definition	The link state of a certain switch Port group is accumulated by embracing the link state of each port that is part of the port group. The rule how to embracing the link state is specified in the Ethernet Interface.
Initiator	Communication

▽

△

Further Explanations	–
Comment	–
Example	–
Reference	–

4.186 Link Time Configuration

Definition	The configuration of the SW module is done until link time.
Initiator	Methodology and Templates
Further Explanations	The object code of the SW modules receives parts of its configuration from another object code file or it is defined by linker options.
Comment	–
Example	Initial value of a signal.
Reference	–

4.187 Logical Execution Time (LET)

Definition	Is a fixed time interval. Input data is read at the beginning of this interval and output data is written at the end of the interval. Processing of the data is limited within the time interval.
Initiator	Software and `Architecture`
Further Explanations	The logical execution time (LET) is a real-time programming abstraction. Conceptually, a LET program execution has three steps: read the program input (in zero time), then execute, and finally write the output (in zero time) exactly when the time has elapsed since reading input. Hence, communication logically happens instantaneously at fixed points in time and the program executes within a time window (`Execution Time`) with a `Deadline` represented by the LET. When the execution of a program finishes before the `Deadline`, writing the output is (logically) delayed until the `Deadline`. This makes the `Deadline` a logical upper and lower bound for the execution. Thus, using a faster processor does not result in lower `Response Time`, but in decreased core utilization. Therefore, the behavior of the application is the same on any platform able to execute the program within the LET interval.
Comment	Practical implementation of the LET may use buffers to avoid write bursts at the end of a LET interval. Instead, only buffers are switched.
Example	See example in Figure 4.5.
Reference	Henzinger, T.A. et al: Giotto: A Time-Triggered Language for Embedded Programming. In: Proceedings of the IEEE, vol.91, 2003. Pp. 94-99

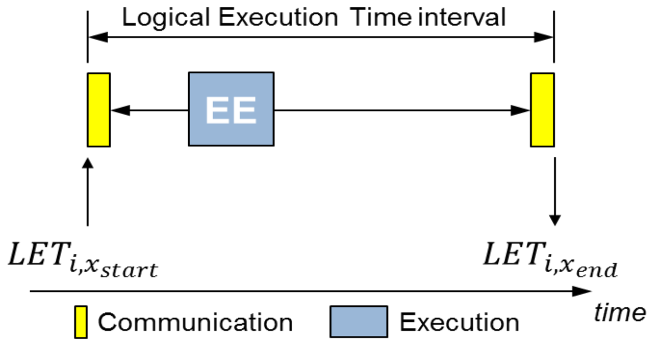


Figure 4.5: Example of Logical Execution Time

4.188 Log and Trace

Definition	Log and Trace provides interfaces for applications to forward logging and `Tracing` information onto the communication bus, the console, or to the file system.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.189 Logging

Definition	Logging is the activity of collecting information about arbitrary, not necessarily correlated events within a `System` during runtime.
Initiator	Software and `Architecture`
Further Explanations	The purpose of logging is to get information for finding and resolving defects of one or multiple programs running on a `System`. Therefore, it should also be usable in the field within a released product and the created logs are intended to be human readable.
Comment	Based on the currently active log level thresholds, logging may have a considerable timing and/or load impact on the `System`, which must be considered during further analysis. In contrast to measuring, logging does not focus on raw data values but on aggregated information within a `System` that is represented by events. In contrast to tracing, logging does not focus on recording internal program flow or state variables but focuses on collecting events that are explicitly added by a software developer on source code level. In contrast to diagnostics, logging does not make a statement on a system's health state but focuses on debug information that allows for detection and resolution of defects.
Example	Operation reporting, `Error` logging, printf output
Reference	–

4.190 Machine

Definition	A Machine is a single instantiation of an `AUTOSAR Adaptive Platform` stack, that may run on physical ECU-HW (without `Hypervisor`) or on virtual ECU-HW (with hypervisor).
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.191 Manifest

Definition	A Manifest represents a piece of AUTOSAR model description that is created to support the `Configuration` of an `AUTOSAR Adaptive Platform` product and which is uploaded to the AUTOSAR Adaptive Platform product, potentially in combination with other artifacts (like binary files) that contain `Executable` code to which the Manifest applies.
Initiator	Methodology and Templates
Further Explanations	Manifests are often used to denote a piece of `Configuration` content that ships along a given piece of software and is used to deploy the software in the field. Three examples of manifest are: - Execution Manifest - `Service` Instance Manifest - `Machine` Manifest
Comment	The Manifest may contain platform implementation dependent data, as well as generic data derived from Application System Description.
Example	–
Reference	–

4.192 Mappable Element

Definition	A mappable element is a part of a MCAL module which can be assigned to a partition via a reference parameter in the Base Software Module Description (BSWMD) of the module. Mappable elements allow the formal description on how MCAL modules are available respectively- distributed on partitions (and thus cores).
Initiator	General
Further Explanations	The type- and scope of the `Functionality` represented by a mappable element strongly depends on the MCAL module. A mappable element may closely represent the hardware (e.g. a channel) but can also represent subsets of HW units as well as groups of HW units.
Comment	Single partition MCAL driver define the complete driver as mappable element. Thus indicating that advanced multi-partition use-cases are not supported.
Example	Adc channel, CAN controller
Reference	–

4.193 Mapping

Definition	Mapping designates the distribution of elements in the logical view to elements in the physical view.
Initiator	Software and `Architecture`
Further Explanations	In general several entities may be allocated to one container but an entity may be allocated to only one container.
Comment	–
Example	a) Mapping of AUTOSAR Signals onto Frames (for inter-ECU communication). b) Mapping of SWC onto ECUs (Distribution of the SW-Components to the ECUs).
Reference	–

4.194 Master Switch

Definition	A Master Switch is an Ethernet switch where the host `Port` is located.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.195 MCAL Signal

Definition	The MCAL signal is the software representation of the `Conditioned Signal`. It is provided by the `Microcontroller Abstraction Layer` (MCAL) and is further processed by the ECU abstraction.
Initiator	General
Further Explanations	The processing unit is accessing the `Conditioned Signal` through some peripheral device that typically digitizes the `Conditioned Signal` into a software representation. The transformation from the `Conditioned Signal` to the MCAL Signal has to take the digitalization `Error` into account in order to provide information about the quality loss between the `Technical Signal` and the MCAL Signal.
Comment	–
Example	–
Reference	–

4.196 Meta-data

Definition	Meta-data is data about data
Initiator	Methodology and Templates
Further Explanations	Meta-data includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc..
Comment	–
Example	–
Reference	–

4.197 MetaDataItem

Definition	Defined item of Meta-data for a `Protocol Data Unit` e.g. a diagnostic address, a Message address, a CAN ID, or a J1939 node address.
Initiator	Communication
Further Explanations	An ordered list of Meta Data Items defines the layout of `PDU Meta Data`. Each MetaDataItem has a fixed type and length, and enables the accessing modules to parse the `PDU Meta Data`, and to access items of the types that are relevant for the module.
Comment	Meta-data was revised with AUTOSAR 4.3.
Example	A PDU exchanged between CanIf and PduR can carry the CAN ID as a MetaDataItem to enable range routing of CAN messages.
Reference	–

4.198 Microcontroller

Definition	`Hardware Element` that integrates computing and communication resources as well as peripheral circuits in a single chip, including memories.
Initiator	General
Further Explanations	Microcontrollers are normally designed for small embedded systems and allow hardware designs with minimal amount of external parts. Microcontroller designs are normally optimized for silicon area and often support hard real-time and high-`Integrity` demands.
Comment	Classic AUTOSAR is intended for Microcontroller based embedded systems.
Example	–
Reference	–

4.199 Microcontroller Abstraction Layer

Definition	Software layer containing drivers to enable the access of onchip peripheral devices of a `Microcontroller` and offchip memory mapped peripheral devices by a defined `Application Programming Interface`. Task: make higher software layers independent of the `Microcontroller`.

▽

△

Initiator	Software and Architecture
Further Explanations	The Microcontroller Abstraction Layer is the lowest software layer of the Basic Software. The Microcontroller Abstraction Layer consists of the following parts: • I/O Drivers • Communication Drivers • Memory Drivers • Crypto Drivers • Microcontroller Drivers Properties: • Implementation: μC dependent • Upper Interface (API): standardizable and μC independent
Comment	–
Example	Examples of drivers located in the Microcontroller Abstraction Layer are: • onchip eeprom driver • onchip adc driver • offchip flash driver
Reference	[23]

4.200 Middleware

Definition	A software layer that separates application SW from more technical layers like software stack, operating system or drivers. Middleware allows the communication of heterogenous applications with each other by providing an abstraction.
Initiator	Safety
Further Explanations	Middleware describes SW in a layered structure. It serves as a connection between at least two other software parts. In embedded software environments they usually consist of an upper layer, which is closer to application or user and a lower layer, which is closer to technical features or components. The upper layer passes commands or requests to the middleware whereas the middleware processes commands or requests and passes them to the lower layer. A Middleware can be seen as a proxy and allows a higher grade of abstraction, supports portability and maintainability of software.
Comment	The middleware approach is also used in non-embedded software systems.
Example	The complete AUTOSAR stack is middleware. It serves an application as upper layer and uses the MCAL (Microcontroller Abstraction Layer) software as lower layer (physical layer representation).
Reference	Refer to the description of the CP architecture in [23].

