Document Title	Specification for the ECU Resource Template
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	060
Document Classification	Standard

Document Version	1.0.3
Document Status	Final
Part of Release	3.0
Revision	0002

Document Change History

Date	Version	Changed by	Change Description
22.01.2008	1.0.3	AUTOSAR Administration	• Correction of References
05.12.2007	1.0.2	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
31.01.2007	1.0.1	AUTOSAR Administration	• Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
09.05.2005	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

Table of Figures and Tables

Table of Figures and Tables

1 Introduction

One of the most prominent goals of AUTOSAR is the standardization of descriptions relevant for automotive software applications. In this context, the description of underlying Electronic Control Units (ECU[1]) is one of the major topics to resolve.
This document contains a description of all hardware modelling elements required to describe an ECU to the necessary extent. One aspect of this description is to provide configuration tools with information needed to configure the microcontroller and ECU abstraction layer residing on a particular ECU[2].
The description of an ECU is based on the ECU Resource Template. The latter (in form of a XML DTD) is directly derived from the AUTOSAR meta-model by generation tools.
Where applicable, please consult the glossary (chapter 1.4) and the abbreviation list (chapter 1.5) contained in this document. The general characteristics of the ECU resource description are introduced followed by a detailed description of hardware components inside the ECU is provided. The main subchapters are about the modelling elements ECU, memory, the processing unit, peripherals, ECU electronics and sensors and actuators.

1.1 Goals of the ECU Resource Template

The main goal of the ECU Resource Template is to provide a solid basis for describing and checking the consistency of characteristics and features of automotive ECUs. The scope of ECU resource description is focussed to characteristics and features that are relevant for the configuration of the microcontroller and ECU abstraction layer on a particular ECU.
In other words: ECU resource descriptions created on the basis of the ECU Resource Template shall enable AUTOSAR tools to properly configure the microcontroller and ECU abstraction layer residing on the described ECU. Please note that the configuration of the microcontroller and ECU abstraction layer will supposedly not be possible as an automatic process but require more or less substantial engineering skills. A tool, for example a "AUTOSAR ECU Configuration Generator" will be used to support the user to manually process information contained in the ECU Resource Description and set up a suitable configuration of the microcontroller and ECU abstraction layer. This configuration can then be picked up by a generation tool that creates or configures the actual microcontroller and ECU abstraction layer (see Figure 1-1).

[1] An ECU is an embedded computer system consisting of at least one processing unit, corresponding periphery and memory which is placed in one package.
[2] Please note, however, that the ECU Resource Template is not to be confused with a schematic, thermal design specification, EMC guideline, or hardware model for simulation purposes.

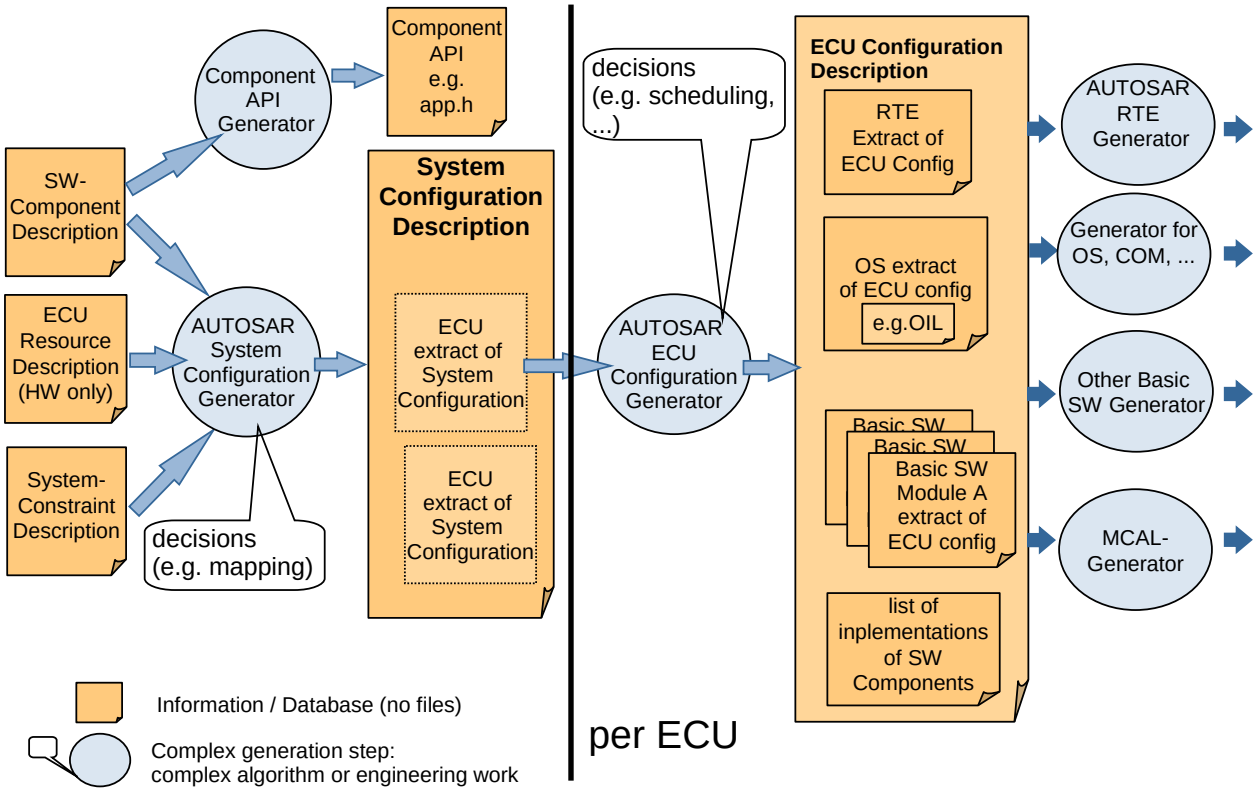

Figure 1-1: Configuration of microcontroller and ECU abstraction layer on the basis of an ECU Resource description

Tools for configuring abstraction layers of particular µCs are already available on the market. Admittedly, this concept requires the AUTOSAR ECU Configuration Generator to master an additional level of complexity because external peripheral devices have to be covered as well. As a result, the configuration editor will have to provide some sort of generic mechanism for the description of hardware resources.
Please note, that the workflow sketched in Figure 1-1 is contained in this document as a mere motivation of the approach for the current definition of the ECU Resource Template. The definition as well as possible enhancements of the workflow is actually out of the scope of the ECU Resource Template and will not be further discussed.
In the majority of cases, ECU resource descriptions based on the ECU Resource Template will be provided by ECU and hardware manufacturers for further usage in the AUTOSAR development process. For example, the description of software components has interrelations with ECU resource descriptions, in particular the comparison of available (ECU Resource Template) and required (Software Component Template) memory.
The consideration of interrelations between the ECU Resource Template and other AUTOSAR templates is another major goal of this document.
In general, typical automotive ECUs might be equipped with a wide variety of different microcontrollers and peripheral devices. In other words: the architecture of a typical automotive ECU cannot be described using a fix description scheme. Consequently, the ECU Resource Template shall provide enough flexibility to cover different microcontroller- and ECU-architectures.

1.2 Scope of the ECU Resource Template

The scope of the ECU Resource Template is the description of ECUs by means of the following basic building blocks:

- Hardware Elements (HW Elements)
- Hardware Ports (HW Ports)
- Hardware Connections (HW Connections)

The HW Elements are the main describing elements of an ECU, For example: Processing units, memory, peripherals and sensors/actuators. HW Elements have a unique name and can be identified within an ECU.
HW Elements do not necessarily have to be described on the level of an ECU. It is possible to describe HW Elements as parts of other HW Elements. By this means a hierarchical description of HW Elements can be created.
HW Elements provide HW ports for being interconnected among each others. Each HW Port has a name which has to be unique within the HW Element it is located in. In order to address a specific HW Port, the name of the HW Element and then the name of the HW Port have to be concatenated (e.g. "PU1/ADCPort3").
Please note that each HW Element provides a name-space. In other words: names of ports within HW Elements are required to be unique within the scope of the very HW Element[3].
HW Connections are used to describe the connection of HW Elements among each others. The provision of the entire information that is later needed to configure the microcontroller and ECU abstraction layer is in the scope of the ECU Resource Template.
The contents of ECU Resource Template directly correspond to the package "ECUResourceTemplate" in the AUTOSAR meta-model.

1.3 Overview ECU Resource Template

Figure 1-2 depicts the main HW elements of an ECU resource description and their interrelations.

[3] This concept works pretty much like a hierarchical file-system where the file names are required to be unique only in the scope of a particular folder. The same file name can be used in **different** folders without any problem.

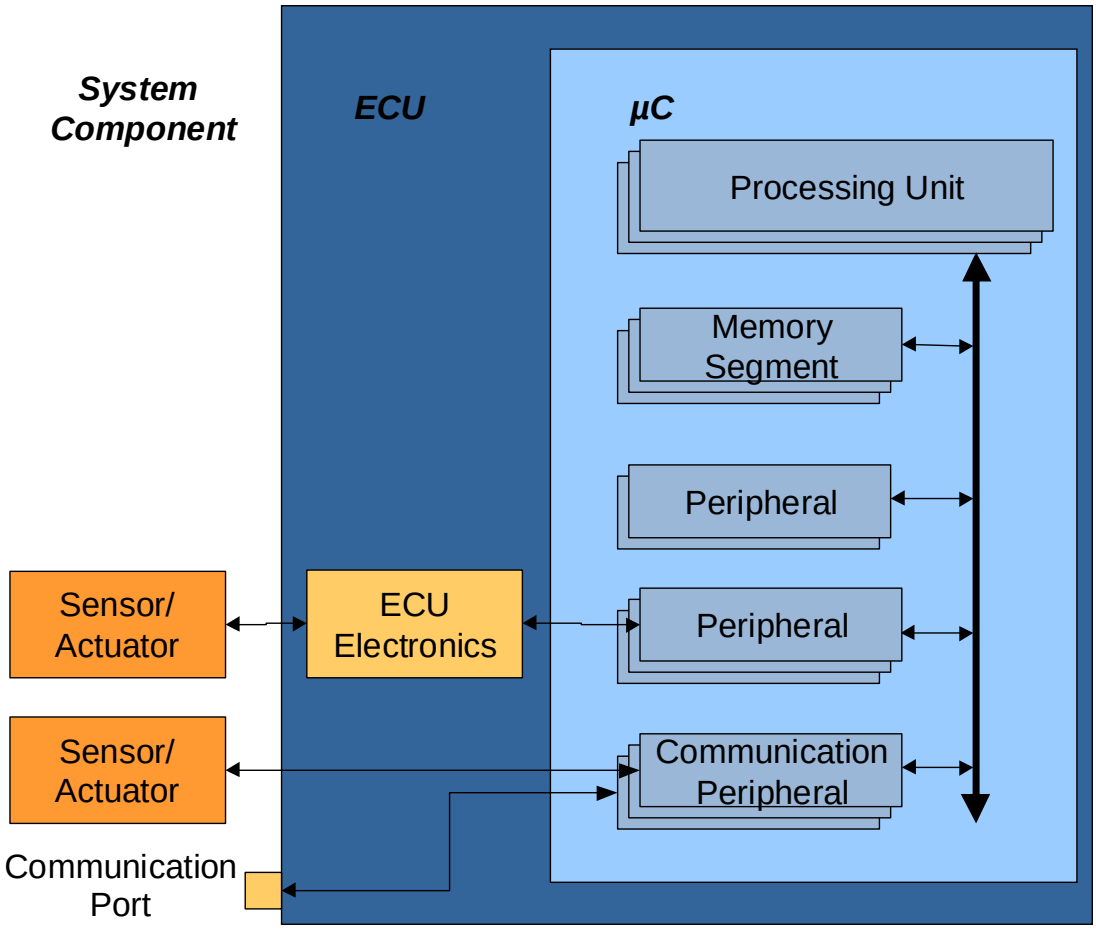

Figure 1-2: Overview ECU Template

Modelling elements in the ECU Resource Template can be hierarchically organized. A particular ECU can be described as a hierarchical composition of microcontrollers and ECU Electronics. Each microcontroller is in turn composed of processing units, memory, peripherals and management units.
The same approach can be used to describe a particular ECU in combination with all sensors and actuators attached to the ECU.
The ECU Electronics is the hardware present on the ECU to guarantee the operation of the Processing Units (clock) as well as the conditioning of signals going out of the ECU or coming in (communication transceiver, amplifier, discrete electronics).
On the application level, each sensor/actuator is represented by exactly one sensor/actuator SW components. A typical purpose of sensor/actuator SW component is to provide conversion formula etc.
Since the SW-Component Template is used to describe the Sensor/Actuator SW-Components, a unidirectional reference from the software description to the actual sensor/actuator description (the description of the hardware) is mandatory.

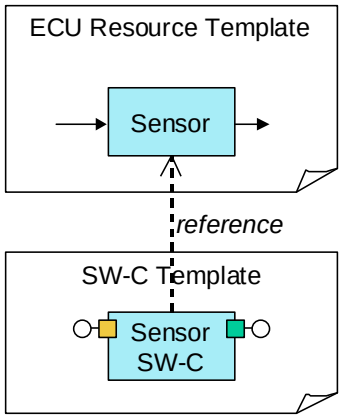

Figure 1-3: Shipment of a Sensor

The reference from the SW-Component description to the ECU Resource description is based on the name of the sensor/actuator provided in the ECU Resource description.

1.4 Glossary

The glossary consists of definitions that relevant for the understanding of the ECU Resource Template but have no impact on the general AUTOSAR terminology. Please refer to the AUTOSAR glossary for a description of general AUTOSAR terms.

Term	Description
AUTOSAR Signal	see AUTOSAR Glossary (Version 1.9)
Conditioned Signal	The Conditioned Signal is the internal electrical representation of the Electrical Signal within the ECU. The Conditioned Signal is delivered to the PU and represented in voltage and time (or, in case of logical signals, by high or low level)..
Electrical Signal	The Electrical Signal is the electrical representation of Technical Signals. Electrical Signals can only be represented in voltage, current and time.
Element-Tree	The element-tree is the complete definition path starting at the highest and ending at the lowest level. The element-tree is a concatenation of elements and at the end of an element-tree primitives only are used.
HW Connection	Abstract modelling means for defining the interrelationship among HW Elements.
HW Container	Abstract modelling means for creating hierarchical descriptions of hardware.
HW Element	A HW Element is the piece or a part of the piece to be described with the ECU Resource Template. It uses other elements or primitives; this means HW Elements can be nested (through HW Containers). At the lowest level a HW Element only uses primitives.
HW Port	The HW Port exposes functionality to the exterior of the HW Element. HW Elements can be connected via HW Connections.
MCAL Signal	The MCAL Signal is the software representation of the Conditioned Signal. The MCAL Signal is provided by the

Term	Description
	Microcontroller Abstraction Layer (MCAL) and is further processed by the ECU Abstraction.
Pin	A pin is one elementary electric interface of an electrical unit. Pins are associated with HW Ports.
Primitive	A primitive is an elementary type of element.
Signal name	In AUTOSAR several kinds of signals exist that represent the same information at different levels during its processing. Therefore it is mandatory that all these different kinds of signals have to be bound together. A specific signal naming convention is supposed to ensure that e.g. a physical, analogue rotation speed of the wheel will result in an AUTOSAR Signal (e.g. “get_v())” that can be provided by a Sensor SW-C.
Technical Signal	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).

1.5 Abbreviation list

Abbreviation	Meaning
ACE	Advanced Communication Engine
ASIC	Application Specific Integrated Circuit
BSD	Bit-serial Data Link
CAN	Controller Area Network
CC	Capture Compare
CCU	Capture Compare Unit
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Digital to Analogue Converter
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signalling Processor
DTD	Document Type Definition
ECC	Error Checking and Correction
EDC	Error Detection Code
EEPROM	Electrically Erasable Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
EMC	Electro Magnetic Compatibility
ESD	Electro Static Discharge
E^2PROM	→ EEPROM
FIFO	First In First OUT
FIT	Failure In Time

Abbreviation	Meaning
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FW	Fire Wire
GSM	Global System for Mobile communication
HF	High Frequency
HIS	Hersteller Initiative Software
I2C	Inter-integrated Circuits serial interface
IC	Integrated Circuit
IEC	International Electrotechnical Commission
IO	Input/Output
ISO	International Standardisation Organisation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LIFO	Last In First Out
LIN	Local Interconnected Network
LSB	Least Significant Bit
LVDS	Low Voltage Differential Signalling
µP	Microprocessor
MCAL	Microcontroller Abstraction Layer
MIMD	Multiple Instruction Multiple Data
MIPS	Millions of Instructions Per Second
MMU	Memory Management Unit
MOST	Media Oriented Systems Transport
MPU	Memory Protection Unit
MSB	Most Significant Bit
MSR	Manufacturer Supplier Relationship
MSRSYS	MSR System
MTBF	Mean Time Between Failure
NV	Non Volatile
OEM	Original Equipment Manufacturer
OLED	Organic Light Emitting Diode
OSI	Open Systems Interconnection
PCP	Programmable Communications Processor
PLL	Phase Locked Loop
PPM	Parts Per Million
PROM	Programmable Read Only Memory
PU	Processing Unit
PWD	Pulse Width Demodulation
PWM	Pulse Width Modulation
RAM	Random Access Memory
RGB	Red/Green/Blue

Abbreviation	Meaning
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SCI	Serial Communication Interface
SIM	Subscriber Identity Module
SIMD	Single Instruction Multiple Data
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
STN	Super Twisted Nematic
TFT	Thin Film Transistor
TPM	Trusted Platform Module
TPU	Time Processor Unit
TTP	Time-Triggered Protocol
USB	Universal Serial Bus
UV	Ultra Violet
VFD	Vacuum Fluorescent Display
WDT	Watchdog Timer
XML	Extensible Markup Language
XOR	Exclusive-OR

2 Basics of Hardware Descriptions

As already mentioned in chapter 1.2 the description mechanism used to create an ECU Resource Description is founded upon a few general building blocks. A General HW Element acts as a base class for all specific HW Elements like ECU, Memory, PU, etc. This General HW Element contains all properties that are common for all specialized HW Elements.
To provide a proper means to describe ECUs as a whole a few other description elements are necessary: HW Ports (see 2.2) and HW Connections (see 2.4). They are used to establish communication or data exchange relations between different HW Elements and exist in several specializations, too. An example is shown in Figure 2-1

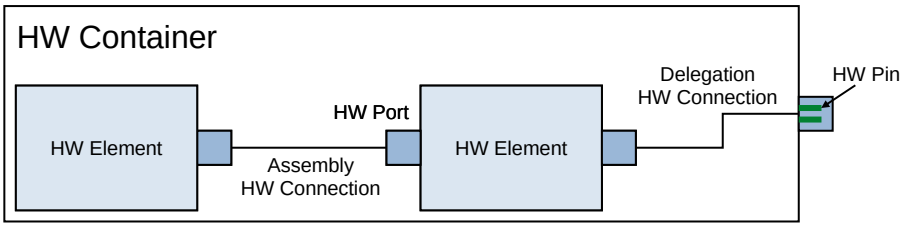

Figure 2-1: HW Elements, HW Ports, and HW Connections

One other description element shown above is the HW Pin. This elements comes handy if a physical connection point has to be described like the specific pins of a µC. The HW Pin is an optional feature of a HW Port and will only be used in visible boundaries (like µCs, ECUs) if appropriate.
With these elements the creation of HW Connections between different HW Elements (see Figure 2-1**)** is optimally supported from the technical point of view.
Chapter 2.1 describes the General HW Element. It is often necessary to group several HW Elements together (e.g. internals of a µC). For this reason the HW Container is explained in chapter 2.5. The definition of the HW Port is similar to the general HW Element. Each HW Element has at least one HW Port. The base class of all HW Ports is characterised in chapter 2.2. The HW Connection is explained in chapter 2.4.
The mapping of electrical pins to signals is depicted in the appendix B (see chapter 5.1).
An iterative process for system generation, specified in the AUTOSAR Methodology, is supported by the given structure. The system generator obtains all available hardware resources and maps these against the requirements from SW Component /System Constraint Template. For example, the system generator accumulates the required memory resources from the software components and checks this against the available memory. In the case of shifting software modules, the generator must subtract the memory demand of this component from the whole amount.
The same method is valid for HW Pin/HW Port/signal assignment. The system generator maps the Conditioned Signal on a microcontroller HW Port, so that this HW Port is no longer available for other signals. In the case of shifting modules, the generator deallocates this HW Port again.

2.1 General Hardware Element

In the ECU Resource Template the General HW Element serves as an abstract base class to describe properties common for all specific HW Elements. The definitions are sketched in Figure 2-2 (Meta Model) and Table 2-1 (class table).

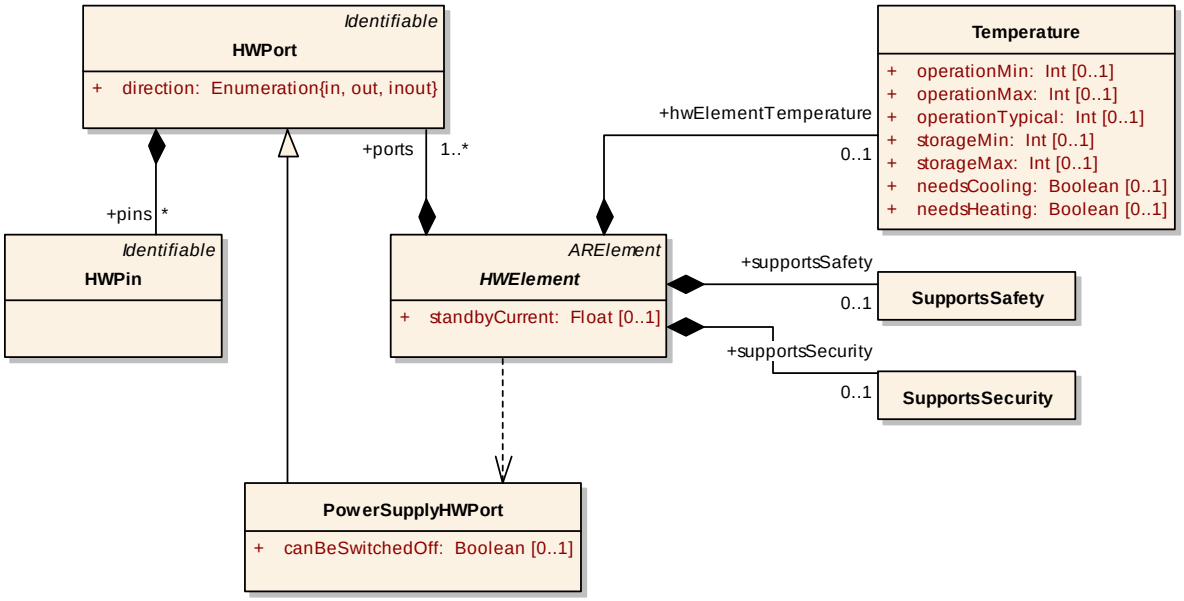

Figure 2-2: General HW Element in the Meta-Model

Class	**HWElement (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	The General HW Element specifies definitions valid for all specific HW Elements.			
Base Class(es)	ARElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
hwElementTemperature	Temperature	0..1	aggregation	For the placement of a HW Element within a car the temperature has to be considered the HW Element has been designed for.
ports	HWPort	1..*	aggregation	All the HWPorts this HWElement exposes. This also includes delegation HWPorts of HWContainers.
signalTransformations	SignalTransformation	0..*	aggregation	The Signal Transformation defines the conversion of the signal attached to a HW Port (direction In) and a signal attached to a HW Port (direction Out).
standbyCurrent	Float	0..1	aggregation	The Standby Current is the current in state OFF. For HW Elements the Standby Current consists of the current needed by the logic to stay in the state OFF and the leakage current of the HW Element. For big HW Elements the leakage current can be up to several 100 microA. Standby Current is used to determine which elements are necessary to be switch off, when the car itself is in a standby mode. This can be done automatically or by the user. The value for the Standby Current can be entered for each HW Element and HW Container respectively. The user has to take care about these values and decides about the entries necessary for the calculation of the complete standby current. On the other side the sum for all HW Elements grouped in a HW Container may have a bigger value than the value for the HW Container itself. In this case the values for the HW Elements represent the worst case for each HW Element. Unit: Ampere (A)

Class	**HWElement (abstract)**			
supportsSafety	SupportsSafety	0..1	aggregation	The ECU Resource Template provides the element Supports Safety storing the information about the supported safety mechanism. In the current version of the ECU Resource Template the element Supports Safety is optional. A HW Element uses this element if it provides a safety mechanism otherwise this element is missing in the description of the HW Element.
supportsSecurity	SupportsSecurity	0..1	aggregation	The ECU Resource Template provides the element SupportsSecurity storing the information about the supported security mechanism. In the current version of the ECU Resource Template the element SupportsSecurity is optional. A HW Element uses this element if it provides a security mechanism otherwise this element is missing in the description of the HW Element.

Table 2-1: General HW Element

A HW Element is referenced by its name defined in the Meta Model base class ARElement. The name consists of two parts: a short-name and a long-name. The short name is a unique identifier and enables the HW Element to be referenced. The long-name is used if the short-name is not self-explaining and to create a readable documentation. A reference to the long-name is forbidden.
Furthermore the Meta Model base class ARElement contains the description and the information about the manufacturer and version of a HW Element along with additional descriptions in natural language and/or references to other documents and files.
Each concrete HW Element (ECU, Memory, PU, etc) has additional elements specific for these HW Elements. These elements are explained in chapter 3. The relation between the specific HW Elements and the General HW Element in the Meta-Model is shown in Figure 2-3.

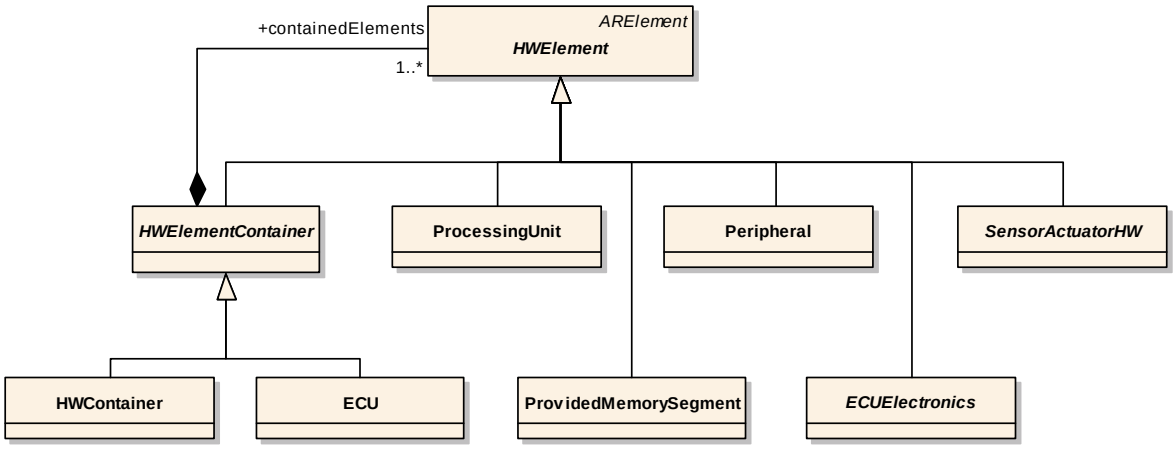

Figure 2-3: Definition of HW Elements in the Meta-Model

You will find some specific properties of a HW Element in the following subchapters.

