Document Title	Basic Software Module Description Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	089

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History

Date	Release	Changed by	Description
2018-10-31	4.4.0	AUTOSAR Release Management	• Added support for Structuring of Measurement and Calibration • Added Use-Case description for Upload and download of data • Added Use-Case description for Hardware Test Manager • Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Added Use-Case description for OBD services • Extended Use-Case descriptions for BSW modules • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Standardization of Rapid Prototyping Support • Improve Callout handling • Extended Use-Case descriptions for BSW modules • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2014-10-31	4.2.1	AUTOSAR Release Management	• Extended splittables for BSW • Added Use-Case descriptions for BSW modules • Editorial changes

— AUTOSAR CONFIDENTIAL —

2014-03-31	4.1.3	AUTOSAR Release Management	• Extended Upstream mapping for BSW • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Added support for indexing of array elements • Various fixes and clarifications • Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Meta-model extensions for multicore use cases • Meta-model extensions for functional modeling of measurement and calibration • Meta-model extensions for rapid prototyping use cases • Improved support for production error modeling • Added several clarifications, explanations and model attributes • Added hyper-links to requirements and model elements • Added appendix C on Upstream Mapping
2011-12-22	4.0.3	AUTOSAR Administration	• Introduced formal specification items and Constraint and Specification History • Added several clarifications, examples and constraints • Improved support for AUTOSAR Services, memory mapping and calibration • New attributes in various parts of the model
2009-12-18	4.0.1	AUTOSAR Administration	• Reworked description of Memory Section • Added chapter on Implementation Conformance Statement

2010-02-02	3.1.4	AUTOSAR Administration	• Harmonized with SW Component Template (triggers, events, local data etc.) • Harmonized with Generic Structure Template • Revision of data types concept • Added variant handling • Added debugging support • Added support for measurement and calibration • General rework of implementation description
2008-08-13	3.1.1	AUTOSAR Administration	• Added OBD Features
2008-02-01	3.0.2	AUTOSAR Administration	• Layout adaptations
2007-12-21	3.0.1	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

References

[1] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[2] Requirements on Basic Software Module Description Template
AUTOSAR_RS_BSWModuleDescriptionTemplate

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[4] Methodology
AUTOSAR_TR_Methodology

[5] Glossary
AUTOSAR_TR_Glossary

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[7] System Template
AUTOSAR_TPS_SystemTemplate

[8] XML Schema Production Rules
AUTOSAR_TPS_XMLSchemaProductionRules

[9] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[10] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[11] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

[12] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions

[13] Specification of RTE Software
AUTOSAR_SWS_RTE

[14] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[15] Meta Data Exchange Format for Software Module Sharing V1.0 (MDX V1.0)
http://www.asam.net
ASAM-AE-MDX-V1_0_0.pdf

[16] Guide to BSW Distribution
AUTOSAR_EXP_BSWDistributionGuide

[17] Virtual Functional Bus
AUTOSAR_EXP_VFB

— AUTOSAR CONFIDENTIAL —

[18] Specification of Operating System
AUTOSAR_SWS_OS

[19] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping

[20] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction

[21] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[22] ASAM MCD 2MC ASAP2 Interface Specification
http://www.asam.net
ASAP2-V1.51.pdf

[23] Overview of Acceptance Tests
AUTOSAR_EXP_AcceptanceTestsOverview

[24] Collection of blueprints for AUTOSAR M1 models
AUTOSAR_MOD_GeneralBlueprints

[25] Specification of Function Inhibition Manager
AUTOSAR_SWS_FunctionInhibitionManager

[26] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager

[27] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager

[28] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager

[29] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[30] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer

1 General Information

1.1 Document Scope

This is the documentation of the template for the Basic Software Module Description (BSWMDT).

The BSWMD is a formal notation of all information belonging to a certain BSW artifact (BSW module or BSW cluster) in addition to the implementation of that artifact. There are several possible use cases for such a description, see 3.1 for details.

The BSWMDT - the *template* to be used for the BSWMD - is the standardized format which has to be used for this description in AUTOSAR. The template is represented in UML as part of the overall AUTOSAR meta-model and is part of the XML schema generated out of this meta-model. This document describes all the elements which belong to this template. These elements are maintained in two different packages of the AUTOSR meta-model:

- The package `BswModuleTemplate` contains all elements which are used exclusively by the BSWMDT.
- Some elements of the BSWMDT, for example for the description of implementation aspects and resource consumption, are used also within the Software Component Template (SWCT). These elements belong to the `CommonStructure` package of the meta-model and are also described within this document.

For clarification, please note that the `GenericStructure` package of the meta-model contains some fundamental infrastructure meta-classes and common patterns that are described in [1]. These elements are also used within the `BswModuleTemplate` but for details refer to [1].

Generic Structure provides details about

- AUTOSAR top level structure
- Commonly used meta-classes and primitives
- Variant handling
- Documentation

This document addresses people who need to have a deeper understanding of the BSWMDT part of the meta-model, for example tool developers and those who maintain the meta-model. It is not intended as a guideline for the BSW developers who will have to provide the actual BSWMD, i.e. who have to "fill out" the template.

For further information on the overall goal of this document refer to the related requirements document, see [2].

Due to the complexity of the meta-model, the text in some class-diagrams in this document is too small to be read on printed paper of normal size. It is recommended to use the electronic document and enlarge these diagrams on a computer screen if required.

1.2 Input Documents

The following input documents have been used to develop the BSWMDT:

- Generic Structure Template [1]
- Requirements on BSW Module Description Template [2]
- General Requirements on Basic Software Modules [3]
- AUTOSAR Methodology [4]
- AUTOSAR Glossary [5]
- Software Component Template [6]
- System Template [7]
- XML Schema Production Rules [8]

1.3 Abbreviations

The following table contains a list of abbreviations used in the scope of this document along with the spelled-out meaning of each of the abbreviations.

Abbreviation	*meaning*
BSW	Basic Software
BSWMD	Basic Software Module Description
BSWMDT	Basic Software Module Description Template
DEM	Diagnostic Event Manager
ECU	Electronic Control Unit
ECUC	ECU Configuration
ICC1, ICC2, ICC3	AUTOSAR Implementation Conformance Class 1...3
ISR	Interrupt Service Routine
ICS	Implementation Conformance Statement
IOC	Inter OS-Application Communication
MC	Measurement and Calibration
MSR	Manufacturer Supplier Relationship
NvM	Non Volatile Memory
NVRAM	Non Volatile RAM
OS	Operating System
RAM	Random Access Memory
ROM	Read-only Memory
SWC	Software Component
SWS	Software Specification
SWCT	Software Component Template

UML	Unified Modeling Language
ARXML	AUTOSAR XML
XML	Extensible Markup Language

Table 1.1: Abbreviations used in the scope of this Document

1.4 Document Conventions

Technical terms are typeset in mono spaced font, e.g. `PortPrototype`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `PortPrototype`s. By this means the document resembles terminology used in the AUTOSAR XML Schema.

This document contains constraints in textual form that are distinguished from the rest of the text by a unique numerical constraint ID, a headline, and the actual constraint text starting after the ⌈ character and terminated by the ⌋ character.

The purpose of these constraints is to literally constrain the interpretation of the AUTOSAR meta-model such that it is possible to detect violations of the standardized behavior implemented in an instance of the meta-model (i.e. on M1 level).

Makers of AUTOSAR tools are encouraged to add the numerical ID of a constraint that corresponds to an M1 modeling issue as part of the diagnostic message issued by the tool.

The attributes of the classes introduced in this document are listed in form of class tables. They have the form shown in the example of the top-level element AUTOSAR:

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:** xml.globalElement=true			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:** xml.sequenceOffset=10
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
fileInfo Comment	FileInfoComment	0..1	aggr	This represents a possibility to provide a structured comment in an AUTOSAR file. **Stereotypes:** atpStructuredComment **Tags:** xml.roleElement=true xml.sequenceOffset=-10 xml.typeElement=false

▽

△

Class	AUTOSAR			
introduction	DocumentationBlock	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to rpresent disclaimers and legal notes. **Tags:** xml.sequenceOffset=20

Table 1.2: AUTOSAR

The first rows in the table have the following meaning:

Class: The name of the class as defined in the UML model.

Package: The UML package the class is defined in. This is only listed to help locating the class in the overall meta model.

Note: The comment the modeler gave for the class (class note). Stereotypes and UML tags of the class are also denoted here.

Base Classes: If applicable, the list of direct base classes.

The headers in the table have the following meaning:

Attribute: The name of an attribute of the class. Note that AUTOSAR does not distinguish between class attributes and owned association ends.

Type: The type of an attribute of the class.

Mul.: The assigned multiplicity of the attribute, i.e. how many instances of the given data type are associated with the attribute.

Kind: Specifies, whether the attribute is aggregated in the class (`aggr` aggregation), an UML attribute in the class (`attr` primitive attribute), or just referenced by it (`ref` reference). Instance references are also indicated (`iref` instance reference) in this field.

Note: The comment the modeler gave for the class attribute (role note). Stereotypes and UML tags of the class are also denoted here.

Please note that the chapters that start with a letter instead of a numerical value represent the appendix of the document. The purpose of the appendix is to support the explanation of certain aspects of the document and does not represent binding conventions of the standard.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([9]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([9]).

2 Requirements Traceability

The following table references the requirements specified in [10] and denotes how they are satisfied in this document.

Requirement	Description	Satisfied by
[RS_BSWMD_00001]	Main source of information on BSW Module ECU Configuration activity and integration	[TPS_BSWMDT_04000] [TPS_BSWMDT_04001] [TPS_BSWMDT_04016] [TPS_BSWMDT_04017] [TPS_BSWMDT_04030] [TPS_BSWMDT_04031] [TPS_BSWMDT_04036] [TPS_BSWMDT_04039] [TPS_BSWMDT_04040] [TPS_BSWMDT_04045] [TPS_BSWMDT_04071] [TPS_BSWMDT_04079] [TPS_BSWMDT_04085] [TPS_BSWMDT_04086]
[RS_BSWMD_00005]	Description of the memory needs of the software implementation	[TPS_BSWMDT_04045] [TPS_BSWMDT_04046] [TPS_BSWMDT_04048] [TPS_BSWMDT_04049] [TPS_BSWMDT_04080]
[RS_BSWMD_00007]	Provide vendor-specific published information	[TPS_BSWMDT_04033] [TPS_BSWMDT_04034]
[RS_BSWMD_00008]	BSW Module Description SHALL be tool processable	[TPS_BSWMDT_04126]
[RS_BSWMD_00009]	Description of peripheral register usage	[TPS_BSWMDT_04032]
[RS_BSWMD_00010]	Compiler version and settings	[TPS_BSWMDT_04043] [TPS_BSWMDT_04068]
[RS_BSWMD_00011]	Guaranteed execution context of API calls	[TPS_BSWMDT_04007] [TPS_BSWMDT_04156]
[RS_BSWMD_00013]	Describe configuration class of ECU Configuration Parameters	[TPS_BSWMDT_04076]
[RS_BSWMD_00014]	Support of BSW Module clusters	[TPS_BSWMDT_04020] [TPS_BSWMDT_04047] [TPS_BSWMDT_04049] [TPS_BSWMDT_04071]
[RS_BSWMD_00015]	Timing requirements	[TPS_BSWMDT_04077]
[RS_BSWMD_00016]	Timing guarantees	[TPS_BSWMDT_04050] [TPS_BSWMDT_04051] [TPS_BSWMDT_04052] [TPS_BSWMDT_04053] [TPS_BSWMDT_04054] [TPS_BSWMDT_04055] [TPS_BSWMDT_04077]
[RS_BSWMD_00024]	Support description of module specific published information	[TPS_BSWMDT_04035] [TPS_BSWMDT_04069]

[RS_BSWMD_00025]	Support for shipment information	[TPS_BSWMDT_04001] [TPS_BSWMDT_04030] [TPS_BSWMDT_04031] [TPS_BSWMDT_04040] [TPS_BSWMDT_04068] [TPS_BSWMDT_04085] [TPS_BSWMDT_04086] [TPS_BSWMDT_04092] [TPS_BSWMDT_04097]
[RS_BSWMD_00026]	Description of supported hardware	[TPS_BSWMDT_04032] [TPS_BSWMDT_04068]
[RS_BSWMD_00027]	Provide Vendor-Specific Module Definition	[TPS_BSWMDT_04033] [TPS_BSWMDT_04069]
[RS_BSWMD_00028]	Development according to the AUTOSAR Generic Structure Template document	[TPS_BSWMDT_04016] [TPS_BSWMDT_04017] [TPS_BSWMDT_04126]
[RS_BSWMD_00029]	Transformation of BSWMD template modeling according to the AUTOSAR XML Schema Production Rules	[TPS_BSWMDT_04126]
[RS_BSWMD_00030]	Publish resource needs for the BSW Scheduler	[TPS_BSWMDT_04006] [TPS_BSWMDT_04019] [TPS_BSWMDT_04020] [TPS_BSWMDT_04027] [TPS_BSWMDT_04067] [TPS_BSWMDT_04072] [TPS_BSWMDT_04128]
[RS_BSWMD_00031]	Description of used memory section names	[TPS_BSWMDT_04046] [TPS_BSWMDT_04047] [TPS_BSWMDT_04049] [TPS_BSWMDT_04080]
[RS_BSWMD_00032]	Recommended ECU Configuration Values	[TPS_BSWMDT_04034]
[RS_BSWMD_00033]	Pre-configured ECU Configuration Values	[TPS_BSWMDT_04034] [TPS_BSWMDT_04035]
[RS_BSWMD_00034]	ECU Configuration Editor and Generation supported tool version information	[TPS_BSWMDT_04041] [TPS_BSWMDT_04042]
[RS_BSWMD_00035]	Provide Standardized Module Definition	[TPS_BSWMDT_04033] [TPS_BSWMDT_04069]
[RS_BSWMD_00037]	Needed libraries	[TPS_BSWMDT_04041] [TPS_BSWMDT_04042]
[RS_BSWMD_00038]	Required execution context of API calls	[TPS_BSWMDT_04007] [TPS_BSWMDT_04156]
[RS_BSWMD_00039]	Identification of implemented API and functions	[TPS_BSWMDT_04000] [TPS_BSWMDT_04002] [TPS_BSWMDT_04008] [TPS_BSWMDT_04009] [TPS_BSWMDT_04028] [TPS_BSWMDT_04066] [TPS_BSWMDT_04130] [TPS_BSWMDT_04153]

[RS_BSWMD_00040]	Identification of required API and functions	[TPS_BSWMDT_04008] [TPS_BSWMDT_04009] [TPS_BSWMDT_04066]
[RS_BSWMD_00041]	Declaration of the provided API argument data types	[TPS_BSWMDT_04002] [TPS_BSWMDT_04007] [TPS_BSWMDT_04009] [TPS_BSWMDT_04010] [TPS_BSWMDT_04011] [TPS_BSWMDT_04012] [TPS_BSWMDT_04066] [TPS_BSWMDT_04091] [TPS_BSWMDT_04130] [TPS_BSWMDT_04153] [TPS_BSWMDT_04156]
[RS_BSWMD_00042]	Description of the required API argument data types	[TPS_BSWMDT_04007] [TPS_BSWMDT_04009] [TPS_BSWMDT_04010] [TPS_BSWMDT_04011] [TPS_BSWMDT_04012] [TPS_BSWMDT_04066] [TPS_BSWMDT_04091] [TPS_BSWMDT_04156]
[RS_BSWMD_00043]	Support description of common published information	[TPS_BSWMDT_04030] [TPS_BSWMDT_04031] [TPS_BSWMDT_04035]
[RS_BSWMD_00044]	Description of generated artifacts	[TPS_BSWMDT_04041] [TPS_BSWMDT_04042]
[RS_BSWMD_00045]	Publish resources needed from AUTOSAR Services	[TPS_BSWMDT_04026] [TPS_BSWMDT_04029] [TPS_BSWMDT_04110] [TPS_BSWMDT_04111] [TPS_BSWMDT_04112] [TPS_BSWMDT_04113] [TPS_BSWMDT_04127]
[RS_BSWMD_00046]	Publish OS resource usage	[TPS_BSWMDT_04006] [TPS_BSWMDT_04072]
[RS_BSWMD_00047]	Modeling of call-chain dependencies between BSW Modules	[TPS_BSWMDT_04018]
[RS_BSWMD_00048]	Tagging of Vendor-Specific Module Definition	[TPS_BSWMDT_04076]
[RS_BSWMD_00049]	Describe optional and required elements	[TPS_BSWMDT_04063] [TPS_BSWMDT_04064] [TPS_BSWMDT_04065] [TPS_BSWMDT_04070] [TPS_BSWMDT_04090]
[RS_BSWMD_00050]	Allow vendor-specific modification of Standardized Module Definition	[TPS_BSWMDT_04033]
[RS_BSWMD_00051]	Description of libraries	[TPS_BSWMDT_04071]
[RS_BSWMD_00052]	Description of the generated RTE	[TPS_BSWMDT_04026] [TPS_BSWMDT_04048]

— AUTOSAR CONFIDENTIAL —

[RS_BSWMD_00053]	Cyclic time based scheduling of BSW Main Functions	[TPS_BSWMDT_04021] [TPS_BSWMDT_04022] [TPS_BSWMDT_04023]
[RS_BSWMD_00054]	Mode Switches for BSW modules shall be supported	[TPS_BSWMDT_04004] [TPS_BSWMDT_04013] [TPS_BSWMDT_04021] [TPS_BSWMDT_04025]
[RS_BSWMD_00055]	Simultaneous Mode transitions	[TPS_BSWMDT_04000] [TPS_BSWMDT_04074]
[RS_BSWMD_00056]	API for Mode switch notification of BSW modules	[TPS_BSWMDT_04004] [TPS_BSWMDT_04013] [TPS_BSWMDT_04014] [TPS_BSWMDT_04019] [TPS_BSWMDT_04025]
[RS_BSWMD_00057]	Triggering of BSW Main Functions by Triggered Events	[TPS_BSWMDT_04005] [TPS_BSWMDT_04015] [TPS_BSWMDT_04021] [TPS_BSWMDT_04023] [TPS_BSWMDT_04024]
[RS_BSWMD_00058]	Simultaneous Triggering by Triggered Events	[TPS_BSWMDT_04000] [TPS_BSWMDT_04074]
[RS_BSWMD_00059]	API for Triggering BSW modules by Triggered Events	[TPS_BSWMDT_04015] [TPS_BSWMDT_04019]
[RS_BSWMD_00060]	Support exclusive areas in BSW Modules and Application Software Components	[TPS_BSWMDT_04073]
[RS_BSWMD_00062]	Provide Measurement and Calibration Support	[TPS_BSWMDT_04026] [TPS_BSWMDT_04027] [TPS_BSWMDT_04056] [TPS_BSWMDT_04057] [TPS_BSWMDT_04058] [TPS_BSWMDT_04059] [TPS_BSWMDT_04060] [TPS_BSWMDT_04061] [TPS_BSWMDT_04062] [TPS_BSWMDT_04078] [TPS_BSWMDT_04087] [TPS_BSWMDT_04088] [TPS_BSWMDT_04114] [TPS_BSWMDT_04115] [TPS_BSWMDT_04128] [TPS_BSWMDT_04168] [TPS_BSWMDT_04169] [TPS_BSWMDT_04170]
[RS_BSWMD_00063]	Allow enabling of providing Activating Bsw Event API	[TPS_BSWMDT_04089]
[RS_BSWMD_00064]	Support optional configuration of ExclusiveArea usage within BSWModuleEntities	[TPS_BSWMDT_04081] [TPS_BSWMDT_04082] [TPS_BSWMDT_04083] [TPS_BSWMDT_04084] [TPS_BSWMDT_04154] [TPS_BSWMDT_04155]

[RS_BSWMD_00065]	Provide Rapid Prototyping Support	[TPS_BSWMDT_04094] [TPS_BSWMDT_04095] [TPS_BSWMDT_04096] [TPS_BSWMDT_04159] [TPS_BSWMDT_04160] [TPS_BSWMDT_04161] [TPS_BSWMDT_04162] [TPS_BSWMDT_04163] [TPS_BSWMDT_04164]
[RS_BSWMD_00066]	BSW inter-partition client-server communication	[TPS_BSWMDT_04098] [TPS_BSWMDT_04099] [TPS_BSWMDT_04100] [TPS_BSWMDT_04102] [TPS_BSWMDT_04103] [TPS_BSWMDT_04104] [TPS_BSWMDT_04105]
[RS_BSWMD_00067]	BSW inter-partition sender-receiver communication	[TPS_BSWMDT_04101] [TPS_BSWMDT_04106] [TPS_BSWMDT_04107]
[RS_BSWMD_00068]	BSW Service Execution on Local or Remote Partition	[TPS_BSWMDT_04108] [TPS_BSWMDT_04109]
[RS_BSWMD_00069]	Configuration for production errors and extended production errors	[TPS_BSWMDT_04110] [TPS_BSWMDT_04111] [TPS_BSWMDT_04112]

Some input requirements cannot (or not completely) be traced down to single specification items found in this document. They are satisfied by BSWMDT in a general way together with other documents as listed in the following:

[TPS_BSWMDT_04126] General meta-model methodology ⌈These requirements are implicitly fulfilled because the BSWMDT follows the general methodology of the AUTOSAR meta-model defined in [1] and [8]. ⌋*(RS_BSWMD_00008, RS_BSWMD_00028, RS_BSWMD_00029)*

[TPS_BSWMDT_04076] ECUC features ⌈These requirements are fulfilled by BSWMDT in general due to the possibility of linking ECU configuration artifacts with a BSWMD. For the specific features see [11]. ⌋*(RS_BSWMD_00013, RS_BSWMD_00048)*

[TPS_BSWMDT_04077] Timing requirements and guarantees ⌈These requirements are fulfilled by the Specification of Timing Extensions, see [12] due to the fact, that timing models can be linked to a BSWMD. The BSWMDT supports this by the specification of meta-model elements for execution time values. ⌋*(RS_BSWMD_00015, RS_BSWMD_00016)*

3 Use Cases and Modeling Approach

3.1 Use Cases

There are several possible use cases for the BSWMDT. The following uses cases can be applied for BSW modules (ICC3 conformance class) or for BSW clusters (ICC2 conformance class) and for libraries. For convenience we often use the word "module" in this document as a synonym for all three types of artifacts.

A library can be seen as a special kind of module which provides services to be used within the basic or application software and which are accessed via direct function calls. Thus the following use cases can also be applied to a library. The main difference between a library and a "normal" BSW module is, that library services can directly be called from application SWCs without going via the RTE. As a consequence, there will be certain restrictions on the model elements which can be used for libraries, e.g. a library should not have scheduled functions. However, these restrictions are currently not formalized.