4.201 Minimum Send Interval

Definition	The Minimum Send Interval specifies the minimum amount of time that shall pass between two consecutive transmissions of an `Event`, field, trigger or method call, i.e., the Minimum Send Interval limits the maximum Tx frequency of `Service Interface` elements in the `Network`. The Minimum Send Interval can be configured individually for each event, field and method of a service interface, through the associated value of ServiceInterfaceDeploymentElement.minimumSendInterval, referenced by the ServiceEventDeployment, ServiceFieldDeployment and ServiceMethodDeployment.
Initiator	Communication
Further Explanations	For each field, event and method, ara::com shall monitor the frequency of transmission requests by the application. If a transmission is requested faster than allowed and specified by the configured ServiceInterfaceDeploymentElement.minimumSendInterval, ara::com shall discard the request and return with an error code. Minimum Send Interval monitoring shall be disabled for a given event, field or method, if the configured value ServiceInterfaceDeploymentElement.minimumSendInterval = 0. This shall also be the default value, to avoid unintended Tx limitation, unless the user explicitly configured a suitable value for its application.
Comment	–
Example	If a field has a configured Minimum Send Interval of 0.010 (10 ms), and the application Updates() the field every 50 ms, ara::com will accept the update and hand over the updated field to the lower transmission layers and will never discard a transmission request. However, if the application tries to update every 5 ms, ara::com will discard every second update and return with an error, because the configured Minimum Send Interval is higher than the time between request.
Reference	ara::com::ComErrc::kminimumSendIntervalViolationError ([SWS_CM_10432] of [43]) ServiceInterfaceDeploymentElement.minimumSendInterval in TPS Manifest ([44])

4.202 Mistake

Definition	Human `Error`
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	[45]

4.203 Mode

Definition	A Mode is a certain set of states of the various state machines that are running in the vehicle that are relevant to a particular entity, e.g. a SWC, a BSW module, an application or a whole vehicle. In its lifetime, an entity changes between a set of mutually exclusive Modes. These changes are triggered by environmental data, e.g. signal reception, operation invocation.
Initiator	Runtime Environment
Further Explanations	–
Comment	–
Example	–
Reference	–

4.204 Multimedia Stream

Definition	A consistent sequence of digital data versus time which is suited as input for devices which transfer these data into a continuous visible or audible impression to humans. When transferred over a physical link, multimedia stream data typically are produced at the same rate (by the data source), as they are consumed (by the data sinks).
Initiator	General
Further Explanations	A multimedia stream usually follows a certain standard (e.g. MPEG-x). When transferred over a physical link, a multimedia stream needs a certain minimum bandwidth (in terms of bits/second) in order to allow continuous impressions. A multimedia stream in a car typically exists for several seconds (a warning signal, a navigation hint) up to several hours (a video film, a phone call, playing a radio program). Resources (e.g. bus system channels) needed by the stream have to be allocated continuously over this lifetime (this is a difference to e.g. file transfer, which may be split into several chunks of data). The source of a multimedia stream typically is a specialized device and/or software program (a tuner, a microphone, a text-to-speech engine, etc.). The same holds for the sinks (an audio amplifier or mixer, a voice recognition software, an MPEG decoder, etc.).
Comment	The term "visible or audible impression to humans" should not be taken too literally, because streams can also be used to transfer machine readable data (e.g. modem, encrypted signals). But it is this condition, which defines the standards and technology used in multimedia streams.
Example	Audio stream as output of or input to a telephone (mono, low bandwidth) Audio stream as output of a radio tuner (stereo, high bandwidth) Video stream as output of a television tuner An example for the physical implementation on a multimedia bus is the Firewire isochronous stream. see reference
Reference	[46]

4.205 Multiplexed PDU

Definition	A multiplexed PDU is a PDU with a configurable number of different payload layouts.
Initiator	Communication
Further Explanations	Each instance of a multiplexed PDU has a distinct layout. The set of possible layouts is statically defined. A selector signal defines which layout is used in a PDU instance. The selector signal must reside at the same position in all layouts. Each layout is identified by a unique selector value. The length of each instance of a multiplexed PDU is fixed.
Comment	–
Example	–
Reference	–

4.206 Network

Definition	Communication infrastructure between AUTOSAR ECUs/Machines.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.207 Network Interface

Definition	A Network Interface is the sum of all interfaces offered by the `Basic Software` towards its connected `Network`.
Initiator	Communication
Further Explanations	The interface that the Basic Software shares via the communication lines with other systems that behave like AUTOSAR ECUs in order to - allow distributed Software Components) to exchange inter-ECU signals and to - operate the communication lines (the `Network`) is called Network Interface. A Network Interface (NWI) denotes the interface between the `Basic Software` and the physical network (OSI Layer 0) to which the ECU executing the Basic Software is connected to (e.g. CAN, LIN, FlexRay). The NWI therefore transports network data packets between the `Basic Software` and the physical network. The interfaces included within the term NWI are: - Logical interfaces, including • Network Management • Data Management • Data transmission/reception The interfaces excluded from the term NWI are: - The physical network interface (CAN, FlexRay etc). Note that, attention must be given to the physical form of the network, since it is not formally specified by AUTOSAR. The NWI provided by a given ECU supports the transfer of data to and from the ECU, and management of the `Network`. For the purposes of this definition, the `Basic Software` can be designed according to ICC1, ICC2 or ICC3.
Comment	The term has been introduced as a short-hand to aid in discussion of the conformance of the content of ICC1 / 2 and to define the backward compatibility between releases and revisions. However, since from the `Network` perspective, the clustering of the `Basic Software` is invisible, the `Network` Interface is applicable to all potential `Basic Software` conformance classes (ICC1, ICC2, ICC3) in the same way.
Example	–
Reference	`Software Component Interface`

4.208 NM Coordination Cluster

Definition	A discrete set of NM Channels on which shutdown is coordinated.
Initiator	Communication
Further Explanations	The NM Coordinator will keep all presently awake NM Channels of an NM Coordination Cluster awake until it is possible to `Coordinate Network` sleep on all the awake channels.
Comment	–
Example	–
Reference	[47]

4.209 NM Coordinator

Definition	A `Functionality` of the Generic NM Interface which allows coordination of `Network` sleep for multiple NM Channels.
Initiator	Communication
Further Explanations	Depending on `Configuration`, different level of synchronous `Network` sleep can be achieved. The NM Coordinator is using a generic coordination algorithm which, by means of individually configured `Timeout` and `Synchronization` indications can coordinate a synchronized shutdown of multiple NM Channels.
Comment	–
Example	–
Reference	[47]

4.210 Non-repudiation

Definition	Concept that is used in information `Security` that assures that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	NIST SP 800-18 Rev. 1 under Non-repudiation from CNSSI 4009

4.211 Notification

Definition	Informing a software entity about a state change of a hardware and/or software entity which has occurred.
Initiator	Software and `Architecture`
Further Explanations	The informing about a state change can be done by an activation of a software part or by setting a `Flag`.
Comment	–
Example	–
Reference	–

4.212 Onboard Security Event

Definition	Analogous to ISO/IEC 27000:2018 ([48]) an onboard security event (SEv) is the identified occurrence of an onboard system, `Service` or `Network` state indicating a possible breach of information `Security` policy or `Failure` of controls, or a previously unknown situation that can be security relevant.
Initiator	Security
Further Explanations	–
Comment	–
Example	The failed `Verification` of a single secured PDU is typically an onboard security event.
Reference	

4.213 OS Application

Definition	A block of software including tasks, interrupts, hooks and user services that form a cohesive `Functional Unit`.
Initiator	Software and `Architecture`
x ***Further Explanations***	Trusted: An OS-Application that may be executed in privileged `Mode` and may have unrestricted access to the API and hardware resources. Only trusted applications can provide trusted functions. Non-trusted: An OS-Application that is executed in non-privileged mode has restricted access to the API and hardware resources.
Comment	The trusted / non-trusted attribute of an OS-Application is not related to `Automotive Safety Integrity Levels` (ASIL)/non-ASIL.
Example	–
Reference	[20]

4.214 OS Event

Definition	The event mechanism • is a means of `Synchronization` • is only provided for extended tasks • initiates state transitions of tasks to and from the waiting state.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	[2] (OSEK/VDX Operating System) [2] (OSEK/VDX Operating System)

4.215 Partitioning

Definition	See ISO-26262 ([5]), ID 1.85
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.85

4.216 Protocol Control Information

Definition	Information which is needed to pass a `Service Data Unit` from one instance of a specific protocol layer to another instance. E.g. it contains source and target information.
Initiator	Communication
Further Explanations	The PCI is added by a protocol layer on the transmission side and is removed again on the receiving side.
Comment	–
Example	–
Reference	–

4.217 Protocol Data Unit (PDU)

Definition	The Protocol Data Unit (PDU) contains `Service Data Unit` and `Protocol Control Information`.
Initiator	Communication
Further Explanations	On the transmission side the PDU is passed from the upper layer to the lower layer, which interprets this PDU as its SDU (as shown in Figure 4.6).
Comment	–
Example	[41] (OSEK/VDX Communication)
Reference	[41] (OSEK/VDX Communication)