2.1.1 Temperature

The placement of a HW Element within a car depends on the temperature of the place the HW Element is located in. If a HW Element has to be at a specific place cooling and heating may be necessary respectively.

Class	**Temperature**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	General element to describe temperature ranges and need for cooling/heating.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
needsCooling	Boolean	0..1	aggregation	These attributes describe that there is an active cooling system required for this HW Element.
needsHeating	Boolean	0..1	aggregation	These attributes describe that there is an active heating system required for this HW Element.
operationMax	Int	0..1	aggregation	It describes the temperature range (max) in which the HW Element can be operated. Unit: Kelvin (K)
operationMin	Int	0..1	aggregation	It describes the temperature range (min) in which the HW Element can be operated. Unit: Kelvin (K)
operationTypical	Int	0..1	aggregation	It describes the temperature range (typical) in which the HW Element can be operated. Unit: Kelvin (K)
storageMax	Int	0..1	aggregation	The storage temperature defines maximum temperature the device is allowed to be stored. The HW Element is not in an operational state. Unit: Kelvin (K)
storageMin	Int	0..1	aggregation	The storage temperature defines minimum temperature the device is allowed to be stored. The HW Element is not in an operational state. Unit: Kelvin (K)

Table 2-2: Temperature

2.1.2 Support Security

The ECU Resource Template provides the element SupportsSecurity storing the information about the supported security mechanism and it is stored in the Meta Model class SupportsSecurity.
In the current version of the ECU Resource Template the element SupportsSecurity is optional. A HW Element uses this element if it provides a security mechanism otherwise this element is missing in the description of the HW Element.
In a later version of the ECU Resource Template the Meta Model class SupportsSecurity is expanded due to the requirements to security. At the moment any requirements exist.
Examples for affected HW elements are

- internal memory
- external memory (e.g. memory stick)
- disk drives
- internal and external communication interfaces

- external diagnosis interface
- PU
- ECU itself

Class	**SupportsSecurity**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	The ECU Resource Template provides the element SupportsSecurity storing the information about the supported security mechanism. In the current version of the ECU Resource Template the element SupportsSecurity is optional. A HW Element uses this element if it provides a security mechanism otherwise this element is missing in the description of the HW Element.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 2-3: Supports Security

2.1.3 Supports Safety

Regarding the safety level of an ECU some information needs to be described during filling the ECU Resource Template. For this reason references to according guidelines of [ImpactsSafety] will be provided if relevant. As the current AUTOSAR methodology mostly cares about software aspects only a few safety requirements deal with hardware at all.
In the current version of the ECU Resource Template the element SupportsSafety is optional.

Class	**SupportsSafety**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	In the current version of the ECU Resource Template the element SupportsSafety is optional. A HW Element uses this element if it provides a safety mechanism, otherwise this element is missing in the description of the HW Element.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 2-4: Supports Safety

One of the major properties is the information about the safety integrity level (SIL) of the ECU itself. Referencing the IEC 61508 a safety-relevant SW-C can only be mapped to an ECU that has the same or a higher SIL. Due to software enhancements an ECU may get a higher SIL than the electronics provide. It must be assured that software and hardware descriptions use the same SIL specifications because different standards, for example IEC 61508 and MISRA, are using more or less the same levels but provide different meanings to them.
Please note that while the automotive interpretation of IEC61508 is still not available, the SIL level will supposedly become more important once the announced safety standard for automotive systems is put into operation.

The meta-class SupportsSafety is currently empty but will be filled with attributes once the implications of the automotive safety standard become more visible[4].
Another important property for hardware-related safety aspects is the documentation of mapping results due to power sources and packaging (AR-DS-77). During mapping of redundant SW-Cs to ECUs this information provides crucial hints if to ECUs may fail due to a common power failure. This property may also identify that a specific ECU is located in a crash-sensitive area and therefore is not an ideal location for a safety-relevant SW-C that requires a high availability during crash situations.
Other safety means like hardware watchdogs (AR-DS-12) or MMU/MPU usage (AR-DS-30) can be found in the peripherals section (see chapter 3.10).

2.2 HW Port

A HW Port defines a connection endpoint for HW Elements and contains common elements for all specialized kinds of HW Ports. They are defined in the base class HW Port in the Meta-Model and are shown in Table 2-5.

Class	**HWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	The general element HW Port is necessary to connect HW Elements and contains common elements for all different kind of HW Ports. HW Ports are required to be uniquely identifiable within the scope of a HW Element. This means to identify a specific HW Port it is necessary to prefix the HW Port name by the name of the HW Element (e.g."PU1/ADCPort3").			
Base Class(es)	Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
direction	Enumeration{in, out, inout}	1	aggregation	The direction of a HW Port defines signal flow in the point of view of the HW Element the HW Port belongs to. The following attributes are allowed: - In: The signal flow goes into the HW Element. The HW Element is the target for a signal. - Out: The signal flow goes from the HW Element away. The HW Element is the source of a signal. - InOut: Both, incoming and outgoing signal flows are allowed.
pins	HWPin	*	aggregation	These are the pins the HWPort assembles. If the HWPort does not have HWPins (because the HWPort is inside a micro) the set is empty.

Table 2-5: HW Port

Each HW Element may have specific HW Port elements containing the general HW Port. They are described in the corresponding chapter where their according specialized HW Element is introduced. The relation between the HW Port its optional HW Pins and the corresponding HW Element is shown in Figure 2-4.

[4] In addition, this approach is supposed to optimally support the migration of existing documents to a new template definition that supports the safety aspect to a greater extent.

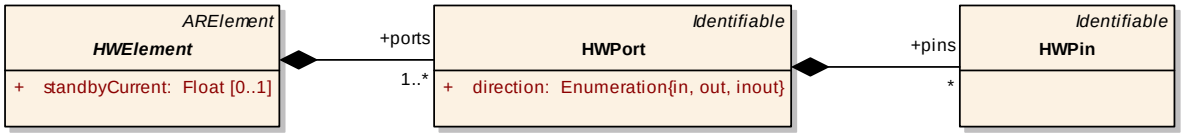

Figure 2-4: Definition of HW Port in the Meta-Model

Specialized HW Ports store information typical for the specialized HW Element they are correlated to. In Figure 2-5 an example for the usage is shown. One specialized HW Port that more or less every dedicated HW Element will have is a Power Supply HW Port (see chapter 3.12.8). This HW Port represents the plug that will be connected to a power source.

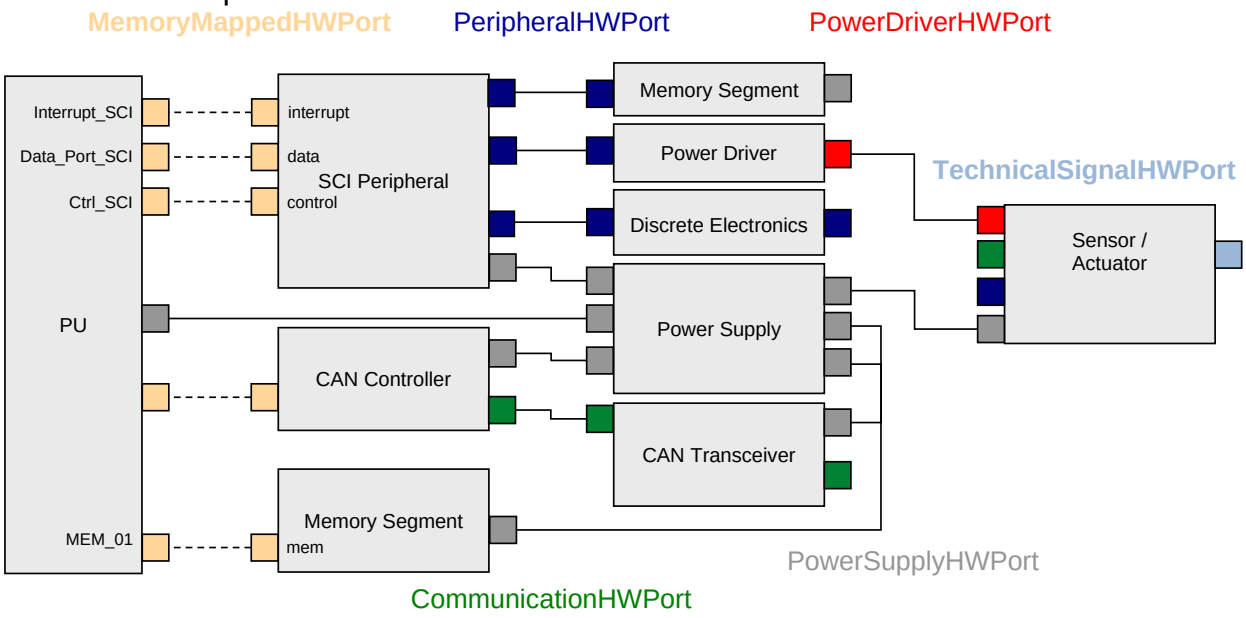

Figure 2-5: Example for the usage of the specific HW Ports and HW Connections

Note: It is not allowed to establish two HW Connections on the same hierarchical level to the same HW Port. In the description using the ECU Resource Template two different HW Ports in the same hierarchical level have to be used.
Note2: The TechnicalSignalHWPort is represented by a General HW Port assigned to a Sensor/Actuator (see 3.13.1).

2.2.1 Direction

The signal flow in the point of view of the HW Element is shown in Figure 2-6.

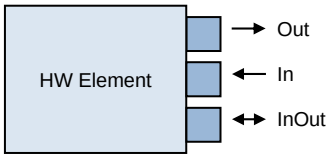

Figure 2-6: Definition of HW Port attribute direction

2.3 HW Pin

A pin is an elementary electrical interface of the HW Port and is identified by its name. The definition is shown in Table 2-6. The short name derived from the base class Identifiable represents the pin no. of that pin within its HW Port. The long name

derived by Identifiable will provide a more descriptive name that may be unique throughout the HW Element the according HW Port belongs to.

Class	**HWPin**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	A HWPin is an elementary electrical interface of the HWElement. The HWPins of a HWPort can be clustered if there are some HWPins with the same behaviour.			
Base Class(es)	Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 2-6: HW Pin

2.4 HW Connections

HW Elements are associated to each other by means of HW Connections (as already sketched in Figure 2-1). The HW Ports of the HW Elements serve as connection endpoints for this purpose. An example is shown in Figure 2-5.
There are two kinds of connections defined:

- The Assembly HW Connection (see Table 2-9) is used to connect several HW Elements via their HW Ports.
- The Delegation HW Connection (see Table 2-10) connects the HW Port of an HW Element with a HW Port of an enclosing HW Container (see 2.5 for details).

An abstract class HW Connection exists, which generalizes all common properties of the mentioned HW Connections above. This includes the optional description of the Pin HW Connection (see Table 2-8). This connection represents the pin assignment within a HW Connection between HW Ports. This may be used to describe what pins of an internal HW Element are mapped to what pins of the HW Container. It is even possible to describe the pin assignment on one side of this Pin HW Connection only if e.g. the pins of the HW Elements within a µC are not applicable to assign but the pins on the outside of the µC do exist.
Figure 2-7 shows the definition of the HW Connection in the Meta-Model.

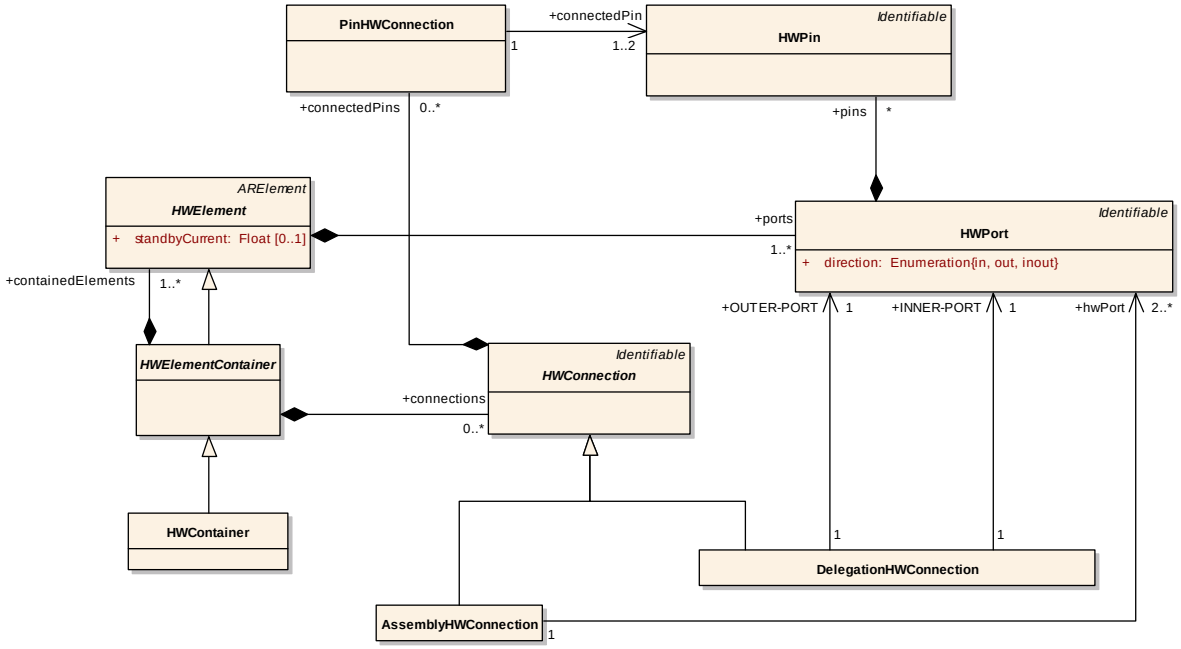

Figure 2-7: HW Connection in the Meta-Model

Class	**HWConnection (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	Abstract class to specify the ability to connect HWPorts.			
Base Class(es)	Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
connectedPins	PinHWConnection	0..*	aggregation	A set of connections between HWPins of the connected HWPorts.

Table 2-7: HW Connection

Class	**PinHWConnection**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	Is used to connect the HWPins of the HWPorts that are referenced by the HWConnection.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
connectedPin	HWPin	1..2	reference	Reference to the connected pin. When the PinHWConnection is used in the context of a DelegationHWConnection there might be only ONE pin known (typically this is a pin of the OUTER-PORT), so only one pin has to be specified.

Table 2-8: PinHWConnection

Class	**AssemblyHWConnection**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	An AssemblyHWConnection is established between at least two HWPorts provided by different HWElements. The AssemblyHWConnection can only be established between HWPorts on the same hierarchical level. To connect accross hierarchical levels use the DelegationHWConnection.			
Base Class(es)	HWConnection			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
hwPort	HWPort	2..*	reference	References to HWPorts connected by this AssemblyHWConnection.

Table 2-9: AssemblyHWConnection

Class	**DelegationHWConnection**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	Is used to connect HWPorts of a HWContainer with HWPorts of the contained HWElements.			
Base Class(es)	HWConnection			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
INNER-PORT	HWPort	1	reference	Reference to the HWPort of the connected inner HWElement. A single HWPort can only be referenced by exactly one DelegationHWConnection.
OUTER-PORT	HWPort	1	reference	Reference to the HWPort of the connected outer HWElement. A single HWPort can only be referenced by exactly one DelegationHWConnection.

Table 2-10: DelegationHWConnection

Figure 2-8 depicts an example how the different HW Connections may be used:

- **AC1**
  represents a µC-internal connection between HW Elements. This is represented by an Assembly HW Connection. As there are no internal details this HW Connection does not provide a Pin HW Connection.
- **DC1**
  represents a Delegation HW Connection to the outer boundary of the µC. The outside HW Port of the µC does provide information about the HW Pins that belong to it. As there is no information about the internal details the Pin HW Connection does only provide the pin information on one side of the connection.
- **AC2**
  represents an Assembly HW Connection between HW Elements where the Pin HW Connection is used to map the pins of the µC to the pins of the CAN transceiver.
- **DC2**
  represents a Delegation HW Connection where the pins of the CAN transceiver are mapped to the pins of the outer ECU port (an ECU Communication HW Port in this example).

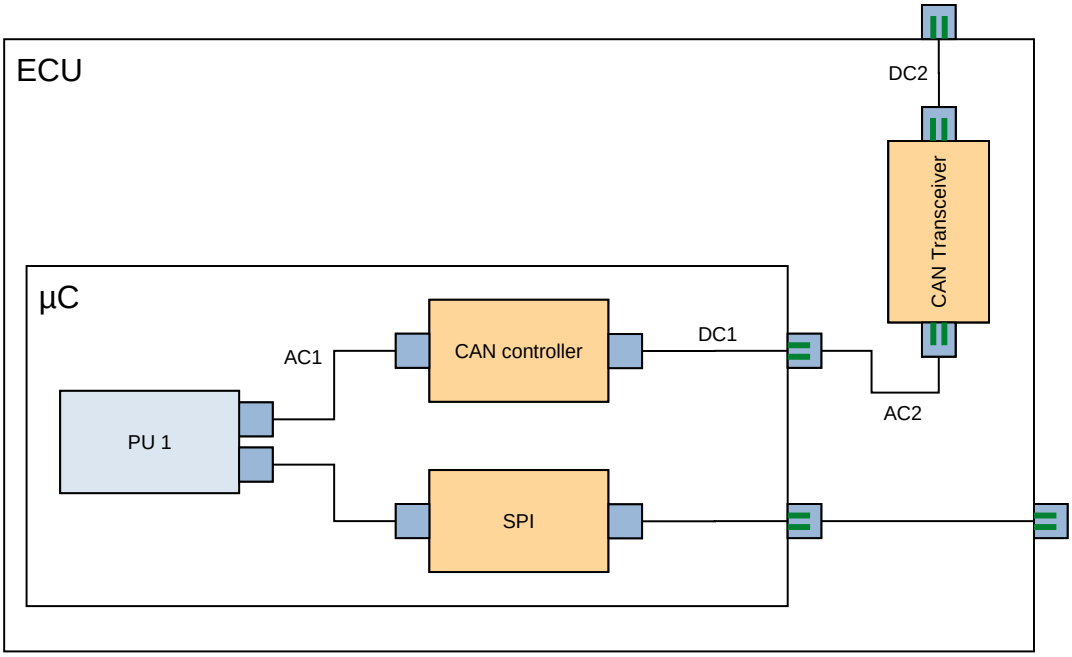

Figure 2-8: Example of different connection types and their usage

HW Connections between particular HW Elements have different characteristics. Therefore, it is necessary to derive different subclasses (with different elements) for connecting particular HW Elements in the Meta-Model. The HW Element specific HW Connection elements are described in the corresponding chapter, if there exists one.
The assignment of HW Connections has to be verified by the AUTOSAR Template Editor: only HW Ports of the same type are allowed to be connected.
Please note that the main advantage of the generic HW Connection concept over a fixed ECU layout is the flexibility. Especially the consideration of sensors and actuators is perfectly supported by this concept. In particular, it is possible to describe the path from a sensor to the PU (and the path from the PU to an actuator) with virtually unlimited detailing (although in most cases the level of detail is expected to be rather low).

2.4.1 Connection Topology

An Assembly HW Connection can be used to connect multiple (more than 2) HW Ports to create a bus-architecture. Therefore the Assembly HW Connection references all the HW Ports it is connected to. A simple example is shown in Figure 2-9.

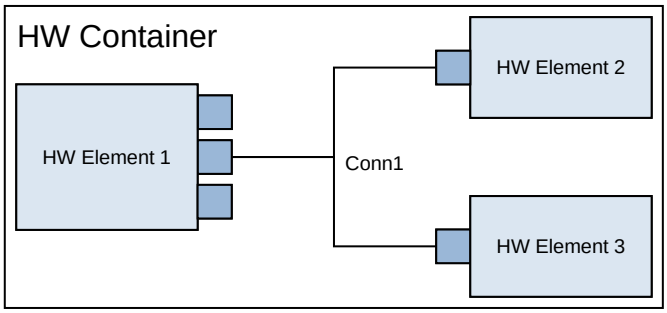

Figure 2-9: Allowed assignment of one HW connection to multiple HW ports

This method allows n:m communication of Assembly HW Connections, too, as shown in Figure 2-10

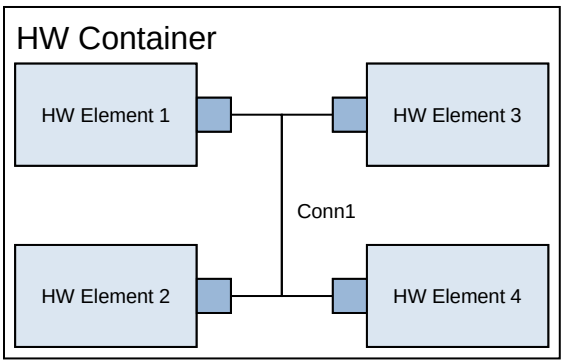

Figure 2-10: Example of Bus Topology

In the examples above each HW Element is able to communicate with another HW Element over the designated Assembly HW Connection. But as this Assembly HW Connections is shared, appropriate access control has to be taken into account during runtime.
On the other hand, it is **not** permitted to associate multiple Assembly HW Connections to a single HW Port, like shown in Figure 2-11.

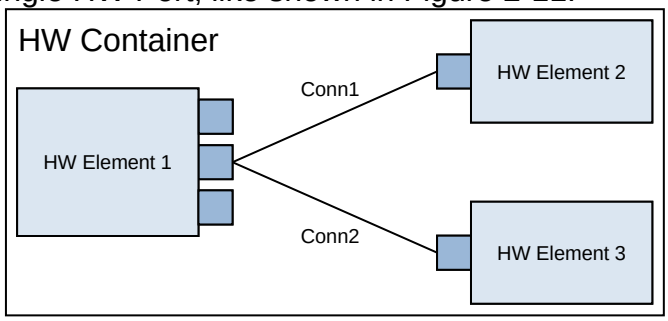

Figure 2-11: Forbidden assignment of multiple HW Connections to one HW Port

To specify a situation that one HW Element is connected to two other HW Elements, independently, a slightly different approach has to be taken. In Figure 2-12 the situation is shown where PU1 and PU2 have to communicate with a Memory that is shared. But on the other hand both PUs are prohibited to communicate with each other. Therefore each Assembly HW Connection needs its own HW Port at the shared memory.

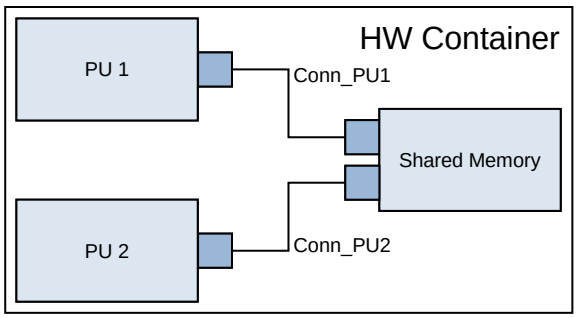

Figure 2-12: Example of Shared Memory

2.5 HW Element Container and HW Container

To be able to describe hardware that is structured hierarchically (e.g. a microcontroller consists of a processing unit, memory and peripherals, an ECU consists of a microcontroller and ECU Electronics, etc.) the Container Elements are introduced. The HW Element Container is abstract and defines the containing

relationship of HW Elements and HW Element Containers only. In addition the HW Element Container stores all HW Connections between its contained HW Elements (including Delegation HW Connections to outer HW Ports), too.

Class	**HWElementContainer (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	Abstract class to enable the collection of HW Elements.			
Base Class(es)	HWElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
connections	HWConnection	0..*	aggregation	The connections gathered within the HWElementContainer.
containedElements	HWElement	1..*	aggregation	The HW Elements contained in the HW Element Container.

Table 2-11: HW Element Container

The HW Element Container is used to derive two other elements: the general HW Container and the ECU. The main difference is that the HW Container provides all kinds of HW Ports (including Memory Mapped HW Port), while the ECU is not allowed to provide the Memory Mapped HW Port.

Class	**HWContainer**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	A HW Container is a group of HW Elements on the same hierarchical level and is an abstraction of composite HW Elements. The HW Container is a specialisation of a HW Element and therefore the HW Container has all elements of a HW Element. The values of elements of the HW Elements grouped in a HW Container may differ from the values of the elements of the HW Container.			
Base Class(es)	HWElementContainer			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 2-12: HW Container

A HW Container provides at least one HW Port. The provided HW Ports are connected to the HW Ports of the grouped HW Elements via Delegation HW Connections. The grouped HW Element's HW Ports can also be connected to each other internally.
Examples for the HW Containers are:

- Microcontroller, which consists of PU, memory and peripherals
- complete ECU, which consists of µC, external memory, external peripherals
- complete sub-system, which consists of a complete ECU and associated Sensors/Actuators

A good example for different HW Containers was already depicted in Figure 2-8.

2.6 Signal Transformation

The signal transformation between two HW Ports of a HW Element is described with the Meta-Model mechanism shown in Figure 2-13.

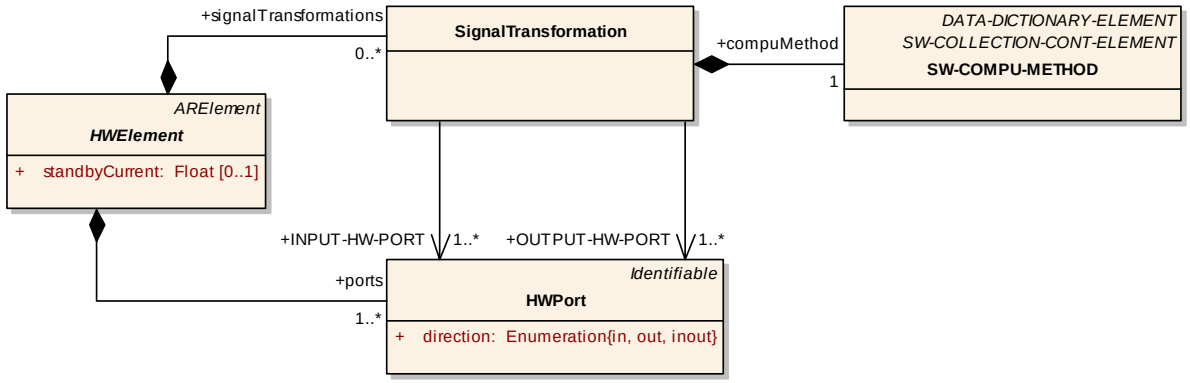

Figure 2-13: Signal Transformation

Class	**SignalTransformation**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	The Signal Transformation defines the conversion of the signal attached to a HW Port (direction In) and a signal attached to a HW Port (direction Out).			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
INPUT-HW-PORT	HWPort	1..*	reference	The input port the signal transformation is refering to.
OUTPUT-HW-PORT	HWPort	1..*	reference	The output port the signal transformation is refering to.
compuMethod	SW-COMPU-METHOD	1	aggregation	The description of the transformation between inpot and output port as a MSR-SW SW-COMPU-METHOD.

Table 2-13: Signal Transformation

This 1:1 transformation is realised with the MSR SW-COMPU-METHOD. All other signal transformations use other mechanisms like formal description, program code, etc.
In the scope of the ECU Resource Template the Technical Signals are one of the source/sink of signal transformations, e.g. sensors are at the start, actuator are at the end.