- The BSWMDT can be used to *specify* a BSW module or cluster (or a set of those) in terms of interfaces and dependencies before it is actually implemented. Details of the internal behavior and implementation are not filled out for this use case. Since the BSWMDT includes variation points, several variants of a BSW module or cluster can be described by a single specification (for details see chapter 11). According to the Methodology [4], artifacts on this level are delivered as **BSW Design Bundle** as a result of the activity **Design Basic Software**.

- The BSWMDT can be used as input for a *conformance test* which tests the conformance of the product (a module, cluster or library) with respect to the AUTOSAR standard. In other words this means that for a conformance test the BSWMD must be usable as an ICS (implementation conformance statement). See 12 for details. According to the Methodology, artifacts on this level are delivered as **BSW Module ICS Bundle**. Note that this delivery has to be distinguished from the following one (the BSW Module Delivered Bundle) because conformance tests require completely configured software.

- The BSWMDT can be used to describe an *actually implemented* BSW module or cluster delivered to the integrator of an AUTOSAR ECU. It will contain details of the internal behavior, the implementation and constraints w.r.t. the specification. Especially, there may be more than one implementation (for example for different processors) which have the same specification. According to the Methodology, artifacts on this level are part of a **BSW Module Delivered Bundle** as a result of the activity **Develop BSW Module** (the same delivery also contains the code, as far it is not generated during integration).

- The BSWMDT does not only serve as an "upstream" template - i.e. as a format for information provided prior to ECU configuration time - but certain parts of the BSWMD can be used by the *integrator* to add further information or adjust information which was not available at the delivery time of the module. In the Methodology, artifacts on this level are part of the **BSW Module Integration Bundle** and they are created or refined during the activity **Integrate Software for ECU**.

This use case includes for example adding documentation about the actual resource consumption and adding information in response to the needs of software components and other BSW modules integrated on the ECU (see chapter 5.4).

- Similar to the last case, the BSWMDT allows to add data which are generated from the 'upstream" descriptions in order to support measurement and calibration tools (see chapter 10).
- The source code which implements the RTE and the BSW Scheduler is typically generated completely during ECU integration. Therefore the parts of the BSWMD which documents the implementation of this code (e.g. version information, memory sections, data structures for calibration support), shall be generated or updated by the RTE generator (see [13] for mandatory parts to be generated).

Details of the work flow for the different use cases are not in the scope of this document (please refer to [4]), but the information to be provided in these various steps influences the meta-model of the BSWMDT.

There is only limited use for the BSWMDT to describe software according to ICC1 conformance class, because in this case the complete BSW (including RTE) on an ECU consists of one single cluster, so that no interfaces or dependencies within the BSW can be described by this template, which means that the relevant parts of the template will be empty. However, even in this case the BSWMDT may be used to document implementation aspects (e.g. the required compiler, resource consumption or vendor specific configuration parameters).

3.2 Three Layer Approach

The meta-model of the BSWMDT consists of three abstraction layers similar to the SWCT. This approach allows for a better reuse of the more abstract parts of the description. An overview is shown in Figure 3.1.

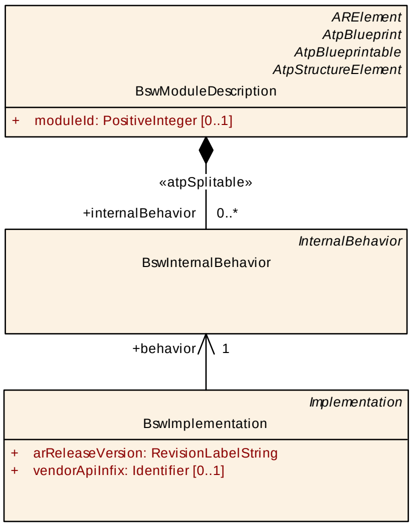

Figure 3.1: Three Layers of the BSW Module Description

The upper layer, the `BswModuleDescription`, contains the specification of all the provided and required interfaces including the dependencies to other modules.

The middle layer, the `BswInternalBehavior`, contains a model of some basic activity inside the module. This model defines the requirements of the module for the configuration of the OS and the BSW Scheduler. There may be several different instances of `BswInternalBehavior` based on the same `BswModuleDescription` (even on the same CPU, for example several drivers adhering to the same `BswModuleDescription`). The term "behavior" has been chosen in analogy to a similar term in the SWCT. Note that it is restricted only to the scheduling behavior here and does not describe the algorithmic behavior of the module or cluster.

The bottom layer, the `BswImplementation` contains information on the individual code. Again, there may be several instances of `BswImplementation` for the same `BswInternalBehavior`.

The usage of `splitable` aggregations resp. references between these layers instead of “ordinary” aggregations allows for more flexibility in the XML artifacts: If for example the `BswInternalBehavior` would aggregate `BswImplementation`, a concrete XML artifact of a `BswInternalBehavior` would have to be duplicated for every instance of `BswImplementation`. By using `splitable` aggregations and references, the layers may be kept in separate files and also the lower layers can be modified in later project phases. This is analog to the inclusion of header files in a C-source file: Several implementation files can share the same header file which typically declares more abstract things as function prototypes and the like. The relation from `BswModuleDescription` to `BswInternalBehavior` is a `splitable` aggregation instead of a reference for semantical reasons and in analogy to the SWCT.

3.3 Several Implementations of the same BSW Module or BSW Cluster

According to the three layer approach, the meta-class BswModuleDescription and an aggregated BswInternalBehavior describe a type of a BSW module or cluster, for which different implementations may exist which are represented by different BswImplementations (note that the name of the meta-class BswModuleDescription is misleading here, because this meta-class does not contain the complete description of a module or cluster).

In case the different implementations of a BSW module or cluster are compiled for different CPUs, the corresponding BSWMDs can be treated as separate artifacts which may share the BswModuleDescription and/or BswInternalBehavior.

In case the implementations are compiled for the same CPU, i.e. are integrated on the same ECU and same address space (for example CAN drivers for several CAN channels), their BSWMDs still should share the BswModuleDescription and (in case it is equal) the BswInternalBehavior, but there must be a mechanism to ensure, that the globally visible C symbols derived from the BswModuleDescription and BswInternalBehavior are unique. This is handled with infixes defined in the implementation part of the BSWMDT (see chapters 5.1 and 7).

3.4 Relation to SwComponentType

Some BSW modules or clusters not only have interfaces to other BSW modules or clusters, but have also more abstract interfaces accessed from Application SW-Cs via the RTE. These BSW modules or clusters can be AUTOSAR Services, part of the ECU Abstraction, or Complex Drivers.

The more abstract interfaces required here are called AUTOSAR Interfaces (see [6] and [5]).

These AUTOSAR Interfaces are described by means of the Software Component Template (SWCT), they consist of ports, port interfaces and their further detailing. The root classes of the SWCT used to describe these elements for BSW modules are ServiceSwComponentType, EcuAbstractionSwComponentType and ComplexDeviceDriverSwComponentType (see [6]) which all are derived from AtomicSwComponentType.

In addition, the function calls from the RTE into these BSW module must be modeled as RunnableEntity-s which are also contained in the SWCT. The root class of the SWCT used to describe the RunnableEntity-s (and a few other things) is called SwcInternalBehavior.

[TPS_BSWMDT_04000] BSW modules with AUTOSAR Interfaces ⌈ Thus for BSW modules or clusters which can be accessed via AUTOSAR Interfaces there must be an XML-artifact defining an AtomicSwComponentType and an SwcInternalBehavior *in addition* to the BSWMD. ⌋*(RS_BSWMD_00001, RS_BSWMD_00039, RS_BSWMD_00055, RS_BSWMD_00058)*

These additional descriptions are required to generate the RTE. Note that in the case of AUTOSAR Services the content of these additional descriptions can vary between different ECUs (for example due to the number of ports the RTE has to create for an AUTOSAR Service) and thus must be created per ECU. The detailed steps for creating these artifacts are described in [6].

In order to trace the dependencies between these additional SWCT descriptions and the associated BSWMD, there is a mapping between the classes SwcInternalBehavior and BswInternalBehavior, see chapter 6.11 for details.

Due to the usage of two different templates for the description of modules mentioned above (i.e. those which have ports for connection to the application software) there is a certain ambiguity how to described the scheduling: With the help of an event model defined in the BSWMDT (see chapter 6 in this document) or with an event model defined in the SwcInternalBehavior of the SWCT. The two different event models result in different interfaces toward the RTE (the BSW-Scheduler-style C-interfaces resp. the SWC-style C-interfaces which are both generated during RTE contract phase). For the standardized AUTOSAR Services defined up to now the SWC-style interfaces are only used for function calls directly related to communication via ports, whereas for e.g. cyclic events the BSW-Scheduler interfaces shall be used. Note, that there is no such rule for the BSW parts which are not standardized (ECU Abstraction and Complex Drivers).

Another special case arises when the BSW Scheduler or an interrupt routine triggers a cyclic function which then has to call into the RTE in order to access an SWC. In order to generate the RTE API with the means of the current SWCT, it is required to specify a RunnableEntity in this case even if it is not triggered by an RTE event.

4 BSW Module Description Overview

Figure 4.1 and the following class table show all the relations of the BSWMDT top layer, the BswModuleDescription.

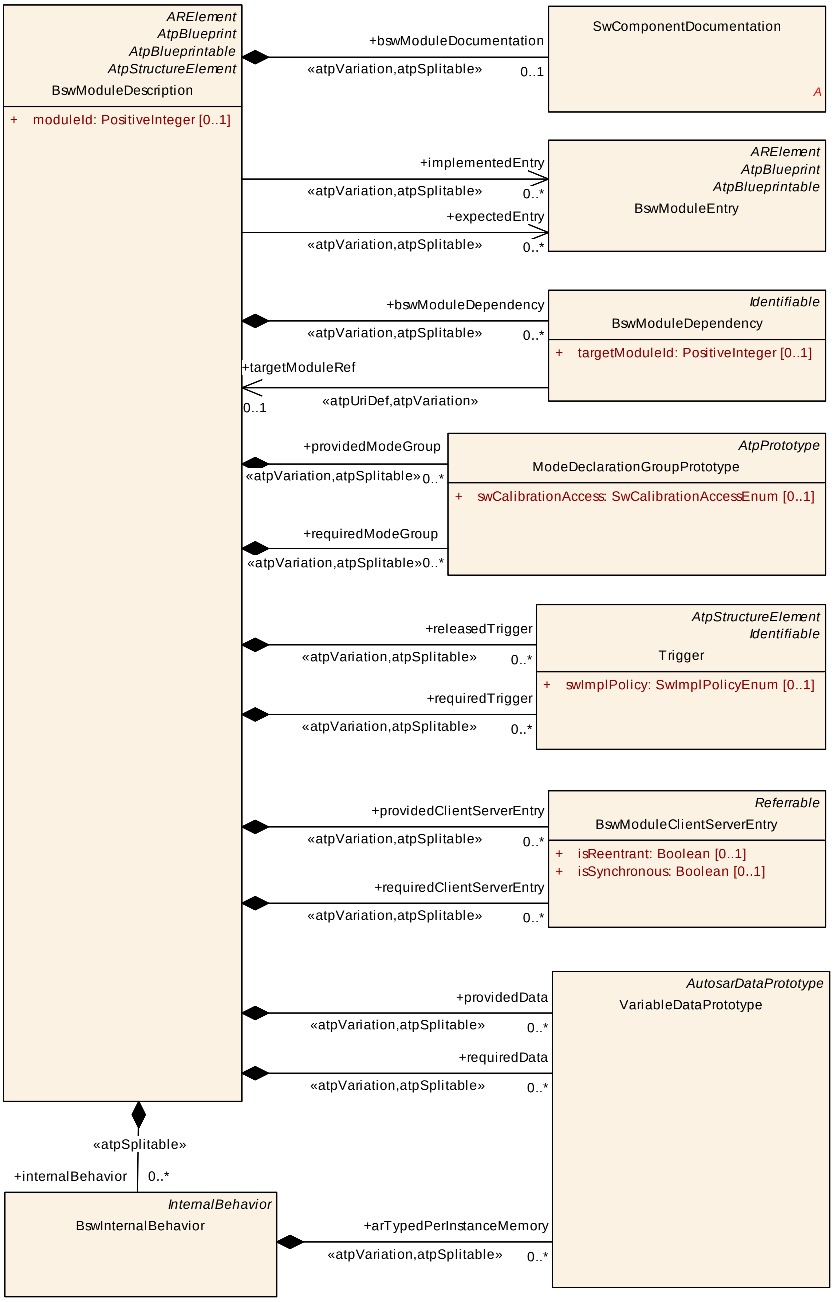

Figure 4.1: BSW Module Description Overview

[TPS_BSWMDT_04079] Usage of module `shortName` ⌈ For a standardized module of ICC3 conformance class the `BswModuleDescription.shortName` shall be chosen identical to the module abbreviation (resp. library abbreviation) defined in [14]. ⌋ *(RS_BSWMD_00001)*

In addition, the `BswModuleDescription` contains an attribute `moduleId`:

[constr_4019] BSW module identifier ⌈ BswModuleDescription.moduleId shall refer to the identifier of the standardized AUTOSAR modules according to [14], if applicable[1]. Otherwise (e.g. for ICC2 clusters) the identifier must either be empty or chosen differently from the ones given in [14]. ⌋*()*

[TPS_BSWMDT_04071] Usage of module identifier and category ⌈ In any case, this identifier in the BSWMD shall be used to document the relation of an artifact to the standard and thus is a useful information for the conformance test. In addition to this, the generic category attribute (inherited from Identifiable) shall be used for a general classification of a BswModuleDescription as shown in the following table. This allows to check for constraints. ⌋*(RS_BSWMD_00001, RS_BSWMD_00014, RS_BSWMD_00051)*

[constr_4020] Categories of BswModuleDescription ⌈ Only categories listed in table 4.1 are allowed. Other values or an empty value are not allowed. ⌋*()*

category	*Explanation*
BSW_MODULE	Specifies a single BSW module (ICC3 granularity).
BSW_CLUSTER	Specifies a BSW module cluster (ICC2 granularity).
LIBRARY	Specifies a Library (not restricted to be used within the BSW).

Table 4.1: BSWMD Categories

[TPS_BSWMDT_04001] Attaching SwComponentDocumentation to a BSWMD ⌈It is possible to attach documentation to a BswModuleDescription by using the meta-class SwComponentDocumentation. This uses the same concept as the documentation for software components and is described in detail in [6].⌋*(RS_BSWMD_00001, RS_BSWMD_00025)*

The meta-class BswModuleEntry describes a single C-function prototype (see chapter 5.1) and is used here as follows:

[TPS_BSWMDT_04002] Provision of BswModuleEntry ⌈The interface exported by a BswModuleDescription is the set of implementedEntry-s provided for the usage by other modules (including "main"-functions called by the BSW Scheduler). ⌋*(RS_BSWMD_00039, RS_BSWMD_00041)*

[TPS_BSWMDT_04153] Usage of BswModuleEntry ⌈The interface required by a BswModuleDescription is the set of expectedEntry-s implemented by other modules. ⌋*(RS_BSWMD_00039, RS_BSWMD_00041)*

[TPS_BSWMDT_04130] Linkage of BswModuleEntry ⌈BswModuleEntry referenced as implementedEntry by one BswModuleDescription and a BswModuleEntry referenced as expectedEntry by another BswModuleDescription are matching if one of the following applies:

- The identical BswModuleEntry is referenced

or

[1] Note that there may be more than one module in an ECU software with the same identifier, e.g. according to the standard Complex Drivers all have the same identifier.

- the 2 BswModuleEntry.shortNames are identical.

⌋*(RS_BSWMD_00039, RS_BSWMD_00041)*

[constr_4093] Entries linked to BswModuleEntrys shall have compatible signature ⌈ Matching BswModuleEntrys according to [TPS_BSWMDT_04130] are compatible if the following condtions are fullfilled:

- both or neither of them define a returnType
- when the returnTypes are defined, the SwServiceArgs in the role returnType shall be compatible
- both define the same number of compatible arguments in same order

⌋*()*

[constr_4094] compatibility of SwServiceArg in role returnType ⌈ SwServiceArg in role returnType are compatible if they are identically typed ⌋*()*

[constr_4095] Compatibility of SwServiceArg in role argument ⌈ SwServiceArg in role returnType are compatible if:

- they are identically typed

and

- if both do have the same shortName

⌋*()*

[constr_4096] Matching BswModuleEntrys should have compatible attributes ⌈ Matching BswModuleEntrys according to [TPS_BSWMDT_04130] should be defined with identical values of the attributes

- callType
- executionContext
- isReentrant
- isSynchronous
- serviceId
- swServiceImplPolicy
- bswEntryKind

⌋*()*

[TPS_BSWMDT_04004] BswModuleDescription.providedModeGroup ⌈With the optional attribute providedModeGroup a BSW module can provide a set of modes (mode group) in order to control other BSW modules which in turn have to declare a corresponding requiredModeGroup.⌋(*RS_BSWMD_00054*, *RS_BSWMD_00056*)

[TPS_BSWMDT_04005] **BswModuleDescription.releasedTrigger** ⌈With the optional attribute releasedTrigger a BSW module can declare a trigger which it releases. A trigger is used to raise events in other BSW modules which in turn have to declare a corresponding requiredTrigger.⌋(*RS_BSWMD_00057*)

[TPS_BSWMDT_04006] **BswModuleDescription.internalBehavior** ⌈By the aggregation of class BswInternalBehavior in BswModuleDescription it is possible to add scheduling aspects to the description.⌋(*RS_BSWMD_00030*, *RS_BSWMD_00046*)

The declaration of function calls, dependencies, triggers and modes make up the interface of a module or cluster to be used for communication among modules on the same memory and processor core. The details are described in chapter 5.

For communication between partition and/or core boundaries, additional declarations are required, see chapter 5.6

For BswInternalBehavior see chapter 6.

Class	**BswModuleDescription**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswOverview			
Note	Root element for the description of a single BSW module or BSW cluster. In case it describes a BSW module, the short name of this element equals the name of the BSW module. **Tags:** atp.recommendedPackage=BswModuleDescriptions			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bswModule Dependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
bswModule Documentation	SwComponent Documentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=bswModuleDocumentation, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=6
expectedEntry	BswModuleEntry	*	ref	Indicates an entry which is required by this module. Replacement of outgoingCallback / requiredEntry. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=expectedEntry, variationPoint.short Label vh.latestBindingTime=preCompileTime

▽

△

Class	BswModuleDescription			
implemented Entry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions, interrupt routines, and callbacks. Replacement of providedEntry / expectedCallback. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=implementedEntry, variation Point.shortLabel vh.latestBindingTime=preCompileTime
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a Bsw ModuleDescription can be distributed over several physical files. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName xml.sequenceOffset=65
moduleId	PositiveInteger	0..1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be optionally chosen. **Tags:** xml.sequenceOffset=5
providedClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module provides a client server entry which can be called from another parition or core.This entry is declared locally to this context and will be connected to the requiredClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
providedData	VariableDataPrototype	*	aggr	Specifies a data prototype provided by this module in order to be read from another partition or core.The providedData is declared locally to this context and will be connected to the requiredData of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
providedMode Group	ModeDeclarationGroup Prototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the required ModeGroups of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with modes provided via ports by an associated ServiceSwComponentType, EcuAbstraction SwComponentType or ComplexDeviceDriverSw ComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
releasedTrigger	Trigger	*	aggr	A Trigger released by this module or cluster. It can be connected to the requiredTriggers of other modules or clusters via the configuration of the BswScheduler. It can also be synchronized with Triggers provided via ports by an associated ServiceSwComponentType, Ecu ▽

▽

△

Class	**BswModuleDescription**			
				△ AbstractionSwComponentType or ComplexDeviceDriver SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=35
requiredClient ServerEntry	BswModuleClientServer Entry	*	aggr	Specifies that this module requires a client server entry which can be implemented on another parition or core.This entry is declared locally to this context and will be connected to the providedClientServerEntry of another or the same module via the configuration of the BSW Scheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
requiredData	VariableDataPrototype	*	aggr	Specifies a data prototype required by this module in oder to be provided from another partition or core.The required Data is declared locally to this context and will be connected to the providedData of another or the same module via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60
requiredMode Group	ModeDeclarationGroup Prototype	*	aggr	Specifies that this module or cluster depends on a certain mode group. The requiredModeGroup is local to this context and will be connected to the providedModeGroup of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=30
requiredTrigger	Trigger	*	aggr	Specifies that this module or cluster reacts upon an external trigger.This requiredTrigger is declared locally to this context and will be connected to the providedTrigger of another module or cluster via the configuration of the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

Table 4.2: BswModuleDescription

5 BSW Interface

This chapter describes the meta-model elements which are used to define the interface level of a BSW module: The description of implementedEntry-s, expectedEntry-s, declaration of mode groups, declaration of triggers, dependencies from other modules and the interfaces for inter-partition communication.

5.1 BSW Module Entry

[TPS_BSWMDT_04007] **BswModuleEntry** ⌈The meta-class BswModuleEntry is used to model the signature of a C-function call⌋*(RS_BSWMD_00011, RS_BSWMD_00038, RS_BSWMD_00041, RS_BSWMD_00042)*, see figure 5.1.