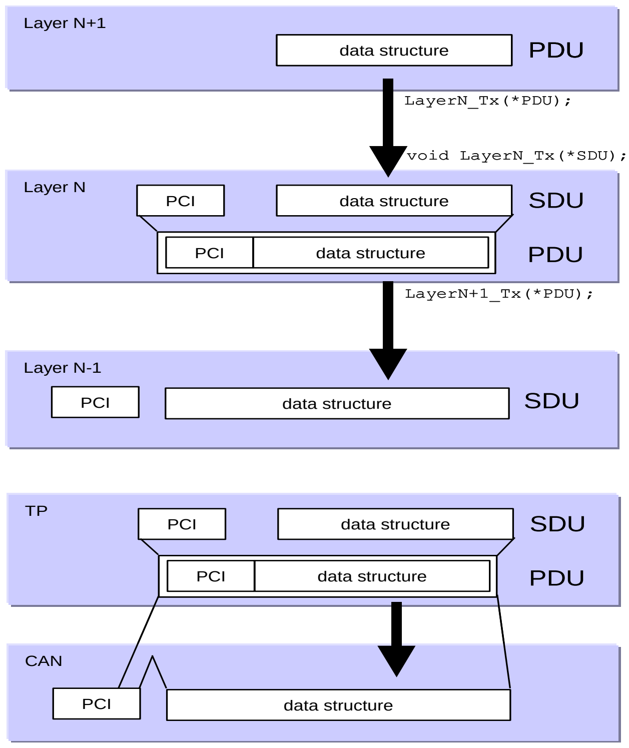


Figure 4.6: Explanation of Protocol Data Unit

4.218 PDU Meta-Data

Definition	Additional data of a `Protocol Data Unit`, which is not part of the payload.
Initiator	Communication
Further Explanations	Meta-data is placed alongside the PDU payload in a separate buffer. The layout of the Meta-data is determined by an ordered list of `MetaDataItems`.
Comment	Meta-data was introduced to transport parts of the CAN ID or addressing information alongside the data of a PDU.
Example	Diagnostics according to ISO 15765/14229, J1939 parameter group handling.
Reference	–

4.219 PDU Timeout

Definition	Maximum time between the receptions of two instances of one PDU is exceeded.
Initiator	Communication
Further Explanations	This `Timeout` indicates that the last reception of a PDU instance is too long in the past. As a consequence it can be concluded that the data in the last PDU instance is outdated.
Comment	–

▽

△

Example	–
Reference	–

4.220 Performance

Definition	Performance is a set of measurable characteristics (e.g. time, memory, resources usage, power consumption, etc.) which may be used to compare different system, SW element, algorithm, etc. implementations.
Initiator	Safety
Further Explanations	`Scalability` refers to the characteristic of a system to increase performance by adding additional resources. If the software performance requirements change (e.g more functions that impact the `Response Time`), scalability comes into play. Scalability is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases.
Comment	–
Example	–
Reference	–

4.221 Peripheral Hardware

Definition	Hardware devices integrated in micro-controller architecture to interact with the environment.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Memory, CAN-Controller, ADC, DIO, etc.
Reference	–

4.222 Personalization

Definition	User-specific and memorized adjustment of SW data or selection of functional alternatives.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Seat parameters (position, activation status of drive-dynamic seat) can be stored in correlation to a user ID. For a given user ID the seat can be adjusted according to the stored position parameters and the drive-dynamic seat can be activated or deactivated.
Reference	–

4.223 Plausibility

Definition	Runtime Plausibility check is a method to verify during runtime if inputs for a computation/ algorithm or results of a computation/algorithm are reasonable against corresponding values of a simplified reference model.
Initiator	Safety
Further Explanations	Range checks are a subset of plausibility checks. The additional knowledge can be taken from various sources, e.g. the physical domain of the value or from a model representing the computation/algorithm more roughly and calculating in parallel to the actual computation/ algorithm.
Comment	–
Example	• Range Check: for determination that a value for a car velocity is plausible, the knowledge that a normal vehicle cannot be faster than 400km/h. • Plausibility: for determination that that a value for a car velocity is plausible, the history of the values can be used and the knowledge that a certain acceleration for a car cannot be exceeded. E.g. velocity was 10 km/h and increases within 1 Sec to 100 km/h.
Reference	–

4.224 PNC Leaf Node

Definition	A PNC leaf node is located beneath a `Top-level PNC Coordinator` or `Intermediate PNC Coordinator`. A PNC leaf node represents the lowest level within the hierarchy of a PN topology. A PNC leaf node does not coordinate PNC request across its local communication channels. It receives and transmits Nm frames with PN information, but it does not forward received PNC requests to other local communication channels. A PNC leaf node processes PN shutdown messages as usual Nm frames with PN information.
Initiator	Partial Networking
Further Explanations	–
Comment	A PNC leaf node represents the lowest level within a PN topology.
Example	–
Reference	–

4.225 PN shutdown message

Definition	A `Top-level PNC Coordinator` transmits the PN shutdown messages to indicate a PNC shutdown. A PN shutdown message is a NM message where the PNSR bit (resides in the control bit vector) and all PNC bits (resides in the PN info) which are indicated for a synchronized shutdown set to “1”. A receiving `Intermediate PNC Coordinator` has to process the PN shutdown message immediately, reset the PN reset timer and forward the PN information as fast as possible to ensure a synchronized shutdown of the affected PNCs at nearly the same point in time. A `PNC Leaf Node` acts independently of the PNSR bit and resets the PN reset timer.
Initiator	Partial networking
Further Explanations	–
Comment	A PN shutdown message always indicates a shutdown of the indicated PNCs.
Example	–
Reference	–

4.226 Policy Decision Point (PDP)

Definition	The PDP represents the logic in which the access control decision is made. It determines if the application is allowed to perform the requested Task.
Initiator	Security
Further Explanations	The PDP provides an Access Control Decision.
Comment	–
Example	–
Reference	–

4.227 Policy Enforcement Point (PEP)

Definition	The PEP represents the logic in which the Access Control Decision is enforced. It communicates directly with the corresponding Policy Decision Point to receive the Access Control Decision.
Initiator	Security
Further Explanations	–
Comment	–
Example	–
Reference	–

4.228 Port

Definition	A port belongs to a Software Component and is the interaction point between the component and other components. The interaction between specific ports of specific components is modeled using Connectors. A port can either be a Provide Port or an Require Port.
Initiator	Software and Architecture
Further Explanations	–
Comment	–
Example	–
Reference	For more information see [9]

4.229 Port Interface

Definition	A Port Interface characterizes the information provided or required by a Port of a Software Component.
Initiator	Software and Architecture
Further Explanations	A Port Interface is either a Client-Server Interface in case Client-Server Communication is chosen or a Sender-Receiver Interface in case Sender-Receiver Communication is used.
Comment	–

▽

△

Example	–
Reference	For more information see [9]

4.230 Post-build Time Configuration

Definition	The `Configuration` of the SW module is possible after building the SW module.
Initiator	Methodology and Templates
Further Explanations	The SW may receive its `Configuration` file that can be downloaded to the `Electronic Control Unit` (ECU) separately, avoiding a re-compilation and re-build of the ECU SW modules. In order to make the post-build time re-configuration possible, the re-configurable elements shall be stored at a known position in the ECU storage area
Comment	–
Example	Identifiers of the CAN frames
Reference	–

4.231 Post-build Hooking

Definition	The insertion of Hooks to facilitate `Rapid Prototyping` support into a (complete) `Electronic Control Unit` (ECU) hex image.
Initiator	Runtime Environment
Further Explanations	–
Comment	–
Example	Detection of reads and/or writes of ECU variables by analysis of the instruction stream.
Reference	–

4.232 Pre-build Hooking

Definition	The insertion of Hooks to facilitate Rapid Prototyping support into software source prior to creating an `Electronic Control Unit` (ECU) hex image.
Initiator	Runtime Environment
Further Explanations	–
Comment	–
Example	–
Reference	–

4.233 Pre-Compile Time Configuration

Definition	The Configuration of the SW module is done at source code level and will be effective after compile time.
Initiator	Methodology and Templates
Further Explanations	The source code contains all the Electronic Control Unit (ECU) Configuration data and when compiled together, it produces the given SW.
Comment	–
Example	Preprocessor switch for enabling the development Error detection and reporting
Reference	–

4.234 Predictabiliy

Definition	Predictability is the degree to which a correct prediction or forecast of a system's state / behavior can be made either qualitatively or quantitatively.
Initiator	Safety
Further Explanations	Important type of predictability occurs in the design of systems that are subject to real-time requirements. A good overview of predictability criteria and how to achieve them can be found in
Comment	–
Example	–
Reference	John A. Stankovic, Krithi Ramamritham, What is predictability for real-time systems?, Journal of Real-Time Systems, Volume 2 Issue 4, Nov. 1990, 247-254

4.235 Pretended Networking

Definition	Method to reduce energy consumption in an existing active Network without changing network infrastructure.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.236 Private Interface

Definition	A private interface is an interface within the Basic Software of AUTOSAR which is neither standardized nor defined within AUTOSAR.
Initiator	Software and Architecture

▽

△

Further Explanations	The goal of the private interface is to enable a more efficient implementation of `Basic Software Modules`. Basic software modules sharing a private interface have to be distributed as one package. This package has to behave exactly the same as separate modules would. It must provide the same `Standardized Interfaces` to the rest of the `Basic Software` and/ or RTE as separate modules would. It has to be configured exactly the same as separate modules would be configured.
Comment	Private interfaces contradict the goal of exchangeability of `Standard Software` modules and should be avoided.
Example	–
Reference	–

4.237 Probability of Failure

Definition	Probability of the occurrence of a `Failure` in a system or `Functional Unit`.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.238 Procedure Call