SW-COMPU-METHOD:

- **Scaling:**
  Describes the form of transforming the physical data into data
  - **Linear:**
    The relation of the physical data and the logical is in a linear way
  - **Non-linear:**
    The relation of the physical data has to be transferred via a formula or a translation table.
- **Signal Limits:**
  Defines the range of the physical signal which can be correctly handled

3 Hardware description

Chapter 2 introduced the basic building blocks that are necessary to specify hardware elements. But to specify a complete ECU several specialisations of HW Element, HW Container, HW Port and HW Connection have to be defined.
The following sections deal with the special elements that are necessary to specify a partly or complete engineered ECU with the ECU Resource Template.
Chapter 3.4 will describe the special HW Element Container that constitutes the ECU as a whole.

3.1 Chapter 3.6 characterises the most important features of the

Memory. There is no distinguishing between embedded memory and external from the microcontroller.

3.2 Chapter 3.8 describes the

Processing Unit (PU) in detail to give a deeper understanding of the architecture and structure of the processing core. Some general information, the basic programming model, information about performance issues and features from peripheral devices are illustrated.

3.3 Chapter 3.10 describes the

Peripherals. The focus is set to the components that are encapsulated by the MCAL.
Chapter 3.12 characterise the ECU Electronics that connects a peripheral electrically with e.g. a sensor/actuator. Here the mapping between the Technical Signal and the Electrical Signal can be established (in case of a simple sensor/actuator). Special HW Elements with no connection to the outer world (e.g. Oscillators) are described here, too.
Chapter 3.13 describes the Sensors and Actuators. They are the source and target of physical events
All subsections specify the mentioned HW Elements and according specialised HW Ports and HW Connections if appropriate.
For the description of the element the automatically generated class-tables are used. In these tables you will find information where to find the element in the Meta-Model, what the inheritance relationships are and details about each attribute. If the description fields of the class-tables do not provide enough space for the explanations additional text is provided below the class-tables.

3.4

3.5 ECU

The ECU Element is the top-most container for an ECU resource description. All contained HW Elements like memory, processing units, peripherals etc. belong to exactly one ECU.
If sensors and actuators are located inside the ECU they will also be described within the ECU description. If however the elements are located physically outside the ECU they should also be described in a HW Container that holds the ECU and the external elements.

3.5.1 Detailed ECU Resource Description

In the detailed ECU Resource description in addition to the before mentioned elements also the peripherals are described in a very detailed manner. This information is basically needed to generate and configure the Basic Software Modules covering the MCAL and the ECU Abstraction in a detailed way.
Not only information about the peripheral hardware elements and their attributes is provided, but also information about how the peripherals and ECU Electronics is interconnected. This information provides details about the net-list structure of the ECU. But also the connection of the sensors and actuators to the ECU and the peripherals is described.
Due to the very detailed description means of the peripherals and its connections a more accurate estimation of the timely behaviour can be provided for each sensor and actuator from the Basic Software Module and hardware point of view.

3.5.2 Minimal ECU Resource Description

The consideration of the most important ECU hardware elements from the software point of view the list can probably be limited to only a few elements:

- Processing Unit
- Memory
- Sensors / Actuators
- Communication Peripherals
- Reference to the ECU Abstraction (SW-Component Template)

In this approach only the processing unit and the memory are described in detail. This information is mandatory for the mapping of AUTOSAR SW-Components onto ECUs. The purpose of the hardware description is then the verification that enough processing power and memory are provided by each ECU.
Additionally the HW Connection of the sensors and actuators to the ECUs is described to be able to choose the appropriate Sensor/Actuator SW-Components.
The most crucial simplification is made due to the available Peripherals of the ECU. The Peripherals are not described in detail, but only the software-relevant view is provided.

The software view on the hardware is the ECU Abstraction. The interface of the ECU Abstraction is described with the means provided by the AUTOSAR Software Component Template and only a reference to that description is placed in the ECU Resource Template.
Since the ECU manufacturer will probably develop the whole ECU hardware he will also be in the position to deliver the software (ECU Abstration and SPAL) for accessing it.
For the communication peripherals however a more detailed description is needed to allow the highest flexibility for the usage of the ECU in the mapping process.

3.5.2.1 Structure of the ECU Element

In Figure 3-1 the structure of the ECU Element is shown. The ECU provides several HW Ports to describe the outside interface. And since the ECU is a HW Element Container it can contain other HW Elements to describe hierarchical structures. Typically the ECU will contain descriptions for the Processing Unit, the Memory Segments and the Peripherals.

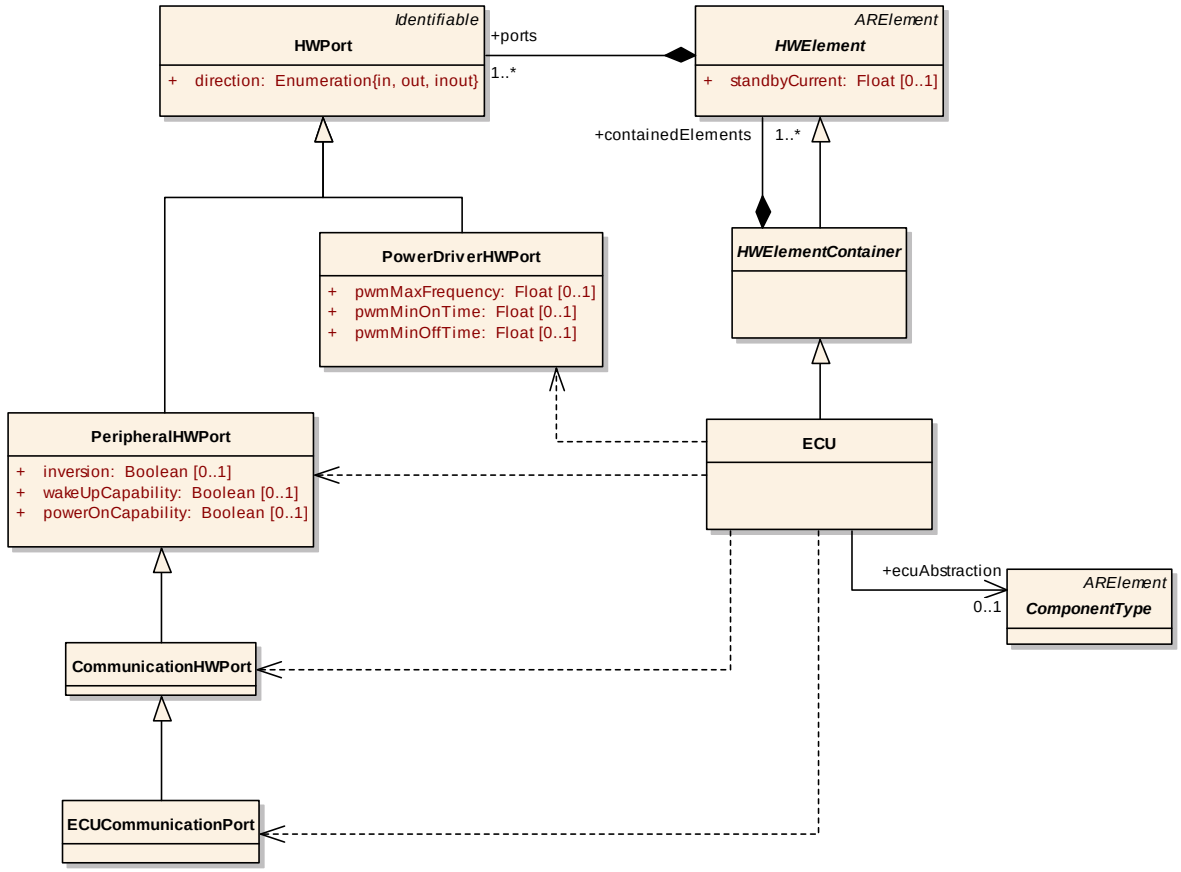

Figure 3-1: ECU Resource Overview

Please note that the reference to communication ports as sketched in Figure 3-1 implies some characteristics that need to be carefully observed. Please consult chapter 3.5.4.1 for more details.

3.5.3 ECU HW Element

An ECU HW Element is a specialisation of a HW Element Container. Besides the capability of aggregating other HW Elements it has got several other attributes to characterise this special type of ECU.

Class	**ECU**			
Package	AUTOSAR Templates::ECUResourceTemplate			
Class Description	The ECU provides information about an ECU and its internal hardware elements. The described hardware elements are related to some extent to basic software configuration and the AUTOSAR generation process.			
Base Class(es)	HWElementContainer			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
ecuAbstraction	ComponentType	0..1	reference	Reference to the software component describing the ECU Abstraction software module for this ECU.

Table 3-1: ECU

3.5.4 ECU HW Port

The type of HW Port of an ECU depends more ore less on the peripherals and the ECU abstraction it provides (see also Figure 3-1). So an ECU can have ECU Communication Ports, Power Driver HW Ports, Peripheral HW Ports, Communication HW Ports and Power Supply HW Ports.
Each HW Element is able to have Power Supply HW Ports – an ECU however must have at least one Power Supply HW Port.

3.5.4.1 ECU Communication Port

The ECU Communication Port is used for referencing from the System-Constraint Template to the according communication peripheral. This is shown in Figure 3-2. The ECU Communication Port is a special kind of Communication HW Port.
The rule for the Communication HW Ports is that if the Communication Port will be used for inter ECU communication (effecting System Constraint Template) it is described in the ECU Communication Port section, but if the port is used to access some local sensors or actuators by means of a local communication bus (e.g. private LIN or SPI) it will be described in the Peripheral HW Port section of the ECU description.

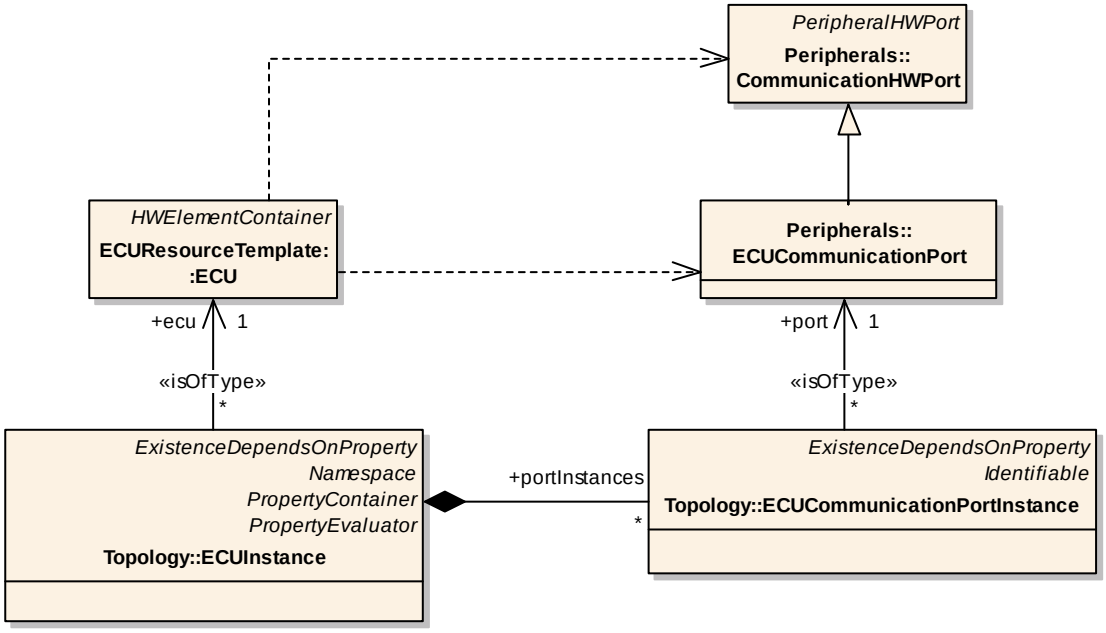

Figure 3-2: Coupling between the ECU Resource and the System Constraint Template

3.5.5 ECU HW Connection

The description for a HW Connection of an ECU (as described with ECU HW Port) depends heavily on the peripherals it provides. For describing a generic connection of ECUs the general HW Connection (see 2.3) can be used.

3.5.6 Application Domain

The Application Domain contains a list of possible uses. Each ECU is developed for a specific domain and controls one or more specific HW Elements of this domain. The following domains are possible:

- body/comfort
- power train
- chassis/driving dynamics
- safety
- telematics/multimedia
- man-machine-interface

This list is open and is not restricted to one entry per ECU.
Meta-Model Note: For accomplishing this goal the SW-COLLECTION structure of MSR is used. The important relations are presented in Figure 3-3.

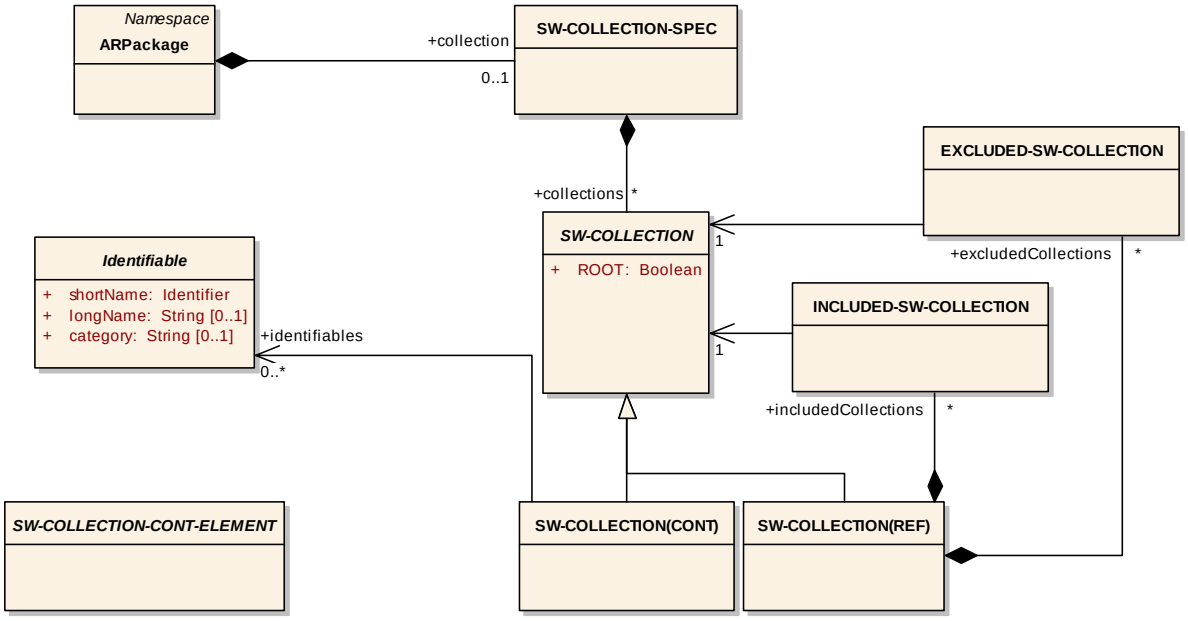

Figure 3-3: Representation of the Group relation

The approach to specify which domains an ECU can be used in is that for each domain there is a SW-COLLECTION and all ECUs supporting this domain are referenced from the SW-COLLECTION.

3.6

3.7 Memory

In this section the overall amount of available memory within an ECU is described. The memory can be included in a processing unit itself or can be located in the ECU and is available for one or more processing units.
The main topics have been selected in certain granularity in order to achieve a common understanding and general validity.
Within the memory all available data information and program code are stored during operation of the ECU. Data information and program code may have a different life cycle within the ECU. Some parts are stored permanently, others are only generated and valid for a short time during operation. Memory generically covers different types of technologies and storage forms. A detailed description of important attributes follows.
This chapter describes memories as used in modern embedded computing systems. Older or very uncommon technologies are not covered.
All definitions below are based on a digital representation of information.
All descriptions in the next are referenced to memory segments instead of a dedicated single chip, as a chip can contain different types of memory or the same type of memory in a different arrangement. Different type of memory is the most general case where memory is implemented in a microcontroller. Different arrangement is e.g. the case where a flash memory device is arranged in blocks with different size.

Type of memory and the access type of memory can restrict the usage of memory within a software component.
For memory description we use elements to describe the behaviour and not the name of the implementation of the memories like RAM, ROM, Flash, etc.

The Memory HW Element is described in the subchapters of this section.

3.7.1 HW Elements for memory

There is at least one Memory HW Element in a flash or an EEPROM memory. If an ECU memory contains more than one segment must each segment be defined in the format described in Table 3-2 and Table 3-4. To have the total number of segments in a specific type of memory the number of tables must be counted.

3.7.1.1 Volatile vs. Non-volatile Memory

The kind of the Memory HW Element is defined through the usage of either the class Provided Memory Segment for volatile memory or Provided NV Memory Segment for non-volatile memory:

- **Volatile:**
  Volatile memory is capable of storing data and program code only during the operation of the ECU. In the non-operational mode all data stored in volatile memory is lost and not valid anymore. In most cases RAM is the form of a volatile memory. If the power supply is turned off, all data stored in RAM get invalid, corrupted or completely lost (see Table 3-2).

- **Non-volatile:**
  Non-volatile memory is capable of storing data and program code during the operational and non-operational mode of the ECU. The data stays valid during the non-operational mode and is available if the ECU is set to operational mode again.
  ROM, Flash, EEPROM and any kind of disk drives are implementations of non-volatile memory.
  The implementation of non-volatile memory differs widely in performance and data-retention. The most important characteristics are endurance and data retention (see Table 3-4).

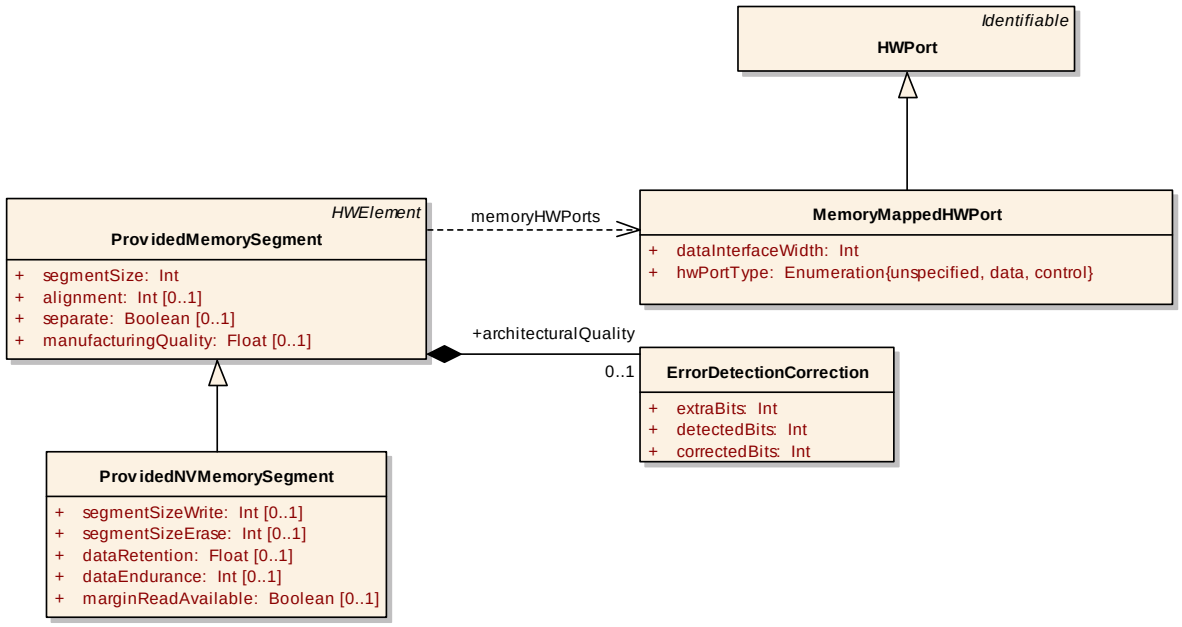

Figure 3-4: Provided Memory Segment

Class	**ProvidedMemorySegment**			
Package	AUTOSAR Templates::ECUResourceTemplate::Memory			
Class Description	Describes one volatile memory segment.			
Base Class(es)	HWElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
alignment	Int	0..1	aggregation	Some memory architectures are organised for bigger natural data size than accessed by the PU. In this case memory might not be used completely and some memory locations are left blank due to misalignment. The overall memory size is not used by code or data. Misalignment may be recovered by special mechanisms provided by hardware or software. In this case the access time will be increased. Certain PU architectures do not allow misalignment in code or data generally. Unit: Byte
architecturalQuality	ErrorDetectionCorrection	0..1	aggregation	The error detection and error correction facilities provided by the memory segment itself.
manufacturingQuality	Float	0..1	aggregation	As memory holds the executable program as well as the according data, memory needs a level of quality and reliability according to the purpose of the ECU. In most applications today memory is the biggest homogeneous hardware block; therefore quality and reliability of memory have the biggest impact on ECU performance and availability. With the introduction of non-volatile memory and the use of huge DRAM memories the aspect of quality and reliability gains more importance. Despite all Zero-defect programs of the semiconductor suppliers a defect rate in the range of 1 to 5 ppm must be considered as given even for critical applications. Separate quality data for Endurance and Data retention can be available also. The biggest value for ppm rates is used as a quality factor. Unit: ppm

Class	**ProvidedMemorySegment**			
segmentSize	Int	1	aggregation	A value that determines the amount of bytes within the memory segment. An ECU for AUTOSAR always contains at least one memory segment. Within a single ECU different memory types and access types can be used simultaneously. An ECU may consist of different parts that carry one or more memories. Segment size is defined in the terms of Bytes. The overall amount of memory is the sum of all available memory segments. For different types of memory only the sum of the same type gives a useful overall sum. Unit: Byte
separate	Boolean	0..1	aggregation	Separate memories are required for safety-relevant functions and redundancy requirements.

Table 3-2: Provided volatile memory segment

Class	**ErrorDetectionCorrection**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	Provides information on what extra bits are used for error detection and correction.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
correctedBits	Int	1	aggregation	How many error bits can be corrected. Unit: Bit
detectedBits	Int	1	aggregation	How many bit errors can be reconginsed. Unit: Bit
extraBits	Int	1	aggregation	How many extra bits are used for ensuring error detection and correction. Unit: Bit

Table 3-3: Error Detection and Correction

Class	**ProvidedNVMemorySegment**			
Package	AUTOSAR Templates::ECUResourceTemplate::Memory			
Class Description	Describes one non-volatile memory segment.			
Base Class(es)	ProvidedMemorySegment			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataEndurance	Int	0..1	aggregation	Program and erase cycles are a stress to the memory cell, therefore endurance is characterised by the number of erase/program cycles the memory can survive exceeding the prescribed failure rate. Unit: No of cycles.
dataRetention	Float	0..1	aggregation	Data retention time is the time between programming a sample of non-volatile memory and the observation of a prescribed failure rate when verifying the programmed pattern. The time is dependent on the chosen technology for the integrated memory cell. Unit: Years
marginReadAvailable	Boolean	0..1	aggregation	Additional to verify, all data are read back with a different set-up of the threshold of the sense amplifier of the programmed cell. With this method besides the logical correctness, the sufficient amount of electrical charges, which represents the stored data is checked. Furthermore the content of the memory and the correctness and the quality of the programming tools can be verified. Performing the margin read allow the prediction of the data retention. The feature of margin read is a characteristic of a memory, which is not available at all devices. Margin read is available for flash technology only.
segmentSizeErase	Int	0..1	aggregation	For an erasable memory it's usually the same size of the segment as for the segment size which can be erased at a time. If the size of an erasable segment is different shall the size of the erasable segment be entered in the table. Unit: Byte
segmentSizeWrite	Int	0..1	aggregation	Some memory types have not the same segment size as in the case when data is written to the memory. For example, a specific flash memory you must write 128 bytes once even if the segment size is 16 kBytes. Unit: Byte

Table 3-4: Provided non-volatile memory segment

3.7.1.2 Architecture

Most of the following characteristics are covered within interface, access type and access time.

- **Shared:**
  Memory that is used from more than one PU concurrently. The memory resource is available to all PUs connected to that memory.
  - **Multi ported:**
    Memory that can be accessed by more subscribers at one time. Additional access control provides a control mechanism if a specific memory location is accessed by more than one write access simulta-neously.

The most common form is dual-ported RAM for the massive and fast exchange of information between two independent PUs in a single microprocessor.

Note 1: Shared memory has an effect on execution time, memory size calculation, synchronisation and communication.
Note 2: The software concept does not allow shared memory (e.g. multiple connections) on a single HW Port. To establish a working solution multiple HW Ports have to be defined on the according Memory HW Element.
Meta-Model Note: This attribute can be calculated from the amounts of HW Ports defined for a Memory HW Element. This means if more than one HW Port is defined, the Memory HW Element is shared.

3.7.1.3 Data retention

Data retention time is the time between programming a sample of non-volatile memory and the observation of a prescribed failure rate when verifying the programmed pattern. The time is dependent on the chosen technology for the integrated memory cell.
Temperature has influence on memory characteristics and must be defined for memory devices if memory is not implemented within a µC and the definition is made there.
For standalone memory the specifications below are mandatory (definition see 2.1.1).

- **Operation temperature**
- **Storage temperature**

For calculating the data retention and data endurance an application specific temperature profile can be applied.
The data retention time is mostly evaluated at accelerated tests at elevated temperature. The time acceleration factor is given by the Arrhenius formula:

$K = \exp [Ea/k(1/Top – 1/Ttest)]$

E_a = Activation energy, k = Boltzmann constant, T_{op} = Operating temperature, T_{test} = Elevated temperature at test.

3.7.1.4 Architectural Quality

Special hardware implementations can help to improve the overall quality of an ECU. ECC and Parity are usual technologies.
The choices of the storage media and the according quality and reliability requirements have to match. For example to store some graphical data, with a lot redundancy or a high update rate in a single cell-multi bit Flash is a good choice, whereas storing important system parameter without other precautions is clearly not recommended.
Quality is not only subject to the semiconductor device itself, but must be also reflected in the whole development and manufacturing process. Even in the implementation of software the quality and reliability aspect are subject to be considered. Coding schemes like EXCESS-3 codes, where a valid digital number is always a sequence of different bit-levels, and all bits to a single level are illegal digital numbers may help to increase system reliability.

- **ECC:**
  ECC Error Correction uses extra data besides the normal data in order to detect and correct errors. ECC keeps some structural information of the data and the representation of the data in a different code scheme. With this a

limited set of errors can be detected and even corrected. For detection of errors and correction of one error over 32 Bit array 6 extra bits are necessary at least.

- **Parity:**
  Parity is a simple form of a plausibility check of data and therefore for error detection. Parity simple stores in one single bit the information about the majority of logical levels that represent a digital value. An error correction is not possible.

3.7.2 Examples of memory

This section is not an element of the Memory HW Element but there are just examples of memory implementations.