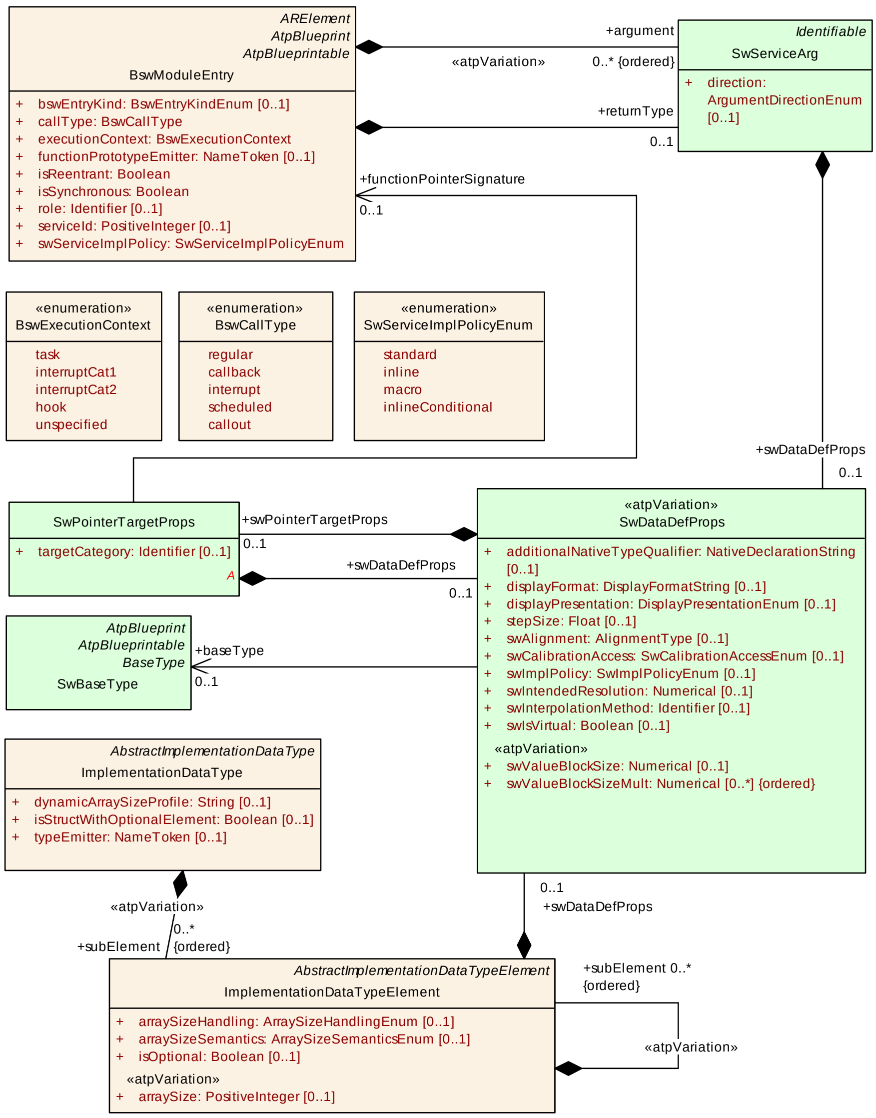

Figure 5.1: Details of meta-class BswModuleEntry

The attributes of meta-class BswModuleEntry are shown in the following table. The attribute serviceId is used to identify the C-function and thus is an important information for an AUTOSAR conformance test.

[constr_4013] BSW service identifier ⌈ For Standardized Interfaces, this identifier is defined in the AUTOSAR Software Specification (SWS) of the module. In case the C-function prototype represented by the entry is not standardized, it still can be used optionally, but its value must differ from the standardized ones. ⌋*()*

[TPS_BSWMDT_04156] Usage of functionPrototypeEmitter ⌈If attribute functionPrototypeEmitter is set to "RTE" the RTE shall generate the function prototypes in the Module Interlink Header File. If the attribute is set to any other value or does not exist, the BSW module shall generate and provide the prototype in its header file(s). ⌋*(RS_BSWMD_00011, RS_BSWMD_00038, RS_BSWMD_00041, RS_BSWMD_00042)*

Class	**BswModuleEntry**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This class represents a single API entry (C-function prototype) into the BSW module or cluster. The name of the C-function is equal to the short name of this element with one exception: In case of multiple instances of a module on the same CPU, special rules for "infixes" apply, see description of class BswImplementation. **Tags:** atp.recommendedPackage=BswModuleEntrys			
Base	*ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, Multilanguage Referrable, PackageableElement, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=45
bswEntryKind	BswEntryKindEnum	0..1	attr	This describes whether the entry is concrete or abstract. If the attribute is missing the entry is considered as concrete. **Tags:** xml.sequenceOffset=40
callType	BswCallType	1	attr	The type of call associated with this service. **Tags:** xml.sequenceOffset=25
execution Context	BswExecutionContext	1	attr	Specifies the execution context which is required (in case of entries into this module) or guaranteed (in case of entries called from this module) for this service. **Tags:** xml.sequenceOffset=30
function Prototype Emitter	NameToken	0..1	attr	This attribute is used to control the generation of function prototypes. If set to "RTE", the RTE generates the function prototypes in the Module Interlink Header File.
isReentrant	Boolean	1	attr	Reentrancy from the viewpoint of function callers: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=15

▽

△

Class	**BswModuleEntry**			
isSynchronous	Boolean	1	attr	Synchronicity from the viewpoint of function callers: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=20
returnType	SwServiceArg	0..1	aggr	The return type belonging to this bswModuleEntry. **Tags:** xml.sequenceOffset=40
role	Identifier	0..1	attr	Specifies the role of the entry in the given context. It shall be equal to the standardized name of the service call, especially in cases where no ServiceIdentifier is specified, e.g. for callbacks. Note that the ShortName is not always sufficient because it maybe vendor specific (e.g. for callbacks which can have more than one instance). **Tags:** xml.sequenceOffset=10
serviceId	PositiveInteger	0..1	attr	Refers to the service identifier of the Standardized Interfaces of AUTOSAR basic software. For non-standardized interfaces, it can optionally be used for proprietary identification. **Tags:** xml.sequenceOffset=5
swServiceImpl Policy	SwServiceImplPolicy Enum	1	attr	Denotes the implementation policy as a standard function call, inline function or macro. This has to be specified on interface level because it determines the signature of the call. **Tags:** xml.sequenceOffset=35

Table 5.1: BswModuleEntry

Enumeration	**BswEntryKindEnum**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	Denotes the mechanism by which the entry into the Bsw module shall be called.
Literal	***Description***
abstract	This BswModuleEntry specifies an abstract signature of C-functions. The signature needs to be implemented by concrete BswModuleEntrys **Tags:** atp.EnumerationValue=0
concrete	This BswModuleEntry specifies a concrete C-function with its signature. **Tags:** atp.EnumerationValue=1

Table 5.2: BswEntryKindEnum

Enumeration	**BswExecutionContext**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	Specifies the execution context required or guaranteed for the call associated with this service.
Literal	***Description***
hook	Context of an OS "hook" routine always **Tags:** atp.EnumerationValue=0

▽

△

Enumeration	**BswExecutionContext**
interruptCat1	CAT1 interrupt context always **Tags:** atp.EnumerationValue=1
interruptCat2	CAT2 interrupt context always **Tags:** atp.EnumerationValue=2
task	Task context always **Tags:** atp.EnumerationValue=3
unspecified	The execution context is not specified by the API **Tags:** atp.EnumerationValue=4

Table 5.3: BswExecutionContext

The RTE and *Basic Software Scheduler* do support the invocation of triggered ExecutableEntity via direct function call in some special cases. Nevertheless it shall be prevented that an ExecutableEntity from a particular execution context calls a triggered ExecutableEntity which requires an execution context with more permissions. The table 5.4 lists the supported combinations.

BswExecutionContext[1]	callee's BswExecutionContext[2]				
	task	**interruptCat2**	**interruptCat1**	**hook**	**unspecified**
task	Supported	Supported	Supported		Supported
interruptCat2		Supported	Supported		Supported
interruptCat1			Supported		Supported
hook					
unspecified	Supported				Supported

Table 5.4: Possible invocation of ExecutableEntitys by direct function call dependent from BswExecutionContext

[constr_4086] invocation of ExecutableEntitys by direct function call dependent from BswExecutionContext ⌈

For example, if we take the fourth column in table 5.4 , the invocation of an ExecutableEntity with an interruptCat1 BswExecutionContext can be implemented with a direct function call if the BswExecutionContext of the caller BswModuleEntry is set to task, interruptCat2, or interruptCat1.

This applies to the invocation of a triggered ExecutableEntity by the SchM_Trigger, SchM_ActMain or Rte_Trigger APIs, or to the invocation of an OnEntry ExecutableEntity, OnTransition ExecutableEntity, OnExit ExecutableEntity or mode switch acknowledge ExecutableEntity by the SchM_Switch or Rte_Switch APIs. For more information about the technical terms refer to [13]

⌋*()*

[1]The execution context of a RunnableEntity is considered as task
[2]The execution context of a RunnableEntity is considered as task

Enumeration	BswCallType
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	Denotes the mechanism by which the entry into the Bsw module shall be called.
Literal	**Description**
callback	Callback (i.e. the caller specifies the signature) **Tags:** atp.EnumerationValue=0
callout	Callout - provide defined means to extend the functionality of an existing module. In this case caller specifies the signature. **Tags:** atp.EnumerationValue=4
interrupt	Interrupt routine **Tags:** atp.EnumerationValue=1
regular	Regular API call **Tags:** atp.EnumerationValue=2
scheduled	Called by the scheduler **Tags:** atp.EnumerationValue=3

Table 5.5: BswCallType

Enumeration	SwServiceImplPolicyEnum
Package	M2::MSR::DataDictionary::ServiceProcessTask
Note	This specifies the legal values for the implementation policies for services (in AUTOSAR: BswModule Entry-s).
Literal	**Description**
inline	inline service definition. **Tags:** atp.EnumerationValue=0
inlineConditional	The service (in AUTOSAR: BswModuleEntry) is implemented in a way that it either resolves to an inline function or to a standard function depending on conditions set at a later point in time. This could be handled by using the AUTOSAR compiler abstraction macros (INLINE, LOCAL_INLINE) and/or by further compiler switches depending on ECU configuration values. **Tags:** atp.EnumerationValue=1
macro	macro service definition. **Tags:** atp.EnumerationValue=2
standard	Standard service and default value, if nothing is defined. **Tags:** atp.EnumerationValue=3

Table 5.6: SwServiceImplPolicyEnum

[constr_4014] Call type and execution context ⌈ Within a given `BswModuleEntry`, the following constraint holds for its attributes:

- `callType=='interrupt'` is not allowed together with `executionContext=='task' or =='hook'`
- `callType=='scheduled'` is not allowed together with `executionContext=='interruptCat1' or =='interruptCat2'`
- other combinations of these two enums are allowed

⌋*()*

[TPS_BSWMDT_04008] C-symbol of BswModuleEntry ⌈The shortName of a BswModuleEntry shall be equal to the name of the C-function implementing it, with one exception: In case of several instances of the same module (e.g. several CAN drivers) on a single CPU, the C-function names must be made unique by inserting additional characters called "infixes". Since each BSW module instance is implemented by a separate piece of code, the infixes are defined as part of each single BswImplementation of the providing module.⌋*(RS_BSWMD_00039, RS_BSWMD_00040)* For details see 7.

As a result, also the code of a module requiring a BswModuleEntry with infixes needs some adjustment, but this adjustment can be made only at integration time. Currently there is no standardized mechanisms for this task in AUTOSAR, but it can be solved with vendor specific configuration parameters (of the requiring modules) whose values are set at integration time according to the infixes of the actually providing modules.

[TPS_BSWMDT_04009] Usage of SwServiceArg ⌈Class SwServiceArg [3] is used to declare the properties of the function arguments as well as of the return type. ⌋*(RS_BSWMD_00039, RS_BSWMD_00040, RS_BSWMD_00041, RS_BSWMD_00042)*

Class	**SwServiceArg**			
Package	M2::MSR::DataDictionary::ServiceProcessTask			
Note	Specifies the properties of a data object exchanged during the call of an SwService, e.g. an argument or a return value. The SwServiceArg can also be used in the argument list of a C-macro. For this purpose the category shall be set to "MACRO". A reference to implementationDataType can optional be added if the actual argument has an implementationDataType.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
direction	ArgumentDirectionEnum	0..1	attr	Specifies the direction of the data transfer. The direction shall indicate the direction of the actual information that is being consumed by the caller and/or the callee, not the direction of formal arguments in C. The attribute is optional for backwards compatibility reasons. For example, if a pointer is used to pass a memory address for the expected result, the direction shall be "out". If a pointer is used to pass a memory address with content to be read by the callee, its direction shall be "in". **Tags:** xml.sequenceOffset=10
swArraysize	ValueList	0..1	aggr	This turns the argument of the service to an array. **Tags:** xml.sequenceOffset=20
swDataDefProps	SwDataDefProps	0..1	aggr	Data properties of this SwServiceArg. **Tags:** xml.sequenceOffset=30

Table 5.7: SwServiceArg

[3] SwServiceArg and its attributes belong to the meta-model part re-engineered from MSR-SW. This subset of MSR-SW is defined by the AUTOSAR meta-model and the XML schema published as part of an AUTOSAR release. The relevant classes are shown as green in the class diagrams. See [6] and [15] for more explanation.

[TPS_BSWMDT_04010] **SwServiceArg.swDataDefProps.implementationDataType** ⌈ shall be used to relate the data definition to a reusable type definition (corresponds to a C typedef). Because ImplementationDataType is an ARElement and itself contains SwDataDefProps, it is possible to declare the required data properties as part of an ImplementationDataType and reuse it as a data type by referring to it. ⌋*(RS_BSWMD_00041, RS_BSWMD_00042)*

ImplementationDataTypeElement within an ImplementationDataType allows to declare composite types (corresponding to C-structs or -arrays).

[TPS_BSWMDT_04011] **SwServiceArg.swDataDefProps.swPointerTargetProps** ⌈ together with its category (see [6]) is used to declare an argument or return type as a pointer to either another data object or to a function: ⌋*(RS_BSWMD_00041, RS_BSWMD_00042)*

Class	**SwPointerTargetProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This element defines, that the data object (which is specified by the aggregating element) contains a reference to another data object or to a function in the CPU code. This corresponds to a pointer in the C-language. The attributes of this element describe the category and the detailed properties of the target which is either a data description or a function signature.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
functionPointerSignature	BswModuleEntry	0..1	ref	The referenced BswModuleEntry serves as the signature of a function pointer definition. Primary use case: function pointer passed as argument to other function. **Tags:** xml.sequenceOffset=40
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of the target data type. **Tags:** xml.sequenceOffset=30
targetCategory	Identifier	0..1	attr	This specifies the category of the target: • In case of a data pointer, it shall specify the category of the referenced data. • In case of a function pointer, it could be used to denote the category of the referenced Bsw ModuleEntry. Since currently no categories for BswModuleEntry are defined it will be empty. **Tags:** xml.sequenceOffset=5

Table 5.8: SwPointerTargetProps

[constr_4021] Implementation policy of function pointer target ⌈
A BswModuleEntry can only be used as target of a function pointer (SwPointerTargetProps.functionPointerSignature), if its swServiceImplPolicy is 'standard'. ⌋*()*

For more information on ImplementationDataType, SwBaseType and the usage of SwServiceArg.category in relation to SwDataDefProps see [6]. Note that due to constraints on SwServiceArg.category (the category VALUE is not allowed), it is not possible to base the declaration of SwServiceArg directly on a SwBaseType, i.e. SwServiceArg.swDataDefProps.baseType must never be set.

— AUTOSAR CONFIDENTIAL —

Function signatures containing the keyword **void** in C deserve special attention:

[constr_4056] BswModuleEntry with no returnType ⌈
In case of an empty return type ("void" in C) the reference BswModuleEntry.returnType shall not be set. ⌋*()*

[constr_4057] BswModuleEntry with no argument ⌈
In case of an empty argument list ("void" in C) no reference BswModuleEntry.argument shall be set. ⌋*()*

Note that nonetheless a SwBaseType exists which represents the **void** type as a pointer target.

[constr_4087] Usage of category "MACRO" ⌈
It is only allowed to use the category "MACRO" for SwServiceArg if the owning BswModuleEntry has its swServiceImplPolicy attribute set to macro.⌋*()*

Furthermore the usage of category "MACRO" defined in chapter "Data Categories" in [6] is restricted to SwServiceArg like defined in [constr_4087]. It is still supported that BswModuleEntry being a macro describes its SwServiceArg with other categories defined in table 5.7 in [6] in order to express the assumed type of the return value and macro argument.

[TPS_BSWMDT_04012] SwServiceArg.direction ⌈ allows to declare the direction of data flow ⌋*(RS_BSWMD_00041, RS_BSWMD_00042)* (the attribute was introduced in R4.0.3 and is optional for backwards compatibility reasons):

Enumeration	**ArgumentDirectionEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	Use cases: • Arguments in ClientServerOperation can have different directions that need to be formally indicated because they have an impact on how the function signature looks like eventually. • Arguments in BswModuleEntry already determine a function signature, but the direction is used to specify the semantics, especially of pointer arguments.
Literal	***Description***
in	The argument value is passed to the callee. **Tags:** atp.EnumerationValue=0
inout	The argument value is passed to the callee but also passed back from the callee to the caller. **Tags:** atp.EnumerationValue=1
out	The argument value is passed from the callee to the caller. **Tags:** atp.EnumerationValue=2

Table 5.9: ArgumentDirectionEnum

This value must be chosen compatible to the role and the formal signature of the SwServiceArg instance:

[constr_4052] BswModuleEntry returnType direction ⌈
BswModuleEntry.returnType.direction must not have the value **in** or **inout**. ⌋*()*

[constr_4053] BswModuleEntry argument direction ⌈
If BswModuleEntry.argument.direction has the value **out** or **inout**, the corresponding BswModuleEntry.argument.swDataDefProps plus eventually referred ImplementationDataType must be such that they result in a pointer declaration.
⌋*()*

It is also possible to specify function signatures containing the keyword **enum** in C[4]:

[TPS_BSWMDT_04091] Function signature containing the keyword enum in C ⌈The respective ImplementationDataType or ImplementationDataTypeElement has to include the string "enum" in the associated SwDataDefProps.additionalNativeTypeDeclaration and use an associated CompuMethod with category TEXTTABLE.

Hints: This information can be used by a code generator to create the correct signature. In case this method is applied to generate C-style enums it should be avoided to use the same CompuMethod as input to a generator (for example the RTE generator) that produces preprocessor literals instead. Otherwise, the enum-literals and the preprocessor-literals might get in conflict. ⌋*(RS_BSWMD_00041, RS_BSWMD_00042)*

5.2 BSW Mode Declaration

[TPS_BSWMDT_04013] Usage of BswModuleDescription.providedModeGroup ⌈ With the optional attribute providedModeGroup a BSW module can declare one or more ModeDeclarationGroupPrototypes, each defining a set of modes (mode group) which is used to control the activity of other BSW modules. Those other modules which require to be controlled by the mode group, must declare a compatible ModeDeclarationGroupPrototype as attribute requiredModeGroup. See figure 5.2. ⌋*(RS_BSWMD_00054, RS_BSWMD_00056)*

[4] Note that the usage of C-enum types is not allowed for signatures created by the RTE generator.

— AUTOSAR CONFIDENTIAL —

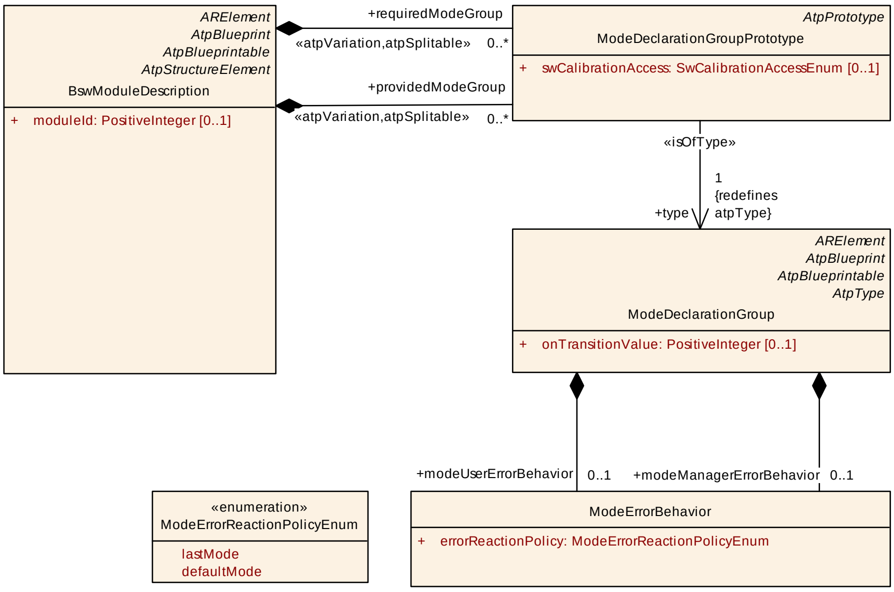

Figure 5.2: Details of BSW Interfaces for modes

For the compatibility of ModeDeclarationGroupPrototypes see [6] [constr_1074]. These declarations allow for the appropriate API generation and coordination of mode switches by the BSW Scheduler. Note that the configuration of the BSW Scheduler actually determines which provided mode group is connected to which required one. This makes the specification of the individual module independent of the overall BSW setup.

A ModeDeclarationGroupPrototype is based on a type definition by meta-class ModeDeclarationGroup. It is possible to use the same ModeDeclarationGroup within the basic software and for software components above the RTE as well, therefore ModeDeclarationGroupPrototype and ModeDeclarationGroup are part of the CommonStructure package of the meta-model. For more information on the semantics of modes see [6].

By aggregation of ModeErrorBehavior a ModeDeclarationGroup can define the behavior of mode managers and/or mode users in case of errors. This is further explained in [6], chapter "Mode Error Behavior".

Class	**ModeDeclarationGroupPrototype**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context. **Tags:** atp.ManifestKind=ExecutionManifest,MachineManifest			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	This allows for specifying whether or not the enclosing ModeDeclarationGroupPrototype can be measured at run-time.

▽

△

Class	**ModeDeclarationGroupPrototype**			
type	ModeDeclarationGroup	1	tref	The "collection of ModeDeclarations" (= ModeDeclaration Group) supported by a component **Stereotypes:** isOfType

Table 5.10: ModeDeclarationGroupPrototype

Note that by aggregating `SwCalibrationAccessEnum` in the role `swCalibrationAccess` `ModeDeclarationGroupPrototype` gains the ability to become measurable. For the constraint on the possible values of `swCalibrationAccess` please refer to [6].

Class	**ModeDeclarationGroup**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:** atp.ManifestKind=ExecutionManifest,MachineManifest atp.recommendedPackage=ModeDeclarationGroups			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
initialMode	ModeDeclaration	1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.
mode Declaration	ModeDeclaration	1..*	aggr	The ModeDeclarations collected in this ModeDeclaration Group. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
modeManager ErrorBehavior	ModeErrorBehavior	0..1	aggr	This represents the ability to define the error behavior expected by the mode manager in case of errors on the mode user side (e.g. terminated mode user).
modeTransition	ModeTransition	*	aggr	This represents the avaliable ModeTransitions of the ModeDeclarationGroup
modeUserError Behavior	ModeErrorBehavior	0..1	aggr	This represents the definition of the error behavior expected by the mode user in case of errors on the mode manager side (e.g. terminated mode manager).
onTransition Value	PositiveInteger	0..1	attr	The value of this attribute shall be taken into account by the RTE generator for programmatically representing a value used for the transition between two statuses.

Table 5.11: ModeDeclarationGroup

Class	**ModeDeclaration**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model. **Tags:** atp.ManifestKind=ExecutionManifest,MachineManifest			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
value	PositiveInteger	0..1	attr	The RTE shall take the value of this attribute for generating the source code representation of this Mode Declaration.