Definition	A simple statement that provides the actual parameters for and invokes the execution of a procedure (software `Function`).
Initiator	Software and `Architecture`
Further Explanations	A `Synchronous Communication` mechanism can be implemented by a procedure call.
Comment	–
Example	–
Reference	[49]

4.239 Process

Definition	An `Executable` unit managed by an operating system `Scheduler` that has its own name space and resources (including memory) protected against use from other processes.
Initiator	Software and `Architecture`
Further Explanations	A process consists of n Task (n>=1)
Comment	–
Example	–
Reference	–

4.240 Processed Manifest

Definition	A Processed Manifest is a Manifest that is stored in the implementation specific format on the AUTOSAR Adaptive Platform product. Usually this is done in combination with other artifacts (like binary files) that contain Executable code to which the Manifest applies.
Initiator	Execution Management
Further Explanations	Manifests are often used to denote a piece of Configuration content that ships along a given piece of software and is used to deploy the software in the field. There are several kinds of manifest, this list includes but is not limited to: • Execution Manifest • Machine Manifest • Service Instance Manifest
Comment	The Manifest may contain platform implementation dependent data, as well as generic data derived from Application System Description.
Example	–
Reference	–

4.241 Profiling

Definition	Profiling refers to the process of evaluating Performance.
Initiator	Software and Architecture
Further Explanations	Profiling aggregates descriptive statistics of Performance measures of items running in an AUTOSAR system based on recordings, e.g. measurements or traces. The evaluation can be done online, i.e. during runtime, or offline.
Comment	Items subject to profiling could be functions, tasks, runnables, modules, buses etc.
Example	Statistics (min/max/average), worst case analysis
Reference	–

4.242 Proven In Use Argument

Definition	See ISO-26262 ([5]), ID 1.90
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.90

4.243 Provide Port

Definition	Specific Port providing Data or providing a Service of a Server.
Initiator	Software and Architecture
Further Explanations	The Provide Port is sometimes abbreviated as PPort or P-Port.
Comment	–
Example	• Server Port • Server Port
Reference	–

4.244 Rapid Prototyping

Definition	The experimental incorporation of new Functionality.
Initiator	Runtime Environment
Further Explanations	Rapid Prototyping (RP) permits a user to quickly perform experiments to add new Functionality, or to replace/bypass existing functionality, without requiring an ECU image to be built.
Comment	–
Example	–
Reference	–

4.245 Rapid Prototyping Memory Interface

Definition	The memory access pattern necessary for Rapid Prototyping Tool.
Initiator	Runtime Environment
Further Explanations	The RP memory interface provides the well-defined memory access pattern required by RP tool to ensure consistent and complete access to bypass values.
Comment	–
Example	A mandated "write-read" cycle within RTE APIs provides the RP tool with an opportunity to bypass (i.e. substitute with value generated from an alternative algorithm) the written value before it is read and then subsequently used within the generated code.
Reference	–

4.246 Rapid Prototyping Tool

Definition	Software and/ or hardware tools to support Rapid Prototyping.
Initiator	Runtime Environment
Further Explanations	–
Comment	–
Example	Dedicated prototyping interfaces on ECUs accessed by PC-based software tools.

▽

△

Reference	–

4.247 Rate Conversion

Definition	Operation to change the timing between two transmissions of the same PduId on one physical `Network`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.248 Raw Data Stream

Definition	A Raw Data Stream is a collection of data that is unprocessed, basically a series of bytes without any information on how to interpret it.
Initiator	Communication
Further Explanations	Raw Data Streams are used in communication links, mainly over ethernet reading streaming data from sensors. An API for Raw Data Streams is defined for AUTOSAR `Adaptive Applications` in SWS Communication Management.
Comment	–
Example	–
Reference	–

4.249 Recovery

Definition	Returning to intended `Functionality` after `Fault Detection` without violating the `Safety` goals.
Initiator	Safety

▽

△

Further Explanations	• Restart Mode: Restart Operation from the initial state of operation • Continue Mode: Restart Operation from the last known state of operation • Recovery by repetition: repeat until Timeout to cope i.e. random transmission errors. • Forward Error recovery: relies on continue from an erroneous state by making selective corrections to the system state. This includes making the controlled environment safe, which may be damaged because of the Failure • Backward Error recovery: relies on restoring the system to a previous Safe State and executing an alternative section of the program. This has the same Functionality but uses a different algorithm (c.f. N-Version Programming) • Recovery Point: The point to which a Process is restored is called a recovery point and the act of establishing it is termed check-pointing (saving appropriate system state) • Recovery testing is the forced Failure of the software in a variety of ways to verify that recovery is properly performed.
Comment	–
Example	–
Reference	–

4.250 Redundancy

Definition	Existence of means in addition to the means that would be sufficient for an element to perform a required Function or to represent information. Hardware Element redundancy includes replicated or additional hardware means added to the system to support Fault Tolerance. Software element redundancy includes the additional SW units and/or data used to support fault tolerance.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.94

4.251 Reentrancy

Definition	In AUTOSAR a Function is called reentrant if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution. AUTOSAR differs between • (full) reentrancy • non reentrancy and • conditional reentrancy.
Initiator	Software and Architecture
Further Explanations	Reentrancy is always considered from the viewpoint of the caller. A Function which is conditional reentrant has to document the conditions for the reentrancy. Typical cases for conditional reentrancy are functions which are reentrant as long as a function parameter is different to (possible) ongoing calls.

▽

△

Comment	An implementation of a Function might be reentrant for one system but only conditional reentrant (or even non reentrant) for another one. It always depend how the reentrancy was realized (e.g. locks). As an example, just consider a function which uses Interrupt locks to realize full reentrancy on a single core System. If this implementation is used in a multi core system its reentrancy will only be conditional reentrant for calls from the same core.
Example	–
Reference	–

4.252 Reliability

Definition	Probability of a System or Functional Unit to perform as expected under specified conditions within a time interval.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.253 Relocatability

Definition	Capability of a software part being executed on different hardware environments without changing the code of the software part.
Initiator	Software and Architecture
Further Explanations	–
Comment	–
Example	–
Reference	–

4.254 Require Port

Definition	Specific Port requiring Data or requiring a Service of a Server.
Initiator	Software and Architecture
Further Explanations	The Require Port is sometimes abbreviated as RPort or R-Port.
Comment	–
Example	• Client Port • Receiver Port
Reference	–

4.255 Required Property

Definition	A required property or quality of a design entity (e.g. `Software Component` or `System`) is a property or quality which has to be fulfilled by the environment of this design entity.
Initiator	Software and `Architecture`
Further Explanations	A property or quality can be required by a stakeholder (e.g. customer) or another design entity.
Comment	–
Example	1) In order to meet its `Functionality`, a `Software Component` A requires a minimum temporal resolution of a signal (information on a required `Port`) which has to be fulfilled by SW component B. 2) SW component requires to be activated by the runtime environment every 100ms with a jitter of 10ms.
Reference	Compare term `Asserted Property`

4.256 Residual Error Rate

Definition	The ratio of the number of bits, unit elements, or blocks incorrectly received and undetected, to the total number of bits, unit elements, characters, or blocks sent.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.257 Resource

Definition	A resource is a required but limited hardware entity of an `Electronic Control Unit`, which in general can be accessed concurrently, but not simultaneously, by multiple software entities.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	The definition from [2] (OSEK/VDX Operating System) cannot be used, due to the specific usage in ISO 17356.
Example	CPU-load, interrupts (mechanism itself and the resulting CPU-load), memory, peripheral hardware, communication, ...
Reference	–

4.258 Resource-Management

Definition	Entity which controls the use of Resources.
Initiator	Software and Architecture
Further Explanations	The main Functionality of resource management is the control of simultaneous use of a single Resource by several entities, e.g. scheduling of requests, multiple access protection.
Comment	–
Example	OS-Scheduler (CPU-load management)
Reference	–

4.259 Response Time

Definition	Time between receiving a stimulus and delivering an appropriate response or reaction.
Initiator	Software and Architecture
Further Explanations	The response time describes the time between a stimulus like e.g. the state change of hardware or software entity and the expected reaction of the system (e.g. response, actuator activation). Synonym: reaction time See also: Execution Time, Worst Case Execution Time and Worst Case Response Time.
Comment	–
Example	–
Reference	–

4.260 Risk

Definition	See ISO-26262 ([5]), ID 1.99
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.99

4.261 Robustness

Definition	Ability of a System or Functional Unit to perform as expected also under unexpected conditions.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	–

4.262 RTE Event

Definition	An RTE Event encompasses all possible situations that can trigger execution of a `Runnable Entity` by the RTE. Thus they can address timing, data sending and receiving, invoking operations, call `Server` returning, `Mode` switching, or external events. RTE Events can either activate a runnable entity or wake-up a runnable entity at its waitpoints.
Initiator	Runtime Environment
Further Explanations	Note: '`Event`' in this context is not necessarily synonymous with 'RTEEvent' as defined in the `Virtual Functional Bus` (VFB) specification. In particular, RTE Events that result from communication are handled by communication-triggered runnable entities.
Comment	Events can have a variety of sources including time.
Example	Scheduling of runnable entities from angular position, e.g. a crankshaft, that are used to trigger an `Interrupt` and hence an RTE `Notification`. A software component needs to perform a regular interval, e.g. flash an LED, reset a watchdog, etc.
Reference	–