- **ROM (Read Only Memory):**
  The program and constant data are fixed onto the chip during the manufacturing process. This data cannot be modified.
- **PROM (Programmable Read Only Memory):**
  data can only be written once. Programming is not part of the operational mode. Used for program code and constant data.
- **EPROM (Erasable Programmable Read Only Memory):**
  Data can be erased completely by UV light and then written one time until next erasure. Erasure and programming is not part of the operational mode. Used for program code and constant data.
- **Flash:**
  Electrically Erasable Memory. Data is stored in form of electrical charges via tunnelling effects in floating gate. Flash cells are erased and programmed by different effects. The programming is generally a very fast process, whereas erasure takes a certain amount of time. A single Flash cell can be used to store more than one bit per cell. But multiple bits per cell reduce the reliability of the information and are used for information with lower safety and quality requirement.
  Flash cells can only be erased in defined array sizes. The capability to erase single flash cells is only the exception. Today array sizes from 128 Bytes to 32 KB are used. Flash cells can be written in byte or word size depending on the implementation. Usually Flash is used to store program code and constant data.
  Characteristic of a Flash is an array size for erasure and a array size for write access. The size for write access might be smaller than the size for erasure.
- **EEPROM (Electrical Erasable Memory):**
  The data is stored as the presence of electrical charges via tunnelling effects in floating gates. The erasure and programming use similar physical effects. Erasure and programming takes the same amount of time approximately. Each EEPROM cell represents only a single bit. Depending on the internal architecture EEPROM can be erased and programmed in bit, 4-bit, byte or word size. Usually EEPROM is used to store data that must be kept after a power-down sequence (e.g. serial number).
  Characteristic of EEPROM is the size of data to be accessed.

- **RAM (Random access memory):**
  Data is stored in electrical form either in the switching state of a Flip-flop or in the charge of a capacitor. RAM is used for temporary program code and variables data. This is a volatile memory.
- **SRAM (Static Random Access Memory):**
  Data is stored in the switching state of a Flip-flop. Data can be accessed at any time and very fast. The data is valid as long as the power is supplied. This is a volatile memory.
- **DRAM (Dynamic Random Access Memory):**
  Data is stored in a capacitor. DRAM can be implemented cheaply and in a high density. Due to the leakage of the capacitor the DRAM needs a refresh cycle in a defined time frame. Therefore the access to data is sometimes delayed for a short time. Data is valid as long as the refresh cycles are performed continuously and the power is available. This is a volatile memory.
- **Cache:**
  Usually implemented as fast SRAM. Caches are used to increase performance of memory implementations. Most frequently used program code and data are stored in the fast, size-limited cache, closely coupled with the PU, whereas the major part of the program code and data are stored in cheaper and slower memory types like DRAM or Flash. The increase of performance is influenced by the usability of the data in the cache. The loading of caches is done automatically with different kinds of replace strategies (e.g. FIFO, LIFO, etc.). Cache is not usable for general memory. This is a volatile memory.
- **Shadowed NV-RAM:**
  Such a Memory HW Element has a good endurance and access time. It's a special form of non-volatile memory which uses extra mechanisms to increase the performance. Shadowed NV-RAM uses non-volatile and volatile memory in parallel. Such mechanism incorporates both volatile and non-volatile memory in one application. To achieve better performance the volatile memory is used in the operational mode. The non-volatile memory is used to keep the data in the non-operational mode. A cyclic update or at least an update before power-down mirrors the data of volatile memory into the non-volatile memory. During power up the volatile memory is loaded with the content of the non-volatile memory.

3.7.3 Memory Mapped HW Port

This special type of HW Port is used if the information could be mapped to memory. The following list of attributes must be considered for this HW Port type.

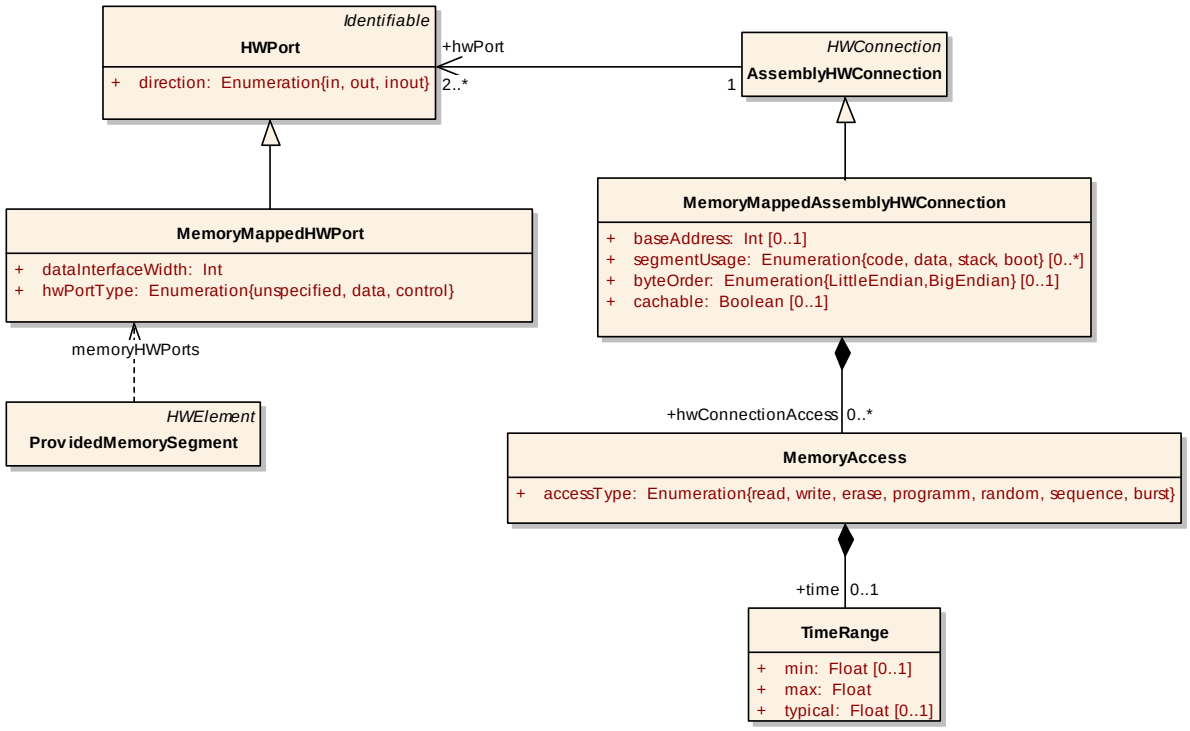

Figure 3-5: Memory Mapped HW Port

Class	**MemoryMappedHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::Memory			
Class Description				
Base Class(es)	HWPort			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
dataInterfaceWidth	Int	1	aggregation	The element defines the number of bits that can be transferred from the PU to the memory device or vice versa. The width of data bus are important characteristics. All technical characteristics of the memory are covered by the access time. Unit: Bit
hwPortType	Enumeration{unspecified, data, control}	1	aggregation	The HW Port Type of a HW Port can be one of the following attributes - unspecified: The attribute is used as a placeholder in the iterative process or if a detailed specification of the HW Port is not required, for example a low-level driver supplied by the semiconductor manufacturer. - Data: Data is the part of the information processed later in the SW Application. - Control: Control is needed by the hardware and the basic SW Routines to manage the overall operation. The attributes Data and Control are defined in the HIS Specification.

Table 3-5: Memory Mapped HW Port

Class	**MemoryAccess**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	Describes the different possible kinds of access to the memory and the time these access takes.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
accessType	Enumeration{read, write, erase, programm, random, sequence, burst}	1	aggregation	The element access type describes the different possible kinds of access to the memory. - Read Access: Read Access is available to all memory by default. The data is copied from the memory to another location. - Write Access: Possible only with RAM devices. Data is copied to the memory. - Erase Access: Data stored in specific memory areas is erased and the memory is transferred to a defined initial-state. The erase access is a complex process. The erase access is in many cases prerequisite of a program access. Used on EPROM, EEPROM and Flash. - Program Access: The transfer of data to PROM, EPROM, EEPROM or Flash. In the difference to write access the program access takes more time and is a more complex process to transfer the data. Data is copied to the memory. - Random Access: Data access is not sequenced in memory. Time restrictions are valid in order to address the appropriate memory location and getting the right to access this location with single access. The access time of single data is the important characteristic. - Sequenced Access: The wanted data can only be achieved by reading a defined sequence of data in advance. Typical examples are tapes, certain EEPROM or hardware implemented stacks. - Burst-Mode: A group of data is accessed. The data is stored in continuous addresses and is copied in continuous set of data transfer without an interruption. Access time for the overall data transfer is minimised. For the set-up of a burst mode time restrictions are valid in order to address the appropriate memory location and getting the right to access this location in burst-mode. Set-up time for burst-mode, transfer time of first byte, transfer time of last byte, transfer time for intermediate bytes of a burst and the number of bytes per burst-access are the characteristics of burst-mode access. Burst-mode is used to transfer great amount of data e.g. from a memory with slow access times to a memory with fast access time. Note: The access is denied during erasure and programming of Flash and EEPROM as some technologies do not allow a simultaneous read access to that specific memory area. Also during the simultaneous access of different subscribers on the same memory, array or byte the access is either denied or at least delayed. For example if the CPU and the DMA write to the same memory block.

Class	**MemoryAccess**			
time	TimeRange	0..1	aggregation	Access time is the timing characteristic for each access type. Access time at the Memory Mapped HW Port is only theoretical. The actual Access time is given in the Memory Mapped HW Connection.

Table 3-6: Memory Access

3.7.3.1 Access time

Access time is the timing characteristic for each access type. Access time at the Memory Mapped HW Port is only theoretical. The actual Access time is given in the Memory Mapped HW Connection.
A typical value or an average value of access time is necessary and used for ECU and system design,
A peak value of access time is used to represent the burst mode. A peak value may only be achieved for a limited time period.
A maximal value of access time is necessary and used for timing critical calculation i.e. for safety critical worst-case calculation.
Different access times may be valid for accessing code or data in ECUs using Harvard architectures. Separate access times are then defined for accessing code or accessing data.
Delays during programming and erasure are also handled as a separate access time.
Example:

	Time range / Byte	**Memory Device**
Read Access	500…5ns	All
Write Access	500…5ns	RAM
Erase Access	10s…10ms	EEPROM, Flash
Program Access	10s…10µs	EPROM, EEPROM, Flash

Table 3-7: Examples for access time

Note: The processing of ECC and Parity Data (see 3.7.2) slows down the Access time by some small amounts. The attributes random access, sequenced access, burst-mode and access denied/access delayed have to be covered by the sum of access time.

3.7.4 Memory Mapped HW Connections

This types of HW Connections are used to connect Memory HW Elements to e.g. PUs. They are used connecting different HW Elements (other than memory) if the information could be mapped to memory.

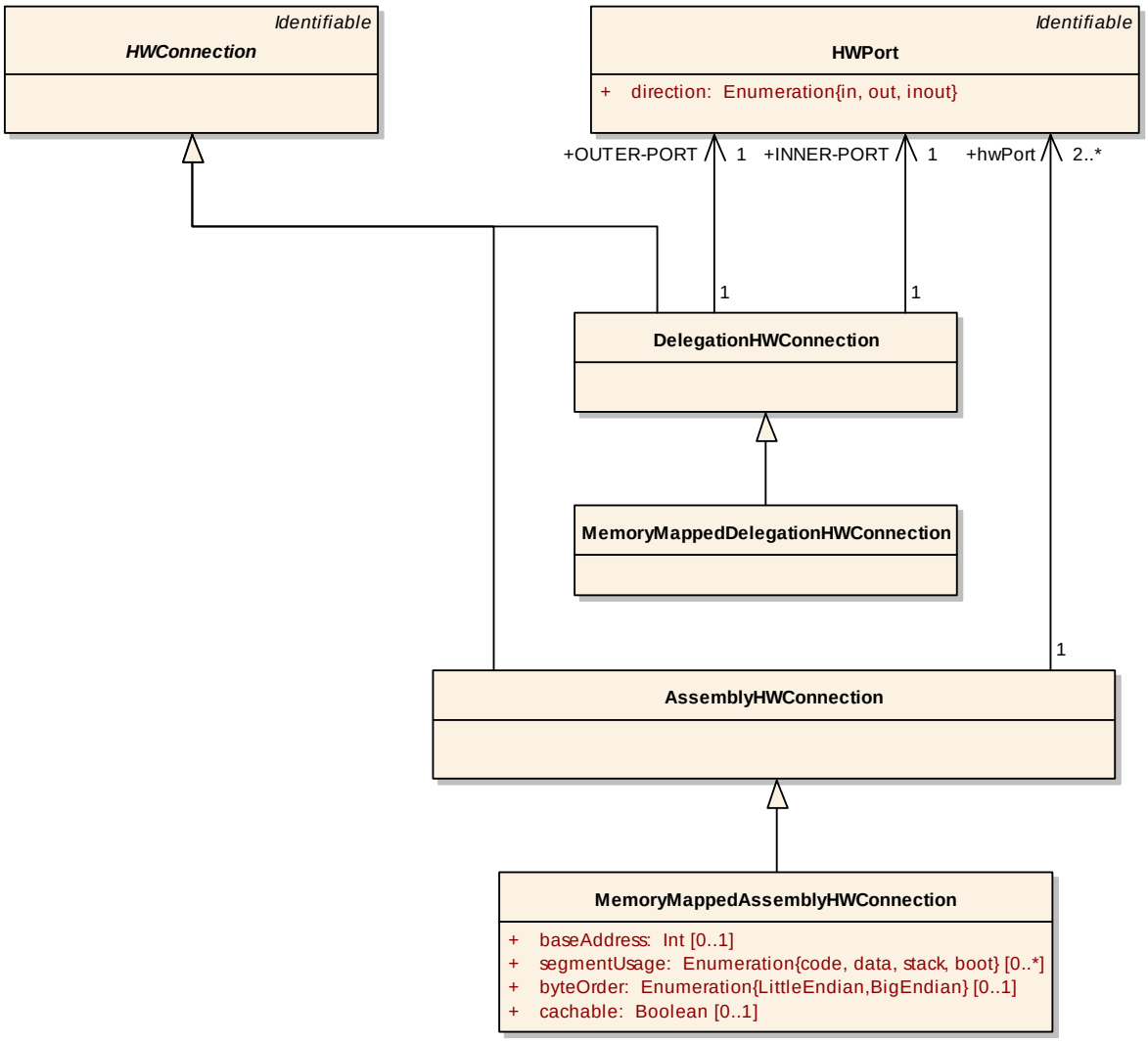

Figure 3-6: Memory Mapped HW Connections

Two different cases must be considered for the establishment of memory-mapped HW Connections:

- Connect HW Elements within a specific HW Container. This leads to the definition of the Memory Mapped Assembly HW Connection. A typical use case for this variant is the connection of a PU with (internal) memory inside a microcontroller.
- Connect a HW Element within a HW Container with the outside of the HW Container. This case is represented for example by the definition of a microcontroller that might have a Memory Mapped HW Port on the outside connected to a Memory Mapped HW Port of the contained PU within the microcontroller. Please note that the Memory Mapped Delegation HW Connection does not provide any attributes comparable to the Memory Mapped Assembly HW Connection.
-

The following list of attributes must be considered for the HW Connection types.

Class	**MemoryMappedAssemblyHWConnection**			
Package	AUTOSAR Templates::ECUResourceTemplate::Memory			
Class Description	This connection is used to connect Memory HWPort on the same hierarchical level (inside one HW Container).			
Base Class(es)	AssemblyHWConnection			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
baseAddress	Int	0..1	aggregation	Defines the base address of the memory within the ECU template. The same base address might be defined multiple times according to the access type and the architecture. A base address is uniquely bound to a defined memory segment.
byteOrder	Enumeration{LittleEndian, BigEndian}	0..1	aggregation	When a PU stores data in a memory the ordering of the bytes can be done in two different ways. This storing scheme is normally fixed in the PU when more than one byte is stored in the memory. When the least significant byte is stored at the lowest address this architecture is called little endian and when the most significant byte is stored at the lowest address this is called big endian. Some CPU architectures like ARM and PowerPC can be configured to handle both little and big endian. This is called bi-endian. ByteOrder is very important when you communicate between different PUs or ECUs.
cachable	Boolean	0..1	aggregation	A memory segment may be cacheable. This has to be defined in the description of the segment connection.
hwConnection Access	MemoryAccess	0..*	aggregation	Access time is the timing characteristic for each access type. Access time at the Memory Mapped HW Port is only theoretical. The actual Access time is given in the Memory Mapped HW Connection.

Class	**MemoryMappedAssemblyHWConnection**			
segmentUsage	Enumeration{code, data, stack, boot}	0..*	aggregation	The segment usage is defined in the linker memory map. Usually it is fixed within an operation mode but different operations modes have different memory usage. The element segment usage contains the following attributes. - Data: The attribute data is used for all kind of data: constant, variable, calibration, system initialisation, system configuration, etc. - Code: The attribute code is used for the program code itself. - Stack: Memory usable for a system or software stack is in some cases only available on limited number of segments within a PU. - Boot: The boot memory is used to store program and data for the start up or test of an ECU. Within the boot program the basic ECU initialisation is performed. Furthermore some system maintenance programs, like the set-up for flash programming or loading of data via communication interface, can be part of the boot program. Boot memory is in general not a subject of AUTOSAR SW. -- Boot sector: Boot sector is the location in memory, preferable Flash or ROM, where the boot program is stored. The location of the boot sector might be dependant on the PU architecture as reset and interrupt vectors must be accessible during boot. The boot sector should be separated from other user accessible Flash or implemented as ROM. In Flash the boot sector should have the appropriate array size, some few kilobyte and provide protection mechanism against unintended erasure or system manipulation. A system may have different boot sectors, one accessible only for testing the PU during the manufacturing and configuration of the PU, in factory test mode or some initial program mode, and a user boot mode. -- Mapping of boot sector: Mapping of the boot sector is useful to enhance the boot sequence and the release of the used resources during the boot phase at the later user mode, e.g. the use of reset vector and interrupt vector can be improved.

Table 3-8: Memory Mapped Assembly HW Connection

Class	**MemoryMappedDelegationHWConnection**			
Package	AUTOSAR Templates::ECUResourceTemplate::Memory			
Class Description	This connection is used to delegate a Memory HW Port to the outside of a HW Container. Typically this will used internally of a micro-controller.			
Base Class(es)	DelegationHWConnection			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 3-9: Memory Mapped Delegation HW Connection

3.7.5 Provided Memory overall model

Since in the Meta-Model there are a lot of inheritance relationships it is not immediately visible how the memory describing classes are interrelated. In Figure 3-5 these classes are shown with their relationships.
The Provided Memory Segment is a specialisation of a HW Element so it can be contained in a HW Container or an ECU. The Provided Memory Segment contains the Memory Mapped HW Ports.
The ECU and the HW Container both inherit from HW Element Container and through the HW Element Container the HW Connections are aggregated. So also the ECU and each HW Container can contain HW Connections and especially the Memory Mapped HW Connections. The HW Connections connect HW Ports, which the Memory Mapped HW Port is a specialisation of. So with these relationships it is possible to describe all of the provided memory segments within an ECU.

3.7.6 Constraints/Services

Following sections are not a part of the ECU Resource Template because they are part of the ECU configuration and describe the usage of memory.

- **Stack:**
  The stack buffer or often simply the stack is a reserved part of RAM. Stack keeps the return addresses, the system status flags in normal PU architectures for interrupt-routines and sub-routine calls. Characteristic for the stack is the indexed access via stack pointer and a set of special by hardware supported op-codes. Stack is also used for passing data between software routines. Passing data via the stack is commonly done for sub-routine calls. Stack also is used very often used to keep local variables. Stack is used for context saving in interrupts and sub-routine calls.
  A stack is one of the very heavily used and important memory usage, fast access is necessary and care and precautions have to be taken in order to ensure system integrity.
  Sometimes there are several stacks supported by the PU architecture and op-codes. Most known are one user stack for normal operation and a supervisor stack for system maintenance and administration or error recovery. As an exception, there are PU architectures, which do not need a stack or have a hardware support for a stack.
  As characteristics of the stack, stack size, the address of the stack pointer, the address range of the stack pointer, the number of stack pointers, alignment requirement, the lower and upper stack boundaries are important.

- **Heap:**
  Heap is a buffer, which is implemented similar to the stack in RAM. In opposite

to the stack, a heap is fully controlled by software. Heap is used to implement a form of memory management. It is very common to use a heap only for temporary storage of data and program code.
As characteristics of a heap, heap size, the addresses of the used heap pointer, the address range of the heap pointer, alignment requirement, the lower and upper heap boundary are important.

- **Dynamic Memory Allocation:**
  In embedded real time systems dynamic memory allocation is not recommended as no reliable and predictable system behaviour can be achieved and guaranteed. The real time requirement prevents the necessary continuing memory resource management.
  A most static memory usage is required in order to guarantee the system integrity and predictability.

3.7.7 Mass-Storage Devices

Data stored on CD-ROM, DVD, as well as memory cards are covered also by the memory template. The special hardware for according drives has to be handled in the peripheral section or by special complex device drivers. The presence of the mentioned data carrier is not subject of this template and must be covered in the higher application layer. The usage of the mass-storage device is also subject of a memory management device.

3.8

3.9 Processing Unit (PU)

A processing unit is a device that can execute instructions like arithmetic and logical operations and data moving operations. Characteristic parts of a processing unit are internal data registers and an instruction pointer.
The description for a HW Connection to/from a PU is totally covered by the Memory Mapped HW Connection (see chapter 3.7.4).
The HW Port of a Processing Unit is described with the Memory Mapped HW Port (see chapter 3.7.3), the Power Supply HW Port (see chapter 3.12.8) and the Clock (see chapter 3.12.3).

3.9.1 Processing Unit HW Element

In this section the main characteristics of a PU and its connections to memory, peripherals and power supply are described.

Class	**ProcessingUnit**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	A ProcessingUnit describes the plain processor core. The identifier provides the actual type name of the processing unit.			
Base Class(es)	HWElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
addressingModes	Enumeration{direct, indirect, indexed}	0..*	aggregation	The addressing modes describe the options to access the data from the memory. There are several addressing modes possible. Typical attributes are: - Direct - Indirect - Absolute - Base + offset (relative) - Near / far - Pre/Post-Increment/decrement The addressing modes take important influence to the performance index of a PU, but are not main topic of the software development, as far as high level programming languages are used, due to the fact that these are covered by compiler settings and compiler strategies. The main advantage is available at the level of assembler programming or compiler optimization. The Enumeration is open.
architectureKind	Enumeration {HARVARD, vNEUMANN}	0..1	aggregation	There are two kinds of architectures: - Harvard: A computer architecture in which program instructions are stored in different memory from data. Each type of memory is accessed via a separate bus, allowing instructions and data to be fetched in parallel. - Von Neumann: A computer architecture in which each successive operation can read or write any memory location, independent of the location accessed by the previous operation.
architectureName	String	0..1	aggregation	The name of the architecture (e.g. ARM, MIPS, PowerPC).

Class	**ProcessingUnit**			
architectureType	Enumeration{RISC, CISC, SIMD, MIMD}	0..1	aggregation	The architecture type take influence to the performance index of a PU, but are not necessarily a main topic of software development, as some of the aspects are transparent to the user as the are handled with in the compiler. The last two types are slightly different as they require a detailed planning during system design, how the data must be prepared in order to use the features of an SIMD or MIMD very efficiently. Application like video processing use in general this features by their data format, as every pixel information is formed by the three basic colours and intensity, thus each pixel is represented by four different values, which have to be processed simultaneously.
bootTime	BootTime	0..1	aggregation	Provide information about the boot time of the PU.

Class	**ProcessingUnit**			
dmaChannelCount	Int	0..1	aggregation	Direct Memory Access is a mechanism for off-loading a PU and having transfer directly between memory and peripherals without involving the PU. In most common PU the DMA mechanism is implemented with a specialised cell (DMA unit) that transfers data between a peripheral device and memory, independent of the processor. The DMA controller becomes the bus master and directs the reads or writes between itself and memory. A typical DMA transfer can be described in three steps: 1. The PU sets up the DMA transfer by supplying the identity of the periphery module, the operation to perform on that device, the memory address that is the source or the destination of the data to be transferred, and the number of bytes to transfer. 2. The DMA starts the operation on the periphery device and arbitrates for the bus. When the data is available, from the periphery device or memory, it transfers the data. The DMA unit supplies the memory address and initiates the next transfer. With this technique, a DMA unit can complete an entire transfer, which may be Kbytes of data, without bothering the PU. Many DMA units contain some memory to allow them to deal flexibly with delays either in transfer or those incurred while waiting to become bus master. 3. Once the DMA transfer is complete, the DMA unit interrupts the processor, which can then determine by interrogating the DMA unit or examining memory whether the entire operation completed successfully. During an active DMA transfer the PU cannot access the involved resources concurrently.
implementationTechnology	String	0..1	aggregation	Name of the technology used to implement this PU.

Class	**ProcessingUnit**			
maxInterruptNestingLevels	Int	0..1	aggregation	Defines how many interrupts are allowed to be activated at one time. This entry is mainly defined by the size of the according stack as the data saving mechanism during interrupts and the data passing mechanism during function calls use the stack.
mpu	MPU	0..1	aggregation	Describes the Memory Protection Unit, if available.
naturalDataSize	Int	1	aggregation	The natural data size prvides information which data manipulations are optimized on this PU. The natural data size is typically the Data Register size. The PU is optimised for this data size. Typical natural data size is one of following (in bits): 8, 16, 32, 64, other (12, 20, 80)
opcodeSize	String	0..1	aggregation	The attribute describes the typical length of the opcodes. In CISC machines the opcode size is variable due to the opcode itself. In RISC machines the opcode is by definition limited to a defined size. Type definition is used if the PU has more than one opcode set is incorporated. E.g. at the ARM architecture there is the ARM opcode set (size of 32 bit) and the Thumb opcode set (size of 16 bit). Code compression is a technology where the opcodes are stored in a compressed form and are automatically decompressed before execution.
puClock	Clock	0..1	aggregation	The clock source for the PU.
registerModel	Register	0..*	aggregation	Describes the register model of the PU. For each register type availabe one 'Regsiter' element is used to give the number available.

Class	**ProcessingUnit**			
specialOpcodes	Enumeration{ bitManipulation, MUL, DIV, MAC, Float}	0..*	aggregation	The following attributes are opcodes, which are not supported by all PUs. The support of this opcodes has an influence on the performance of the PU. - Bit-manipulation - Multiplication - Division - Multiply Accumulate - Floating-point Support In the future further opcodes may be possible. The special opcodes take important influence to the performance index of a PU, but are not main topic of the software development, as far as high level programming languages are used, due to the fact that these are covered by compiler settings, compiler strategies and software libraries where the direct usage or a SW emulation is available for almost any general use case. For some of these opcodes the main advantage is available only at the level of assembler programming or compiler optimization.
supportedDataTypes	SupportedDataType	0..*	aggregation	This element provides a list of all supported data types and its according data sizes of the described PU.

Table 3-10: Processing Unit

Class	**BootTime**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	Time information for ECU and PU startup.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
coldStartTime	Float	0..1	aggregation	Time it takes to come from a completely powered down state to the HWInit state in seconds.
resetTime	Float	0..1	aggregation	Time it takes from the reset initiation to the HWInit state in seconds.

Table 3-11: Boot Time[5]

[5] Please note: HWInit is also known as the Startup state defined by the ECU State Manager

Class	**MPU**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	A MPU provides means where the memory access within a defined program flow of a PU is controlled. In case of a failure an exception routine e.g. interrupt is activated. A MPU is implemented as an independent hardware block in a device separate to the processing unit. A MPU does not perform any data manipulation, only in case of an error a change in the processor control flow is forced. The number of individual checked memory ranges is limited to the registers available for specifying the memory ranges. One individual memory range is in the following referenced as MPU Channel. In most cases the MPU consists of a set of registers, which do range checking of the addresses asserted by the program on the address bus as well as the access type to the according address. The checking of the address ranges is also coupled to the address range of the program, which runs under the control of the MPU. The MPU also can be coupled with one of the different processing elements of a device like the PU itself, a DMA or a Coprocessor. In order to have a broad usage of a MPU a specification of filter values is often allowed, so very user specific selection methods are possible. Typical use case for a MPU: A 3rd party software routine is allowed only to access a defined memory-area. Any access outside the defined memory area will be denied and control will be transferred to the supervising operating system. A error handling routine will be started.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
channelCount	Int	1	aggregation	Defines the number of individual MPU channels are available.
protectedMemoryHWPorts	MemoryMappedHWPort	1..*	reference	Describes which Memory Mapped HW Ports are obserable by this MPU. The actual configuration will be done by SW.
raisedInterrupts	InterruptProduceHWPort	*	aggregation	Defines the interrupts this MPU can raise to indicate the violation of memory access.