Table 5.12: ModeDeclaration

— AUTOSAR CONFIDENTIAL —

Class	ModeTransition			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	This meta-class represents the ability to describe possible ModeTransitions in the context of a Mode DeclarationGroup.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
enteredMode	ModeDeclaration	1	ref	This represents the entered model of the ModeTransition.
exitedMode	ModeDeclaration	1	ref	This represents the exited mode of the ModeTransition

Table 5.13: ModeTransition

Class	ModeErrorBehavior			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	This represents the ability to define the error behavior in the context of mode handling.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
defaultMode	ModeDeclaration	0..1	ref	This represents the ModeDeclaration that is considered the error mode in the context of the enclosing Mode DeclarationGroup.
errorReaction Policy	ModeErrorReaction PolicyEnum	1	attr	This represents the ability to define the policy in terms of which default model shall apply in case an error occurs.

Table 5.14: ModeErrorBehavior

Enumeration	ModeErrorReactionPolicyEnum
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	This represents the ability to specify the reaction on a mode error.
Literal	***Description***
defaultMode	This represents the ability to switch to the defaultMode in case of a mode error. **Tags:** atp.EnumerationValue=0
lastMode	This represents the ability to keep the last mode in case of a mode error. **Tags:** atp.EnumerationValue=1

Table 5.15: ModeErrorReactionPolicyEnum

In order to avoid conflicts in generated header files which might be included in the same C-file, the following constraint holds:

[constr_4059] Different mode groups referred by a BSWM must have different names ⌈ A `BswModuleDescription` may not refer to different `ModeDeclarationGroup`s (via `requiredModeGroup` and/or `providedModeGroup`) having the same `shortName` but different elements. ⌋*()*

The attributes `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue` and the `category` of `ModeDeclarationGroup` allow to determine the generation of source code from the formal definition. For constraints on these attributes refer to [6].

[TPS_BSWMDT_04014] ModeRequestTypeMap in BSW ⌈Furthermore, it is required to define a ModeRequestTypeMap in order to explicitly specify by which data type a ModeDeclarationGroup is implemented: ⌋*(RS_BSWMD_00056)*

Class	ModeRequestTypeMap			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Specifies a mapping between a ModeDeclarationGroup and an ImplementationDataType. This ImplementationDataType shall be used to implement the ModeDeclarationGroup.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
implementationDataType	AbstractImplementationDataType	1	ref	This is the corresponding AbstractImplementationDataType. It shall be modeled along the idea of an "unsigned integer-like" data type.
modeGroup	ModeDeclarationGroup	1	ref	This is the corresponding ModeDeclarationGroup.

Table 5.16: ModeRequestTypeMap

[constr_4063] Restrictions of ModeRequestTypeMap in BSW ⌈ For every ModeDeclarationGroup referenced by a ModeDeclarationGroupPrototype used in a BswModuleDescription a ModeRequestTypeMap shall exist that points to the ModeDeclarationGroup and also to an eligible ImplementationDataType.

The ModeRequestTypeMap shall be aggregated by a DataTypeMappingSet which is referenced from the BswInternalBehavior that is aggregated by the BswModuleDescription. ⌋*()*

Refer to [6] for restrictions on the ImplementationDataType that can be used for such a mapping. Since provided and required modes are connected via ECU configuration, it is not possible to check constraints on these ImplementationDataTypes on the level of BSWMDs only.

5.3 BSW Trigger Declaration

[TPS_BSWMDT_04015] Usage of Trigger in BSW ⌈With the optional attribute releasedTrigger a BSW module can declare that it releases one or more Triggers which are used to trigger events across BSW modules. Other modules which want to react on such a trigger, must declare a compatible Trigger as attribute requiredTrigger (for the compatibility of Triggers refer to [6] [constr_1038]). These declarations together with the associated event model (see chapter 6.7) allow for the appropriate API generation and coordination by the BSW Scheduler. ⌋*(RS_BSWMD_00057, RS_BSWMD_00059)*

Note that the configuration of the BSW Scheduler actually determines which released trigger is connected to which required one. This makes the specification of the individual module independent of the overall BSW setup.

Class	Trigger			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	A trigger which is provided (i.e. released) or required (i.e. used to activate something) in the given context.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table 5.17: Trigger

A Trigger declaration can optionally set an attribute to define its queuing behavior. This is in more detail explained in [6]. The usage of the enumeration type SwImplPolicyEnum in Trigger.swImplPolicy is restricted in the following way:

[constr_4060] Allowed values of Trigger.swImplPolicy for BSW ⌈ The **only** allowed values for the attribute Trigger.swImplPolicy are either STANDARD (in which case the Trigger processing does not use a queue) or QUEUED (in which case the processing of Triggers positively uses a queue). ⌋*()*

5.4 BSW Module Dependency

5.4.1 General

Figure 5.3 and the following table show the details of class BswModuleDependency. This class represents the expectations of one BSW module or cluster on another BSW module or cluster.

It should be noted, that in order to define a dependency it is not required to have a complete model of the the targeted BswModuleDescription. This allows to maintain each BSWMD separately. Nonetheless, the target module needs to be identified by the attribute BswModuleDependency.targetModuleId and/or the «atpUriDef» reference BswModuleDependency.targetModuleRef. Of course, if both attributes are used their values must be consistent.

Because the module identifier is not always sufficient to identify the target module (e.g. Complex Drivers all have the same module ID), the usage of targetModuleRef is recommended.

A module cannot state a dependency to itself:
[constr_4038] bswModuleDependency must refer to a different module ⌈

- BswModuleDescription.bswModuleDependency.targetModuleId (if given) must differ from BswModuleDescription.moduleId. This does not hold if the value is 254 (used for IO Hardware Abstraction modules) or 255 (used for Complex Driver modules).

- BswModuleDependency.targetModuleRef (if given) must differ from the package location of the BswModuleDescription that owns the BswModuleDependency.

⌋()

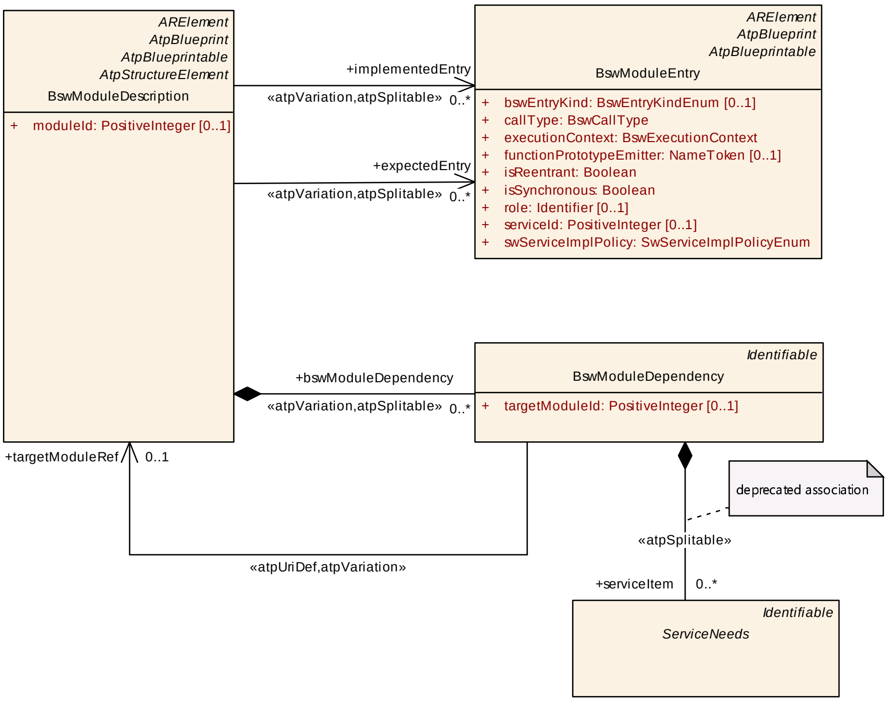

Figure 5.3: Details of class BswModuleDependency

Class	**BswModuleDependency**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This class collects the dependencies of a BSW module or cluster on a certain other BSW module.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
serviceItem	ServiceNeeds	*	aggr	A single item (example: Nv block) for which the quality of a service is defined. The aggregation is marked as «atpSplitable» to allow for extension during the ECU configuration process. This association is deprecated since R4.0.3, since ServiceNeeds shall be associated with the new element BswServiceDependency within the BswInternalBehavior. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName atp.Status=removed xml.sequenceOffset=20

▽

△

Class	**BswModuleDependency**			
targetModuleId	PositiveInteger	0..1	attr	AUTOSAR identifier of the target module of which the dependencies are defined. This information is optional, because the target module may also be identified by targetModuleRef. **Tags:** xml.sequenceOffset=5
targetModule Ref	BswModuleDescription	0..1	ref	Reference to the target module. It is an «atpUriDef» because the reference shall be used to identify the target module without actually needing the description of that target module. **Stereotypes:** atpUriDef; atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=7

Table 5.18: BswModuleDependency

The set of expectedEntry-s represent the interface imported from another module in terms of function calls.

5.4.2 Dependency and Packages

It is important to note that via BswModuleDependency the module description that owns the dependency refers to model elements which are also referred by the description of the module it depends on. This holds especially for instances of BswModuleEntry but also for other ARElements like data types referred from there. In order to avoid inconsistencies, one should put such mutually used M1 elements under a well defined location in terms of ARPackages.

Rules for the package location of standardized M1 model elements are given in [1], chapter *Identifying M1 elements in packages*. As a consequence we can state:

[TPS_BSWMDT_04016] Location of standardized BswModuleEntry-s ⌈ Instances of standardized BswModuleEntrys defined for an AUTOSAR module <module>[5] shall be located under a package AUTOSAR_<module>/BswModuleEntrys/⌋ *(RS_BSWMD_00001, RS_BSWMD_00028)*

for example

AUTOSAR_Can/BswModuleEntrys/Can_SetControllerMode

[TPS_BSWMDT_04017] Reference to standardized BswModuleEntry-s ⌈ If a BSWMD refers to a standardized BswModuleEntry via implementedEntry or expectedEntry it shall also use the path AUTOSAR_<module>/BswModuleEntrys/ thus indicating that it relies on the AUTOSAR compliant implementation of the referred API functions. ⌋*(RS_BSWMD_00001, RS_BSWMD_00028)*

[5]Here <module> is the module abbreviation of the standardized ICC3 module to which the API is belongs.

It is highly recommended to follow an analog pattern (but not starting with AUTOSAR) for the package names of non-standardized ARElements too.[6] If a BSWMD refers in its dependency to a path like

```
<vendor_specific_prefix>_<module>/BswModuleEntrys/
```

for example

```
VendorX_Can/BswModuleEntrys/Can_SpecialFunction
```

this would indicate that the BSWMD relies on a vendor specific function resp. callback of the referred module (for example *Can*).

In addition, the value of targetModuleRef should be set to

```
VendorX_Can/BswModuleDescriptions/Can
```

In this example, we would instead of *Can* use a non-standardized module name if the referred module is a Complex Driver. In this case, the module name would be equal to the BswModuleDescription.shortName of the BSWMD of that Complex Driver.

5.4.3 Dependency: Examples and Constraints

Note that expectedEntry-s do also include calls in interrupt context. An example could be as follows:

Consider we want to describe the callback-dependencies of an external EEPROM driver module from the (standardized) AUTOSAR SPI module. Consider the SPI driver offers an outgoing callback "EndJobNotification" always called in interrupt context. To describe the dependency we would have to create an instance BswModuleDescription.bswModuleDependency and do the following assignments:

- bswModuleDependency.targetModuleId = module identifier of the SPI driver (alternatively, we could use bswModuleDependency.targetModuleRef)

Figure 5.4 shows another example for an M1 model of a dependency between two hypothetical BSW modules. The dependency includes one regular function implemented by the lower layer module “Any” (which could stand for an MCAL module) and two callbacks implemented by the upper layer Module “MyComplexDriver”[7].

[6] The recommended name of the package that should be the immediate container of instances of a given meta-class derived from ARElement is defined as an UML-tag and can be seen in the respective class table.

[7] The AUTOSAR BSW architecture distinguishes the semantics of *callback* and *callout*: Whereas a *callback* notifies something to an upper layer module, a *callout* is used to add functionality to the calling module. Within the BSWMD, these two mechanisms can be described in the same way.

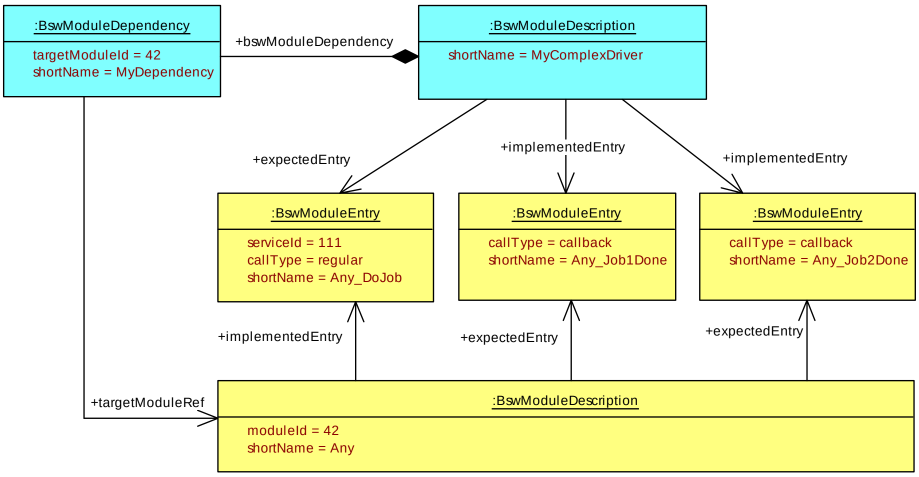

Figure 5.4: Example for an M1 model of a dependency between two modules

Note that the model of the outgoing callbacks can (in general) only be completed at configuration time, because the number and names of the BswModuleEntrys used as callbacks might be unknown at the time the BSWMD of the lower level module is delivered. However at that point in time it is still possible to describe the signature of the callback function by using an AtpBlueprint of the intended BswModuleEntry and to deliver this description together with the BSWMD of the lower level module. For more details on the blueprint concept refer to [9].

In addition to direct function calls, two BSW modules can also be connected via triggers or modes declared in their interfaces. This does not show up as a dependency, because the actual connection is created by the configuration of the BSW Scheduler.

Note that a BswModuleDependency can also contain ServiceNeeds. However, this is a deprecated relationship (only allowed for backwards compatibility) since the declaration of ServiceNeeds has been moved to the internal behavior level, see chapter 13.

5.5 BswModuleEntry Relationship Set

The BswEntryRelationshipSet describes a collection of BswEntryRelationships. A BswEntryRelationship describes a relationship between two BswModuleEntrys and the type of relationship. This is typically used to express that a concrete BswModuleEntry is derived from an abstract BswModuleEntry. In this case the bswEntryRelationshipType is set to drivedFrom, the BswEntryRelationship.from references the abstract BswModuleEntry and the BswEntryRelationship.to references the concrete BswModuleEntry.

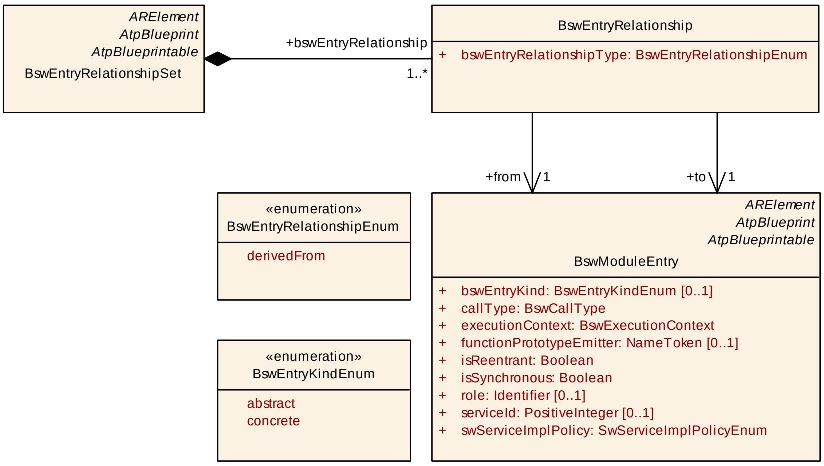

Figure 5.5: BswEntryRelationshipSet

Class	**BswEntryRelationshipSet**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	Describes a set of relationships between two BswModuleEntrys. **Tags:** atp.recommendedPackage=BswEntryRelationshipSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bswEntry Relationship	BswEntryRelationship	1..*	aggr	Relationship between two BswModuleEntrys.

Table 5.19: BswEntryRelationshipSet

Class	**BswEntryRelationship**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	Describes a relationship between two BswModuleEntrys and the type of relationship.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bswEntry Relationship Type	BswEntryRelationship Enum	1	attr	Denotes the type of the relationship. **Tags:** xml.sequenceOffset=5
from	BswModuleEntry	1	ref	Type of relationship that refers to the abstract BswModule Entry. Please notice that in this case the bswEntryRelationship Type shall be set to drivedFrom.
to	BswModuleEntry	1	ref	Type of relationship that refers to the concrete Bsw ModuleEntry

Table 5.20: BswEntryRelationship

Enumeration	**BswEntryRelationshipEnum**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces
Note	
Literal	*Description*
derivedFrom	Describes that the BswModuleEntry referenced as "to" needs to have the same signature as the "abstract" BswModuleEntry referenced as "from". **Tags:** atp.EnumerationValue=0

Table 5.21: BswEntryRelationshipEnum

5.6 BSW Inter-Partition Interface

5.6.1 Overview

AUTOSAR BSW has the ability to communicate across partition boundaries which includes communication across processor core boundaries.[8]

While this is in general possible over the RTE by using Ports and Software Components (e.g. Complex Drivers) on top of the BSW modules, there exist more efficient mechanisms of doing this with the help of “glue code” provided by the BSW Scheduler part of the RTE. See [16] for a detailed guideline.

These mechanisms follow the Client-Server communication pattern or the Sender-Receiver communication pattern of the VFB - see [17] - but cannot be used for inter-ECU communication.

The required meta-model part is shown in Figure 5.6.

[8]AUTOSAR currently supports at most one BSW partition per core. However, the meta-model part described here is independent on this restriction.

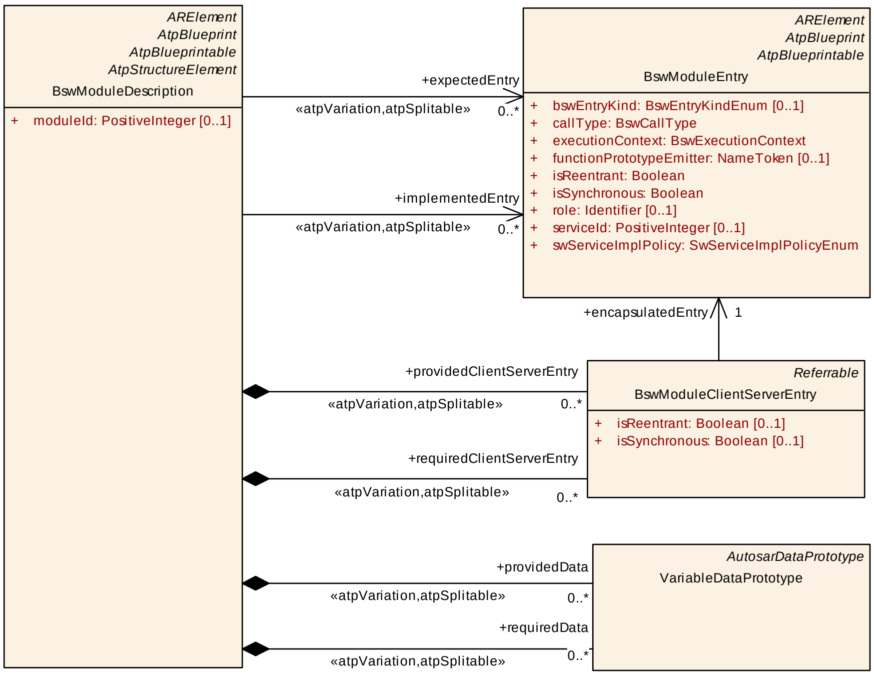

Figure 5.6: BSW Interfaces for inter-partition and multicore communication

5.6.2 Client-Server

Class	**BswModuleClientServerEntry**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswInterfaces			
Note	This meta-class represents a single API entry into the BSW module or cluster that has the ability to be called in client-server fashion via the BSW Scheduler. In this regard it is more special than BswModuleEntry and can be seen as a wrapper around the Bsw ModuleEntry to which it refers (property encapsulatedEntry). **Tags:** atp.recommendedPackage=BswModuleEntrys			
Base	*ARObject*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
encapsulated Entry	BswModuleEntry	1	ref	The underlying BswModuleEntry. **Tags:** xml.sequenceOffset=5
isReentrant	Boolean	0..1	attr	Reentrancy from the viewpoint of clients invoking the service via the BSW Scheduler: • True: Enables the service to be invoked again, before the service has finished. • False: It is prohibited to invoke the service again before is has finished. **Tags:** xml.sequenceOffset=10

▽

△

Class	BswModuleClientServerEntry			
isSynchronous	Boolean	0..1	attr	Synchronicity from the viewpoint of clients invoking the service via the BSW Scheduler: • True: This calls a synchronous service, i.e. the service is completed when the call returns. • False: The service (on semantical level) may not be complete when the call returns. **Tags:** xml.sequenceOffset=15

Table 5.22: BswModuleClientServerEntry

[TPS_BSWMDT_04098] Declaration of BswModuleClientServerEntry ⌈ With the optional attribute providedClientServerEntry a BSW module can declare that it provides a BswModuleClientServerEntry that can be used in the server role for client-server communication across partition boundaries.[9]. The client module (which may be a different or the same module) must declare a compatible BswModuleClientServerEntry as attribute requiredClientServerEntry. These declarations together with the associated event model (see chapter 6.7) allow for the appropriate API generation and coordination by the BSW Scheduler. ⌋*(RS_BSWMD_00066)*

[constr_4074] Compatibility of BswModuleClientServerEntry-s ⌈Two BswModuleClientServerEntry-s are compatible if and only if all of the following conditions hold:

- Their reentrancy values are identical. These values are taken from the attribute isReentrant or, if this is undefined, from encapsulatedEntry.isReentrant.
- Their synchronicity values are identical. These values are taken from the attribute isSynchronous or, if this is undefined, from encapsulatedEntry.isSynchronous.
- The two BswModuleEntry-s referred as encapsulatedEntry have completely identical attributes.

⌋*()*

Note that the configuration of the BSW Scheduler determines which providedClientServerEntry is actually connected to which requiredClientServerEntry. This makes the specification of the individual module independent of the overall BSW setup.