4.263 Runnable Entity

Definition	A Runnable Entity is a part of an `Atomic Software Component` which can be executed and scheduled independently from the other Runnable Entities of this Atomic Software-Component. It is described by a sequence of instructions that can be started by the RTE. Each runnable entity is associated with exactly one (1) `Entry Point`.
Initiator	Software and `Architecture`
Further Explanations	A Runnable Entity contains at least two points for the `Scheduler`: 1 `Entry Point` and 1 `Exit Point`. Due to the reason that an `Atomic Software Component` is not dividable, all its Runnable Entities are executed on the same `Electronic Control Unit` (ECU).
Comment	In general a `Task` in the runtime system consists out of n Runnable Entities of m Atomic Software-Components.
Example	`Server Function` of a `Software Component`.
Reference	–

4.264 SAE J1939

Definition	SAE J1939 is a vehicle bus standard created by the SAE (Society of Automotive Engineers, a USA standards body) for car and heavy duty truck industries.
Initiator	Communication

▽

△

Further Explanations	The J1939 standard encompasses the following areas: - bus physics (J1939/11, J1939/15) - CAN message layout (J1939/21) - request/response and multi packet transport protocols (J1939/21) - Network management used to assign a unique address to each node (J1939/81) - diagnostics layer comparable to UDS in complexity (J1939/73) - standardized application signals and messages (J1939/71)
Comment	The J1939 standard is used by most truck manufacturers worldwide and is prescribed for OBD in some states of the USA. It is also used as a base for other standards for maritime (NMEA 2000), agricultural (ISO 11783), and military (MilCAN A) applications.
Example	–
Reference	http://www.sae.org/

4.265 Safe State

Definition	See ISO-26262 ([5]), ID 1.102
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.102

4.266 Safety

Definition	See ISO-26262 ([5]), ID 1.103
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.103

4.267 Safety Analysis

Definition	The objective of Safety Analysis is to examine the consequences of faults and failures on items and elements considering their functions, behaviour and design.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	ISO 26262 - 9: Automotive Safety Integrity Levels (ASIL)-oriented and safety-oriented analyses (see also: Safety Analysis)

4.268 Safety Case

Definition	See ISO-26262 ([5]), ID 1.106
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.106

4.269 Safety Goal

Definition	See ISO-26262 ([5]), ID 1.108
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See I[5], ID 1.108

4.270 Safety Measure

	See ISO-26262 ([5]), ID 1.110
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.110

4.271 Safety Mechanism

Definition	See ISO-26262 ([5]), ID 1.111
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.111

4.272 Safety Protocol

Definition	A communication protocol defining the necessary mechanisms to ensure the integrity of transmitted data and to detect any communication related `Error`.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	–

4.273 Sample Application

Definition	Defined system used for evaluation purposes.
Initiator	Software and `Architecture`
Further Explanations	The application may be simplified for better understanding within the evaluation phase.
Comment	–
Example	Diagnosis Application Exterior Light Management
Reference	–

4.274 Scalability

Definition	The degree to which assets can be adapted to specific target environments for various defined measures.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	Target environment introduced compared to [3].
Example	–
Reference	[3]

4.275 Scheduler

Definition	The scheduler handles the scheduling of the Tasks / Runnable Entitys according to the priority and scheduling policy (pre-defined or configurable). It has the responsibility to decide during run-time when which task can run on on the CPU of the Electronic Control Unit (ECU).
Initiator	Software and Architecture
Further Explanations	There are many strategies (priority-based, time-triggered, round-robin, ...) a scheduler can use, depending of the selected and/or implemented algorithms
Comment	–
Example	–
Reference	See AUTOSAR Specification of the Virtual Functional Bus ([9]

4.276 Service Data Unit

Definition	Service Data Unit is the data passed by an upper layer, with the request to transmit the data. It is as well the data, which is extracted after reception by the lower layer and passed to the upper layer.
Initiator	Communication
Further Explanations	A SDU is part of a Protocol Data Unit.
Comment	–
Example	–
Reference	–

4.277 Security

Definition	Protection of data, software entities or resources from accidental or malicious acts.
Initiator	Software and Architecture
Further Explanations	–
Comment	Slightly adapted norm.
Example	–
Reference	[50]

4.278 Secure Channel

Definition	A secure channel is a communication channel between two parties. A secure channel shall at least provide Integrity and Authenticity of the communication. It shall provide means to authenticate at least one participant of the communication. It may provide means to mutually authenticate both participants of the communication. A secure channel may additionally provide the Confidentiality of the communication.
Initiator	Security

▽

△

Further Explanations	–
Comment	–
Example	A TLS or IPsec channel for secure SOME/IP communication between two Adaptive Platform (AP) instances.
Reference	–

4.279 Security Event

Definition	An `Event` which is related to the `Security` of the `Electronic Control Unit` (ECU) and should be reported for later analysis.
Initiator	Security
Further Explanations	–
Comment	–
Example	Failed negotiation of a shared secret, failed `Verification` of a download signature, successful update of a certificate
Reference	–

4.280 Sender-Receiver Communication

Definition	A communication pattern which offers asyncronous distribution of information where a sender communicates information to one or more receivers, or a receiver receives information from one or several senders.
Initiator	Software and `Architecture`
Further Explanations	The process of sending data does not block the sender and the sender usually gets no response from the receivers
Comment	Often used for data or `Event` distribution
Example	–
Reference	[9]

4.281 Sender-Receiver Interface

Definition	A sender-receiver interface is a special kind of a `Port Interface` for the case of `Sender-Receiver Communication`. The sender-receiver interface defines the data-elements which are sent by a sending component (which has a p-`Port` providing the sender-receiver interface) or received by a receiving component (which has an r-port requiring the sender-receiver interface).
Initiator	Software and `Architecture`
Further Explanations	–
Comment	A special kind of `Port Interface`
Example	–

▽

△

Reference	[9]

4.282 Sensor-Actuator SW-Component

Definition	`Software Component` dedicated to the control of a sensor or actuator.
Initiator	Software and `Architecture`
Further Explanations	There will be several Sensor/ Actuator SW-Cs in each ECU. In general there will be one Sensor/ Actuator SW-C for each sensor and one for each actuator (=> number of Sensor/Actuator SW-C = number of sensors + number of actuators).
Comment	–
Example	–
Reference	–

4.283 Server

Definition	Software entity which provides services for Clients
Initiator	Software and `Architecture`
Further Explanations	The Server and the clients using its `Service` might be located on one ECU or distributed on different calculation units (e.g. ECU).
Comment	Adapted from [14].
Example	–
Reference	[14]

4.284 Service

Definition	A service is a type of operation that has a published specification of interface and behavior, involving a contract between the provider of the capability and the potential clients.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Diagnosis service, ...
Reference	[3]

4.285 Service Discovery

Definition	Generic `Functionality` provided by the Communication Management to Applications that allows Applications at runtime to find locally or remotely available `Service Instances` providing the requested `Service`.
Initiator	Communication
Further Explanations	Based on Application query, the Communication Management provides list of compatible `Service Instances`. Compatibility is defined by compatibility rules and may consider version or QoS attributes.
Comment	–
Example	–
Reference	–

4.286 Service Instance

Definition	The properties of a service instance are described by a specific `Service Interface`. A service instance has a unique identity.
Initiator	Communication
Further Explanations	It is accessible by other Applications by using a Service Requester Proxy at runtime and is typed by a specific `Service Interface` It is addressable within the vehicle `Network` by its Service Instance ID, an abstraction from of the physical location. Optionally, authentication data is associated with a Service Instance for authentication at runtime.
Comment	–
Example	–
Reference	–

4.287 Service Interface

Definition	A service interface is a special kind of `Port Interface` used in the Adaptive platform. It defines both data elements for `Event`-based communication and operations that are provided by the service provider and that can be used by the service requester.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.288 Service-Oriented Communication

Definition	Communication, for which communication partners are generally not defined during design time, but dynamically discovered and bound during runtime.
Initiator	Communication

▽

△

Further Explanations	Communication partners are generally not defined during design time, but dynamically discovered and bound during runtime. Adaptive Applications are therefore developed agnostic to the concrete context, assuming model of loosely-coupled components.
Comment	This is the standard communication paradigm for communication between AUTOSAR Adaptive Applications.
Example	–
Reference	–

4.289 Service Port

Definition	Special kind of a Port from a Atomic Software Component used to describe Service communication.
Initiator	Software and Architecture
Further Explanations	The interface of a Service Port has to be a Standardized AUTOSAR Interface. A Service Port does not need to be connected to another Port in the Virtual Functional Bus View.
Comment	If a Service is provided by the ECU where a specific Atomic Software Component is located the VFB View is sufficient. If a service is provided by another ECU the connection of the service call to the service has to be done explicitly during the Mapping step.
Example	Write data to non volatile memory.
Reference	–

4.290 Service Proxy

Definition	A facade that represents a specific Service on code level from the perspective of the service consumer providing methods for all functionalities offered by the represented service.
Initiator	Communication
Further Explanations	The service consumer side Application code interacts with this local facade, which then knows how to propagate these calls to the real service implementation and back. The Service Proxy is typically an instance of a service proxy class which itself is potentially generated from Service Interface according to standardized patterns and implemented by platform-specific Communication Management. The Service Proxy provides placeholder for Service Instance ID, which is set at runtime by requesting Application implementation using Service Discovery or statically based on Planned Dynamics.
Comment	–
Example	–
Reference	–

4.291 Service Skeleton

Definition	A representation of a specific Service on code level from the perspective of the service implementation, which provides functionalities according to the service definition and allows to connect the service implementation to the Communication Management transport layer, so that the service implementation can be contacted by service consumers.