Table 3-12: MPU

Class	**SupportedDataType**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	This element provides a list of all supported data types and its according data sizes of the described PU. Examples: Boolean (1, 8 bit), Bit String / Bit Field (x bit), Character (8, 16 bit), Signed / unsigned integer (8, 16, 24, 32, 48, 64 bit), Single / double float (56, 64 bit), Fixed point (24 bit), Fractional, Packed (two values handled in one register)			
Base Class(es)	Identifiable			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
supportedDataSize	Int	1..*	aggregation	The size of the data type. Unit: Bit

Table 3-13: Supported Data Type

Class	**Register**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	The user available registers are a part of the programming model that consists of: Memory organisation and segmentation, Data types, Registers, Instruction format, Operand selection, Interrupts and exceptions. Basically the register model contains the registers that are interest to the applications programmer; these registers can be grouped into three basic categories: - General registers: These general-purpose registers are used primarily to contain operands for arithmetic and logical operations. - Segment registers: These special-purpose registers permit software designers to choose either a flat or segmented model of memory organisation. These registers determine, at any given time, which segments of memory are currently addressable. - Status and instruction registers: These special-purpose registers are used to record and alter certain aspects of the processor state. The register model is only valuable data for low level programming, e.g assembler programming. The register model, along with the available OP codes takes influence on the performance of the PU. The most general impact of the register model to software is number of registers which have to be handled during interrupts or function calls.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
count	Int	1	aggregation	Describes how many registers are available of the specified type.
type	Enumeration{data, address, control, status, stackPointer, special}	1	aggregation	Describes the kind of register.

Table 3-14: Register

3.9.1.1 Architecture

3.9.1.1.1 Architecture Type

Type specifies the processor architecture. The most common examples are:

- **RISC:**
  A processor whose design is based on the rapid execution of a sequence of simple instructions rather than on the provision of a large variety of complex instructions.
- **CISC:**
  A processor where each instruction can perform several low-level operations such as memory access, arithmetic operations or address calculations.
- **Vector PU; SIMD PU:**
  Type of a PU, which can process different operands and instructions at the same clock cycle. This is achieved by combining and synchronising different PU to a cluster or by split up powerful PUs in smaller units.
  Vector PU is chosen according to the usage of this type of PUs to process vector data e.g. performing addition, multiplication on mathematical scalars and vectors.
  This form of operation is often used in processing of calibration curves or graphic data which are often represented in scalars according to there RGB representation.
  Most DSP have the capability of vector PUs, same as the TriCore, the

PowerPC architecture (PPC 555x) and the PC like Processors with MMX extension.

- **MIMD PU:**
  Multiple **I**nstruction **M**ultiple **D**ata (MIMD) machines have a number of processors that work asynchronously and independently. The different processors execute different instructions on different pieces of data. MIMD machines can be separated in shared memory or distributed memory categories. These classifications are based on how MIMD processors access their memory.

3.9.1.1.2 Architecture Name

These are examples of architecture names:

- **ARM:**
  A series of low-cost, 32-Bit RISC microprocessors cores for embedded control. It was the first commercial RISC. The ARM has a small instruction set, as do most RISC processors. Every instruction includes a four Bit code, which specifies a condition (of the processor status register), which must be satisfied for the instruction to be executed.
  Instructions are split into load and store, which access memory and arithmetic and logic instructions, which work on registers.
  The ARM cores have 27 registers of which 16 are accessible in any particular processor mode. R15 combines the program counter and processor status byte, the other registers are general purpose except that R14 holds the return address after a subroutine call and R13 is conventionally used as a stack pointer. There are four processor modes: user, interrupt, fast interrupt and supervisor.
  Most ARM processor cores offers a special Thumb extension, by identifying the critical subset of the ARM instruction set and encoding it into 16 bits, ARM has succeeded in reducing typical program size by 30-40%.
  All Thumb-aware processor cores combine the capability to execute both the 32 Bit ARM and the 16 Bit Thumb instruction sets. Careful design of the Thumb instructions allows them to be decompressed into full ARM instructions transparently during normal instruction decoding without any performance penalty.
  The ARM license includes the complete core and not a instruction set architecture like MIPS or PowerPC.

- **MIPS:**
  A project at Stanford University intended to simplify processor design by eliminating hardware interlocks between the five pipeline stages. This means that only single execution cycle instructions can access the thirty-two 32-bit general registers, so that the compiler can schedule them to avoid conflicts. This also means that LOAD/STORE and branch instructions have a one-cycle delay to account for.

- **PowerPC:**
  The PowerPC standard specifies a common instruction set architecture (ISA), allowing anyone to design and fabricate PowerPC processors, which will run the same code. The PowerPC architecture is based on the IBM POWER architecture, used in IBM's RS/6000 workstations.

3.9.1.1.3 Implementation/Technology

Some of these implementation technologies are customisable. Therefore a reference to the manufacturer of the raw HW element could be necessary (e.g. FPGA comes from Xilinx, but the customisation is done by a different supplier).

- **Microprocessor (µP):**
  A microprocessor is a processing unit without any peripherals and a significant amount of memory. The whole PC-world deals with µPs.
  A coprocessor which assists the main processor by performing certain special functions, usually much faster than the main processor, is a special µP. The coprocessor executes his instructions parallel and often independent from his assigned main processor.
  Examples are:
  - o TPU Time Processing Unit (Freescale)
  - o PCP Periphery Control Processor (Tricore)
  - o FPU Floating Point Unit
  - o XGATE Peripheral Co-Processor (Freescale)
  - o ACE advanced communication engine
  - o Crossbar multipath switch, allows parallel processing
- **Application specific integrated circuit (ASIC):**
  A chip that implements a very special functionality in hardware. An ASIC is a microchip designed for a special application, such as a particular kind of transmission protocol or a lambda-IC. You might contrast it with general integrated circuits, such as microcontroller.
  An ASIC can be manufactured for a special application or it can be custom manufactured for a particular customer application.
- **Field programmable gate array (FPGA):**
  A reconfigurable chip, which implements a digital function in hardware. The FPGA is an integrated circuit that contains many identical logic cells that can be viewed as standard components. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit.
  Field Programmable means that the FPGA's function is defined by a user's program rather than by the manufacturer of the device. A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the FPGA's function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either 'burned' in permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits.

- **Digital Signal Processor (DSP):**
  A digital signal processor is a processing unit that is specialised for limited functions to process signals. They are very efficient at this function.

3.9.1.2 Interrupt HW Ports

In some cases, the normal data and control flow of a processing unit must be suspended in order to respond to an asynchronous electronic event, also known as interrupt. The context switch caused by an interrupt is in most cases of a very limited duration. Only in exceptional cases it might happen that the context switch applies for a longer period of time.
The concept of interrupts is mainly implemented by means of hardware (although certain processor architectures permit even software to raise an interrupt).
Interrupts are usually handled by PU specific low level interrupt handlers. The counterpart of the interrupt concept in software, where the original interrupts are finally processed, is a software element like an event.
If an interrupt occurs, the processing unit may temporarily give control to an interrupt handler routine. The suspended process is resumed after the interrupt has been handled.

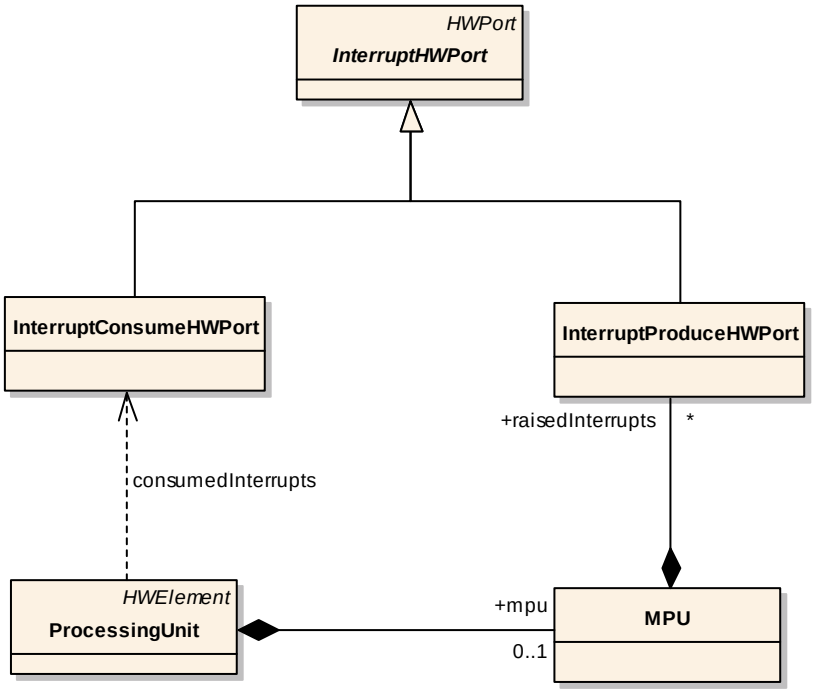

Figure 3-7: Interrupt HW Ports

The ECU Resource Template implements the interrupt concept by means of two meta-classes: InterruptConsumePort and InterruptProducePort (see Figure Figure 3-7 for more details).
As sketched by Figure 3-7, the processing unit is an example of a HW-Element that provides an InterruptConsumePort while an MPU is a typical example for a HW-Element that is capable of raising interrupt requests and consequently sports an InterruptProducePort.

Class	**InterruptConsumeHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	This port represents the drain of an interrupt request, i.e. the HWElement (in the majority of cases this will be a ProcessingUnit) to which this HWPort is attached is capable of accepting interrupt requests.			
Base Class(es)	InterruptHWPort			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
activationSource	String	0..1	aggregation	Defines which hardware resource activate the interrupt. Typically any peripheral HW Element can activate an interrupt. E.g. CAN, SCI, SPI, Timer, ADC. Even simple digital I/O can activate an interrupt. Some microC have in this context dedicatated pins which act only as activation for interrupt. The connection from the according HW Element is made by the HW Port.
latency	Float	0..1	aggregation	Interrupt Latency: time passed from the occurrence of the interrupt request until the interrupt service routine is entered - HW: Register organisation; operations are necessary for context save. - SW: As SW can control the enabling/disabling of the interrupt SW determines a great part of the according time. - System: several constraints define the min/max times allowed for software to enable / disable interrupts.
maskable	Boolean	1	aggregation	Defines if interrupts can be selected to be processed by the interrupt controller.
priority	Int	1	aggregation	Some processing units offer different interrupt levels. It describes the priority in which the incoming interrupts are handled.

Table 3-15: InterruptConsumeHWPort

Class	**InterruptProduceHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::ProcessingUnit			
Class Description	This port represents the source of an interrupt request, i.e. the HWElement to which this port is attached is capable of raising interrupt requests.			
Base Class(es)	InterruptHWPort			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***

Table 3-16: InterruptProduceHWPort

3.9.1.3 Power modes

The power modes are controlled from the PU-Side by PU clock and the Power Supply. The usage of the described elements is not scope of the ECU Resource Template.

3.9.2 Constraints/Service

This subchapter describes some special services that are supported by the PU.

3.9.2.1 Operation mode

Different execution modes on processing units which enable execution of all, some, none instructions, including privileged instructions. Depending on the mode, access

to different address space, to memory management hardware and to other peripherals is possible.

- **User mode:**
  normal operating mode; not all resources are available
- **Supervisor mode:**
  mode accessible via authorisation, all system resources are available. For example Data Encryption might be a operation available only in Supervisor Mode, after a password has allowed the entry into the supervisor mode
- **Exception mode:**
  State of the PU after a exceptional event occurred in the system. Only limited resources are available at this state. Exception mode is most cases used to handle severe error condition. The limitation of resources is made in order to protect the system of further corruption.

Examples:
The operating system usually runs in the supervisor mode, where as the normal SW components are running in user mode.
The limp home mode after a clock failure is running in an exception mode

3.10

3.11 Peripherals

Peripherals are hardware modules accessible from the processing unit. This includes modules inside a processing unit (like a timer inside a µC) but also external modules connected to the processing unit (like an external watchdog timer). In the introduction chapter of this document the relationships between, signals, peripherals and sensors / actuators already have been described shortly.
The Peripheral HW Element and Peripheral HW Port are described in the subchapters of this section.
Standard communication interfaces can be used to connect external peripherals to a PU (e.g. SCI, SPI, LIN, etc.). The necessary mechanism is described in the according section of communication peripherals (see section 3.11.4).

3.11.1 Connecting Peripherals

Peripherals are described as HW Elements. Attributes that are defined by the peripheral are described within the description of the Peripheral HW Element. So each type of peripheral has a set of attributes that characterise it.
To connect the peripherals to processing units, ECU Electronics and sensors/actuators all of them have to provide HW Ports. HW Connections between those HW Ports establish the signal flow like shown in Figure 3-8.

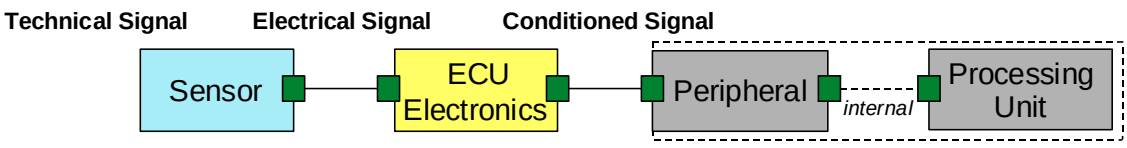

Figure 3-8: Connecting a Sensor to PU

Peripherals and HW Connections between the processing unit and peripherals can be "internal" if they are integrated in a microcontroller, but they are handled in the same way as external HW Connections.
Rather complex net-lists can be described with this approach where a signal is coming from a sensor, going through some ASIC which is connected through a serial communication line with the processing unit.

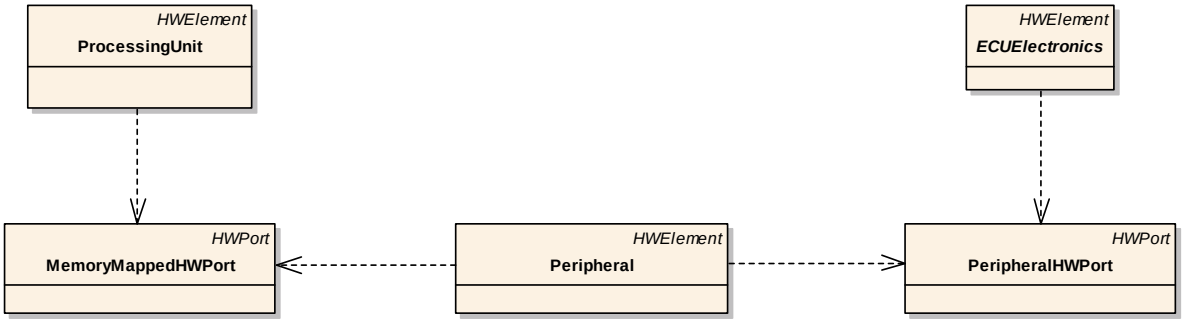

Figure 3-9: How peripherals are connected in the meta-model

The Figure 3-9 shows the class view, and in an actual description the Processing Unit and the Peripheral each have a Memory Mapped HW Port which will be connected with a HW Connection. Also between the Peripheral and the ECU Electronics two Peripheral HW Port instances are involved.

3.11.2 Peripheral HW Element

In the following figure the relationship of the Peripheral is shown. On one side the peripheral has a Memory Mapped HW Port to be connected to the PU. On the other side the Peripheral provides a Peripheral HW Port which is used to connect to the following hardware. Additionally each Peripheral can signal its status through an Interrupt Produce HWPort.

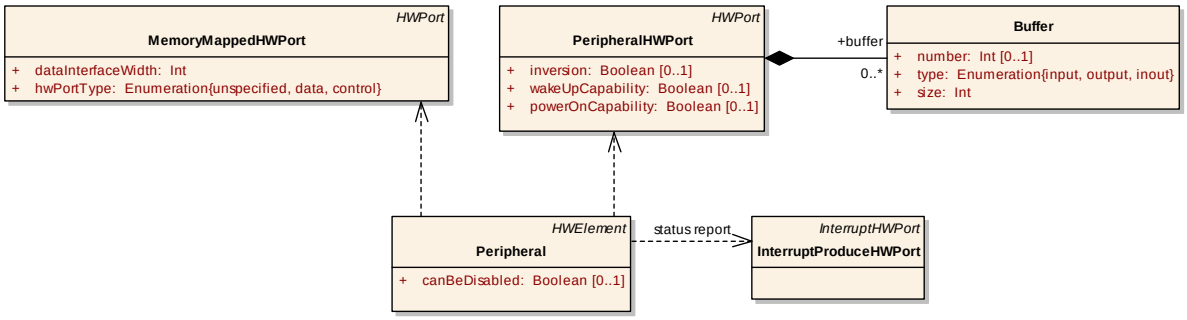

Figure 3-10: Peripherals and their HW Ports

Following table shows the Peripheral HW Element's attributes in overview form:

Class	**Peripheral**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	General Element for all peripherals.			
Base Class(es)	HWElement			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
canBeDisabled	Boolean	0..1	aggregation	Defines if the peripheral can be logically enabled or disabled.

Table 3-17: Peripheral HW Element

3.11.2.1 Peripheral Types

A signal can be gathered using different types of peripherals (taken partly from HIS API IO Driver Specification [1]):
Necessary information for the different peripheral types are:

	DIO	ADC	DAC	PWM	PWD	CCU	WDT	Timer
Inversion	X			X	X	X		
Electrical range	X	X	X	X	X	X	X	X
Resolution		X	X	X	X	X		X
Accuracy		X	X					
External reference voltage		X	X					
FrequencyRange	X	X	X	X	X	X	X	X
Phase detection					X	X		X
Count direction								X
Timer clock source						X	X	X
Threshold						X	X	X
Mode		X			X		X	X
Status report	X	X	X	X	X	X	X	X
Multiplexed		X	X					
Time		X	X				X	
Can be disabled	X	X	X	X	X	X	X	X
Can be re-init.						X		X

Table 3-18: Necessary elements for the peripheral types

The attributes of the different types of peripherals are described below. Within the attribute description only the additional information (restrictions, special values) for used elements are noticed. Please note that some of the attributes shown in Table 3-18 may not be found within the specific Peripheral HW Element directly, but within the associated Peripheral HW Port. The inheritance relationships between the different peripherals are shown in Figure 3-11.

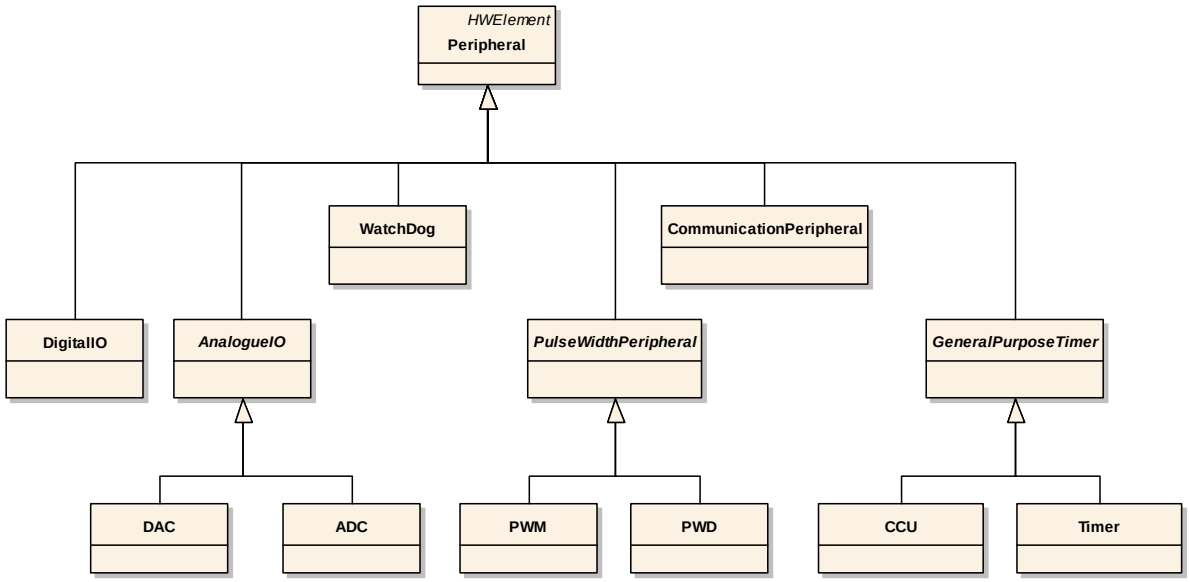

Figure 3-11: Inheritance Relationships of Peripheral HW Elements

3.11.2.1.1 Digital IO

Class	**DigitalIO**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Digital Input/Output peripherals provide the capability to send/receive data in a digital manner. Each digital input has an electrical range. When the voltage is less than a specified level, the input is represents logical low '0'. On the opposite when the voltage is higher than a specified value the input represents logical high '1'. A digital I/O can usually be programmed to be either an input or output. It can also be programmed to logically invert the input or output. The maximum frequency of a digital input or output is related to how fast the PU can read or write to the digital I/O register. It's very common for a digital I/O to have different functionality. One example is when a timer has an external clock signal associated with a digital I/O. In this case the maximum frequency is defined by the performance of the timer counter and can be found in the electrical characteristic of the peripheral datasheet.			
Base Class(es)	Peripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
edgeDetection	Enumeration{rising, falling, risingFalling}	1	aggregation	A general digital input can be enabled to trigger an interrupt on a falling, rising or both edges of an input signal. A clock, compare or count input on a timer can be programmed to increase, decrease or store the timer value if a change on the input pin has occurred. The condition for this change can be programmed to trigger an event in the same manner as for a digital input.

Table 3-19: Digital IO

3.11.2.1.2 Analogue IO

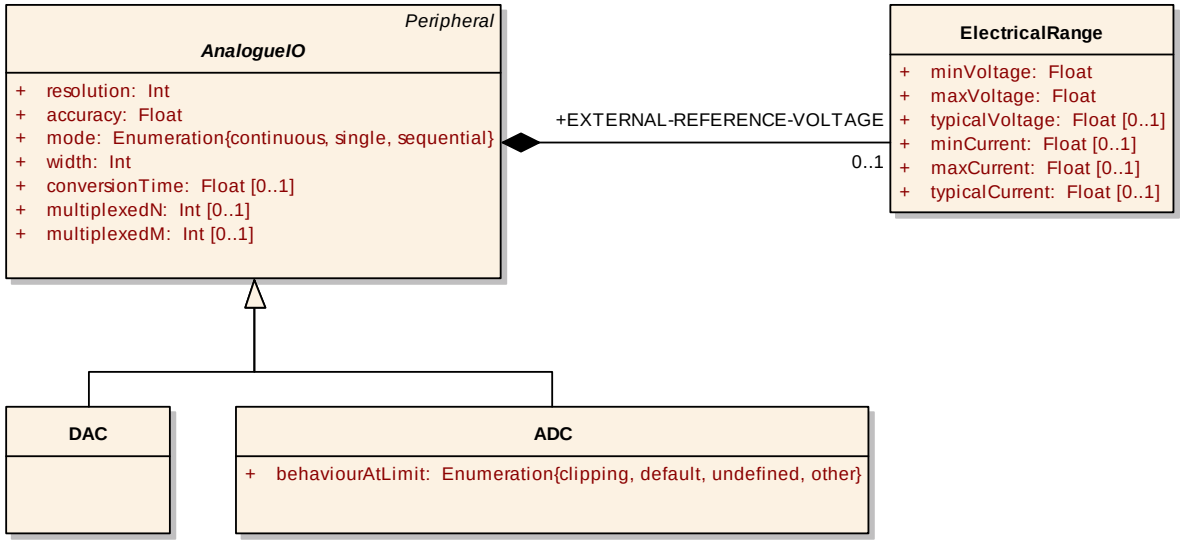

Figure 3-12: AnalogueIO

Class	**AnalogueIO (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	These are the common attributes for both, the ADC and the DAC.			
Base Class(es)	Peripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
EXTERNAL-REFERENCE-VOLTAGE	ElectricalRange	0..1	aggregation	The analogue comparator of an ADC must be connected to a voltage reference. An external voltage is usually connected to one low and one high voltage reference input pin. For an internal DAC it is normally only one voltage reference input. The low voltage reference input is internally connected to the analogue ground.
accuracy	Float	1	aggregation	There is a number of different conversion error in an ADC or DAC like non-linearity and absolute error. The accuracy is defined as the total error and it can be expressed as a deviation from its nominal value by e.g. 0.5, 1.0, 1.5 or 2.0 LSB.
conversionTime	Float	0..1	aggregation	ADC: The conversion time is the definition how long it takes for an ADC to convert an analogue signal to a digital value. DAC: The conversion time is the definition how long it takes for a DAC to convert a digital value to an analogue signal.
mode	Enumeration{continuous, single, sequential}	1	aggregation	An ADC or DAC can operate in continuous, single or sequential mode. In the continuous mode the conversion is automatically repeated after a completion. In single shot only one conversion is performed. For the sequential mode can a predefined number of different analogue channels be converted.
multiplexedM	Int	0..1	aggregation	Defines the design of a simple multiplexer used by an ADC. Example A 2:8-multiplexer has 2 input channels and 8 registers. M is the second value.
multiplexedN	Int	0..1	aggregation	Defines the design of a simple multiplexer used by an ADC. Example A 2:8-multiplexer has 2 input channels and 8 registers. N is the first value.

Class	**AnalogueIO (abstract)**			
resolution	Int	1	aggregation	For an Analogue to Digital Converter the resolution defines the number of different voltage levels, which are converted to a digital value. For example, an ADC with 10-bits resolution have the ability to convert an analogue value into 1024 different digital values. Unit: Bits
width	Int	1	aggregation	The number of Bits used to store the digital value.

Table 3-20: Analogue IO

Class	**ADC**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	An ADC converts an analogue signal at an input pin to a digital value. The analogue signal are quantified to a maximum number of different levels defined by the resolution of the ADC. The Accuracy of the ADC is important for the overall functionality.			
Base Class(es)	AnalogueIO			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
behaviourAtLimit	Enumeration{clipping, default, undefined, other}	1	aggregation	Defines the action that is going to be performed when a limit is reached. - Clipping: Logical value is limited to minimal or maximal range of the value range. E.g. The max. resolution of an ADC is at 5V represented 0xFF; in case off clipping a voltage of 6V is also represented as 0xFF; - Undefined: Exceeding the resolution results in an undefined representation of the logical value. - Default: Exceeding the resolution results in a predefined default value. The default value might be a value, which report the status to the application e.g. as an error condition. - Other: A predefined function occur in order to prevent a undefined logical value. A typical application is a "Soft-Clipping" in a logarithmic form at the limit of the resolutions in audio applications. The according methods have to be specified in the signal description.