[TPS_BSWMDT_04099] Semantics of BswModuleClientServerEntry attributes ⌈ The optional attributes BswModuleClientServerEntry.isReentrant and BswModuleClientServerEntry.isSynchronous can have different values than the corresponding attributes of the referred BswModuleClientServerEntry.encapsulatedEntry, because the first two attributes describe properties seen by a

[9]This does not exclude configurations where client and server are executed in the same partition.

client calling via the BSW Scheduler wheres the latter contains the properties seen by direct callers.

If one of these attributes is undefined, its value is considered as equal to the respective attribute of the referred encapsulatedEntry. ⌋*(RS_BSWMD_00066)*

[TPS_BSWMDT_04100] Different ways of referring BswModuleEntry ⌈ In a given BSWMD a BswModuleEntry, i.e. the declaration of a function signature, can be referred in two different ways:

1. as part of the “direct” module interface, namely as implementedEntry or expectedEntry
2. as part of the client-server “remote” interface via BswModuleClientServerEntry.encapsulatedEntry

The two possibilities may be combined for one BswModuleEntry in the same BSWMD if the entry is called directly and via client-server as well. However, if the BswModuleEntry is *only* used in client-server manner it is recommended not to use the first possibility *in addition*.

Especially, it is not required to state a bswModuleDependency in this case, since the actual connection is done at configuration time and the two module environments need not to exchange header files. ⌋*(RS_BSWMD_00066)*

Client-Server communication via the BSW Scheduler implies some constraints on the nature of the function call on the server side:

[constr_4076] Constraints on BswModuleEntry used for Client-Server ⌈ A BswModuleEntry used in the role BswModuleClientServerEntry.encapsulatedEntry must have attribute values as follows:

- callType must be regular or callback.
- executionContext must be task.

⌋*()*

5.6.3 Sender-Receiver

[TPS_BSWMDT_04101] Declaration of providedData and requiredData ⌈ With the optional attribute providedData a BSW module can declare that it provides a VariableDataPrototype that can be used in the sender role for sender-server communication across partition boundaries.[10] The receiver module (which may be a different or the same module) shall declare a compatible VariableDataPrototype

[10]This does not exclude configurations where sender and receiver are executed in the same partition.

as attribute requiredData (for the compatibility of VariableDataPrototypes refer to [6] [constr_1068]). These declarations together with the associated event model (see chapter 6.7) and ECU configuration allow for the appropriate API generation and coordination by the BSW Scheduler. ⌋*(RS_BSWMD_00067)*

[constr_4075] Constraints for providedData and requiredData ⌈ Sender-Receiver communication in BSW is restricted to the pattern of so-called *explicit communication* (in the same way as described for software components in [6]) with queued behavior. This leads to some constraints for the VariableDataPrototype referred in the role BswModuleDescription.providedData or BswModuleDescription.requiredData:

- It shall not have an initValue.
- Its swDataDefProps.swImplPolicy shall be set to queued.
- Its swDataDefProps.calibrationAccess shall be set to notAccessable.

There are no further formal constraints on the attributes of the VariableDataPrototype to be used in these roles or on the underlying AutosarDataPrototype. ⌋*()*

Note that the ECU configuration of the BSW Scheduler determines which providedData is actually connected to which requiredData. This makes the specification of the individual module independent of the overall BSW setup.

5.7 Count Value Sets

5.7.1 Background

When a high number of software components are integrated on an ECU the allocation of the RTE communication buffers (e.g. for implicit communication) or allocation of specific functions is getting a crucial performance factor. With the knowledge how often RTE API is invoked and consequential how often accesses to data are executed it is possible to optimize the implementation. For instance buffers with a high access frequency are put to a memory with low latency.

5.7.2 AccessCountSets

The meta-class AccessCountSet provides a collection of count values how often an implementation invokes RTE / SchM APIs provided for certain AbstractAccessPoint of a specific ExecutableEntity.

Class	AccessCountSet			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::AccessCount			
Note	This meta-class provides a set of count values evaluated according to the rules of a specific countProfile.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessCount	AccessCount	1..*	aggr	Count value for a AbstractAccessPoint. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
countProfile	NameToken	1	attr	This attribute defines the name of the count profile used to determine the AccessCount.value numbers.

Table 5.23: AccessCountSet

Class	AccessCount			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::AccessCount			
Note	This meta-class provides one count value for a AbstractAccessPoint.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessPoint	AbstractAccessPoint	1	ref	AbstractAccessPoint for which the count value is applicable.
value	PositiveInteger	1	attr	This attribute represents the number of determined accesses **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 5.24: AccessCount

Class	***AbstractAccessPoint*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::AccessCount			
Note	Abstract class indicating an access point from an ExecutableEntity.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	AsynchronousServerCallResultPoint, ExternalTriggeringPointIdent, InternalTriggeringPoint, ModeAccessPointIdent, ModeSwitchPoint, ParameterAccess, *ServerCallPoint*, VariableAccess			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 5.25: AbstractAccessPoint

[TPS_BSWMDT_04140] `AccessCount.value` describes an intrinsic property ⌈
The `AccessCount.value`s in an `AccessCountSet` are statements about the implementation of single `ExecutableEntity`s with respect to RTE/SchM API usage when the code is executed. Those values are independent from the later integration of the respective `AbstractAccessPoint` of a specific `ExecutableEntity`s. ⌋*()*

This means, that the numbers are a characteristic of a single `AbstractAccessPoint` of a specific `ExecutableEntity` and depending on the resulting call graph it might be required to calculate the consolidated numbers of accesses as the basis for the integration decisions. For instance if a server runnable is called **5** times in a loop by direct function call from a periodically scheduled runnable, the intrinsic count values for the data accesses in the server runnable needs to multiplied by **5** in order to get the consolidated effective number of access per time period.

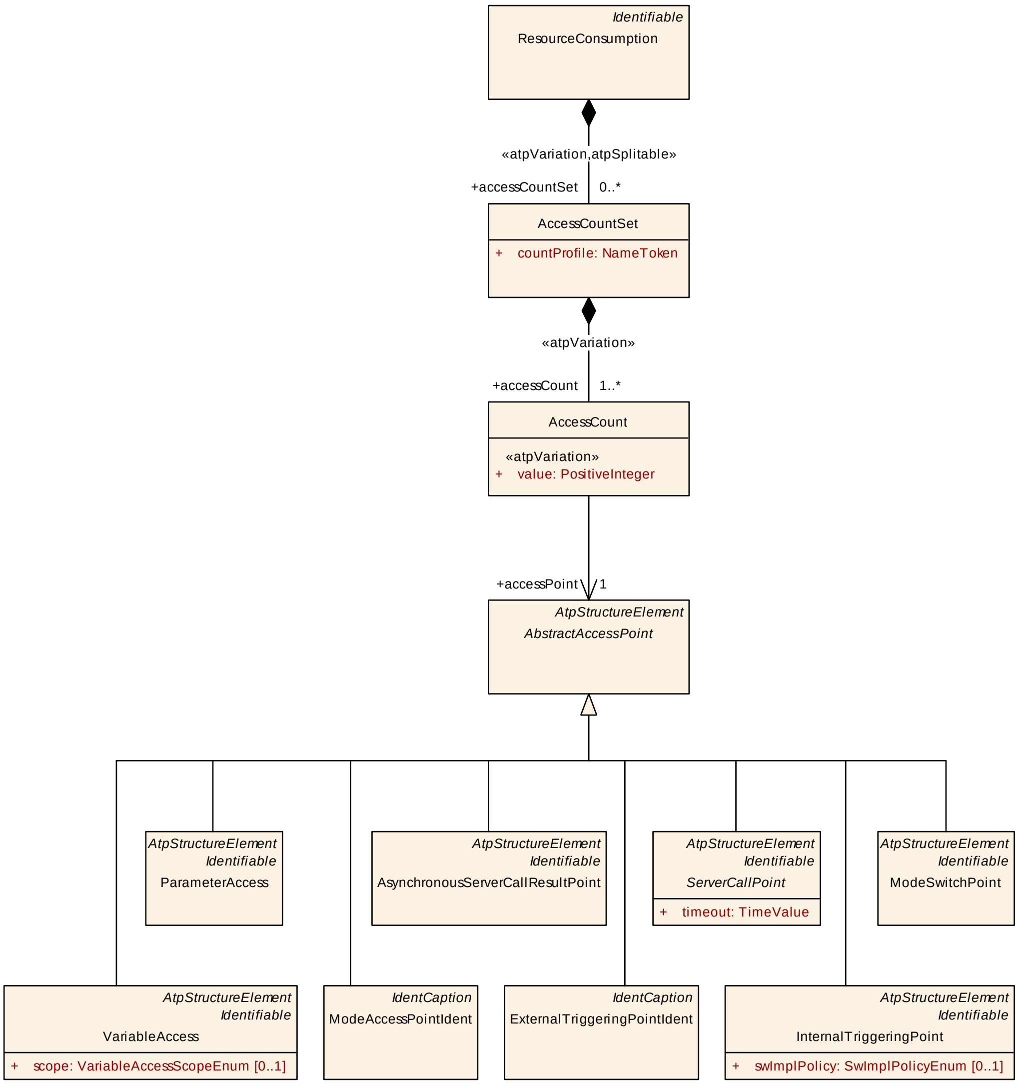

Figure 5.7: Overview AccessCountSet

In general the provider of the count values and the consumer of the count values need a common understanding how the values are determined in order to consider them appropriately for the optimization. Since the topic of optimizations may be a subject of further enhancements the `AccessCountSet` provides information about the counting strategy with the attribute `countProfile`.

[TPS_BSWMDT_04141] The attribute `countProfile` denotes the counting rules ⌈ The attribute `countProfile` denotes the set of applicable counting rules used to determine the `AccessCount.value`s. ⌋*()*

[TPS_BSWMDT_04142] Standardized values of attribute `countProfile` ⌈ AUTOSAR defines following standardized values of the attribute `countProfile`:

- DISTINGUISH_SINGULAR_ACCESSES

⌋*()*

Please note that further count profiles might be defined in future.

5.7.3 Definition of countProfile: DISTINGUISH_SINGULAR_ACCESSES

The purpose of the countProfile DISTINGUISH_SINGULAR_ACCESSES is to determine on one hand the typical frequentness of RTE API invocation supporting the adjustment of the memory allocation. On the other hand it shall be information rich enough to provide precise information about the maximum number of access to data via implicit communication pattern which can also be used to optimize the memory allocation or even to question the existence of buffers at all. The AccessCountSets provide a collection of count values how often an implementation invokes RTE / SchM APIs provided for certain AbstractAccessPoint of a specific ExecutableEntity. In case of implicit communication accesses to data the RTE API may return data references to the location in memory where the data can be accessed. For that kind of AbstractAccessPoint the counting of the API invocations would not be sufficient but rather the number of implemented access to composite data elements via the data reference is important.

[TPS_BSWMDT_04143] countProfile DISTINGUISH_SINGULAR_ACCESSES, Explicit Communication, single access ⌈ The AccessCount.value applied to a VariableAccess in role dataReceivePointByArgument, dataReceivePointByValue, dataSendPoint or a VariableAccess in role writtenLocalVariable / readLocalVariable referencing an explicitInterRunnableVariable shall be given as **1**, if the according implementation of the RunnableEntity invokes the according RTE API at most once per execution of the RunnableEntity in any condition. ⌋*()*

[TPS_BSWMDT_04144] countProfile DISTINGUISH_SINGULAR_ACCESSES, Explicit Communication, multiple accesses ⌈ The AccessCount.value applied to a VariableAccess in role dataReceivePointByArgument, dataReceivePointByValue, dataSendPoint or a VariableAccess in role writtenLocalVariable / readLocalVariable referencing an explicitInterRunnableVariable shall be given greater than **1**, if the according implementation of the RunnableEntity may invoke the according RTE API multiple times per execution of the RunnableEntity. Thereby the AccessCount.value shall state the number of invocations in typically execution conditions. ⌋*()*

[TPS_BSWMDT_04145] countProfile DISTINGUISH_SINGULAR_ACCESSES, Implicit Communication and parameter accesses, single access ⌈ The AccessCount.value applied to a ParameterAccess or a VariableAccess in role dataWriteAccess, dataReadAccess or a VariableAccess in role writtenLocalVariable or readLocalVariable referencing an implicitInterRunnableVariable shall be given as **1**, if the according implementation of the ExecutableEntity access at most one-time one primitive data or at most one-time one primitive composite data element per execution of the RunnableEntity in any condition. ⌋()

[TPS_BSWMDT_04146] countProfile DISTINGUISH_SINGULAR_ACCESSES, Implicit Communication and parameter accesses, multiple accesses ⌈ The AccessCount.value applied to a ParameterAccess or a VariableAccess in role dataWriteAccess, dataReadAccess or a VariableAccess in role writtenLocalVariable or readLocalVariable referencing an implicitInterRunnableVariable shall be given greater than **1**, if the according implementation of the RunnableEntity may access primitive data multiple times or multiple primitive composite data element per execution of the RunnableEntity. Thereby the AccessCount.value shall state the number of accesses to primitive data or accesses to primitive composite data elements in typically execution conditions. ⌋()

For instance accessing a structure with **3** elements of type uint8, uint16 and uint64 in a loop executed **5** times counts a **15**.

For instance a RunnableEntity accesses an array of size **42** in a way, that for each execution of the RunnableEntity exactly one element of this array is read by implicit access. This counts as **1**.

[TPS_BSWMDT_04147] countProfile DISTINGUISH_SINGULAR_ACCESSES, Server calls, issued Triggers, Mode Switch Notifications, single access ⌈ The AccessCount.value applied to a ServerCallPoint, AsynchronousServerCallResultPoint, InternalTriggeringPoint, ExternalTriggeringPoint, ModeSwitchPoint, ModeAccessPoint shall be given as **1**, if the according implementation of the ExecutableEntity invokes the according RTE API at most once per execution of the ExecutableEntity in any condition. ⌋()

[TPS_BSWMDT_04148] countProfile DISTINGUISH_SINGULAR_ACCESSES, Server calls, issued Triggers, Mode Switch Notifications, multiple accesses ⌈ The AccessCount.value applied to a ServerCallPoint, AsynchronousServerCallResultPoint, InternalTriggeringPoint, ExternalTriggeringPoint, ModeSwitchPoint, ModeAccessPoint shall be given greater than **1**, if the according implementation of the ExecutableEntity invokes the according RTE API multiple times per execution of the ExecutableEntity. Thereby the AccessCount.value shall state the number of invocations in typically execution conditions. ⌋()

For instance if a server is invoked in a loop the AccessCount.value is set to the number of typical loop iterations.

5.7.4 Structuring of AccessCountSets

In general the detailed usage how AccessCountSets are used to structure a M1 model is not standardized. Nevertheless this section provides some hints how it might be applied for different use cases. Regardless how the AccessCountSets are sub-structured in detail a valid AUTOSAR model can only provide at most one value according a specific countProfile for a particular AbstractAccessPoint. Otherwise the count values would be ambiguous since multiple values would be stated for one kind of access.

[constr_4091] AccessCount.value needs to be unambiguous ⌈ AUTOSAR model shall define at most one AccessCount.value per countProfile for a specific AbstractAccessPoint. ⌋*()*

[TPS_BSWMDT_04149] Structuring according ExecutableEntitys ⌈ The meta-class AccessCountSet should be used to group the AccessCount.values for one particular ExecutableEntity. ⌋*()*

[TPS_BSWMDT_04150] Structuring according Variants ⌈ The meta-class AccessCountSet should be used to group the AccessCount.values which are valid for one particular variant of the software. The grouping might be used if the AccessCount.values are evaluated by code parsing since the parsing might be done for a specific variant of the C-implementation. ⌋*()*

[TPS_BSWMDT_04151] Structuring according different countProfile definitions ⌈ The meta-class AccessCountSet should be used to group the AccessCount.values which are valid for one particular countProfile value. ⌋*()*

6 BSW Behavior

6.1 BSW Behavior Overview

Figure 6.1 and the following class table show the attributes and description of class BswInternalBehavior. Since several attributes on this level are the same for BSW modules and SWCs, these are aggregated by the abstract class InternalBehavior which is shown in the same figure and in a separate class table.

The following subsections give a more detailed explanation of the various attributes.

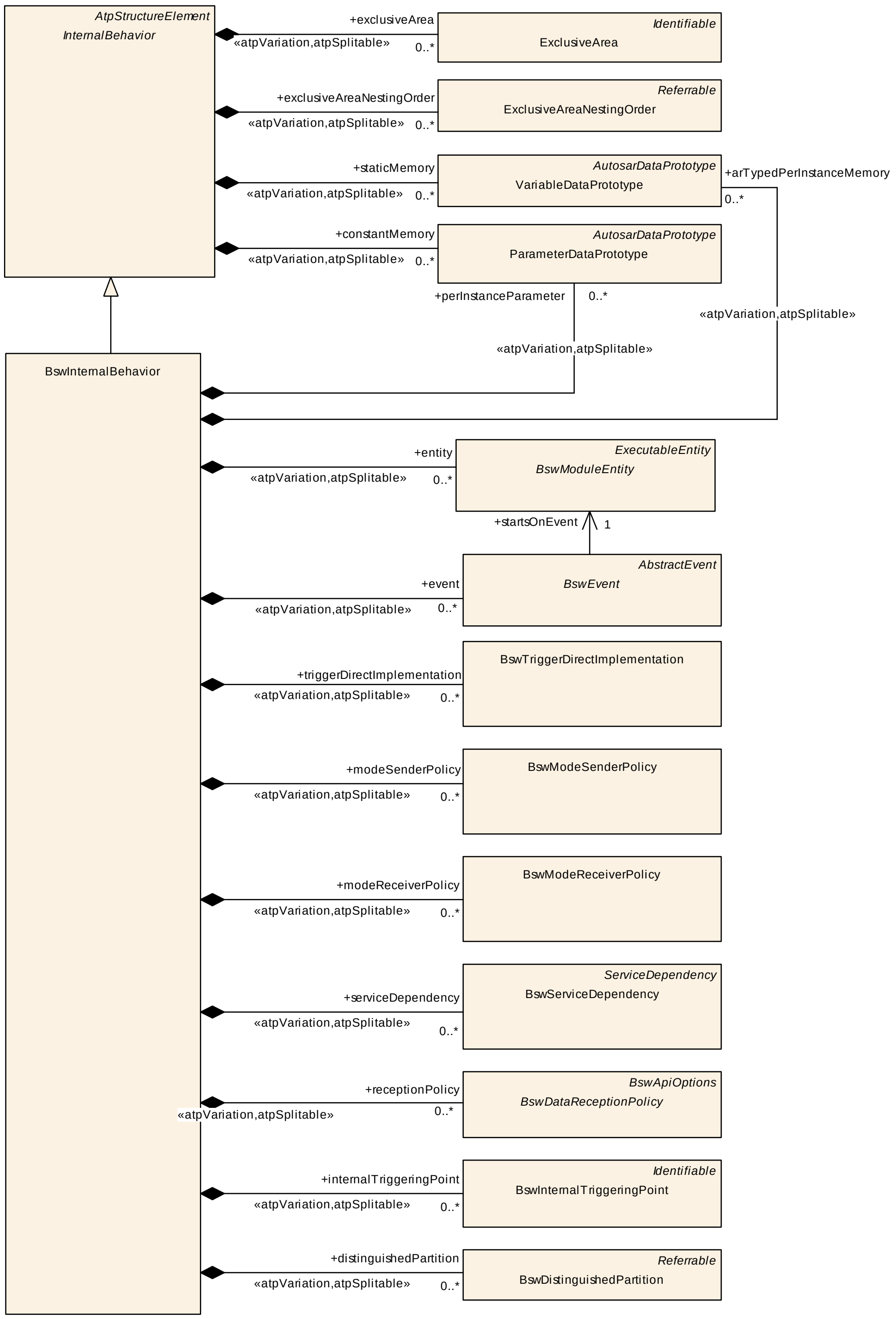

Figure 6.1: Overview of meta-class BswInternalBehavior

Class	***InternalBehavior*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Common base class (abstract) for the internal behavior of both software components and basic software modules/clusters.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswInternalBehavior, SwcInternalBehavior			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
constant Memory	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this Internal Behavior. The shortName of ParameterDataPrototype has to be equal to the "C' identifier of the described constant. The characteristic value(s) might be shared between SwComponentPrototypes of the same SwComponent Type. The aggregation of constantMemory is subject to variability with the purpose to support variability in the software component or module implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
constantValue Mapping	ConstantSpecification MappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=constantValueMapping
dataType Mapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dataTypeMapping
exclusiveArea	ExclusiveArea	*	aggr	This specifies an ExclusiveArea for this InternalBehavior. The exclusiveArea is local to the component resp. module. The aggregation of ExclusiveAreas is subject to variability. Note: the number of ExclusiveAreas might vary due to the conditional existence of RunnableEntities or BswModule Entities. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea NestingOrder	ExclusiveAreaNesting Order	*	aggr	This represents the set of ExclusiveAreaNestingOrder owned by the InternalBehavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurerment variables implemented by this software component. The term "static" is used in the meaning of "non-temporary" and does not necessarily specify a linker encapsulation. This kind of memory is only supported if supportsMultipleInstantiation is FALSE. ▽

▽

△

Class	***InternalBehavior*** (abstract)			
				△ The shortName of the VariableDataPrototype has to be equal with the "C' identifier of the described variable. The aggregation of staticMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 6.1: InternalBehavior

Class	**BswInternalBehavior**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the behavior of a BSW module or a BSW cluster w.r.t. the code entities visible by the BSW Scheduler. It is possible to have several different BswInternalBehaviors referring to the same BswModuleDescription.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *Multilanguage Referrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
arTypedPer Instance Memory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the Basic Software Module. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the Basic Software Module's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
bswPerInstance MemoryPolicy	BswPerInstance MemoryPolicy	*	aggr	Policy for a arTypedPerInstanceMemory The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
clientPolicy	BswClientPolicy	*	aggr	Policy for a requiredClientServerEntry. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=clientPolicy, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
distinguished Partition	BswDistinguished Partition	*	aggr	Indicates an abstract partition context in which the enclosing BswModuleEntity can be executed. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.ShortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=60

▽

— AUTOSAR CONFIDENTIAL —

△

Class	**BswInternalBehavior**			
entity	BswModuleEntity	*	aggr	A code entity for which the behavior is described **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=5
event	BswEvent	*	aggr	An event required by this module behavior. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=10
exclusiveArea Policy	BswExclusiveArea Policy	*	aggr	Policy for an ExclusiveArea in this BswInternalBehavior. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy, variation Point.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a basic software module for its implementation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=includedDataTypeSet
internal TriggeringPoint	BswInternalTriggering Point	*	aggr	An internal triggering point. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=2
internal TriggeringPoint Policy	BswInternalTriggering PointPolicy	*	aggr	Policy for an internalTriggeringPoint in this BswInternal Behavior.. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalTriggeringPointPolicy, variation Point.shortPoint vh.latestBindingTime=preCompileTime
modeReceiver Policy	BswModeReceiver Policy	*	aggr	Implementation policy for the reception of mode switches. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeReceiverPolicy, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=25
modeSender Policy	BswModeSenderPolicy	*	aggr	Implementation policy for providing a mode group. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeSenderPolicy, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
parameterPolicy	BswParameterPolicy	*	aggr	Policy for a perInstanceParameter in this BswInternal Behavior. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=parameterPolicy, variatioPoint.short Label vh.latestBindingTime=preCompileTime

▽

△

Class	BswInternalBehavior			
perInstance Parameter	ParameterData Prototype	*	aggr	Describes a read only memory object containing characteristic value(s) needed by this BswInternal Behavior. The role name perInstanceParameter is chosen in analogy to the similar role in the context of SwcInternal Behavior. In contrast to constantMemory, this object is not allocated locally by the module's code, but by the BSW Scheduler and it is accessed from the BSW module via the BSW Scheduler API. The main use case is the support of software emulation of calibration data. The aggregation is subject to variability with the purpose to support implementation variants. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=atp.Splitkey shortName, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=45
receptionPolicy	BswDataReception Policy	*	aggr	Data reception policy for inter-partition and/or inter-core communication. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=receptionPolicy, variationPoint.short Label vh.latestBindingTime=preCompileTime xml.sequenceOffset=55
releasedTrigger Policy	BswReleasedTrigger Policy	*	aggr	Policy for a releasedTrigger. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=releasedTriggerPolicy, variation Point.shortLabel vh.latestBindingTime=preCompileTime
schedulerName Prefix	BswSchedulerName Prefix	*	aggr	Optional definition of one or more prefixes to be used for the BswScheduler. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=schedulerNamePrefix, variation Point.ShortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=50
sendPolicy	BswDataSendPolicy	*	aggr	Policy for a providedData. The policy selects the options of the Schedule Manager API generation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=sendPolicy, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	BswService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation is subject to variability with the purpose to support the conditional existence of ServiceNeeds. The aggregation is splitable in order to support that ServiceNeeds might be provided in later development steps. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=serviceDependency, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=40

▽

△

Class	BswInternalBehavior			
triggerDirect Implementation	BswTriggerDirect Implementation	*	aggr	Specifies a trigger to be directly implemented via OS calls. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=triggerDirectImplementation, variation Point.shortLabel vh.latestBindingTime=preCompileTime xml.sequenceOffset=15
variationPoint Proxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table 6.2: BswInternalBehavior

6.2 BSW Module Entity

6.2.1 Overview

Figure 6.2 and the next class tables shows the attributes of `BswModuleEntity`, its base class `ExecutableEntity` and its specializations for called, scheduled and interrupt entities. These attributes are mainly required to configure the BSW Scheduler.