▽

△

Initiator	Communication
Further Explanations	The Service Skeleton is typically an instance of a service skeleton class which itself is potentially generated from a Service Interface according to standardized patterns and implemented by platform-specific Communication Management. The skeleton provides placeholder for the Service Instance ID, which is set by platform implementation, e.g. based on Application Description at design time, or by the Vehicle Software Configuration Manager at setup.
Comment	–
Example	–
Reference	–

4.292 Services Layer

Definition	The Services Layer is the highest layer of the Basic Software which also applies for its relevance for the application software: while access to I/O signals is covered by the Hardware Abstraction Layer, the Services Layer offers Operating system services Vehicle Network communication and management services Memory services (NVRAM management) Diagnosis Services (including KWP2000 interface and Error Memory) Electronic Control Unit (ECU) state management Task: Provide basic services for application and Basic Software Modules
Initiator	Software and Architecture
Further Explanations	The Services Layer consists of the following parts: Communication Services Memory Services System Services
Comment	–
Example	Network Management, NVRAM Manager, ECU State Manager
Reference	[23]

4.293 Signal Service Translation

Definition	Signal Service Translation is the standardized way to map the definition of signal-based communication to Service-Oriented Communication, and vice versa.
Initiator	Manifests & Templates
Further Explanations	AUTOSAR Adaptive Platform restricts communication paradigm to Service-Oriented Communication, a major part of the vehicle however still uses signal-based communication means - therefore a translation of these two approaches has to be performed.
Comment	One goal of AUTOSAR is to support the development of a whole vehicle in a seamless way. The translation of communication paradigms is essential for a holistic design. The AUTOSAR methodology supports the translation activity on either a Classic Platform Gateway ECU or on an AUTOSAR Adaptive Platform Machine.
Example	–
Reference	–

4.294 Slave Switch

Definition	A Slave Switch is an Ethernet switch which is connected to a `Master Switch` by uplink ports
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.295 Software Cluster (Adaptive Platform)

Definition	An Adaptive Platform Software Cluster groups all AUTOSAR artifacts which are relevant for deployment on an Adaptive Platform Machine. The content of a Software Cluster is dependent on the category and "may" (but is not limited to) contain artifacts such as: • Adaptive Platform Applications (Executables) • Adaptive Platform Functional Clusters (Modules) • Manifests • Persistent Storage Databases • Platform libraries, e.g. Transformers, Cryptographics • Bootloader
Initiator	Diagnostics
Further Explanations	A Software Cluster is deployed on a Machine via UCM inside a `Software Package` - multiple `Software Packages` are encapsulated inside a Vehicle Package to be deployed on multiple `Machines` in a vehicle. In the context of diagnostics a Software Cluster might be individually addressable via its own set of diagnostic addresses.
Comment	A Software Cluster is used to partition Applications running on a `Machine` into individually updateable clusters.
Example	–
Reference	[51], [44]

4.296 Software Cluster (Classic Platform)

Definition	The purpose of a Software Cluster for CP is to group AUTOSAR artifacts that are relevant for the static deployment of a software part on an ECU (`Electronic Control Unit`).
Initiator	WG-A-CP
Further Explanations	Each Software Cluster is an independent Build Unit and the result of the cluster specific build processes are the Binary Objects.
Comment	The term 'cluster' (or Software Cluster) is also used in the context of ICC2. In this case, it refers to a group of `Basic Software Module`s.
Example	–
Reference	see [52]

4.297 Software Component

Definition	Software Components are architectural elements that provide and/or require interfaces and are connected to each other through the Virtual Functional Bus to fulfill architectural responsibilities.
Initiator	Software and Architecture
Further Explanations	A Software Component (SWC) has a formal description defined by the software component template. SWCs may be atomic components, parameter components or Compositions. Also the software modules providing the Software Component Interface of a Basic Software are called Software Components.
Comment	–
Example	–
Reference	–

4.298 Software Component Interface

Definition	A Software Component Interface (SW-CI) is the sum of all interfaces offered by the Basic Software towards the Software Component.
Initiator	Software and Architecture
Further Explanations	A SW-CI denotes the interface between an SWC and the underlying Basic Software cluster including the RTE. The SW-CI therefore comprises all Application Programming Interface Functions and Callbacks that the Software Component requires from and provides to the Basic Software (generally by means of RTE mechanisms). It includes also the mechanisms allowing Software Components sharing the SW-CI to communicate with one another. For the purposes of this definition, the Basic Software clustered on an Electronic Control Unit (ECU) can be designed according to ICC1, 2 and 3.
Comment	The term has been introduced as a short-hand to aid in discussion of the conformance of the content of Basic Software clusters of conformance class ICC1 / 2 and to define the backward compatibility between releases and revisions. However, since from the Software Component perspective, the clustering of the Basic Software is invisible, the Component Interface is applicable to all potential Basic Software conformance classes (ICC1, ICC2, ICC3) in the same way.
Example	–
Reference	Network Interface (NWI)

4.299 Software Configuration

Definition	The arrangement of software elements in a SW system.
Initiator	Software and Architecture
Further Explanations	A software element is a clearly definable software part. A software configuration is a selection version of software modules, Software Components, parameters and generator configurations. Calibration and Variant Coding can be regarded as subset of Software Configuration.
Comment	–
Example	–
Reference	[3]

4.300 Software Interrupt

Definition	Interrupt triggered by SW Event.
Initiator	Software and Architecture
Further Explanations	SW events are for example calling an operating system service, starting a Process with higher priority.
Comment	–
Example	–
Reference	Translation/Adaptation from [35]

4.301 Software Package

Definition	Unit for deployment of software onto AUTOSAR Adaptive Platform instances containing zero or more Executables and the Meta-data to install and execute it on the Machine.
Initiator	Software and Architecture
Further Explanations	Typically, a software package contains one or more Executables, however it is permitted to have no Executables to enable update of Configuration Meta-data.
Comment	–
Example	–
Reference	Translation/Adaptation from [35]

4.302 Software Platform

Definition	Software environment on which application software is executed.
Initiator	–
Further Explanations	AUTOSAR Adaptive Platform, AUTOSAR Classic Platform
Comment	–
Example	–
Reference	–

4.303 Software Signal

Definition	A Software Signal is an asynchronous Event transmitted between one Process and another.
Initiator	Software and Architecture
Further Explanations	A SW Signal is the software implementation of an (control-) information. Additionally it may have attributes (e.g. Freshness, data type, ...). It is exchanged between Software Components.
Comment	–
Example	–
Reference	–

4.304 Software Unit

Definition	See ISO-26262 ([5]), ID 1.125
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.125

4.305 Special Periphery Access

Definition	Special functions to standard peripheral devices or special peripherals.
Initiator	Software and `Architecture`
Further Explanations	Is only used when, because of technical issues, no standard periphery access can be used
Comment	–
Example	–
Reference	–

4.306 Standard Periphery Access

Definition	Standard functions to typical standard peripheral devices that are available on an ECU (most `Microcontroller` integrated) used in automotive embedded applications.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	Digital Input/Output, Analog/Digital Converter, Pulse Width (De)Modulator, EEPROM, FLASH, Capture Compare Unit, Watchdog Timer
Reference	–

4.307 Standard Software

Definition	Standard Software is software which provides schematic independent infrastructural functionalities on an ECU. It contains only `Standardized AUTOSAR Interfaces`, `Standardized Interfaces` and/or `Private Interfaces`.
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	ISO 17356, MCAL, Services
Reference	[33]

4.308 Standardized AUTOSAR Blueprint

Definition	A Standardized AUTOSAR Blueprint is a Blueprint standardized within the AUTOSAR project. Its derived objects are considered as being standardized within the AUTOSAR project as well.
Initiator	AUTOSAR Application Interfaces
Further Explanations	Blueprints were introduced within the AUTOSAR projects to enable standardization of ports without standardizing the static view of the architecture (i.e. the software components providing or requesting the ports). Sometimes it is not possible to standardize all attributes of an AUTOSAR element because the values of some attributes are project specific. Nevertheless it enables better collaboration if some of the attributes are standardized. Additionally Blueprints enable adding descriptions and long names in different languages.
Comment	–
Example	–
Reference	[53]

4.309 Standardized AUTOSAR Interface

Definition	This is an AUTOSAR Interface which is standardized within the AUTOSAR project.
Initiator	Software and Architecture
Further Explanations	AUTOSAR Services interact with other components through a Standardized AUTOSAR Interface. AUTOSAR Interfaces can be derived from AUTOSAR Application Interfaces.
Comment	–
Example	–
Reference	–

4.310 Standardized Blueprint

Definition	A Blueprint is called a Standardized Blueprint if the derived objects are considered as being standardized as well. It also includes that additionally concrete standardized rules exist how to specify the blueprint as well as how to derive an object from that blueprint. This is typically not done for a specific blueprint but for all blueprints of the same class.
Initiator	AUTOSAR Application Interfaces
Further Explanations	–
Comment	–
Example	–
Reference	–

4.311 Standardized Interface

Definition	A software interface is called Standardized Interface if a concrete standardized API exists.
Initiator	Software and Architecture
Further Explanations	Modules in the Basic Software interact which each other through Standardized Interfaces.
Comment	–
Example	ISO 17356-4 ([41]): COM Interface
Reference	–