Table 3-21: ADC

Class	DAC			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	The processing unit writes a digital value to a register of the DAC and the value is converted to an analogue value at a given pin.			
Base Class(es)	AnalogueIO			
Attribute	Datatype	Mul.	Link Type	Attribute Description

Table 3-22: DAC

3.11.2.1.3 Pulse Manipulation

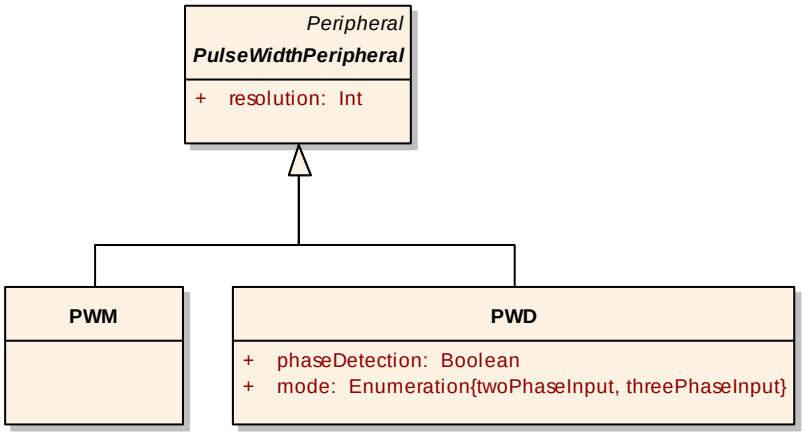

Figure 3-13: Pulse Width Peripheral

Class	PulseWidthPeripheral (abstract)			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Common elements for PWM, PWD and CCU.			
Base Class(es)	Peripheral			
Attribute	Datatype	Mul.	Link Type	Attribute Description
resolution	Int	1	aggregation	Defines the data width of the counter. Unit: Bits

Table 3-23: Pulse Width Peripheral

Class	PWM			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Pulse Width Modulation refers to a method of carrying information on a train of pulses, the information being encoded in the width of the pulses. The frequency is normally not changed. It's only relationship between the high and low level of the signal which are changed. A PWM interface is usually a part of a processing unit. It provides the capability to communicate with another system with an easy protocol.			
Base Class(es)	PulseWidthPeripheral			
Attribute	Datatype	Mul.	Link Type	Attribute Description

Table 3-24: PWM

Class	**PWD**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Pulse Width Demodulation is the inverse method of PWM. Here the processing unit is trying to gather information about a signal. A PWD can detect rising and falling edges and call the software in order to calculate the pulse width. When a PWD have more than one input signal the frequency and phase between the signals can be obtained.			
Base Class(es)	PulseWidthPeripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
mode	Enumeration{twoPhaseInput, threePhaseInput}	1	aggregation	A Pulse Width Demodulation peripheral can be configured to decode a two phase or a three phase input signals.
phaseDetection	Boolean	1	aggregation	Some timers have the possibility to detect signals from a quadrature rotating switch. There are two outputs signal from such a switch; the signals are 90 degrees phase shifted from each other. The rotation of the switch can then automatically be detected by a special function in a timer.

Table 3-25: PWD

3.11.2.1.4 Timer

Class	**GeneralPurposeTimer (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Common attributes shared by all timer peripherals			
Base Class(es)	Peripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
clockSource	Oscillator	0..1	reference	The clock input to a timer can have different sources like the internal system clock or an external clock. If the clock is pre-scaled this information should be entered in the table.
phaseDetection	Boolean	1	aggregation	Some timers have the possibility to detect signals from a quadrature rotating switch. There are two outputs signal from such a switch; the signals are 90 degrees phase shifted from each other. The rotation of the switch can then automatically be detected by a special function in a timer.
resolution	Int	1	aggregation	For a timer the resolution defines the data width of the counter. Unit: Bits
thresholdValue	Int	0..*	aggregation	The threshold is the definition of a counter value. When the counter has reached this value some action is performed, e.g. an interrupt can be generated.

Table 3-26: GeneralPurposeTimer

Class	**Timer**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	The simplest form is an array of directly coupled D-flip flops, which build up a shift register, with separate output from each stage, thus forming a time-register. A clock signal at the input increments the value of the register. Some cases uses some logic in combination of the shift register in order to form a special adopted representation of the information or to fulfil special requirements.			
Base Class(es)	GeneralPurposeTimer			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
canBeReInit	Boolean	0..1	aggregation	Defines if the peripheral can automatically be reset during operation.
countDirection	Enumeration{up, down, selectable}	0..1	aggregation	The direction defines if the counter of a timer counts up, down or if it can be selected.
mode	Enumeration{continuous, single, free}	1	aggregation	single: The timer is counted from a defined start value to a defined end value and then stopped. continuous: The timer is counted from a defined start value to a defined end value and then restarted with the start value. free: The timer is counted always to the available maximum value of the timer structure, then a wrap around occurs and the timer starts again from the HW defined startpoint; e.g. a 16 bit Timer counts from 0(hex)....0FFFF(hex), then after the overflow again from 0(hex)....0FFFF(hex).

Table 3-27: Timer

Class	**CCU**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	The CCU is a special kind of a timer, which has the capability of counting and comparing external signals. The timer is equipped with some extra registers and comparators in order to measure time or frequency.			
Base Class(es)	GeneralPurposeTimer			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
mode	Enumeration{capture, compare}	1	aggregation	A Capture and Compare peripheral can be configured to operate either in capture or compare mode.

Table 3-28: CCU

3.11.2.1.5 Watch Dog

Class	**WatchDog**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	A form of interval timer, which is used to detect a possible malfunction of the processing unit. The timer is started as a countdown and has to be triggered within a specific time. Either the trigger only has to occur before the count-down runs out, or the trigger has to occur within a specific window before the timeout, therefore a trigger too early will also set off the watchdog. If the trigger is not correctly performed the watchdog will set off and will initiate an interrupt or perform a reset of the whole processing unit or ECU. The watchdog can be implemented in software or hardware and if the implementation is in hardware it can be integrated into the processing unit or connected externally. Depending on the complexity of the watchdog the interfacing is either a serial communication or some discrete I/O lines. Also the trigger itself can be just some digital peak or can be some calculated request-response mechanism. The actual handling of the watchdog has to be covered by some specific driver software. The watchdog will use at least an interrupt of the processing unit and also some peripheral connections for the control and set-up. To enable different operating modes of the processing unit and the ECU it is important to specify if the watchdog can be disabled (e.g. during flash programming).			
Base Class(es)	Peripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
clockSource	Oscillator	0..1	reference	The clock input to a timer can have different sources like the internal system clock or an external clock. If the clock is pre-scaled this information should be entered in the table.
mode	Enumeration{normal, windowed, debug}	1	aggregation	A Watchdog can be running in different modes. In normal mode must the watchdog be kicked within a certain maximum time. In the windowed mode there is also a lower limit of the time when the watchdog has to be kicked. In debug mode is the watchdog disabled. Sometimes it's also required that the watchdog can be disabled and enabled during software download. To ensure that this mode is not entered in normal operation there is often some kind of security procedure to enter the mode.
thresholdValue	Int	0..*	aggregation	The threshold is the definition of a counter value. When the counter has reached this value some action is performed, e.g. an interrupt can be generated.

Table 3-29: Watch Dog

3.11.3 Peripheral HW Port

In this section the elements and attributes additional to the general HW Port description (see chapter 2.2) are described. They are specific for peripherals. On the other hand this port is directed to the outside of an ECU namely the ECU Electronics. To a PU the peripherals are connected via a Memory Mapped HW Port.

Class	**PeripheralHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	This port is directed from the peripheral to the ecu electronics and communication.			
Base Class(es)	HWPort			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
buffer	Buffer	0..*	aggregation	Defines the number of elements that can be stored in an input or output buffer. A capture timer can for example have several registers to store the timer value when an external event has occurred.
inversion	Boolean	0..1	aggregation	An input or output can sometimes be programmed to invert the level of a signal.
powerOnCapability	Boolean	0..1	aggregation	The peripheral device has the feature to issue a Power On event to the ECU. Example: Some CAN bus transceivers have a dedicated low power voltage regulator and a feature, which allows to turn on the main voltage regulator of the ECU by detecting a bus activity. Example: A wired-or connection of a termination KL30 and KL15 can fulfil this requirement.
wakeUpCapability	Boolean	0..1	aggregation	The peripheral device has the feature to issue a wake-up event to the processing unit. Example: HW Port with Key-Wake-Up on a PU is still active while the PU is in stop mode and only partially supplied with power. The Key-Wake-Up reacts on any changes at the input of the HW Port with the activation of the power supply, the activation of the clock system and later the resuming of the software, which has to process the input of the HW Port.

Table 3-30: Peripheral HW Port

Class	**Buffer**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Description of a buffer. It is used to describe input and output buffers. Buffers can also be configurable direction.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
number	Int	0..1	aggregation	The amount of buffers with the attributes size and type.
size	Int	1	aggregation	The size of the Buffer. Unit: Byte
type	Enumeration{input, output, inout}	1	aggregation	Either input, output or configurable buffer.

Table 3-31: Buffer

3.11.4 Communication Peripheral

A portion of the ECU that transmits information from the PU to a communication link is mostly a carrier transmitter-receiver.
It can be described by the OSI architecture that is a network architecture and a suite of protocols (a protocol stack) to implement it, developed by ISO [2].

The OSI architecture is split between seven layers as seen in Figure 3-14.

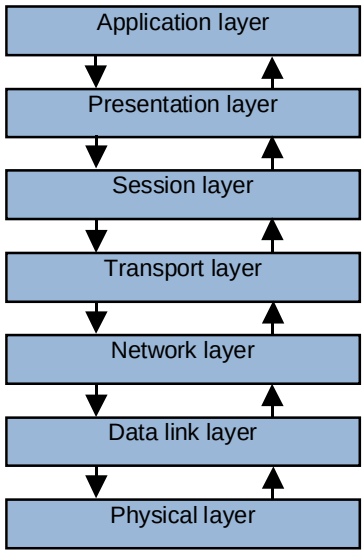

Figure 3-14: ISO / OSI layer model

Each layer uses the layer immediately below it and provides a service to the layer above. In some implementations a layer may itself be composed of sub-layers. Only the physical and the data link layer is relevant for an ECU description. However some bus specifications cover more or even all layers, e.g. LIN, MOST or some extra features like WAKE-UP and Errorhandling in a CAN Network.
The Communication Peripheral is described with a specialisation of the Peripheral HW Element, the Communication HW Element. This means all elements and attributes available for the Peripheral HW Element are contained.
The connection of Communication Peripherals with other HW Elements is done using the Communication HW Connection between the Communication HW Ports of the two HW Elements.

3.11.4.1 Communication Peripheral

The Communication Peripheral has some additional characteristics dealing with principles for data exchange. They are shown in the table below.
The controller shall be bound by name if it is needed by the Basic-SW.

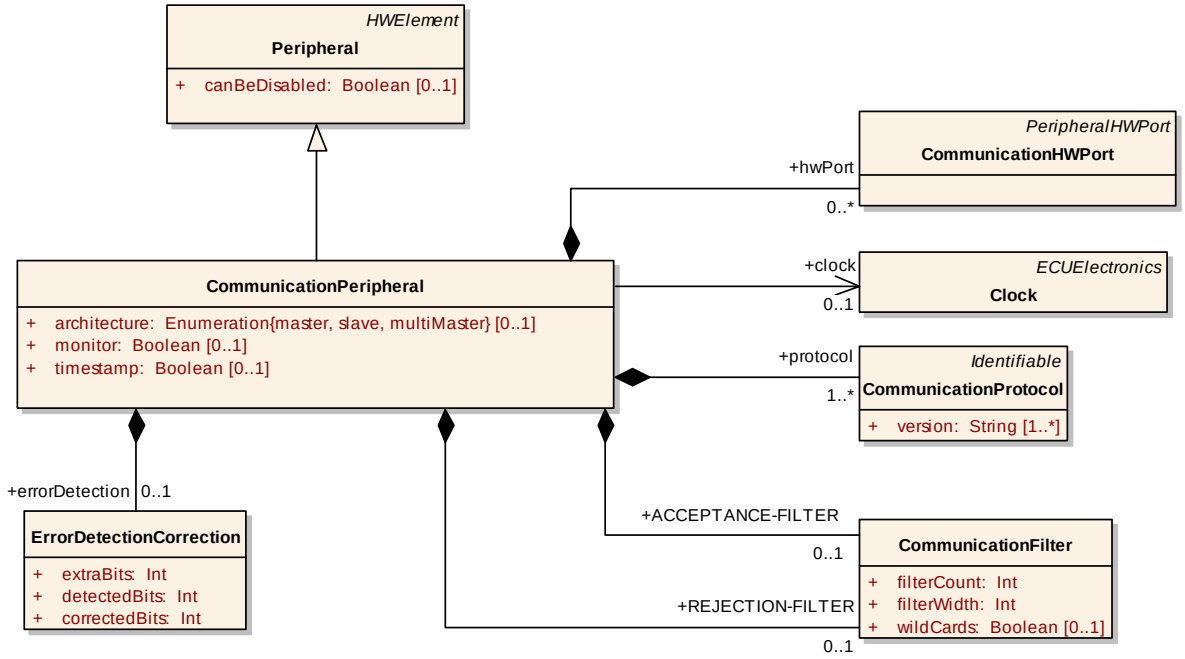

Figure 3-15: Communication Peripheral

Class	**CommunicationPeripheral**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Peripheral representing communication devices.			
Base Class(es)	Peripheral			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
ACCEPTANCE-FILTER	CommunicationFilter	0..1	aggregation	If the Communication Peripheral supports filtering of frames here the amount and quality of filters is described.
REJECTION-FILTER	CommunicationFilter	0..1	aggregation	If the Communication Peripheral supports filtering of frames here the amount and quality of filters is described.
architecture	Enumeration{master, slave, multiMaster}	0..1	aggregation	Distinguish between different implementations. - Master: Defines the position within the ECU, as the main controlling unit. The master defines the schedules and controls the telegram traffic. - Slave: Reacts only on request from the master - Multi Master: The PU has to share the external Interface with other PU. No specific master is defined by hardware.
clock	Clock	0..1	reference	Reference to the clock description. This is used to calculate possible transmission rates.
errorDetection	ErrorDetectionCorrection	0..1	aggregation	Under certain conditions that are likely to appear in automotive applications, the transmission of information between communication endpoints might be unsafe because of particular sources of interference . It is possible to detect the presence of interference by means of error-detection codes (EDC) and at least partly overcome the problem by means of error-correction codes (ECC). Error detection is a method that allows some communication errors to be detected. For this purpose the introduction of redundancy is required such that the information transmitted consists of more bits than the source of information. The extra bits are required for EDC and ECC.
hwPort	CommunicationHWPort	0..*	aggregation	Describes the actual ommunication HW Port of this peripheral.
monitor	Boolean	0..1	aggregation	The Element Monitor specifies if the device supports listen only. In this mode the device is passive on the bus and no acknowledgement or other responds are transmitted.

Class	**CommunicationPeripheral**			
protocol	CommunicationProtocol	1..*	aggregation	Describes which protocols are supported by this hardware.
timestamp	Boolean	0..1	aggregation	Is a timestamp for received frames available.

Table 3-32: Communication Peripheral

For the Error Correction Detection see Table 3-3.

Class	**CommunicationFilter**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Describes the amount and the size of the communication filters. These can be used as acceptance or rejection filters.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
filterCount	Int	1	aggregation	How many acceptance filters are availabe.
filterWidth	Int	1	aggregation	Which bit-size do the filters have.
wildCards	Boolean	0..1	aggregation	Is it possible to provide wildcard identifires allowing to specify ranges for filtering.

Table 3-33: Communication Filter

The element Name of Protocol represents the name of the communication interface type.
The attributes of the element are listed in Table 3-34. Please note that the majority of protocol types share the same set of attributes. Exceptions from this rule are documented in Table 3-34. Some of the attributes are part of the Communication HW Port.

	Protocol Name	**Version of Protocol**	**Architecture**	**Monitor**	**Com. Speed Tolerance**	**Acceptance Filter**	**Timestamp**	**Controller**	**Transmit Buffers**	**Receive Buffers**	**Size of Tx Buffer**	**Size of Rx Buffer**
SCI			X	X	X	X	X	X	X	X	X	X
SPI			X	X	X	X		X	X	X	X	X
I^2C			X	X	X	X		X	X	X	X	X
LIN, K-Line, CAN, TTP, FlexRay, MOST, Ethernet, Firewire (FW), USB, Bluetooth, W-LAN	X	X	X	X	X	X	X	X	X	X	X	X

Table 3-34: Necessary elements for the attributes of element name of protocol of communication interface

3.11.4.2 Communication HW Port

The Communication HW Port is directed to the outside of an ECU. This means that it connects the transceiver with a Remote HW Element that has an according HW Port of its own. Directed to the inside a communication transceiver HW has a Peripheral HW Port to link to a Peripheral HW Element.

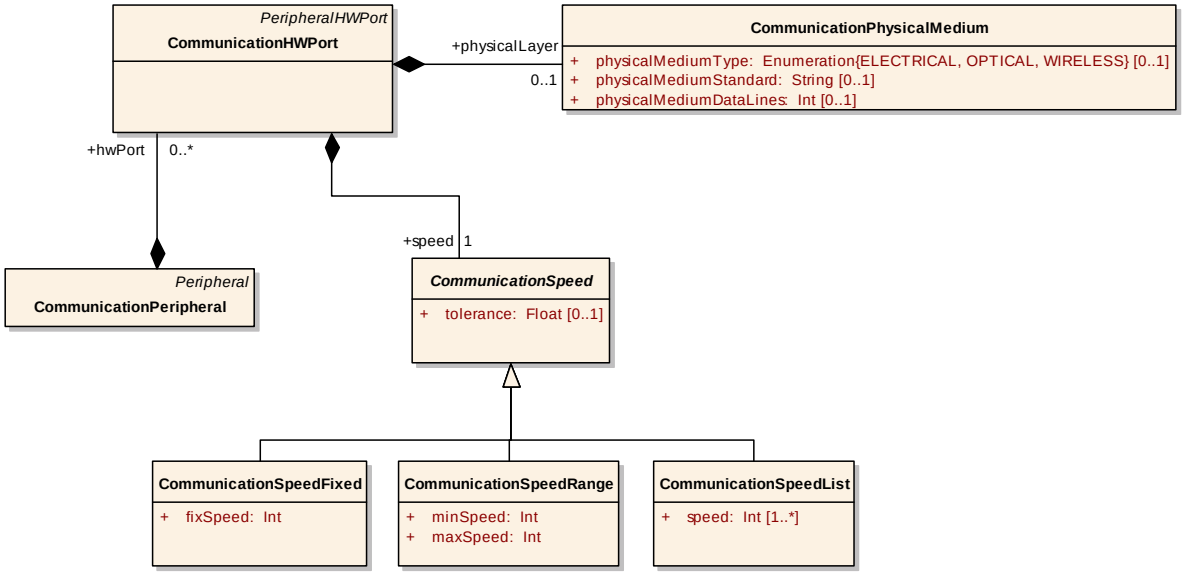

Figure 3-16: Communication HW Port

Class	**CommunicationHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description				
Base Class(es)	PeripheralHWPort			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
physicalLayer	CommunicationPhysicalMedium	0..1	aggregation	Describes the physical medium the bus is working on. In most cases it is not relevant for the SW to know about it.
speed	CommunicationSpeed	1	aggregation	The Communication speed can be provided either as a fixed value if it can not be changed by software. Or there can be a range in which software can configure the communication speed. Also a list of possible speed values can be provided from which the software can choose. If the communication speed is not fixed with one value the System Constraint Description has the freedom to choose from the given range or list of communication speed. In the end only HW Ports with the same communication speed can be connected to a bus system.

Table 3-35: Communication HW Port

Class	**CommunicationSpeed (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Abstract element to describe communication speed. This can be either a fixed value, a range or a list of allowed communication speed.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
tolerance	Float	0..1	aggregation	The element defines the tolerance allowed for the interface. (E.g. K-Line or LIN +/- 2 % at 9.6 Kbd)

Table 3-36: Communication Speed

Class	**CommunicationSpeedFixed**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Fixed value for the communication speed.			
Base Class(es)	CommunicationSpeed			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
fixSpeed	Int	1	aggregation	Unit: bits per second

Table 3-37: Communication Speed Fixed

Class	**CommunicationSpeedRange**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Range of communication speed.			
Base Class(es)	CommunicationSpeed			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
maxSpeed	Int	1	aggregation	Unit: bits per second
minSpeed	Int	1	aggregation	Unit: bits per second

Table 3-38: Communication Speed Range

Class	**CommunicationSpeedList**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	List of possible communication speed.			
Base Class(es)	CommunicationSpeed			
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
speed	Int	1..*	aggregation	Unit: bits per second

Table 3-39: Communication Speed List

Class	**CommunicationPhysicalMedium**			
Package	AUTOSAR Templates::ECUResourceTemplate::Peripherals			
Class Description	Describes the physical medium the bus is working on. In most cases it is not relevant for the SW to know about it, only for compatibility checking.			
Base Class(es)				
Attribute	***Datatype***	***Mul.***	***Link Type***	***Attribute Description***
physicalMediumDataLines	Int	0..1	aggregation	Specifies how many data lines (ground not included) are used for the physical layer. Some bus systems can be implemented with e.g. 1, 2 or 4 data lines.
physicalMediumStandard	String	0..1	aggregation	The element protocol name defines the exact standardised protocol name. Examples are: - ISO 11519 for low-speed CAN - IEEE 802.11g for W-LAN (54 Mbit/s at 2,5 GHz) - IEEE-1394a for Firewire-400
physicalMediumType	Enumeration{ELECTRICAL, OPTICAL, WIRELESS}	0..1	aggregation	Describes the actual medium used for transmission. Can be either electrical, optical or wireless.

Table 3-40: Communication Physical Medium

3.11.4.3 Communication HW Connection

A Communication HW Connection links to Communication HW Ports together. There does not exist any special attributes for this kind of HW Connection therefore a general HW Connection will be used to link these ports.

3.12 ECU Electronics

In this chapter several HW Elements are described. One kind builds the link between a Peripheral HW Port and the outer world in form of sensors or actuators. Another kind are special HW Elements that are relevant for operation but do not fit in the other categories (like an Oscillator HW Element).
The different kind of HW Elements are shown in Figure 3-17:

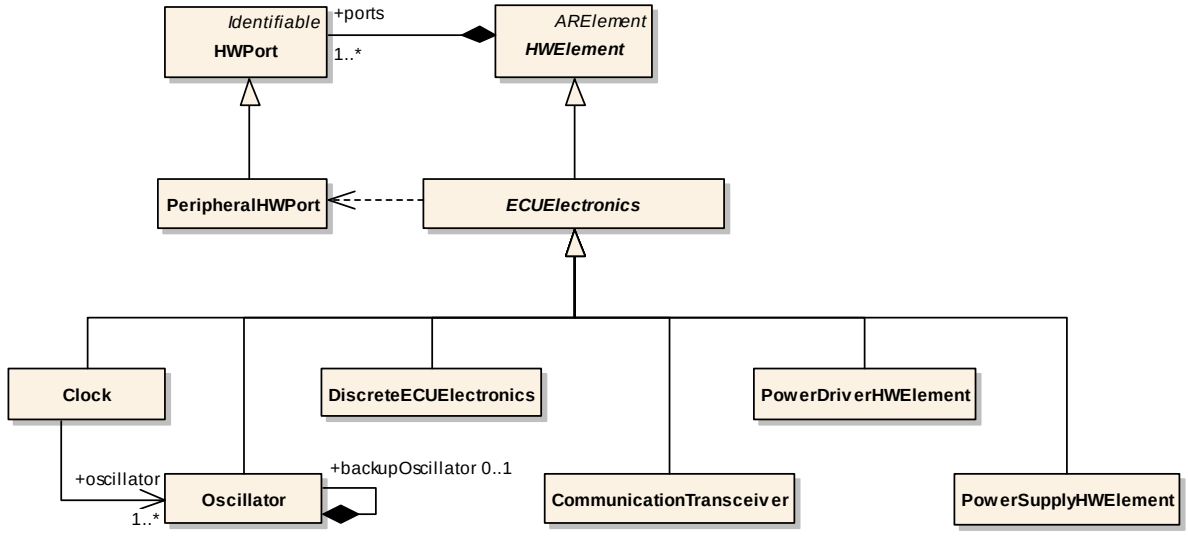

Figure 3-17: Meta-Model of the different ECU Electronics

Not all of these HW Elements need specific HW Port definitions. The necessary HW Port definitions are:

- Power Driver HW Port
- Power Supply HW Port

3.12.1 ECU Electronics HW Element

The abstract class ECU Electronics HW Element contains all elements and attributes common for all kind of ECU Electronics.

Class	**ECUElectronics (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The abstract class ECU Electronics HW Element contains all elements and attributes common for all kind of ECU Electronics.			
Base Class(es)	HWElement			
Attribute	Datatype	Mul.	Link Type	Attribute Description
canBeDisabled	Boolean	1	aggregation	Defines if the ECU Electronic can be enabled and disabled.

Table 3-41: ECU Electronics

3.12.2 Discrete ECU Electronics HW Element

As the ECU Electronics is an abstract class an unspecified ECU Electronics component can be modeled as Discrete ECU Electronics. Table 3-42 shows the specification of that specific ECU Electronics, but it does not introduce any specific attributes.

Class	**DiscreteECUElectronics**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description				
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description

Table 3-42: Discrete ECU Electronics

3.12.3 Oscillator HW Element

The function of an ECU is based on at least one clock element connected to a PU. Clocks are used to establish a sequenced data processing within a HW element and the data transfer between two HW elements.
Clocks can be used at different HW Elements within an ECU in the same way as for a PU, e.g. the Oscillator HW Element is very often assigned to a communication peripheral, ADC, PWM, PDM and most obvious to a timer.
An Oscillator HW Element can be the source of the very important SW Data Element “Time”.
The definition of the Oscillator and the Clock from the meta-model are shown in Figure 3-18.