It is important to understand the difference between `BswModuleEntity` and `BswModuleEntry`: The first one describes properties of a code fragment whereas the second one describes only the interface (i.e. the signature) used to invoke a code fragment.

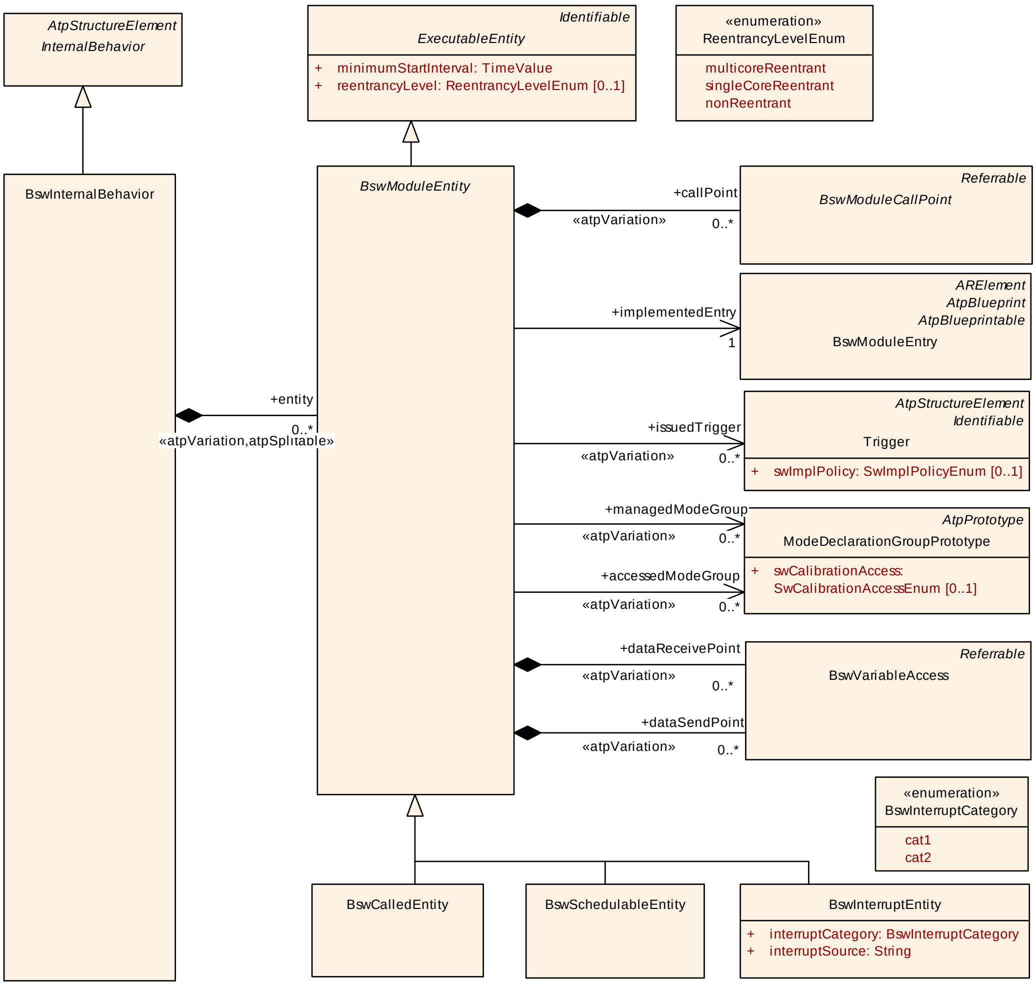

Figure 6.2: Relationships of meta-class BswModuleEntity

[TPS_BSWMDT_04072] Executable entity in BSW ⌈ The abstract meta-class `ExecutableEntity` is not specific for the Basic Software, it is imported from the `CommonStructure` package of the meta-model and is defined as follows: ⌋ *(RS_BSWMD_00030, RS_BSWMD_00046)*

Class	***ExecutableEntity*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Abstraction of executable code.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*BswModuleEntity*, RunnableEntity			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

▽

△

Class	***ExecutableEntity*** (abstract)			
activationReason	ExecutableEntityActivationReason	*	aggr	If the ExecutableEntity provides at least one activationReason element the RTE resp. BSW Scheduler shall provide means to read the activation vector of this executable entity execution. If no activationReason element is provided the feature of being able to determine the activating RTEEvent is disabled for this ExecutableEntity.
canEnterExclusiveArea	ExclusiveArea	*	ref	This means that the executable entity can enter/leave the referenced exclusive area through explicit API calls.
exclusiveAreaNestingOrder	ExclusiveAreaNestingOrder	*	ref	This represents the set of ExclusiveAreaNestingOrders recognized by this ExecutableEntity.
minimumStartInterval	TimeValue	1	attr	Specifies the time in seconds by which two consecutive starts of an ExecutableEntity are guaranteed to be separated.
reentrancyLevel	ReentrancyLevelEnum	0..1	attr	The reentrancy level of this ExecutableEntity. See the documentation of the enumeration type ReentrancyLevelEnum for details. Please note that nonReentrant interfaces can have also reentrant or multicoreReentrant implementations, and reentrant interfaces can also have multicoreReentrant implementations.
runsInsideExclusiveArea	ExclusiveArea	*	ref	The executable entity runs completely inside the referenced exclusive area.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this code entity. Via an association to the same SwAddrMethod, it can be specified that several code entities (even of different modules or components) shall be located in the same memory without already specifying the memory section itself.

Table 6.3: ExecutableEntity

Class	***BswModuleEntity*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the smallest code fragment which can be described for a BSW module or cluster within AUTOSAR.			
Base	*ARObject*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswCalledEntity, BswInterruptEntity, BswSchedulableEntity			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessedModeGroup	ModeDeclarationGroupPrototype	*	ref	A mode group which is accessed via API call by this entity. It must be a ModeDeclarationGroupPrototype required by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
activationPoint	BswInternalTriggeringPoint	*	ref	Activation point used by the module entity to activate one or more internal triggers. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

▽

— AUTOSAR CONFIDENTIAL —

△

Class	***BswModuleEntity*** (abstract)			
callPoint	BswModuleCallPoint	*	aggr	A call point used in the code of this entitiy. The variablity of this association is especially targeted at debug scenarios: It is possible to have one variant calling into the AUTOSAR debug module and another one which doesn't. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
calledEntry	BswModuleEntry	*	ref	The entry of another (or the same) BSW module which is called by this entry (usually via C function call). This information allows to set up a model of call chains. The variablity of this association is especially targeted at debug scenarios: It is possible to have one variant calling into the AUTOSAR debug module and another one which doesn't. Note that this relation has been merked as obsolete, since the more powerful definition of a callPoint should be used. **Stereotypes:** atpVariation **Tags:** atp.Status=removed vh.latestBindingTime=preCompileTime
dataReceive Point	BswVariableAccess	*	aggr	The data is received via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
dataSendPoint	BswVariableAccess	*	aggr	The data is sent via the BSW Scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
implemented Entry	BswModuleEntry	1	ref	The entry which is implemented by this module entity.
issuedTrigger	Trigger	*	ref	A trigger issued by this entity via BSW Scheduler API call. It must be a BswTrigger released (i.e. owned) by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
managedMode Group	ModeDeclarationGroup Prototype	*	ref	A mode group which is managed by this entity. It must be a ModeDeclarationGroupPrototype provided by this module or cluster. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
schedulerName Prefix	BswSchedulerName Prefix	0..1	ref	A prefix to be used in generated names for the Bsw ModuleScheduler in the context of this BswModuleEntity, for example entry point prototypes, macros for dealing with exclusive areas, header file names. Details are defined in the SWS RTE. The prefix supersedes default rules for the prefix of those names.

Table 6.4: BswModuleEntity

6.2.2 BSW Module Entity Attributes

[TPS_BSWMDT_04019] `BswModuleEntity` attributes for exchange of modes and triggers ⌈ The attributes `BswModuleEntity.managedModeGroup`, `BswModuleEntity.accessedModeGroup` and `BswModuleEntity.issuedTrigger` specify, that this BswModuleEntity initiates resp. receives mode switches or activates triggers for other modules by using the BSW Scheduler API. This is mandatory information to configure the BSW Scheduler. ⌋(*RS_BSWMD_00030, RS_BSWMD_00056, RS_BSWMD_00059*)

For an explanation of the attribute callPoint see chapter 6.3

For an explanation of the attributes dataSendPoint and dataReceivePoint see chapter 6.4.

[TPS_BSWMDT_04103] BswModuleEntity reentrancy level ⌈ With the optional attribute reentrancyLevel a BswModuleEntity can state its implemented reentrancy level within the limits given by its interface(see [constr_4077]). This attribute is especially targeted at multicore scenarios.

If this attribute is omitted, reentrancy is assumed to be implemented as defined by the attribute BswModuleEntity.implementedEntry.isReentrant, in which case true means single core reentrancy. ⌋(*RS_BSWMD_00066*)

Enumeration	**ReentrancyLevelEnum**
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior
Note	Specifies if and in which kinds of environments an entity is reentrant.
Literal	***Description***
multicoreReentrant	Unlimited concurrent execution of this entity is possible, including preemption and parallel execution on multi core systems. **Tags:** atp.EnumerationValue=0
nonReentrant	Concurrent execution of this entity is not possible. **Tags:** atp.EnumerationValue=1
singleCore Reentrant	Pseudo-concurrent execution (i.e. preemption) of this entity is possible on single core systems. **Tags:** atp.EnumerationValue=2

Table 6.5: ReentrancyLevelEnum

6.2.3 BSW Module Entity Constraints

The actually implemented reentrancy level can only be "better" than stated on the interface level, as the following constraint says:

[constr_4077] Constraints for BswModuleEntity.reentrancyLevel ⌈

- If the attribute isReentrant of a BswModuleEntry referred by an BswModuleEntity in the role implementedEntry has the value true, then the attribute reentrancyLevel of the same BswModuleEntity (if it exists) can only have the values singleCoreReentrant or multiCoreReentrant.
- If the attribute isReentrant of a BswModuleEntry referred by an BswModuleEntity in the role implementedEntry has the values false, then there are no retrictions for the values of the attribute reentrancyLevel of the same BswModuleEntity (if it exists).

⌋()

A BswModuleEntity can only implement resp. use elements which have been declared on the interface level of the respective module or cluster, in other words:

[constr_4022] BswModuleEntity only uses the module's interface ⌈

- BswModuleEntity.implementedEntry must refer to an element declared as implementedEntry of the enclosing BswModuleDescription
- BswModuleEntity.callPoint.calledEntry - where callPoint is instantiated from BswDirectCallPoint - must refer to an element declared as expectedEntry or implementedEntry of the enclosing BswModuleDescription.
- BswModuleEntity.callPoint.calledEntry - where callPoint is instantiated from BswSynchronousServerCallPoint or BswAsynchronousServerCallPoint - must refer to an element declared as requiredClientServerEntry of the enclosing BswModuleDescription.
- BswModuleEntity.callPoint - where callPoint is instantiated from BswAsynchronousServerCallResultPoint - must refer to an BswAsynchronousServerCallPoint declared in turn as callPoint of the same BswModuleEntity.
- BswModuleEntity.issuedTrigger must refer to an element declared as releasedTrigger of the enclosing BswModuleDescription
- BswModuleEntity.managedModeGroup must refer to an element declared as providedModeGroup of the enclosing BswModuleDescription
- BswModuleEntity.accessedModeGroup must refer to an element declared as requiredModeGroup of the enclosing BswModuleDescription
- BswModuleEntity.dataSendPoint.accessedVariable must refer to an element declared as providedData of the enclosing BswModuleDescription
- BswModuleEntity.dataReceivePoint.accessedVariable must refer to an element declared as requiredData of the enclosing BswModuleDescription
- an accessedModeGroup should be allowed to refer to an element declared as providedModeGroup

⌋()

— AUTOSAR CONFIDENTIAL —

6.2.4 BswCalledEntity

Class	BswCalledEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity which is designed to be called from another BSW module or cluster.			
Base	*ARObject*, *BswModuleEntity*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	Type	Mul.	Kind	Note
–	–	–	–	–

Table 6.6: BswCalledEntity

BswCalledEntity represents an “ordinary” function call for which the following constraints apply:

[constr_4016] BswCalledEntity constraints ⌈

- BswCalledEntity.implementedEntry.callType must be 'regular' or 'callback'
- BswCalledEntity.implementedEntry.executionContext is in general not restricted, but see [constr_4076] for constraints on the server side of a Client-Server communication.

⌋()

6.2.5 BswSchedulableEntity

Class	BswSchedulableEntity			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity, which is designed for control by the BSW Scheduler. It may for example implement a so-called "main" function.			
Base	*ARObject*, *BswModuleEntity*, *ExecutableEntity*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	Type	Mul.	Kind	Note
–	–	–	–	–

Table 6.7: BswSchedulableEntity

BswSchedulableEntity represents a scheduled function call for which the following constraints apply:

[constr_4017] BswSchedulableEntity constraints ⌈

- BswModuleEntity.implementedEntry.callType must be 'scheduled'
- BswModuleEntity.implementedEntry.executionContext must be 'task'

⌋()

6.2.6 BswInterruptEntity

Class	**BswInterruptEntity**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BSW module entity, which is designed to be triggered by an interrupt.			
Base	*ARObject, BswModuleEntity, ExecutableEntity, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
interrupt Category	BswInterruptCategory	1	attr	Category of the interrupt
interruptSource	String	1	attr	Allows a textual documentation of the intended interrupt source.

Table 6.8: BswInterruptEntity

Enumeration	**BswInterruptCategory**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior
Note	Category of the interrupt service
Literal	***Description***
cat1	Cat1 interrupt routines are not controlled by the OS and are only allowed to make a very limited selection of OS calls to enable and disable all interrupts. The BswInterruptEntity is implemented by the interrupt service routine, which is directly called from the interrupt vector (not via the OS). **Tags:** atp.EnumerationValue=0
cat2	Cat2 interrupt routines are controlled by the OS and they are allowed to make OS calls. The Bsw InterruptEntity is implemented by the interrupt handler, which is called from the OS. **Tags:** atp.EnumerationValue=1

Table 6.9: BswInterruptCategory

`BswInterruptEntity` represents an interrupt routine for which the following constraints apply:

[constr_4018] `BswInterruptEntity` constraints ⌈

- `BswInterruptEntity.implementedEntry.callType` must be `'interrupt'`
- `BswInterruptEntity.implementedEntry.executionContext` must be `'interruptCat1'` if and only if `BswInterruptEntity.interruptCategory` is `'Cat1'`
- `BswInterruptEntity.implementedEntry.executionContext` must be `'interruptCat2'` if and only if `BswInterruptEntity.interruptCategory` is `'Cat2'`

⌋*()*

6.3 BSW Module Call Point

6.3.1 Overview

By aggregation of BswModuleCallPoints a BswModuleEntity defines how it uses BswModuleEntry-s in order to call into other (or the same) BSW module.

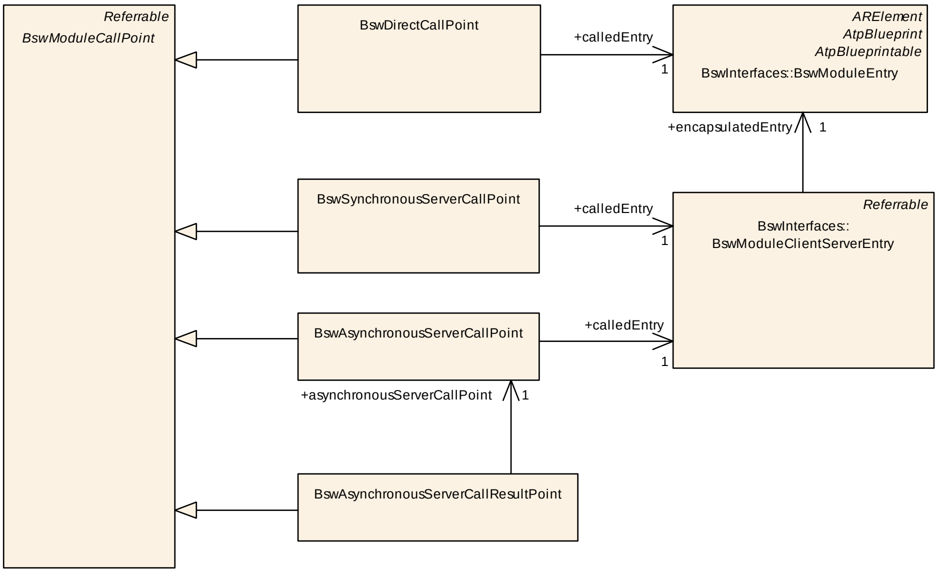

Figure 6.3: Details of BswModuleCallPoint

Class	***BswModuleCallPoint*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents a point at which a BswModuleEntity handles a procedure call into a BswModuleEntry, either directly or via the BSW Scheduler.			
Base	*ARObject*, *Referrable*			
Subclasses	BswAsynchronousServerCallPoint, BswAsynchronousServerCallResultPoint, BswDirectCallPoint, BswSynchronousServerCallPoint			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
contextLimitation	BswDistinguishedPartition	*	ref	The existence of this reference indicates that the call point is used only in the context of the referred BswDistinguishedPartitions.

Table 6.10: BswModuleCallPoint

6.3.2 Direct Call Points

[TPS_BSWMDT_04018] Usage of BswDirectCallPoint ⌈The meta-class BswDirectCallPoint aggregated in the role callPoint in a BswModuleEntity allows to declare which entry of another module (or the same module) is called in the code of the given BswModuleEntity directly, i.e. not via the BSW Scheduler. ⌋ *(RS_BSWMD_00047)*

Class	BswDirectCallPoint			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents a concrete point in the code from where a BswModuleEntry is called directly, i.e. not via the BSW Scheduler. This information can be used to analyze call tree and resource locking scenarios. It is not needed to configure the BSW Scheduler.			
Base	*ARObject*, *BswModuleCallPoint*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
calledEntry	BswModuleEntry	1	ref	The BswModuleEntry called at this point.
calledFromWithinExclusiveArea	ExclusiveAreaNestingOrder	0..1	ref	This indicates that the call point is located at the deepest level inside one or more ExclusiveAreas that are nested in the given order.

Table 6.11: BswDirectCallPoint

Note that this is not a mandatory information in order to be able to integrate a module, but it is a very important information if an integrator wants to analyze a call chain among several modules in order to setup a proper scheduling. It is further important to note that this attribute contains additional information in comparison to `BswModuleDescription.bswModuleDependency`, because the latter only denotes the dependencies between the module interfaces whereas `calledEntry` shows from which code fragment a call is actually invoked.

In addition, a `BswDirectCallPoint` contains information about resource locking see 6.5.

Of course, the execution context (like task, interrupt, etc.) is preserved during a direct call:

[constr_4015] `calledEntry` constraints for direct calls ⌈ The following holds if `callPoint` is aggregated as an instance of `BswDirectCallPoint`:

- `BswModuleEntity.callPoint.calledEntry.executionContext` must be identical to `BswModuleEntity.implementedEntry.executionContext`
- `BswModuleEntity.callPoint.calledEntry.callType` must have the value `'regular'` or `'callback'`

⌋()

6.3.3 Client-Server Call Points

[TPS_BSWMDT_04102] Usage of `BswSynchronousServerCallPoint` ⌈ The meta-class `BswSynchronousServerCallPoint` aggregated in the role `callPoint` in a `BswModuleEntity` allows to declare which entry of another module (or the same module) is called synchronously in the code of the client-side `BswModuleEntity` via the BSW Scheduler.