4.312 Static Configuration

Definition	A setup where the routing Configuration cannot be changed during normal operation of the Gateway.
Initiator	Communication
Further Explanations	Static configuration doesn't allow reconfiguration of the routing during normal operation e.g. during driving. Static configuration does not restrict the update of the configuration in specific maintenance operation modes (e.g. programming Mode).
Comment	–
Example	A software update may change a routing Configuration such that a Protocol Data Unit (PDU) is routed into two instead of one destination Networks.
Reference	–

4.313 Synchronization Points

Definition	Used to synchronize the execution of EEs of different tasks that execute within the same Logical Execution Time (LET) interval. A synchronization point realizes the execution order of EEs that belong to different Executable Entity Clusters (EECs).
Initiator	Software and Architecture
Further Explanations	This pattern is used when EEs on multiple cores are synchronized, e.g. to ensure Execution Order Constraints as shown for example in Figure 4.7. Two kinds of Synchronization are in focus here. The execution of Executable Entity Cluster may be synchronized by actively waiting at a barrier or by advancing the execution until to relative point in time after LET interval start.
Comment	Synchronization points can be derived directly from the EOCs and the Mapping of EEs to tasks. The synchronization can be implemented by time (advance) or as a barrier. A synchronization point between EECs that are mapped to the same Task can already be fulfilled by the EE positions in the task. It is important to known that direct access before the synchronization point does not lead to interference, i.e. no other access to the same memory location can take place. The use of advance requires confident knowledge of the EE's Worst Case Execution Time (WCET).
Example	Execution Order Constraints between Executable Entities imposed by a data dependency. A Runnable Entity C waiting for input from Runnable Entity A and B. A synchronization point before C guarantees that A and B have produced the input for C when it starts execution.
Reference	–

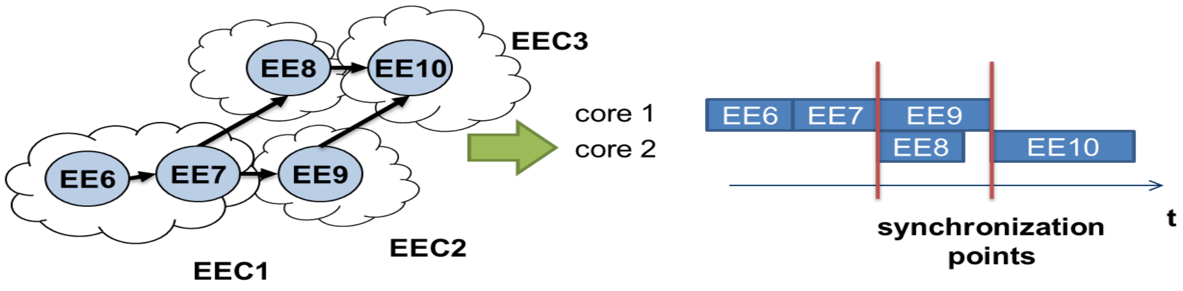


Figure 4.7: Example for Synchronization Points

4.314 Synchronization

Definition	To make two or more events or operations to occur at the same predefined moment in time.
Initiator	Communication
Further Explanations	–
Comment	–
Example	Two NM Channels can enter Bus Sleep `Mode` at the same time ("synchronized network sleep") or they can be ordered to go to sleep at the same time ("synchronized shutdown initiation").
Reference	[47]

4.315 Synchronous Communication

Definition	A communication is synchronous when the calling software entity is blocked until the called operation is evaluated. The calling software entity continues its operation by getting the result.
Initiator	Software and `Architecture`
Further Explanations	Synchronous communication between distributed functional units has to be implemented as Remote `Procedure Call` (RPC).
Comment	
Example	–
Reference	–

4.316 Synchronous Function

Definition	A `Function` is called synchronous if the described `Functionality` is guaranteed to be completed the moment the function returns to the caller.
Initiator	Software and `Architecture`

▽

△

Further Explanations	–
Comment	–
Example	–
Reference	–

4.317 System

Definition	Representation of software related parts in a vehicle and their means to communicate with each other.
Initiator	General
Further Explanations	–
Comment	–
Example	Representation of the Network topology of a vehicle with Software Components deployed on individual Electronic Control Units (ECUs) with a defined description of network communication between the ECUs.
Reference	[32], [44]

4.318 System Constraint

Definition	Boundary conditions that restrict the Design-Freedom of the (cars E/E-) System.
Initiator	Software and Architecture
Further Explanations	The design of Electronic Control Unit (ECU) Networks and the distribution of functionalities to ECUs are limited by several constrains. These constraints result mostly by the communication matrix and Safety requirements.
Comment	–
Example	An existing communication matrix that restricts the distribution of Electrical Signals to frames is a system constraint. Another system constraint is a Safety requirement that does not allow to map a specified Software Component to specific ECU.
Reference	–

4.319 System Health Monitor

Definition	System Health Monitor (SHM) analyzes the health of subsystems.
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	[36]

4.320 System Signal

Definition	The system signal represents the communication system's view of data exchanged between SW components which reside on different ECUs. The system signals allow to represent this communication in a flattened structure, with (at least) one system signal defined for eac`Data Element` sent or received by a `Software Component` instance. If data has to be sent over `Gateways`, there is still only one system signal representing this `Data`. The representation of the `Data` on the individual communication systems is done by the `Cluster Signals`.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.321 Systematic Fault

Definition	See ISO-26262 ([5]), ID 1.131
Initiator	Safety
Further Explanations	–
Comment	–
Example	–
Reference	See [5], ID 1.131

4.322 Task

Definition	A Task is the smallest schedulable unit managed by the OS. The OS decides when which task can run on the CPU of the ECU.
Initiator	Software and `Architecture`
Further Explanations	A `Runnable Entity` of a software component runs in the context of a task. Also the `Basic Software Modules` runs in the context of a task.
Comment	–
Example	–
Reference	[9]

4.323 Technical Signal

Definition	The technical signal is the physical value of an external `Event` coupled to an AUTOSAR system. Technical signals are represented in SI units (e.g. pressure in PA).
Initiator	General

▽

△

Further Explanations	The term Technical Signal is used when we are referring to the "real world" signal that is under consideration. So typical Technical Signals are temperature, velocity, torque, force, electrical current and voltage, etc.
Comment	–
Example	–
Reference	–

4.324 Timed Communication

Definition	Implicit communication with logical publication at the end of a `Logical Execution Time` (LET) interval and logical read at the beginning of a LET interval. Access to the timed communication buffer is available for all EEs that reference the LET interval through EECs.
Initiator	Software and `Architecture`
Further Explanations	Data consistency is guaranteed between EECs which reference the same LET interval (regardless of their `Task` and core `Mapping`). Implicit communication crossing LET interval boundaries is redefined to timed communication. See also the example in Figure 4.8.
Comment	One OS `Task` can have several implicit communication buffers for several LET intervals. Implicit communication between EEs referencing the same LET interval remains unchanged.
Example	–
Reference	–

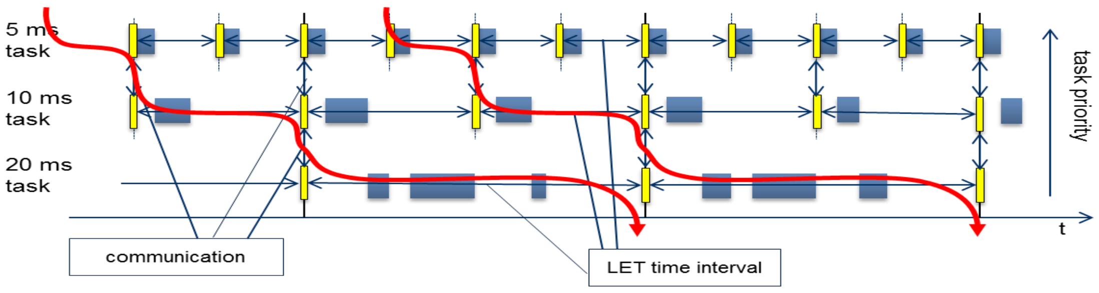


Figure 4.8: Example for Timed Communication

4.325 Timeout

Definition	`Notification` with respect to `Deadline` violation of an `Event` or `Task` (e.g. while working on/ with information: receiving, sending, processing, etc.).
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	–

4.326 Top-level PNC Coordinator

Definition	The top-level PNC (Partial Network Cluster) coordinator is the topmost coordinator in a PN topology, the single root of the directed acyclic graph forming the logical partial `Network` cluster for particular PNCs. The top-level PNC coordinator is responsible for the coordinated shutdown of all PNC members, because it is the only node which has the full overview of the states of all PNCs. Only the PNC top-level coordinator is allowed to send a PN shutdown messages. Note that for different PNCs it is possible to have different top-level PNC coordinators. For the same PNC only one top-level coordinator is supported.
Initiator	Partial Networking
Further Explanations	–
Comment	Partial networking is used to group nodes all over the `Network` topology into logical clusters. The clusters can be activated independently from each other. Each partial network cluster has a defined hierarchy, which spans from top level coordinator across multiple (0...N) levels of `Intermediate PNC Coordinators` to the `PNC Leaf Nodes`. PNC leaf nodes form the lowest level of the PNCs.
Example	–
Reference	–