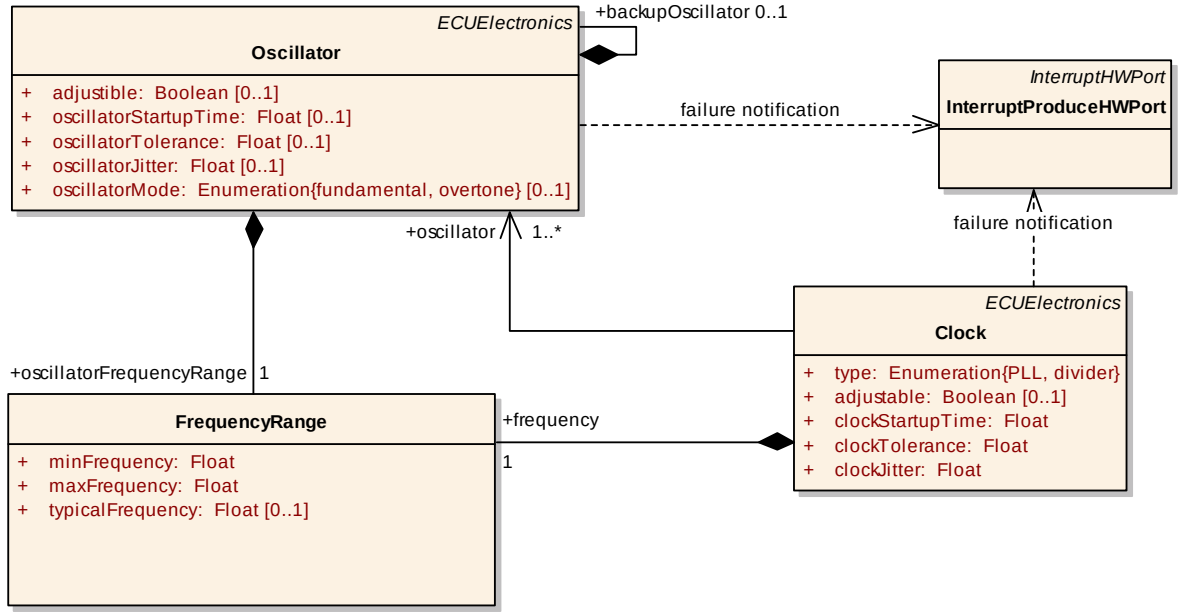

Figure 3-18: Clock and Oscillator

Class	**Oscillator**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The basic source for time in the ECU. Based on the oscillator the PU clock is generated, but also the communication and other peripherals need timing information (ADC, PWM, timer).			
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description
adjustible	Boolean	0..1	aggregation	Defines if the oscillator is adjustable from software.
backupOscillator	Oscillator	0..1	aggregation	An internal, very inaccurate RC-oscillator, which provide the PU with a clock source when the external oscillator is not available.
oscillatorFrequencyRange	FrequencyRange	1	aggregation	The frequency range the oscillator is able to provide.
oscillatorJitter	Float	0..1	aggregation	The short-term deviation of the nominal frequency to the real frequency. Jitter is influenced by the circuitry itself, voltage spikes etc. Jitter is critical for communication elements.

Class	**Oscillator**			
oscillatorMode	Enumeration{fundamental, overtone}	0..1	aggregation	Crystals and other oscillating elements have more than one resonant frequencies, which are usually odd harmonics to the basic, fundamental frequencies. For Automotive a basic requirement should be to use only fundamental mode only. Using overtone modes there is a risk, that the oscillator circuitry starts at the fundamental mode after a error in the start up routine. In order to reach an overtone mode the frequency of the fundamental mode, or lower overtones has to be passed during start up. There is a risk that the oscillator locks to this, lower frequency. Therefore Fundamental modes oscillatorts are not so critical for this issue. Fundamental quartz/crystals can reach up to 40 MHz today, above this frequency all quartz/crystals are running in the 3rd, 5th or higher overtone mode.
oscillatorStartupTime	Float	0..1	aggregation	The time the oscillator needs to deliver a stable and valid signal.
oscillatorTolerance	Float	0..1	aggregation	The long-term deviation of the nominal frequency to the real frequency. Influenced by manufacturing, layout, temperature, external components and voltage. Tolerances are critical for timer application and time synchronisation.

Table 3-43: Oscillator

Class	**FrequencyRange**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	Class describing minimum, maximum and typical frequencies.			
Base Class(es)				
Attribute	Datatype	Mul.	Link Type	Attribute Description
maxFrequency	Float	1	aggregation	Maximum frequency. Unit: Herz
minFrequency	Float	1	aggregation	Minimum frequency. Unit: Herz
typicalFrequency	Float	0..1	aggregation	Typical frequency. Unit: Herz

Table 3-44: Frequency Range

3.12.3.1 Oscillator frequency

The oscillator frequency represents the explicit input to a HW element, e.g. a PU. The oscillator frequency is the base of other derived clocks and the source from which the different system clock to the processing unit are generated with the clock generator circuit. The power mode slow and run is in most cases controlled directly via the clock system, which use a multiplier of the oscillator in run mode and a direct drive of the device with the oscillator in slow mode. The power consumption as well as the performance value of an ECU is a linear function of the clock frequency of the active elements. Some of the devices are designed in the way that they need a minimum of clock for their functions. The maximum value is in general limited by the hardware itself. e.g. signal delays, signal levels and heat dissipation. The most common form of an oscillator is done via quartz or ceramic resonators which can only oscillate at a defined, fixed frequency. Other forms of oscillators are available, which can be adjusted in a wide range, based on RC-types of oscillators.
Therefore minimum, maximum and typical value of the crystal frequency and the accuracy has to be entered in the ECU Resource Template, as well as the range of operation in form of a minimal and maximal ratio value based on the operational frequency. For example 12 kHz, operational, Min value 1.000 kHz, Max Value 40.000 kHz.

Class	**Clock**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The clock delivers the time for the PU and other HW Elements on the ECU.			
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description
adjustable	Boolean	0..1	aggregation	Defines if the clock is adjustable from software.
clockJitter	Float	1	aggregation	The short-term deviation of the nominal frequency to the real frequency. Jitter is influenced by the circuitry itself, voltage spikes etc. Jitter is critical for communication elements. PLL need/generate jitter for operation. Therefore communication interfaces should operate from an oscillator e.g. quartz directly or use only divider, not a PLL, as clock source. Using PLL for communication interfaces is a very common source for problems in ECU test and integration.
clockStartupTime	Float	1	aggregation	The time the clock generation needs to deliver a stable and valid signal.
clockTolerance	Float	1	aggregation	The long-term deviation of the nominal frequency to the real frequency. Influenced by manufacturing, layout, temperature, external components and voltage. Tolerances are critical for timer application and time synchronisation.
frequency	FrequencyRange	1	aggregation	The frequency range the clock is able to provide. This value depends on what the oscillator is delivering.
oscillator	Oscillator	1..*	reference	Specifies which oscillator is used to generate the clock.
type	Enumeration{PLL, divider}	1	aggregation	Defines if a PLL or a divides is used to create the clock signal.

Table 3-45: Clock

3.12.3.2 Clock frequency

The clock to the processing unit is generated from the oscillator frequency by some kind of clock generation circuit. It can be a simple pre-scaler e.g. a divider in steps of 2^n or more sophisticated a PLL circuit which can be used as scaleable multiplier.
Within a PU it is possible from the clock frequency to calculate the execution time of every instruction. Each instruction of the processing unit is executed with a specified number of clock cycles. The minimum, maximum and typical clock frequency has to be entered in the ECU Resource Template.

3.12.4 Communication Transceiver HW Element

The Communication Transceiver has to transfer and receive the data through the physical link and is the implementation of the OSI layer 1. Communication Transceivers have additional links to the controller/PU to support wake-up feature, enabling/disabling and error indication. The Meta-Model of the Communication Transceiver is shown in Figure 3-19.

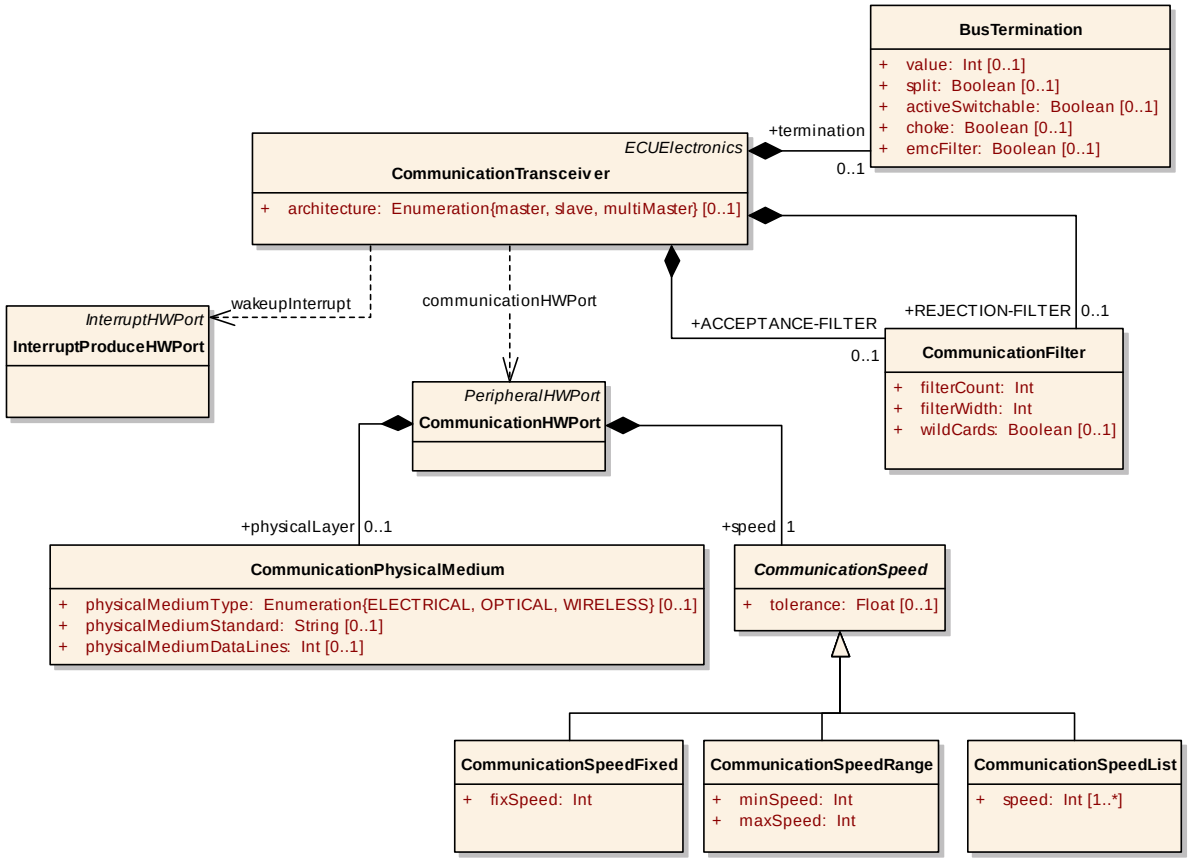

Figure 3-19: Meta-Model of the Communication Transceiver

The Communication Transceiver HW Element is described with the attributes listed in the Table 3-46.

Class	**CommunicationTransceiver**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The Communication Transceiver has to transfer and receive the data through the physical link and is the implementation of the OSI layer 1. Communication Transceivers have additional links to the controller/PU to support wake-up feature, enabling/disabling and error indication.			
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description
ACCEPTANCE-FILTER	CommunicationFilter	0..1	aggregation	Some transceivers have already acceptance filters implemented in hardware. This can be specified here.
REJECTION-FILTER	CommunicationFilter	0..1	aggregation	Some transceivers have already rejection filters implemented in hardware. This can be specified here.
architecture	Enumeration{master, slave, multiMaster}	0..1	aggregation	Distinguish between different implementations. - Master: Defines the position within the ECU, as the main controlling unit. The master defines the schedules and controls the telegram traffic. - Slave: Reacts only on request from the master - Multi Master: The PU has to share the external Interface with other PU. No specific master is defined by hardware.
termination	BusTermination	0..1	aggregation	Description how this network node is terminated.

Table 3-46: Communication Transceiver

The Communication Filter is described in Table 3-33. For Communication Physical Medium see Table 3-40.

Class	**BusTermination**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	In order to avoid signal reflection and oscillation of the Technical Signal represen-tation the impedance on communication lines have to be defined and matched at the signal destination and the signal source. Some examples: A High-speed CAN network with a twisted pair cable works optimal with an overall impedance of 60 Ohms represented by a 120 Ohm resistor in the ECU at the begin-ning of the network and 120 Ohm in the ECU at the end of a linear network topology. For Low-speed CAN networks in many cases termination resistors of 820 Ohm each are distributed on several ECUs to get good balance between system scalability and signal integrity in real networks without a fixed network topology. The lower bandwidth can tolerate some impedance mismatch. For LIN networks recessive termination is used: The inactive bus signal level is defined by resistors tied to a defined voltage level. An important attribute for communication lines is the presence of a termination resistor and the value of this resistor.			
Base Class(es)				
Attribute	Datatype	Mul.	Link Type	Attribute Description
activeSwitchable	Boolean	0..1	aggregation	Switchable termination describes a technology were the termination resistors can be switched or changed by extra hardware and therefore is under control of software. Main scope of a switchable termination is to adopt the termination of an individual ECU to the network topology of a specific car, e. g modules for extra car options can use this method to be incorporated in a wide range of different car types. The active termination is described by the values for each termination value. E.g. 120, 820, 12k. No termination is represented by values greater than 100 k. The attributes value and switchable are exclusive to each other.
choke	Boolean	0..1	aggregation	To improve the signal integrity on communication busses with a differential electrical signal representation on the physical layer, common mode chokes are used. They suppress common mode DC noise on CAN, FlexRay and Ethernet networks. Depending on the network topology common mode chokes of an ECU have to be taken in to account when adding ECUs to a network during design phase. Typically chokes are used on CAN, FlexRay, LVDS.

Class	**BusTermination**			
emcFilter	Boolean	0..1	aggregation	To improve the EMC behaviour of communication busses common dedicated EMC Filters are used. They suppress frequencies outside the range, which is needed for the communication of the specific bus E.g. some small capacitors in parallel to ground, resistors and inductivities in series are added to the bus lines in order to prevent the higher harmonics stimulated by the communication on the network The information about the termination is used with in the system configuration process, where the placement of an ECU within a car architecture is generated.
split	Boolean	0..1	aggregation	Describe a technology were the termination is split in to individual resistors with a capacitive coupling to ground. This technology improves the signal quality.
value	Int	0..1	aggregation	Describe the nominal Value of the termination resistor at the ECU as seen from the communication network.

Table 3-47: Bus Termination

3.12.4.1 Bus Modes

Bus modes describe a unified and simplified model of any communication bus in an ECU where AUTOSAR SW and methods are applied.
Following Bus Modes are assumed for a possible, future model. The creation of a model for communication systems and the according bus modes is not subject to describe within the ECU Resource Template.

- **Bus-Passive:**
  The bus is not supplied with power. The transceiver is not supplied with power. No pull-up/pull-down resistors are active. The termination of the bus via the transceiver is undefined.
- **Bus-Sleep:**
  The bus signal level is in a defined state for the sleep mode. The bus transceivers have switched their physical termination to the most recessive level, in order to avoid unwanted noise and oscillation on the bus line. Some voltage regulators tied to the bus are active. Some of the transceivers are on.
- **Bus-Idle:**
  The bus signal level is in the status as for operation mode but no communication is on the bus. All transceivers are active. The termination is active as in operation mode. The processing unit is in the status ready or run.

- **Bus-Active:**
  The bus signal level is in the status as for operation mode and communication is on the bus. All transceivers are active. Termination is activated. The processing units and the software are active and running.

The Bus modes described above are controlled via the attributes of a peripheral:

- **Termination:**
  Influence the signal level and therefore is prerequisite for the Bus Modes Bus-Active and Bus-Idle.
- **Wake Up Capability:**
  Describes the feature to influence the power mode of an ECU in the Modes Stop, Pseudo Stop, or Slow
- **Power On Capability:**
  Describes the feature to influence the power mode of an ECU in the Modes Off, and other like Stop, Pseudo Stop, or Slow

3.12.5 Power Driver HW Element

Power Driver HW Element describes a group of electronic devices, which perform signal transformations with hardware devices. The main focus of the ECU Resource Template is the logical description of these elements, necessary for the system generation and relevant for the SW development.
Power switches are devices that can switch high voltage/high current with a less resistance. The smart power switches have also some diagnosis features to identify critical operation conditions. Power ASICs, a special kind of Power Driver, combine a programmable microcontroller with a smart power device.
Typical attributes of Power Driver HW Elements are:

Class	**PowerDriverHWElement**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	Power Driver HW Element describes a group of electronic devices, which perform signal transformations with hardware devices. The main focus of the ECU Resource Template is the logical description of these elements, necessary for the system generation and relevant for the SW development.			
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description
currentLimitation	Float	0..1	aggregation	The device limits the maximum current conduction. The limit has to be specified. If this element does not occur the HW Element does not provide current limitation. Unit: Ampere
defaultResetBehaviuor	Enumeration{passive, on, off, other}	0..1	aggregation	- Passive: HW Element does not have an influence on ECU electronics (Tri-State). - On: The HW Element supplies other electronics with power e.g. the switch is closed. - Off: HW Element does not supply other electronics with power e.g. the switch is open. - Other: The HW Element has a complex behaviour and must be described by other means.
notification	PowerDriverNotification	0..1	aggregation	Which kind of notification to the PU is available.

Class	**PowerDriverHWElement**			
onStateResistance	Float	0..1	aggregation	The resistance in ON state can be used together with the current through this element to determine the static thermal power dissipation of the element itself and the ECU completely by summing up each element with power dissipation. As the heating up of an power element follows with an exponential delay, the ratio of on and off times is later needed in the system generation process. This information can give the first hint on critical issues of thermal design, but does not replace a complete static and dynamic verification of the thermal layout. Unit: Ohm

Class	**PowerDriverHWElement**			
powerDriverType	Enumeration{HS, LS, HB, FB, other}	0..1	aggregation	Defines the general type of the element. Type is a most common naming for an element. Several sets of types exist. Type is mandatory for the usage of the ECU Resource Template. Examples are: - High Side Switch (HS): Power supply is switched - Low Side Switch (LS): GND is switched - Half Bridge (HB): Both Power and GND can be switched exclusive - Full-Bridge (FB): Current direction in load can be alternated - Other: Different and complex functions can be included. Examples are Power ASICs that combine smart power devices within one package.
protection	PowerDriverProtection	0..1	aggregation	Which kind of protection is available.

Class	**PowerDriverHWElement**			
resetOnFaultBehaviour	Enumeration{auto, re-trigger, fold-back}	0..1	aggregation	Describe the way the element is reactivated after a self protection sequence was entered - Auto: The element is turned on automatically when the fault situation disappears - Re-Trigger: The element is turned on each time a new trigger is initiated. This new trigger must be initiated by SW. - Fold-Back: The element is turned on automatically or after a new activation trigger with a lower threshold.

Table 3-48: Power Driver HW Element

Class	**PowerDriverProtection**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The Power Driver HW Element / ASIC turns off by internal means in order to protect the device against severe damages.			
Base Class(es)				
Attribute	Datatype	Mul.	Link Type	Attribute Description
lossOfGroundProtection	Boolean	0..1	aggregation	Is protected against loss of ground.
maxCurrentProtection	Boolean	0..1	aggregation	Maximal current limits are exceeded.
maxVoltageProtection	Boolean	0..1	aggregation	Maximal voltage limits are exceeded.
minCurrentProtection	Boolean	0..1	aggregation	Minimal current limits are exceeded.
minVoltageProtection	Boolean	0..1	aggregation	Minimal voltage limits are exceeded.
overTemperatureProtection	Boolean	0..1	aggregation	Is protected against over-temperature.
reverseBatteryProtection	Boolean	0..1	aggregation	Is protected against reverse battery voltage.

Table 3-49: Power Driver Protection

Class	**PowerDriverNotification**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	Describe the status which cause a notification to the controlling devices of an ECU, e.g. to the PU.			
Base Class(es)				
Attribute	Datatype	Mul.	Link Type	Attribute Description
issuedInterruptSource	InterruptProduceHWPort	0..1	reference	If a notification is issued, in which Interrupt will it result.
lossOfGround	Boolean	0..1	aggregation	The device detected a loss of ground situation. The device was forced off, in order to avoid damages by excessive currents.
maxCurrent	Boolean	0..1	aggregation	The device detected a too high current.
maxVoltage	Boolean	0..1	aggregation	The device detected a too high supply voltage.
minCurrent	Boolean	0..1	aggregation	The device detected a too low curent.
minVoltage	Boolean	0..1	aggregation	The device detected a too low supply voltage.
openLoad	Boolean	0..1	aggregation	The device detected open loads. This situation is reported to the controlling portion of an ECU, e.g. to the PU.

Class	**PowerDriverNotification**			
overTemperature	Boolean	0..1	aggregation	The maximum allowed temperature was detected in the device. A protecting hardware has forced the device off.
shortLoad	Boolean	0..1	aggregation	The device was forced off, in order to avoid damages by excessive currents.
shortToBatt	Boolean	0..1	aggregation	The device detected a short to battery situation. The device was forced off, in order to avoid damages by excessive currents.
shortToGround	Boolean	0..1	aggregation	The device detected a short to ground situation. The device was forced off, in order to avoid damages by excessive currents.

Table 3-50: Power Driver Notification

3.12.6 Power Driver HW Port

The following table list the elements and attributes of Power Driver HW Port, describing the interfacing mechanism of power elements to other parts of the ECU electronics.

Class	**PowerDriverHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	Define the option of a Power Driver that the output of a Power Driver can be superposed by a PWM signal asserted by a HW Element e.g. a microprocessor. In most cases, the PWM signal is connected by a separate wire, even when the general control information are transmitted by a bus.			
Base Class(es)	HWPort			
Attribute	Datatype	Mul.	Link Type	Attribute Description
pwmMaxFrequency	Float	0..1	aggregation	It defines the upper limit at which the device can fulfil the PWM capability. EMC and power dissipation are increasing almost exponential with the PWM frequency. Choosing PWM frequencies be aware that some effects can cause secondary effects, which can create a audible noise for the user, e.g. a PWM dimmed lamp can be noticed by the driver. Unit: Hz
pwmMinOffTime	Float	0..1	aggregation	Defines the characteristic, that a Power Driver has to be off at least a minimum time in order to assure the function. Unit: s
pwmMinOnTime	Float	0..1	aggregation	Defines the characteristic, that a Power Driver has to be on at least a minimum time in order to assure the function. Unit: s

Table 3-51: Power Driver HW Port

3.12.6.1 PWM Capability

Minimal and maximal on-time are caused e.g. by the internal status generation of the Power Driver. A minimum time is required to generate stable and valid status information. These data are transferred into the corresponding resolution e.g. duty cycle of a PWM system.
Example: A device for a Power Driver may consist of the following entries: max. PWM frequency 20 kHz, a minimum off-time of 1 µs. Used in a PWM system with 8bit resolution at max frequency this results in: 20khz → T_{cycle} 50us; T_{pause} = 1us→ 1/50 = 2% duty cycle; ➔ max duty cycle is 98%; which result as the max reload value of 249 (decimal) of the PWM register.
Note: Some Power devices implement PWM with the same control mechanisms as normal on / off operation. In this case a strong checking of the status information for overtemperature and overcurrent is necessary. Especially devices with a Reset-on-Fault Behaviour "retrigger" are endangered to be irreversible destroyed by thermal and mechanical stress e.g. they stay in this situation a conductive state. Thermal shutdown features are only valid for a single, static shutdown, not for repetitive and dynamic shutdown behaviour.
Example: In the interior light system a single Power Driver controls 10 to 20 elements each with up to 10 W. The interior light system is dimmed by PWM with low frequencies e.g. 80 Hz. The total system has about 100 W. A short circuit at one of

the many, thin wires will not cause the overcurrent limit to be exceeded, and will not force the fuse element to burn through. But this results in a permanent over-temperature shutdown at the end of each PWM cycle. The lifetime of the Power Driver, due to this stress, is fallen dramatically to only a few thousand PWM cycles and thus only a few hours. This is a very critical situation for the car systems; an empty battery will be one of the minor effects.

3.12.7 Power Supply HW Element

The following table list the important elements and attributes of Power Supply HW Element, necessary for the basic SW routines. Power Supply HW Element can be connected to any other HW Element.

Class	**PowerSupplyHWElement**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The Power Supply HW Element provides information about the power source of the ECU. There can be multiple power supplies within one ECU.			
Base Class(es)	ECUElectronics			
Attribute	Datatype	Mul.	Link Type	Attribute Description
OVER-CURRENT-NOTIFICATION	PowerSupplyCurrentNotification	0..1	aggregation	Status information that the output current exceeds the maximal, defined limit at the output of the Power Supply HW Element. A short circuit indication is a special form of the overcurrent status information.
OVER-VOLTAGE-NOTIFICATION	PowerSupplyVoltageNotification	0..1	aggregation	Notification that the source voltage is too high for the power Supply HW Element to handle.
UNDER-VOLTAGE-NOTIFICATION	PowerSupplyVoltageNotification	0..1	aggregation	Status information that at the input of the Power Supply HW Element has dropped to a critical value. This can be used to switch off functions, which caused that voltage drop or start a power down function like storing important data to NV memory.

Class	**PowerSupplyHWElement**			
backupCapacity	Float	0..1	aggregation	Define the capability assigned to the output of the Power Supply HW Element to maintain the output voltage stable for a certain time after the undervoltage status information is asserted. In the ECU Resource Template the Backup Capacity is described as time. The time is dependent from the current consumed by the HW Elements connected to the according output, the SW-functions controlling the ECU hardware and the implementation of the Power Supply HW Element itself. The time has to be calculated from the available capacitors or measured in the application. Example: For the energy storage for airbag applications capacitors stores energy for the airbag ignition for several seconds even the battery is completely disconnected. Example: A combined step up/step down DC/DC converter can deliver a stable 5V volt output even when the input has already dropped to 3V. The undervoltage status is already set at a Vbatt level of 6V. Unit: Ah
defaultVoltageOnReset	Float	0..1	aggregation	This value defines the nominal output voltage, which is supplied as default after a reset occured. This attribute is only applicable at power supplies with a programmable output voltage.
maxVoltage	Float	1	aggregation	The maximum voltage that can be supplied.
minVoltage	Float	1	aggregation	The minimum voltage that can be supplied.

Class	**PowerSupplyHWElement**			
notificationOverTemperature	Boolean	0..1	aggregation	Status information that the on-chip temperature on the Power Supply Hardware Element exceed the maximal, defined limit.
notificationReset	Boolean	0..1	aggregation	The power on reset is generated depending on the output voltage. Valid operation of the ECU and the connected HW Elements are only possible if the output voltage is stable and within the specified output voltage range. Therefore most of the Power Supply HW Elements contain a circuitry, which generate a output signal which indicates an instable or unspecified voltage at the output. Some Power Supply HW Elements offer the possibility to adjust the reset thresholds.
resetDelay	Float	0..1	aggregation	The reset signal at the power-on reset is delayed by the Power Supply HW Element in order to enable the different ECU HW Elements to enter a stable and defined operation state when the reset signal is released. Unit: s
resetOnFaultBehaviour	Enumeration{auto, reTrigger, foldBack}	0..1	aggregation	Describe the way the element is reactivated after a self protection sequence was entered - Auto: The element is turned on automatically when the fault situation disappears - Re-Trigger: The element is turned on each time a new trigger is initiated. This new trigger must be initiated by SW. - Fold-Back: The element is turned on automatically or after a new activation trigger with a lower threshold.

Class	**PowerSupplyHWElement**			
wakeupCapability	Boolean	0..1	aggregation	Describes the feature of the power supply to turn on the output voltage by an external event. In some cases this feature is coupled with other ECU internal HW elements e.g. transceivers and is the main feature used by the car specific power management. Different to other HW elements with this feature the power supply is the receiver of the signal, as more ECU function is only possible if the power supply is activated. Example: Philips CAN Transceivers have an output called INH, which allow together with some Voltage regulators with wake up capability to wake up the ECU at the presence of a CAN bus telegram. Example: The Wake up Capability is connected to the terminal KL15 (ignition), the ECU is waked up at the presence of KL15.

Table 3-52: Power Supply HW Element

3.12.8 Power Supply HW Port

This special type of HW Port is used if the information could be mapped to power supply. The following list of attributes must be considered for this HW Port type.