The intended use case is inter-partition or inter-core communication.[1] Note that it is a valid use case for a given BswInternalBehavior to have two different BswModuleEntity-s which eventually run on different partitions and/or processor cores. ⌋ (RS_BSWMD_00066)

Class	BswSynchronousServerCallPoint			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents a synchronous procedure call point via the BSW Scheduler.			
Base	*ARObject*, *BswModuleCallPoint*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
calledEntry	BswModuleClientServerEntry	1	ref	The entry to be called.
calledFromWithinExclusiveArea	ExclusiveAreaNestingOrder	0..1	ref	This indicates that the call point is located at the deepest level inside one or more ExclusiveAreas that are nested in the given order.

Table 6.12: BswSynchronousServerCallPoint

In the same way as a BswDirectCallPoint also a BswSynchronousServerCallPoint contains information about resource locking see 6.5.

[TPS_BSWMDT_04104] Usage of BswAsynchronousServerCallPoint ⌈ The meta-class BswAsynchronousServerCallPoint aggregated in the role callPoint in a BswModuleEntity allows to declare which entry of another module (or the same module) is called asynchronously in the code of the client-side BswModuleEntity via the BSW Scheduler.

The intended use case is inter-partition or inter-core communication. Note that it is a valid use case for a given BswInternalBehavior to have two different BswModuleEntity-s which eventually run on different partitions and/or processor cores. ⌋ (RS_BSWMD_00066)

Class	BswAsynchronousServerCallPoint			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents an asynchronous procedure call point via the BSW Scheduler.			
Base	*ARObject*, *BswModuleCallPoint*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
calledEntry	BswModuleClientServerEntry	1	ref	The entry to be called.

Table 6.13: BswAsynchronousServerCallPoint

[TPS_BSWMDT_04105] Usage of BswAsynchronousServerCallResultPoint ⌈ The meta-class BswAsynchronousServerCallResultPoint aggregated in the role callPoint in a BswModuleEntity indicates that the client-side BswModuleEntity has the possibility to retrieve the results (return value and arguments)

[1] This does not exclude configurations where client and server are executed in the same partition within the limits defined by contextLimitation.

of a former asynchronous call done via the associated BswAsynchronousServerCallPoint. ⌋(*RS_BSWMD_00066*)

Class	**BswAsynchronousServerCallResultPoint**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The callback point for an BswAsynchronousServerCallPoint i.e. the point at which the result can be retrieved from the BSW Scheduler.			
Base	*ARObject*, *BswModuleCallPoint*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
asynchronousServerCallPoint	BswAsynchronousServerCallPoint	1	ref	The call point invoking the call to which the result belongs.

Table 6.14: BswAsynchronousServerCallResultPoint

Note that the BswModuleEntity that retrieves such a result may be scheduled in different ways: It may be started via a BswAsynchronousServerCallReturnsEvent and/or by other kind of BswEvents.

[constr_4079] calledEntry constraints for client-server calls ⌈

- The BswModuleClientServerEntry aggregated as calledEntry in a BswSynchronousServerCallPoint must have the attribute isSynchronous = true.
- The BswModuleClientServerEntry aggregated as calledEntry in a BswAsynchronousServerCallPoint must have the attribute isSynchronous = false.

⌋*()*

6.4 BSW Sender-Receiver Data Access

By aggregation of meta-class BswVariableAccess a BswModuleEntity defines how it accesses data for (potential) inter-partition communication with another (or the same) BSW module.

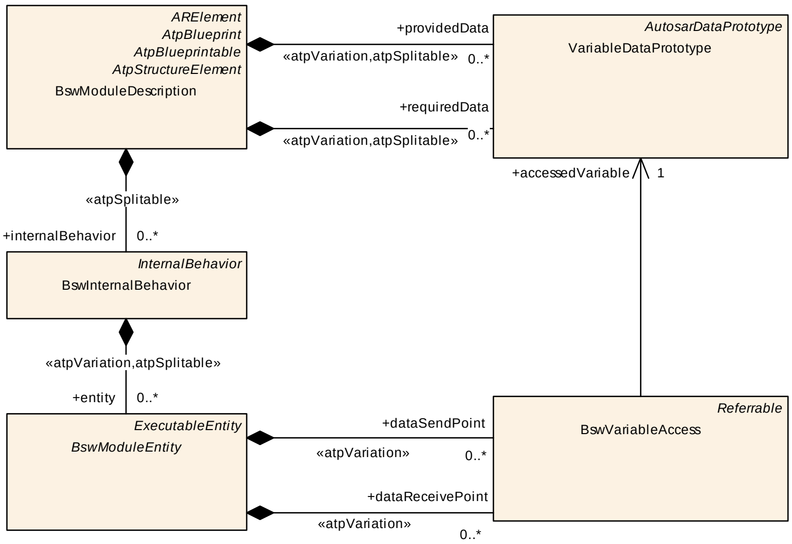

Figure 6.4: Usage of BswVariableAccess

Class	**BswVariableAccess**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The presence of a BswVariableAccess implies that a BswModuleEntity needs access to a VariableData Prototype via the BSW Scheduler. The kind of access is specified by the role in which the class is used.			
Base	*ARObject*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
accessed Variable	VariableDataPrototype	1	ref	The data accessed via the BSW Scheduler.
context Limitation	BswDistinguished Partition	*	ref	The existence of this reference indicates that the variable is recevied resp. sent only in the context of the referred BswDistinguishedPartitions.

Table 6.15: BswVariableAccess

[TPS_BSWMDT_04106] `BswModuleEntity` attributes for sender-receiver data exchange ⌈ The attributes `BswModuleEntity.dataSendPoint` and `BswModuleEntity.dataReceivePoint` specify, that this `BswModuleEntity` has access to the BSW Scheduler in order to send resp. receive the data declared in the referred `VariableDataPrototype`. This is targeted at inter-partition and/or multicore communication scenarios.[2] ⌋*(RS_BSWMD_00067)*

6.5 BSW Exclusive Areas

[TPS_BSWMDT_04073] Exclusive area in BSW ⌈ The meta-class `ExclusiveArea` (including the associations from `ExecutableEntity`) is not specific for the Basic Software, is is imported from the `CommonStructure` package of the meta-model and is defined as follows: ⌋*(RS_BSWMD_00060)*

[2] This does not exclude configurations where sender and receiver are executed in the same partition within the limits defined by `contextLimitation`.

Class	**ExclusiveArea**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Prevents an executable entity running in the area from being preempted.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 6.16: ExclusiveArea

For certain implementations of the ExclusiveArea mechanisms it is advantageous if each BswModuleEntity uses a distinct set of enter and exit APIs. This distinct set of APIs support ExclusiveArea implementations where for the highest prior RunnableEntity(s) the lock is omitted. This is possible when the highest prior BswModuleEntity(s) cannot get interrupted by BswModuleEntitys scheduled with lower priority in any circumstance. To support this kind of implementations the software component description has to state that it requests APIs individually for each BswModuleEntity accessing an ExclusiveArea with the canEnterExclusiveArea manner.

Class	**BswExclusiveAreaPolicy**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The ExclusiveArea for which the BSW Scheduler using this policy.			
Base	*ARObject*, *BswApiOptions*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
apiPrinciple	ApiPrincipleEnum	0..1	attr	Specifies for this ExclusiveArea if either one common set of Enter and Exit APIs for the whole BSW module is requested from the SchM or if the set of Enter and Exit APIs is expected per BswModuleEntity. The default value is "common".
exclusiveArea	ExclusiveArea	1	ref	The ExclusiveArea for which the BSW Scheduler using this policy.

Table 6.17: BswExclusiveAreaPolicy

Enumeration	**ApiPrincipleEnum**
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior
Note	This enumeration represents the ability to control the granularity of API generation.
Literal	**Description**
common	The Rte or SchM API is provided for the whole software component / BSW Module **Tags:** atp.EnumerationValue=0
perExecutable	The Rte or SchM API is provided for a specific ExecutableEntity of a software component / BSW Module **Tags:** atp.EnumerationValue=1

Table 6.18: ApiPrincipleEnum

[TPS_BSWMDT_04154] ExclusiveArea is entered and exit by common set of API ⌈ If the BswExclusiveAreaPolicy.apiPrinciple is set to "common" the SchM provides one sets of enter and exit APIs for the whole BSW module. ⌋ *(RS_BSWMD_00064)*

In this case the same enter and exit code is executed by all affected BswModuleEntitys and there is no way to have a special treatment for the BswModuleEntitys executed in the highest prior context.

[TPS_BSWMDT_04155] ExclusiveArea is entered and exit by individual set of API ⌈ If the BswExclusiveAreaPolicy.apiPrinciple is set to "perExecutable" the SchM provides individual sets of enter and exit APIs for each affected BswModuleEntity. ⌋*(RS_BSWMD_00064)*

In this case enter and exit code for the BswModuleEntity executed in the highest priority context can be left empty.

To avoid contradicting settings of BswExclusiveAreaPolicys for one ExclusiveArea [constr_4097] applies.

[constr_4097] Limitation on the number of BswExclusiveAreaPolicys ⌈ An ExclusiveArea can only be referenced by at most one BswExclusiveAreaPolicy. ⌋*()*

Figure 6.5 shows the detailed meta-model of exclusive areas in BSW.

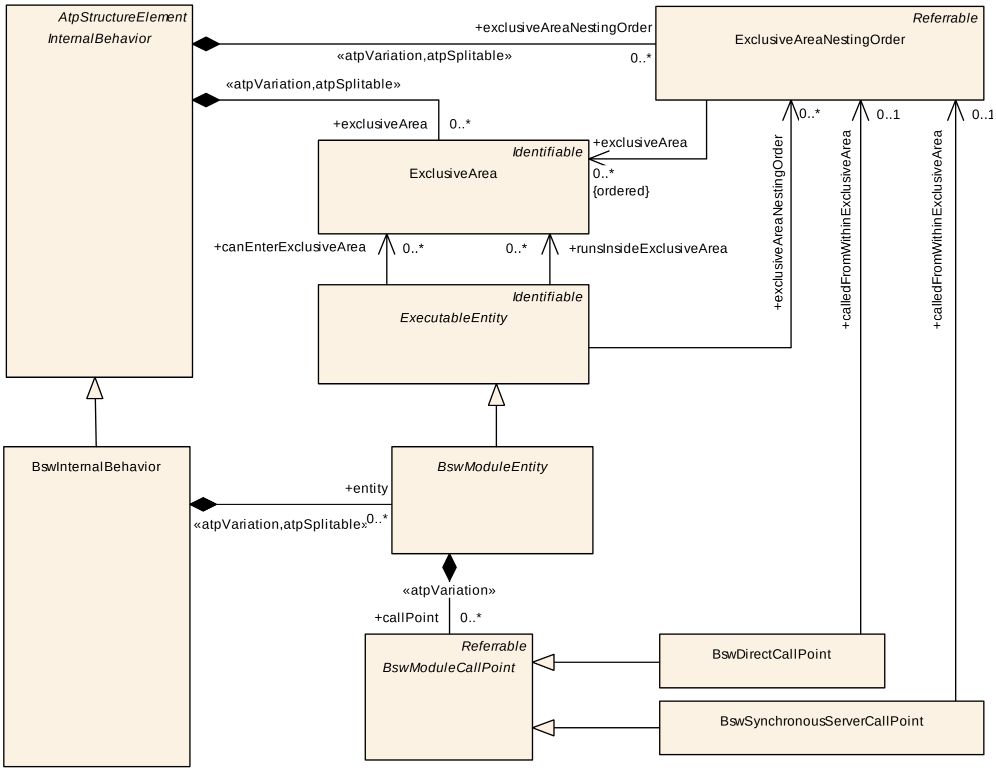

Figure 6.5: Details of defining ExclusiveAreas in BSWMDT

In addition to defining that a BswModuleEntity can enter an exclusive area or completely runs in an exclusive area, it is possible to define possible nesting orders of exclusive areas. Furthermore one can define at which level of a nesting order function calls are invoked from the BswModuleEntity. The information on nesting orders can be used to analyze the call tree with respect to resource locking scenarios.

Class	ExclusiveAreaNestingOrder			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the ability to define a nesting order of ExclusiveAreas. A nesting order (that may occur in the executable code) is formally defined to be able to analyze the resource locking behavior.			
Base	*ARObject*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
exclusiveArea (ordered)	ExclusiveArea	*	ref	This represents a specific scenario of how Exclusive Areas can be used in terms of the nesting order.

Table 6.19: ExclusiveAreaNestingOrder

[TPS_BSWMDT_04081] ExclusiveAreaNestingOrder ⌈ The optional ExclusiveAreaNestingOrders shall (if used at all) describe possible nesting orders (including single ExclusiveAreas) which can occur in the BswModuleEntity. Each possible locking situation requires its own ExclusiveAreaNestingOrder. ⌋ *(RS_BSWMD_00064)*

[TPS_BSWMDT_04082] Indicate that the locking behavior is fully described for BswModuleEntity ⌈ All ExclusiveAreas which are configured in the InternalBehavior should be referenced by an ExclusiveAreaNestingOrder to indicate that the locking behavior is fully described for the corresponding BswModuleEntitys. ⌋*(RS_BSWMD_00064)*

[TPS_BSWMDT_04083] Locking behavior is not described for BswModuleEntity-s ⌈ If ExclusiveAreas are not referenced by any ExclusiveAreaNestingOrder (this is the default scenario), this means that the locking behavior is not described for the corresponding BswModuleEntity-s and the provided information might be incomplete and cannot be used for a global offline analysis of locking behavior. ⌋*(RS_BSWMD_00064)*

[TPS_BSWMDT_04084] Relation of BswModuleCallPoint to ExclusiveAreaNestingOrder ⌈ In case other BswModuleEntitys are called from within the BswModuleEntity the ExclusiveAreaNestingOrder can then be referenced by one or several BswModuleCallPoints to specify the calling environment of the invoked function with regard to ExclusiveAreas. ⌋*(RS_BSWMD_00064)*

6.6 BSW Scheduler Name Prefix

[TPS_BSWMDT_04020] Usage of BswSchedulerNamePrefix ⌈ The Basic Software Scheduler API defines several generated artifacts (macro code and header file names) containing a so-called **module prefix**. This is by default derived from the attribute BswModuleDescription.shortName.

However in order to allow a more fine granular definition of these artifacts, it is possible to specify own prefixes within a BswInternalBehavior and assign them individually to each BswSchedulableEntity. Such an assignment will supersede

the prefix given by BswModuleDescription.shortName. This is especially useful if the BSWMD in questions represents a cluster of several other modules. ⌋
(RS_BSWMD_00014, RS_BSWMD_00030)

Note that this prefix cannot be used to modify any names visible in the module's interface to other modules, namely module abbreviations being part of BswModuleEntry.shortName cannot be superseded by it.

Figure 6.6 and the following class table show how the meta-class BswSchedulerNamePrefix is placed in the meta-model. Refer to [13] for the details how this information is used by the RTE generator.

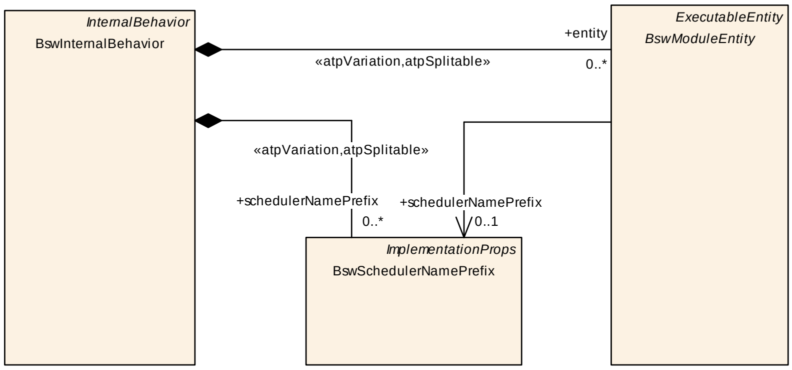

Figure 6.6: Name Prefix for BSW Scheduler artifacts

Class	**BswSchedulerNamePrefix**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A prefix to be used in names of generated code artifacts which make up the interface of a BSW module to the BswScheduler.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 6.20: BswSchedulerNamePrefix

Class	***ImplementationProps*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
symbol	CIdentifier	1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table 6.21: ImplementationProps

6.7 BSW Event

6.7.1 Overview

[TPS_BSWMDT_04021] Usage of BswEvent ⌈ The abstract class BswEvent is used as base class for all kinds of events which can start a BswModuleEntity (which means it does not include direct function calls that are not visible to the BSW Scheduler). Figure 6.7 gives an overview on these events and their association to the different kinds of BswModuleEntity. ⌋*(RS_BSWMD_00053, RS_BSWMD_00054, RS_BSWMD_00057)*

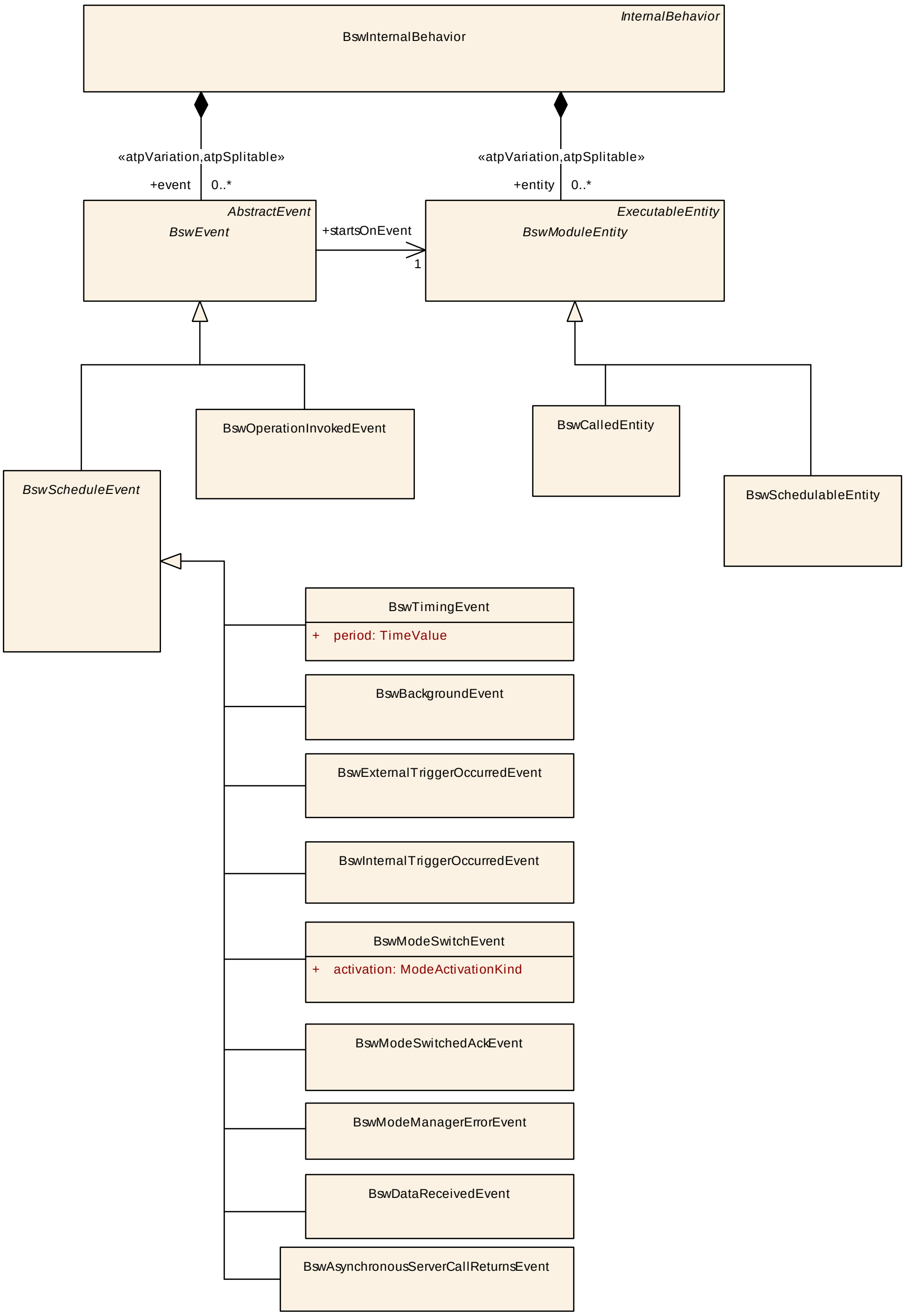

Figure 6.7: Overview on BswEvents

Class	***BswEvent*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Base class of various kinds of events which are used to trigger a BswModuleEntity of this BSW module or cluster. The event is local to the BSW module or cluster. The short name of the meta-class instance is intended as an input to configure the required API of the BSW Scheduler.			
Base	*ARObject*, *AbstractEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswOperationInvokedEvent, *BswScheduleEvent*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
context Limitation	BswDistinguished Partition	*	ref	The existence of this reference indicates that the usage of the event is limited to the context of the referred Bsw DistinguishedPartitions.
disabledInMode	ModeDeclaration	*	iref	The modes, in which this event is disabled. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=disabledInMode
startsOnEvent	BswModuleEntity	1	ref	The entity which is started by the event.

Table 6.22: BswEvent

Class	***BswScheduleEvent*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	BswEvent that is able to start a BswSchedulabeEntity.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswAsynchronousServerCallReturnsEvent, BswBackgroundEvent, BswDataReceivedEvent, Bsw ExternalTriggerOccurredEvent, BswInternalTriggerOccurredEvent, BswModeManagerErrorEvent, Bsw ModeSwitchEvent, BswModeSwitchedAckEvent, BswTimingEvent			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.23: BswScheduleEvent

[constr_1275] Applicability of reference `startsOnEvent` for `BswScheduleEvent` ⌈ The reference `BswScheduleEvent.startsOnEvent` shall only refer to a `BswSchedulableEntity`. ⌋*()*

[constr_1276] Applicability of reference `startsOnEvent` for `BswOperationInvokedEvent` ⌈ The reference `BswOperationInvokedEvent.startsOnEvent` shall only refer to a `BswCalledEntity`. ⌋*()*

6.7.2 Timing and Background Events

[TPS_BSWMDT_04022] Timing and background events for BSW ⌈ A `BswTimingEvent` and `BswBackgroundEvent` are directly driven by the Scheduler resp. OS without external sources. ⌋*(RS_BSWMD_00053)*

Class	BswTimingEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A recurring BswEvent driven by a time period.			
Base	*ARObject, AbstractEvent, BswEvent, BswScheduleEvent, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
period	TimeValue	1	attr	Requirement for the time period (in seconds) by which this event is triggered.

Table 6.24: BswTimingEvent

[constr_4043] Period of BswTimingEvent ⌈ BswTimingEvent.period shall be greater than 0. ⌋*()*

Class	BswBackgroundEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A recurring BswEvent which is used to perform background activities. It is similar to a BswTimingEvent but has no fixed time period and is activated only with low priority.			
Base	*ARObject, AbstractEvent, BswEvent, BswScheduleEvent, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.25: BswBackgroundEvent

6.7.3 Trigger Events

Figure 6.8 and the following tables give a more detailed picture on the events driven by internal or external triggers.