4.327 Tracing

Definition	Tracing is the activity of recording a program's observable state and execution path within the `System` over a certain period of time.
Initiator	Software and `Architecture`
Further Explanations	Tracing collects events of selected types over time and stores the information in a so called "trace buffer", which may be located on-board or off-board. To enable timing measurement and `Performance` optimization, the events may be stored together with a timestamp. Tracing is intended to be mainly used during the development phase, with possibly added extra tracing software and/or hardware.
Comment	The recording may be done by software solutions, e.g. code instrumentation, hardware assisted solutions like CPU instruction flow tracing or Ethernet sniffers, or a combination of both. Depending on this, tracing may have a considerable impact on the system's timing, which must be considered when doing further analysis. In contrast to `Logging`, tracing does not target arbitrary but only correlating events of internal state transitions, variable content, and program flow.
Example	Program flow trace, `Scheduler Event` trace, Ethernet protocol trace, Stack usage trace.
Reference	–

4.328 Trusted Platform

Definition	An execution platform supporting a continuous chain of trust from boot through to application. The trust chain ensures that all execution is both authenticated (that all code executed is from the claimed source) and subjected to `Integrity Validation` (that prevents tampered code/`Data` from being executed)
Initiator	Software and `Architecture`
Further Explanations	–
Comment	–
Example	–
Reference	–

4.329 Uplink Port

Definition	An Uplink Port is a port of an automotive Ethernet switch which is connected to another Ethernet automotive switch (`Cascaded Switch`). An Uplink Port could either be an `Internal Port` or an `External Port`. One Uplink Port is connected to another `Uplink Port`. The Uplink Port has a special role from the perspective of the software.
Initiator	Communication
Further Explanations	–
Comment	–
Example	–
Reference	–

4.330 Use Case

Definition	A use case defines a list of actions defining the interactions between actors and a system to achieve a goal.
Initiator	General
Further Explanations	–
Comment	Use cases are used in the system analysis, for example, to identify and to clarify system requirements, and to define the behavior of a system.
Example	–
Reference	

4.331 Validation

Definition	Confirmation by examination and provisions of objective evidence that the particular requirements for a specific intended use are fulfilled.
Initiator	General

▽

△

Further Explanations	In design and development, validation concerns the process of examining a product to determine conformity with user needs. Validation is normally performed on the final product under defined operating conditions. It may be necessary in earlier stages. "Validated" is used to designate the corresponding status. Multiple validations may be carried out if there are different intended uses. [ISO 8402: 1994]
Comment	–
Example	–
Reference	[54]

4.332 Variability

Definition	Variability of a system is its quality to describe a set of variants. These variants are characterized by `Variant` specific property settings and / or selections.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	As an example, such a system property selection manifests itself in a particular "receive `Port`" for a connection.
Reference	–

4.333 Variant

Definition	A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no `Variability` anymore with respect to the binding time.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.334 Variant Coding

Definition	Adaptation of SW by selection of functional alternatives according to external requirements (e.g. country-dependent or legal restrictions).
Initiator	Software and `Architecture`

▽

△

Further Explanations	The major difference with Calibration is that this later doesn't aim to adapt the SW Functionality itself but only aims to adjust the SW to the HW/SW environment, e.g. the Calibration of engine control SW that is adjusted to the physical parameters of every engine. Variant Coding also includes vehicle-specific (not user-specific) SW adaptation due to end-customer wishes (e.g. deactivation of speed-dependent automatic locking). Variant Coding is always done after compile time. Used techniques to select variants include end-of-line programming and garage programming.
Comment	–
Example	Country related adaptation of MMI with respect to speed and/or temperature unit (km/h vs. mph, ℃ vs. F).
Reference	–

4.335 Variation Binding

Definition	A variant is the result of a variation binding Process that resolves the Variability of the system by assigning particular values/selections to all the system's properties.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.336 Variation Binding Time

Definition	The variation binding time determines the step in the methodology at which the Variability given by a set of variable properties is resolved.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–
Reference	–

4.337 Variation Point

Definition	A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete Variant.
Initiator	Methodology and Templates
Further Explanations	–
Comment	–
Example	–

▽

△

Reference	–

4.338 Vehicle State Manager

Definition	An OEM specific application that controls vehicle level state and interacts with various entities in the vehicle system.
Initiator	Safety and UCM
Further Explanations	–
Comment	The OEMs may just have one such application and send the vehicle state on the communication bus or may have one Master Vehicle State Manager and `Client` Vehicle State Manager applications on each ECU.
Example	–
Reference	[51]

4.339 Vehicle Variant

Definition	A vehicle variant is a fully characterized product and a subset of the artifacts of the product line.
Initiator	–
Further Explanations	–
Comment	–
Example	–
Reference	–

4.340 Vendor ID

Definition	A vendor ID is a unique identification of the vendor of a `Software Component`. All `Basic Software Modules` conforming to the AUTOSAR Standard shall provide a readable vendor ID.
Initiator	General
Further Explanations	AUTOSAR Vendor IDs are used to determine vendors of Basic Software Modules before and during runtime. The mechanism is used to improve bug handling. AUTOSAR currently only provides Vendor IDs to members of the AUTOSAR partnership.
Comment	To apply for an AUTOSAR vendor ID the possible member has to send an E-Mail to request@autosar.org. Within the request name of the company, company address and contact person should be listed.
Example	Vendor ID for EEPROM driver is called: EEP_VENDOR_ID
Reference	SRS_BSW_00374

4.341 Verification

Definition	Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled.
Initiator	General
Further Explanations	In design and development, verification concerns the process of examining the result of a given activity to determine conformity with the stated requirement for that activity. "Verified" is used to designate the corresponding status. [ISO 8402: 1994]
Comment	–
Example	–
Reference	[54]

4.342 VFB View

Definition	The VFB View describes systems or subsystems in the car independently of these resources; in other words, independently of: • what kind of and how many `Electronic Control Units` (ECUs) are present in the car • on what ECUs the entities in the VFB-View run • how the ECUs are interconnected: what kind of `Network` technology (CAN, LIN, ...) and what kind of topology (presence of gateways) is used
Initiator	Software and `Architecture`
Further Explanations	In the VFB-View, the system or subsystem under consideration is a `Composition` which consists out of `Connectors` and Components.
Comment	–
Example	–
Reference	[9]

4.343 Virtual Functional Bus

Definition	The Virtual Functional Bus is an abstraction of the communication between `Atomic Software Components` and `AUTOSAR Services`. This abstraction is such that specification of the communication mechanisms is independent from the concrete technology chosen to realize the communication.
Initiator	Software and `Architecture`
Further Explanations	After compilation and linking of software for a dedicated `Electronic Control Unit` the Virtual Functional Bus interfaces are realized by the AUTOSAR Runtime Environment.
Comment	–
Example	–
Reference	–

4.344 Virtual Integration

Definition	The simulated, modeled and/or calculated (not real) combination of software entities forming a `System`.
Initiator	Software and `Architecture`
Further Explanations	By virtual integration several constraints and/or requirements are checked without the need of real hardware units, like needed CPU load, needed memory, completeness of interfaces, fulfillment of timing requirements etc.).
Comment	–
Example	–
Reference	–

4.345 Virtualization

Definition	Virtualization is a mechanism which hides the physical characteristics of a computing platform from the users, presenting instead another abstract computing platform. It can be used to fulfill functional `Safety` requirements like `Availability`, `Partitioning`, `Resource` conflict management, `Recovery` etc.
Initiator	Safety
Further Explanations	Different types of hardware virtualization include: • Full virtualization - almost complete simulation of the actual hardware to allow software, which typically consists of a guest operating system, to run unmodified. • Partial virtualization - some but not all of the target environment attributes are simulated. As a result, some guest programs may need modifications to run in such virtual environments. • Paravirtualization - a hardware environment is not simulated; however, the guest programs are executed in their own isolated domains, as if they are running on a separate `System`. Guest programs need to be specifically modified to run in this environment.
Comment	–
Example	–
Reference	–

4.346 Wake-up and Sleep on Dataline

Definition	Wake-up request or sleep request which is transmitted over a dataline of an Automotive Ethernet switched `Network`.
Initiator	Communication
Further Explanations	Wake-up on dataline is used for Ethernet switched `Network` and is defined for 100Base-T1. A wake-up on dataline without an established link between the communication partner is called a wake-up pulse (WUP). A wake-up on dataline with an established link between the communication partners is called a wake-up request (WUR). A sleep request is transmitted as a burst of continues LPS (low power signal)
Comment	–
Example	–
Reference	OPEN ALLIANCE Sleep/Wake-up specification for Automotive Ethernet

4.347 Worst Case Execution Time

Definition	Maximum possible time during which a program is actually executing
Initiator	Software and Architecture
Further Explanations	The worst case execution time of a piece of software is the maximum possible time during which the CPU is executing instructions which belong to this piece. The worst case execution time is often identified by analytical methods. It is required to determine if a schedule meets the overall timing requirements. Abbreviation: WCET See also: Response Time, Execution Time, Worst Case Response Time
Comment	This definition has been extended by WP COM.
Example	–
Reference	–

4.348 Worst Case Response Time

Definition	Maximum possible time between receiving a stimulus and delivering an appropriate response or reaction.
Initiator	Software and Architecture
Further Explanations	The worst case response time describes the maximum possible time between a stimulus like e.g. the state change of hardware or software entity and the expected reaction of the System (e.g. response, actuator activation). Typically: worst-case execution-time + infrastructure-overhead + scheduling-policy = worst-case reaction time Synonym: Worst Case Reaction Time (WCRT) See also: Response Time, Execution Time, Worst Case Execution Time
Comment	Worst case reaction time was renamed to worst case response time because response time is the more common terminology. This definition has been extended by WP COM.
Example	–
Reference	–