Class	**PowerSupplyHWPort**			
Package	AUTOSAR Templates::ECUResourceTemplate::ECU Electronics			
Class Description	The Power Supply HW Port is used to model the power distribution within the ECU. If the direction is "in" the port is used to describe the power consumption of an HW Element. If the direction is "out" the port is used to describe a power source. The HW Connection between these two HW Ports can only be established between the Power Supply HW Port of the HW Element with direction "out" and a Power Supply HW Port of the Power Supply HW Element with direction "in".			
Base Class(es)	HWPort			
Attribute	Datatype	Mul.	Link Type	Attribute Description
canBeSwitchedOff	Boolean	0..1	aggregation	Defines if the Power Supply HW Port can be switched off and therefore the supplied HW Element is switched off as well. This feature allows to describe which parts of an ECU are able be disconnected from their power supply based on software interaction. If the Power Supply HW Port has the direction 'in' the supplied HW Element can be switched off. If the Power Supply HW Port has the direction 'out' the power supply can be switched off and therefore all supplied HW Elements are switched off as well (This is used to describe groups of HW Elements that can only be switched off together). When however the whole ECU is disconnected from power the switchable power supply is of course not valid anymore.
supplied	ElectricalRange	1	aggregation	For power source: provided electrical specification. For power consumer: required electrical specification.

Table 3-53: Power Supply HW Port

Class	**ElectricalRange**			
Package	AUTOSAR Templates::ECUResourceTemplate::Basic Elements			
Class Description	Specifies electrical ranges for different applications within the ECU Resource Template.			
Base Class(es)				
Attribute	Datatype	Mul.	Link Type	Attribute Description
maxCurrent	Float	0..1	aggregation	Maximum Current Unit: A
maxVoltage	Float	1	aggregation	Maximum voltage Unit: V
minCurrent	Float	0..1	aggregation	Minimum Current Unit: A
minVoltage	Float	1	aggregation	Minimum voltage Unit: V
typicalCurrent	Float	0..1	aggregation	Typical Current Unit: A
typicalVoltage	Float	0..1	aggregation	Typical voltage Unit: V

Table 3-54: Electrical Range

3.13 Sensors and Actuators

An ECU can manipulate and sense the real world through peripheral IO connected to sensors and actuators. The purpose of this section of the design specification for the ECU Resource Template is to define and add HW Elements which are specific to sensors and actuators.
For the purposes of AUTOSAR any device, including sensor and actuators, that contains its own memory and one processing unit may each be described by an ECU resource template, however this in general will be unnecessary. As is stated in the introduction to this document, the main goal of the ECU Resource Template is to provide a solid basis for describing and checking the consistency of characteristics and features of automotive ECUs. The scope of the ECU description is limited to the characteristics and features that are relevant for the configuration of the MCAL on a particular ECU and it is not the intention of this work to model sensor and actuators for simulation purposes or to provide a placeholder for configuration data, although both types of data maybe referenced from accompanying datasheets. Since the scale and complexity of sensors and actuators varies widely the HW connections to the peripheral units will are very broad. Thus it will not be uncommon for a sensor and actuator to be represented by multiple HW elements, HW ports and pins which will be assigned in software.

3.13.1 Sensor Actuator HW Element

In this section the elements and attributes common to all sensors and actuators are described by the Sensor Actuator HW element. They are grouped in the abstract class Sensor Actuator HW in the Meta-Model. In Figure 3-20 the relevant HW Ports for the Sensor Actuator HW element are shown, however there is no direct relationship between the Sensor Actuator HW element an these ports. This relation is established through the generic HW Element and the generic HW Port.

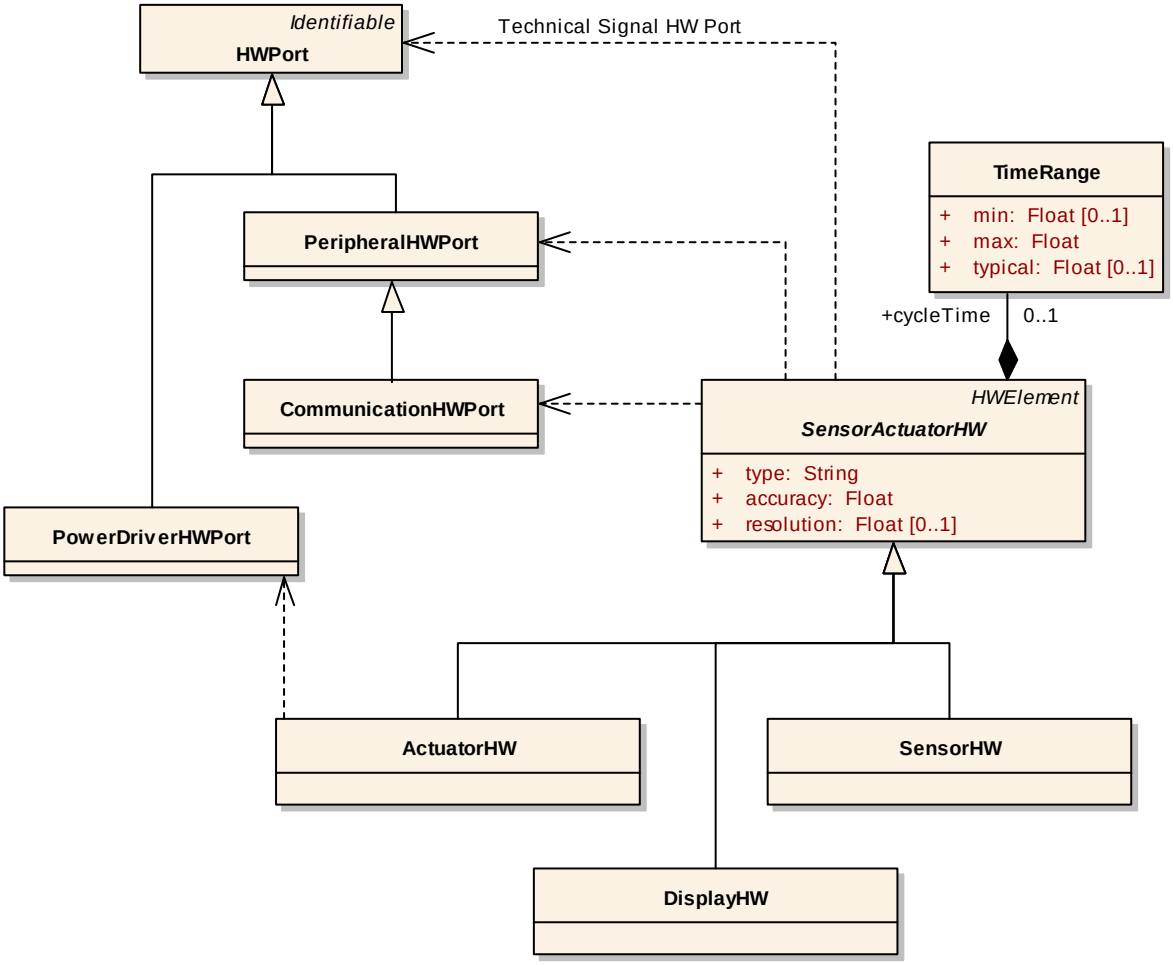

Figure 3-20: HW Sensor Actuator Element

The Sensor Actuator HW element can be connected via different kinds of HW Ports. It is possible to specify the connection through a Power Driver HW Port, a Peripheral HW Port and a Communication HW Port. Also the Sensor Actuator HW element has a virtual HW Port describing the Technical Signal of the real world.

Class	**SensorActuatorHW (abstract)**			
Package	AUTOSAR Templates::ECUResourceTemplate::Sensor Actuator			
Class Description	The common attributes for sensors and actuators. The sensor and actuators can be connected via a Peripheral HW Port, a Communication HW Port or a Power Driver HW Port.			
Base Class(es)	HWElement			
Attribute	Datatype	Mul.	Link Type	Attribute Description
accuracy	Float	1	aggregation	Defines the error in the representation of the Technical Signal in the data format This applies only if the Technical Signal is encoded before it is transferred to the ECU Electronics (e.g. via Communication Transceiver HW Port).
cycleTime	TimeRange	0..1	aggregation	The time the sensor/actuator must be accessed for correct information. It is possible to give a minimum, a maximum and a typical cycle time.
resolution	Float	0..1	aggregation	Defines the granularity of the representation of the Technical Signal in the data format. This applies only if the Technical Signal is encoded before it is transferred to the ECU Electronics (e.g. via Communication Transceiver HW Port).
type	String	1	aggregation	Defines the general type of the sensor/actuator type is a most common naming for a sensor/actuator and is an open list and is not restricted to the following items. Several sets of types exist. Type is mandatory for the usage of the template - Sensor: Temperature, Pressure, Distance, Hall - Actuator: DC Motor, Valve, Relay, Display

Table 3-55: HW Sensor Actuator

3.13.2 Sensor HW Element

In this section the sensor specific definitions are described. They are additional to the general Sensor Actuator HW elements.

Class	**SensorHW**			
Package	AUTOSAR Templates::ECUResourceTemplate::Sensor Actuator			
Class Description	HW Element Sensor definition.			
Base Class(es)	SensorActuatorHW			
Attribute	Datatype	Mul.	Link Type	Attribute Description
signalQuality	Enumeration{raw, filtered, debounced, coded}	1	aggregation	Defines the quality of the data received from a sensor. Depending on this information later processing on the signal has to be done in HW or SW. - Raw: The information are transferred without any signal processing from the sensor to the according electronic. Possible spikes and noise is present on the line. In order to use this signal an appropriate signal processing have to be performed. - Filtered: The information passed some filter mechanism in order to remove noise and spikes from the signal. The signal can be in a discrete form, At least the limit e.g. the -3dB limit, must be available to the software if this limit is close to the used bandwidth, needed for the planed application. If more complex filter like bandpasses or notch filter are used the filter characteristic must be modelled and imported for the according SW-Component as well. - Debounced: The information was already processed by some electronics in order to get a clear and unique representation of the information. Debouncing is a sort of filter which suppress the noises and spikes which are generated by the bouncing when a switch is opened or closed. Debouncing is similar to Filtered, but the input is only represented by digital values instead of analogue values. Typical debouncing electronics are mono- flops or sampling , plus a majority encoder to represent a more stable signal. - Coded: The information is filtered and debounced, but furthermore represented in a digital encoded form. Examples are sensor connected via a serial communication bus like LIN to the ECU. It must be noted that the complexity of the ECU will have a concomitant effect on the complexity of the corresponding ECU SW abstraction. Generally the more complex the ECU electronics, the more complex the abstract interfaces will be for the user to configure.

Table 3-56: Sensor HW Element

3.13.3 Actuator HW Element

In this section the actuator specific definitions are described. They are additional to the general Sensor Actuator HW elements.

Class	**ActuatorHW**			
Package	AUTOSAR Templates::ECUResourceTemplate::Sensor Actuator			
Class Description	HW Element Actuator definition.			
Base Class(es)	SensorActuatorHW			
Attribute	Datatype	Mul.	Link Type	Attribute Description
defaultPositionType	Enumeration{stabel, instable, biStable, multiStable}	0..1	aggregation	Defines the way the actuator is seen after power on. On Bi-Stable or Multistable actuators the software has to know the status of this device after reset or Power On. Each actuator has a control input that determines its behaviour - Stable: The actuator has a unique defined position after a reset or power down. E.g. a spring force the actuator back into a start position or there are means which give a position feedback like a potentiometer or switch. - Instable: The position of the actuator can not be specified after power on or reset or the position of the actuator can be changed by other means - Bi-stable: The actuator has two possible, defined positions after a reset or power down. E.g. the actuator is in one of the two possible end positions. Only one of them is signalled by a feedback, e.g. a switch. The second end position is simply to be determined as the inversion of the feedback. - Multi-stable: The actuator has more possible, defined positions after a reset or power down and they are signalled by appropriate feedback E.g. a door latch might consist of different stable states: pre-locked, locked, double locked, released, open from outside, open from inside, All this states are covered by a feedback, and the states can change in random manner after the wake-up or power-up of a car.
defaultPositionValue	String	0..1	aggregation	If there is a specific default value is available. E.g. Valve status = off;

Class	**ActuatorHW**			
electricalType	Enumeration{resistive, capacitive, inductive}	0..1	aggregation	Defines the most typical electrical behaviour of the system. This information is necessary for the ECU generation process and for tools which check for later interaction of the different component. (avoid resonance's, planning of the ECU power management). - Resistive: Heater, Lamp - Inductive: Coils, Motor - Capacitive: Extra SuperCAPS for bordnet stability
endurance	Int	0..1	aggregation	Number of activation cycles the actuator is designed for.
movement	Enumeration{unidirectional, bidirectional, unidirectionalWithAutoreset}	0..1	aggregation	Defines the way the movement is controlled by AUTOSAR. Positional feedback can be provided by the actuator by the conglomeration of the requisite actuator and sensor primitives. - Unidirectional: The actuator is only forced in one direction by SW. e.g. eject of a tape. The user or a other mechanical system is needed to chance the status of the actuator. Other example is a motor which has a circular movement always in one direction. SW can not alter this behaviour e.g. fan, wiper. - Bidirectional: The actuator can be moved forward and backward by software, e.g. window lift - Unidirectional with autoreset: The actuator is controlled only one direction by electrical means and SW. Bringing the actuator in to the default position other means are available. E.g. unlock of the trunk or gasoline flap with a motor which is brought into the initial position with a spring; a relay or a valve use the same mechanism.

Table 3-57: Actuator HW Element

3.13.4 Connecting Sensors and Actuators via complex ECU electronics

Sensors and actuators are not directly connected to some PU HW Port, but there is some control electronics in the ECU to switch, enable or initialise the ECU electronics, sensors and actuators.
An example is shown in Figure 3-21 (taken from WP1.1.1 document), where several sensors are connected to one ADC pin routed through an external multiplexer. The multiplexer is controlled via some DIO by the PU.

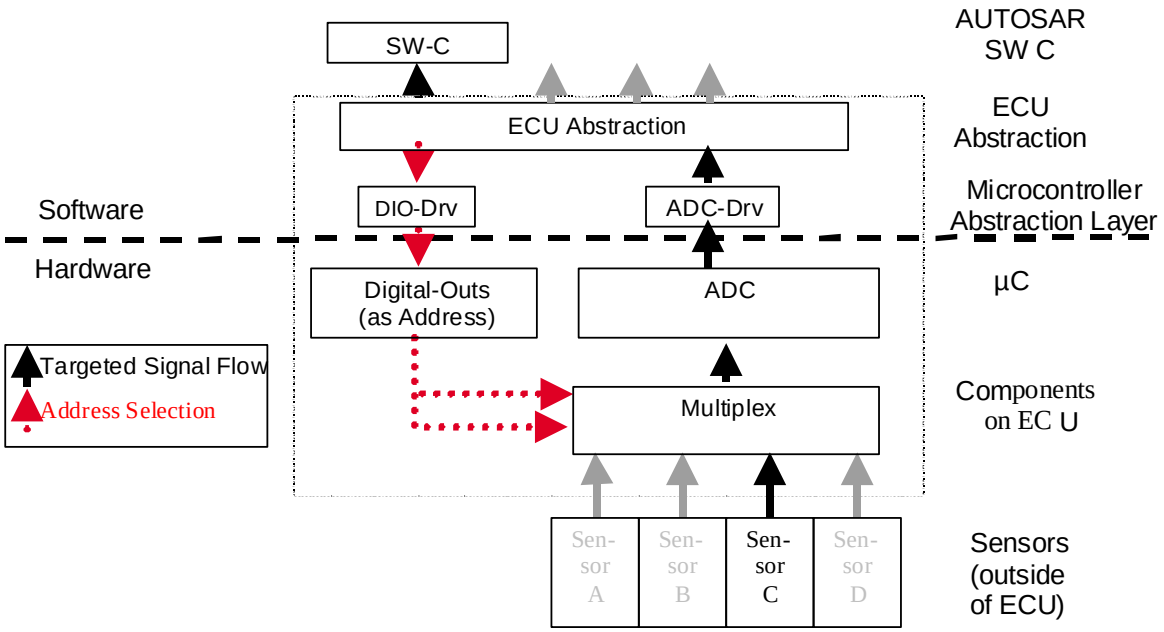

Figure 3-21: Connecting Multiplexed Sensors

Since structures like in Figure 3-21 can become very complex the ECU Resource Template does not provide description mechanisms for complex implementations of the ECU Electronics. In this case it will not be possible to configure the whole ECU Abstraction within AUTOSAR automatically, so the only relevant information is the interface of the ECU Abstraction to the SW-Components and the interface of the sensors and actuators to the ECU. All in between is encapsulated in a HW Element representing the ECU Electronics and by the ECU Abstraction Basic SW Module. In Figure 3-22 the example from Figure 3-21 is shown how it could be described with the ECU Resource Template.

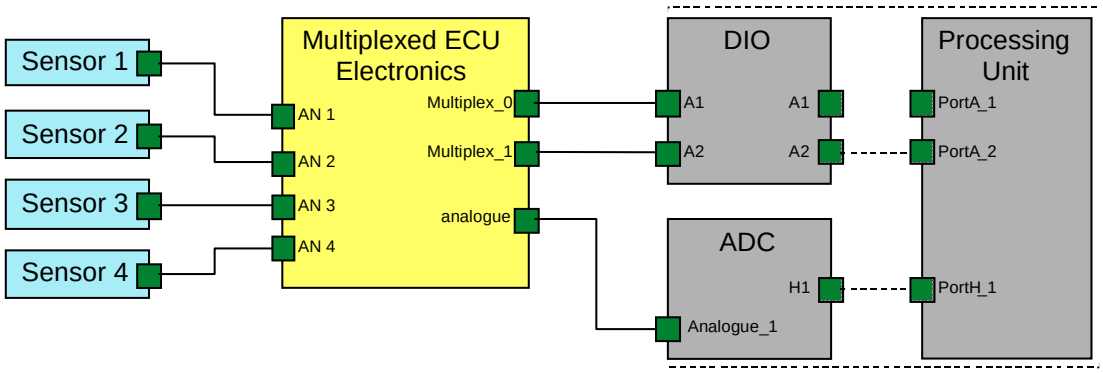

Figure 3-22: Connecting Multiplexed Sensors in the ECU Resource Template

The ECU Electronics performing the multiplexing is encapsulated in one HW Element with several HW Ports describing the interface to the complex ECU Electronics. Now the processing path from the sensors to the processing unit and vice versa can be analysed using the connections between the HW Elements.

3.13.5 Display HW Element

There are several different types of displays. Displays can be divided in three major groups depending of what type of information they can handle.
The simplest ones are the numerical and alpha numerical displays which can show numbers and characters respectively. The most advanced is the graphical, which can

present both characters and graphical objects. Some alphanumerical and graphical displays has an built-in character generator, if this generator is not included in the display the characters must be generated by software.
Displays are produced in the following different technologies:

- **LED-display** Light Emitting Diode display
  LCD displays are in turn divided into two different technologies:
  - TFT Thin Film Transistor
  - STN Super Twisted Nematic
- **OLED-display** Organic Light Emitting Diode display
- **VFD** Vacuum Fluorescent Display
- **LCD** Liquid Crystal Display

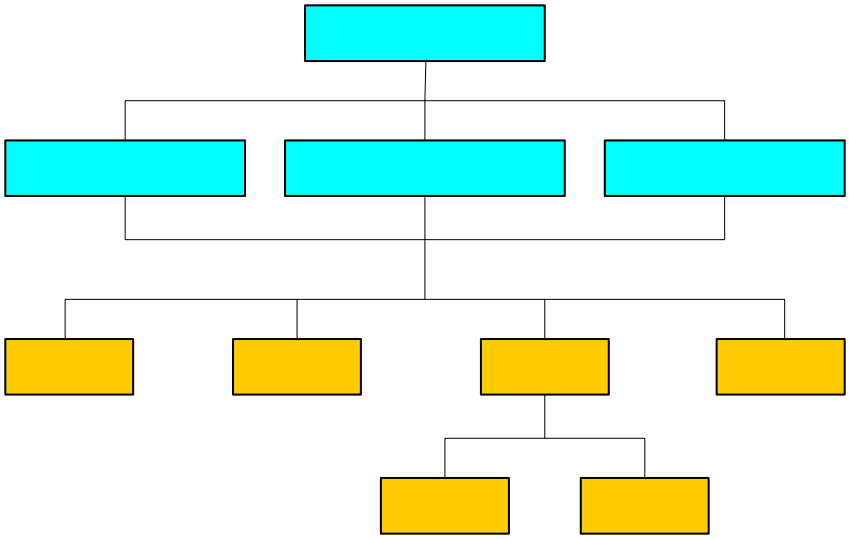

Figure 3-23: Hierarchy of displays

Class	**DisplayHW**			
Package	AUTOSAR Templates::ECUResourceTemplate::Sensor Actuator			
Class Description	The Display HW element derives from HW Sensor Actuator and can be connected to a PU via a serial or parallel interface. For the parallel mode the display can be connected to the data bus and address bus of a PU or to a peripheral. The most common serial interface for displays are: - I2C - SPI The most common physical interface for high resolution displays are the LVDS interface (Low Voltage Differential Signalling). This interface has three different lines for colours (RGB) and additional lines for control signals.			
Base Class(es)	SensorActuatorHW			
Attribute	Datatype	Mul.	Link Type	Attribute Description
backlight	Boolean	1	aggregation	Defines whether the display has a background lighting. This eases the readability of the displays during bad daylight conditions.
brightness	Float	1	aggregation	Defines the visibility of the display.
characterGenerator	Boolean	0..1	aggregation	If the display has an in-built character generator.
characterSet	String	0..1	aggregation	Defines which character set that can be handled by the display. There is a number of standardised character sets like ISO-8859-1 or Unicode.
colour	Enumeration{colour, grey, mono}	1	aggregation	If the display can show objects in monochrome, grey scale or in colour.
resolutionRation	Float	0..1	aggregation	Defines the ratio of the display resolution in horizontal versus vertical direction (e.g. 16 to 9).
resolutionX	Int	0..1	aggregation	Defines the number of the smallest graphical elements which can be displayed in horizontal direction.
resolutionY	Int	0..1	aggregation	Defines the number of the smallest graphical elements which can be displayed in vertical direction.
responseTime	Float	1	aggregation	Defines how long it takes for the display to show a changed element at -20 degree C.
typeOfCharacters	Enumeration{numerical, alpha-numerisal, semi-graphical, graphical}	1	aggregation	If the display can present numerical, alpha-numerical, semi-graphical or graphical objects.
viewingAngle	Int	1	aggregation	Defines the angle range where the display can be observed.

Table 3-58: Display

4 References

4.1 AUTOSAR Documents

For a deeper insight into the modelling and formalization strategies please consult the following documents:

[1] Template UML Profile and Modeling Guide
https://svn2.autosar.org/repos2/22_Releases/
AUTOSAR_TemplateModelingGuide.pdf.

[2] AUTOSAR Glossary
https://svn2.autosar.org/repos2/22_Releases/
AUTOSAR_Glossary.pdf

[3] Specification of the Virtual Functional Bus
https://svn2.autosar.org/repos2/22_Releases/
AUTOSAR_SWS_VFB.pdf

[4] AUTOSAR Technical Overview
https://svn2.autosar.org/repos2/22_Releases/
AUTOSAR_TechnicalOverview.pdf

4.2 External Documents

[1] **HIS API_IODriver Specification**
v2.1.2
http://www.automotive-his.de/download/API_IODriver_2_1_2.pdf

[2] **ISO Open Systems Interconnection (OSI)**
ICS field 35.100

5 Appendix B

5.1 Signal definition

In this subchapter the different kinds of signals related to an ECU are discussed. In Figure 5-1 the signal flow from a sensor to the application software is depicted. Please note that the actuator signal flow resembles the sensor signal flow with reversed signal direction..
The signal labels identify the different kind of signals as discussed below.

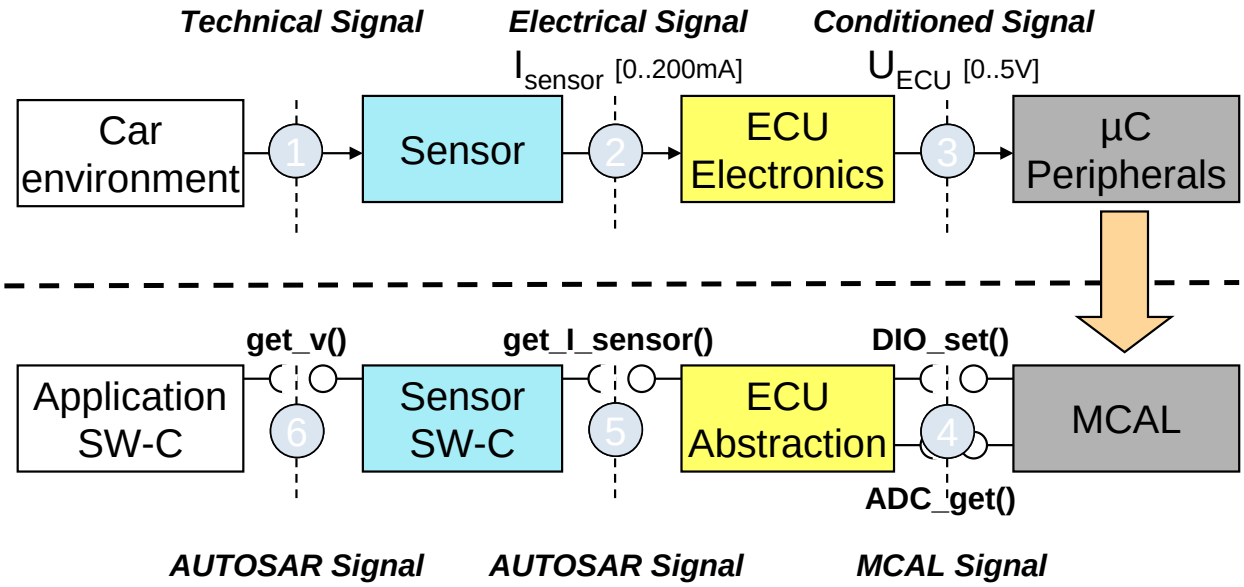

Figure 5-1: Signal Flow in an ECU

5.1.1 Technical Signal (1)

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.

5.1.2 Electrical Signal (2)

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).
To describe the Electrical Signal the same means as for the Technical Signal can be used, limited to electrical current and voltage.

5.1.3 Conditioned Signal (3)

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 electronical 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 and Electrical Signals, but limited to electrical voltage.

5.1.4 MCAL Signal (4)

The processing unit is accessing the Conditioned Signal through some peripheral device that typically digitises the Conditioned Signal into a software representation.
The transformation from the Conditioned Signal to the MCAL Signal has to take the digitalisation error into account in order to provide information about the quality loss between the Technical Signal and the MCAL Signal.

5.1.5 AUTOSAR Signal (5)

The AUTOSAR Signal at point (5) in Figure 5-1 is the software representation of the Electrical Signal. The ECU Abstraction Basic Software Module processes the MCAL Signal and inverts the conversion of the ECU electronics.
The description of the AUTOSAR Signal (5) can be done with the means provided by the AUTOSAR Software Component Template.

5.1.6 AUTOSAR Signal (6)

The AUTOSAR Signal at point (6) in Figure 5-1 is the software representation of the actual Technical Signal that was converted by the sensor.
The description of the AUTOSAR Signal (5) can also be done with the means provided by the AUTOSAR Software Component Template.