Note the difference in the activation of internally triggered events and timing events:

[TPS_BSWMDT_04023] Internal trigger and timing events for BSW ⌈ A BswModuleEntity can trigger a BswInternalTriggerOccurredEvent (of the same module) with the help of an API generated by the BSW Scheduler, whereas a BswTimingEvent is triggered by the BswScheduler via the OS timer. ⌋*(RS_BSWMD_00053, RS_BSWMD_00057)* Further information can be found in [13].

[TPS_BSWMDT_04024] External trigger event for BSW ⌈ The BswExternalTriggerOccurredEvent specifies the fact that the event is raised in response to a trigger issued by another BSW module. This can for example be used to communicate ECU-external events, like wakeup-events or crank-shaft-events directly between BSW modules. ⌋*(RS_BSWMD_00057)*

[constr_4023] External trigger must belong to the interface ⌈ A BswExternalTriggerOccurredEvent must refer to a Trigger that is declared via BswModuleDescription.requiredTrigger for the same module. ⌋*()*

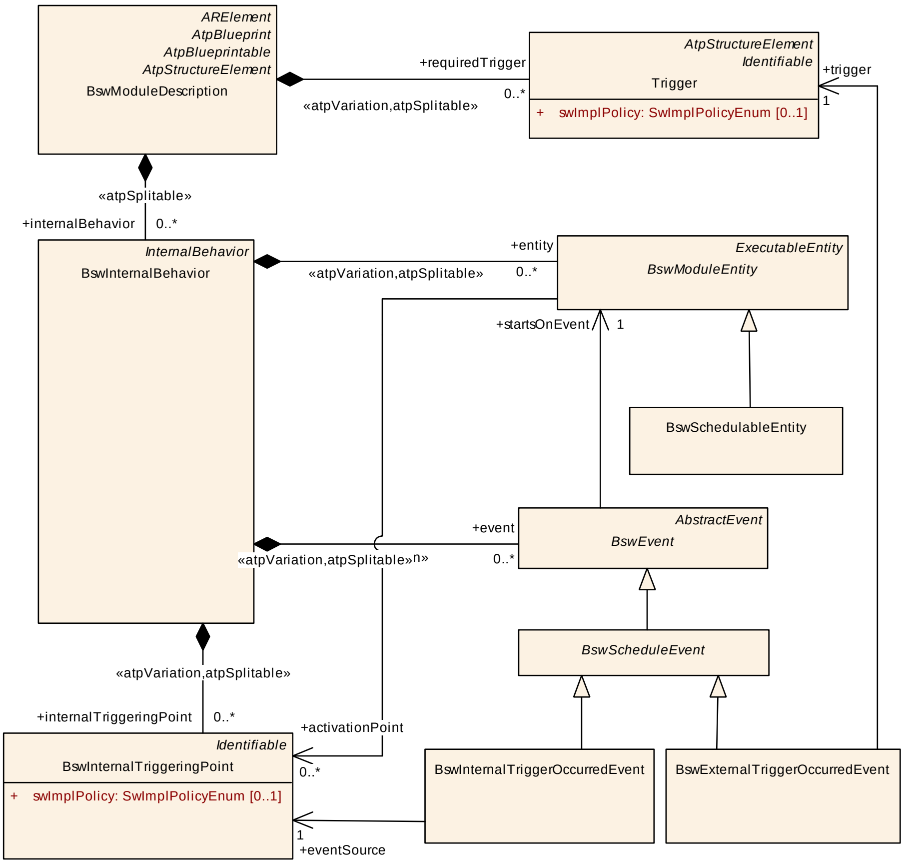

Figure 6.8: Details on BSW Trigger Events

Class	**BswInternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents the activation point for one or more BswInternalTriggerOccurredEvents.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, specifies a queued processing of the internal trigger event.

Table 6.26: BswInternalTriggeringPoint

In a similar way as for external triggers, the `BswInternalTriggeringPoint` can set an attribute to define its queuing behavior:

[constr_4065] Allowed values of `BswInternalTriggeringPoint.swImplPolicy` ⌈ The **only** allowed values for the attribute `BswInternalTriggeringPoint.swImplPolicy` are either `STANDARD` (in which case the internal trigger processing does not use a queue) or `QUEUED` (in which case the internal trigger processing uses a queue). ⌋*()*

Class	BswInternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent, which can happen sporadically. The event is activated by explicit calls from the module to the BSW Scheduler. The main purpose for such an event is to cause a context switch, e.g. from an ISR context into a task context. Activation and switching are handled within the same module or cluster only.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
eventSource	BswInternalTriggering Point	1	ref	The activation point is the source of this event.

Table 6.27: BswInternalTriggerOccurredEvent

Class	BswExternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent resulting from a trigger released by another module or cluster.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
trigger	Trigger	1	ref	The trigger associated with this event. The trigger is external to this module.

Table 6.28: BswExternalTriggerOccurredEvent

In addition to these mechanisms, external events can directly trigger a `BswInterruptEntity` by the means of an interrupt. This situation is not part of the event model, because it is not handled via the BSW Scheduler and is local to a BSW module.

6.7.4 Mode Events

Figure 6.9 and the following tables give a more detailed picture on the events and further classes related to mode switches.

Mode switches can influence the activation of `BswEvent`s by different mechanisms:

[TPS_BSWMDT_04025] Mode switches and events in BSW ⌈

- Via the optional attribute `disabledInMode` a `BswEvent` can specify, that it has to be suppressed in a certain mode.
- A special kind of event, the `BswModeSwitchEvent` can be used to start a `BswModuleEntity` at the entry or exit of a specific mode.
- At the sender side of a mode switch (i.e. in the module managing the mode group), a `BswModeSwitchedAckEvent` can be used to start a `BswModuleEntity` after a mode switch has been acknowledged by the BSW Scheduler.
- At the sender side of a mode switch (i.e. in the module managing the mode group), a `BswModeManagerErrorEvent` can be used to start a `BswModuleEntity` after an error has been announced. This event will be thrown by

the BSW Scheduler after an error that lead to the termination of one of the partitions involved. This could be the partition in which the mode switch was managed or the partition in which it was used.

⌋*(RS_BSWMD_00054, RS_BSWMD_00056)*

The referred ModeDeclaration and the enumeration ModeActivationKind are both imported from the CommonStructure package of the meta-model.

[constr_4024] Semantics of BSW mode switch event ⌈ If BswModeSwitchEvent.activation has the value onTransition BswModeSwitchEvent shall refer to two different modes belonging to the same instance of ModeDeclarationGroup, their order defining the direction of the transition. In all other cases, BswModeSwitchEvent shall refer to exactly one mode. ⌋*()*

[constr_4066] BswModeSwitchEvent and the definition of ModeTransition ⌈ For each pair of ModeDeclarations referenced by a BswModeSwitchEvent with attribute activation set to onTransition a ModeTransition shall be defined in the corresponding direction (i.e. from exitedMode to enteredMode). This constraint shall only apply if the respective ModeDeclarationGroup defines at least one modeTransition. ⌋*()*

[constr_4025] Modes used by BSW mode switch event ⌈ The ModeDeclaration used by BswModeSwitchEvent must belong to the ModeDeclarationGroupPrototype referred as BswInternalBehavior.entity.accessedModeGroup of the enclosing BswInternalBehavior. ⌋*()*

[constr_4026] Mode group used by BSW mode switch acknowledge event ⌈ The ModeDeclarationGroupPrototype used by BswModeSwitchedAckEvent must be referred as BswModuleDescription.providedModeGroup by the same module. ⌋*()*

[constr_4081] Mode group used by BSW mode manager error event ⌈ The ModeDeclarationGroupPrototype used by BswModeManagerErrorEvent must be referred as BswModuleDescription.providedModeGroup by the same module. ⌋ *()*

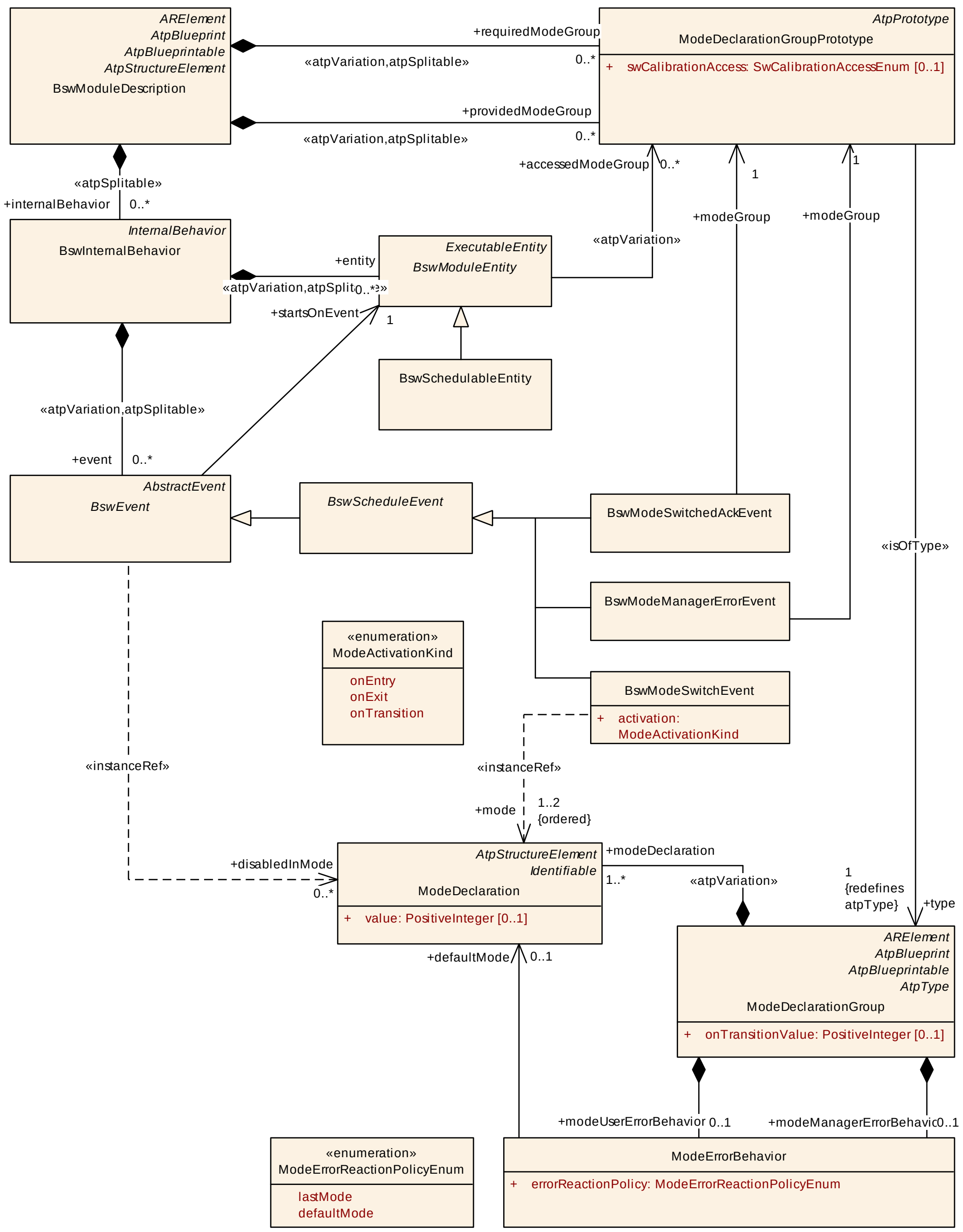

Figure 6.9: Details on BSW Events related to Mode Switches

Class	**BswModeSwitchEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent resulting from a mode switch.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
activation	ModeActivationKind	1	attr	Kind of activation w.r.t. to the referred mode.

▽

△

Class	**BswModeSwitchEvent**			
mode (ordered)	ModeDeclaration	1..2	iref	Reference to one or two Modes that initiate the Mode Switch Event.

Table 6.29: BswModeSwitchEvent

Class	**BswModeSwitchedAckEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	The event is raised after a switch of the referenced mode group has been acknowledged or an error occurs. The referenced mode group must be provided by this module.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	1	ref	A mode group provided by this module. The acknowledgement of a switch of this group raises this event.

Table 6.30: BswModeSwitchedAckEvent

Class	**BswModeManagerErrorEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This represents the ability to react on errors occurring during mode handling.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	1	ref	This represents the ModeDeclarationGroupPrototype for which the error behavior of the mode manager applies.

Table 6.31: BswModeManagerErrorEvent

Enumeration	**ModeActivationKind**
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration
Note	Kind of mode switch condition used for activation of an event, as further described for each enumeration field.
Literal	***Description***
onEntry	On entering the referred mode. **Tags:** atp.EnumerationValue=0
onExit	On exiting the referred mode. **Tags:** atp.EnumerationValue=1
onTransition	On transition of the 1st referred mode to the 2nd referred mode. **Tags:** atp.EnumerationValue=2

Table 6.32: ModeActivationKind

6.7.5 BSW Events for Client-Server Communication

Figure 6.10 and the following tables give a more detailed picture on the events driven by client-server calls. The intended use case is inter-partition and/or inter-core communication.[3]

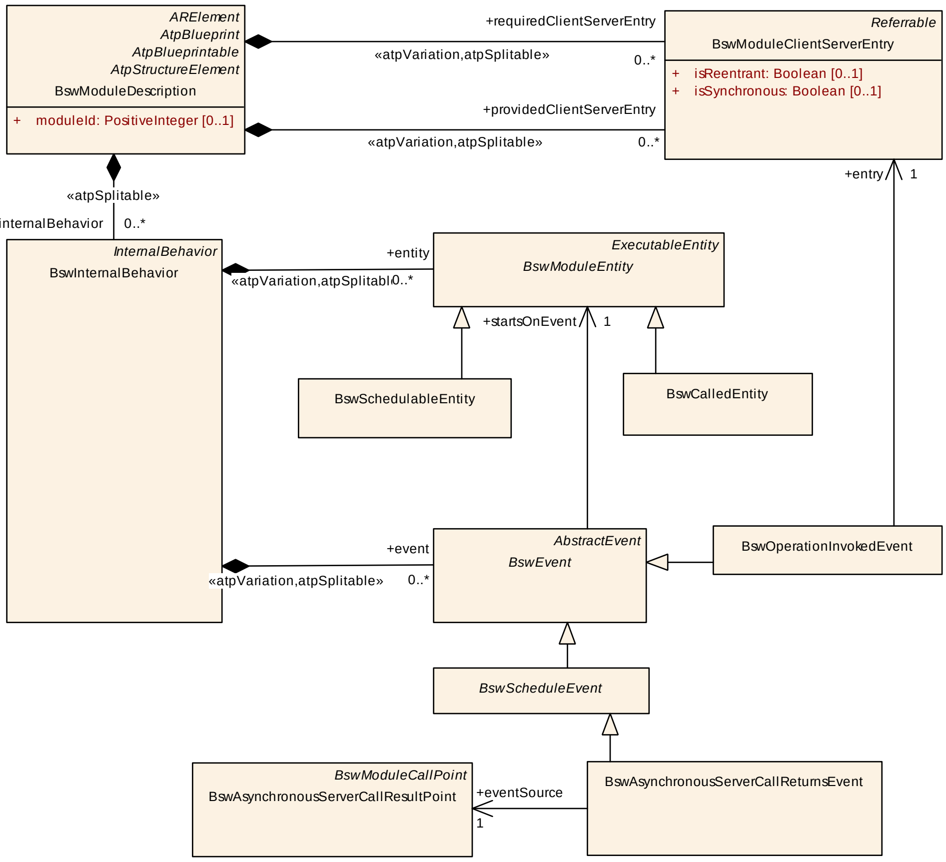

Figure 6.10: Details on BSW Events related to Client-Server Communication

Class	**BswOperationInvokedEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This event is thrown on operation invocation in Client-Server-Communication via the BSW Scheduler. Its "entry" reference provides the BswClientServerEntry that is called subsequently. Note this event is not needed in case of direct function calls.			
Base	*ARObject, AbstractEvent, BswEvent, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
entry	BswModuleClientServer Entry	1	ref	The providedClientServerEntry invoked by this event.

Table 6.33: BswOperationInvokedEvent

[constr_4078] Consistent usage of `BswOperationInvokedEvent` ⌈ The `BswCalledEntity` referred by the attribute `BswOperationInvokedEvent.startsOnEvent` shall refer to the same `BswModuleEntry` (via its atttribute

[3] This does not exclude configurations where client and server are executed in the same partition.

implementedEntry) as the BswOperationInvokedEvent (via its attribute entry.encapsulatedEntry. ⌋()

[constr_4098] No mode disabling for BswOperationInvokedEvent ⌈ A BswOperationInvokedEvent shall not have a reference to a ModeDeclaration in the role disabledInMode. ⌋()

Class	**BswAsynchronousServerCallReturnsEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This is the "callback" event for asynchronous Client-Server-Communication via the BSW Scheduler which is thrown after completion of the asynchronous Client-Server call. Its eventSource specifies the call point to be used for retrieving the result.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
eventSource	BswAsynchronous ServerCallResultPoint	1	ref	The call point to be used for retrieving the result.

Table 6.34: BswAsynchronousServerCallReturnsEvent

6.7.6 BSW Events for Sender-Receiver Communication

Figure 6.11 and the following table give a more detailed picture on the events driven by sender-receiver calls. The intended use case is inter-partition and/or inter-core communication.[4]

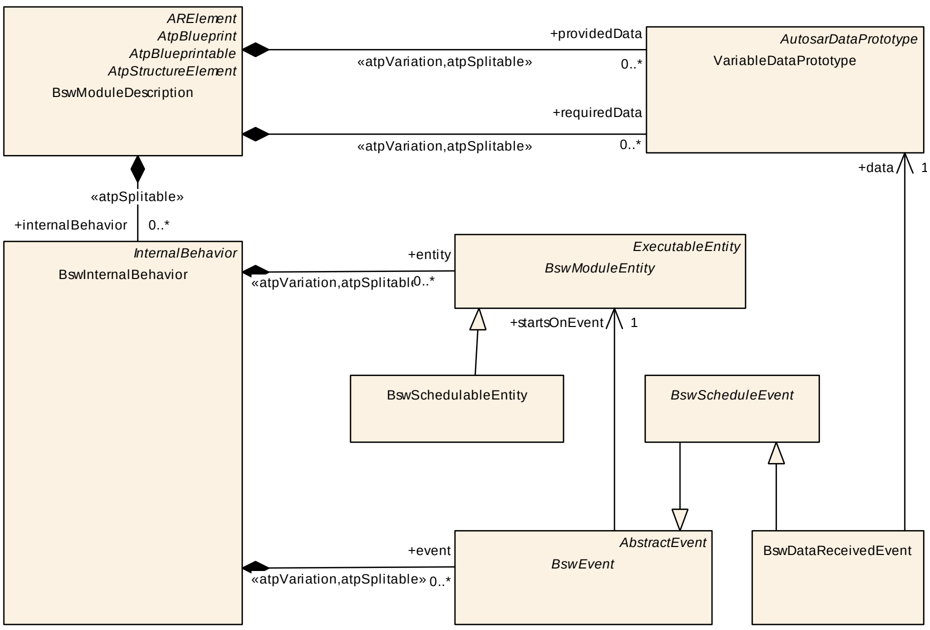

Figure 6.11: Details on BSW Events related to Sender-Receiver Communication

[4]This does not exclude configurations where sender and receiver are executed in the same partition.

Class	BswDataReceivedEvent			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	This event is thrown on reception of the referenced data via Sender-Receiver-Communication over the BSW Scheduler.			
Base	*ARObject*, *AbstractEvent*, *BswEvent*, *BswScheduleEvent*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
data	VariableDataPrototype	1	ref	The received data.

Table 6.35: BswDataReceivedEvent

6.8 Activation Reason of a BSW Module Entity

It is feasible to activate a given `BswModuleEntity` by means of several `BswEvent`s. In many cases, it is therefore necessary to retrieve the information about the activating `BswEvent` from within the implementation of the `BswModuleEntity`.

As a typical use case, consider a `BswSchedulableEntity` that is cyclically activated (by means of a `BswTimingEvent`) and in addition it shall also be executed sporadically, e.g. in response to mode switch (`BswModeSwitchEvent`).

By using the meta-model extract shown in Figure 6.12 (which is further explained in [6]) it is possible to generate the RTE in a way that it provides a bit vector representing the activation reason to the `BswModuleEntity`.

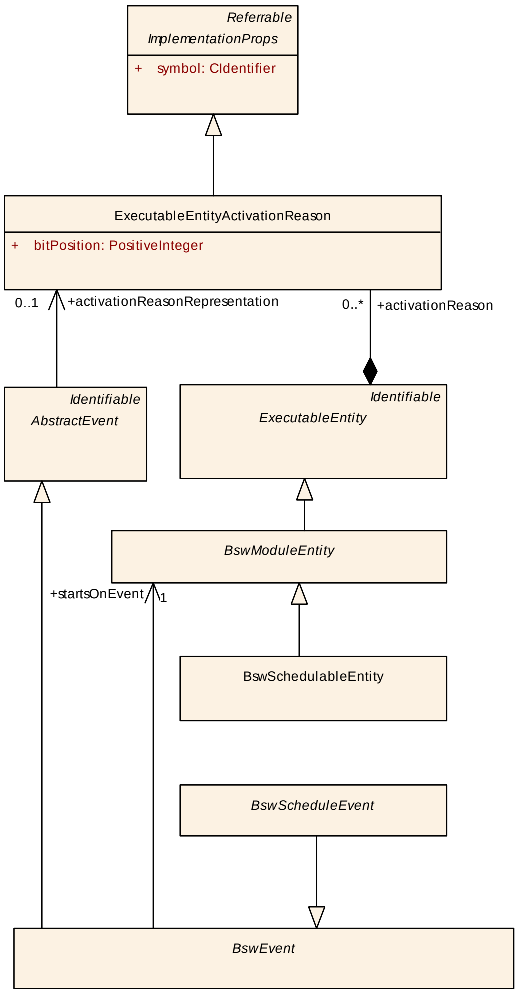

Figure 6.12: BswModuleEntity and activation reason

[TPS_BSWMDT_04089] Access to activation reason ⌈The same mechanism is available for both application software and basic software, therefore the following specification items and constraints defined in [6] also hold for the BSWMDT:

- [TPS_SWCT_01469]
- [constr_1226]
- [constr_1227]

⌋*(RS_BSWMD_00063)*

An activation reason can only be provided to those BswModuleEntity-s that are potentially triggered by BswEvents and thus are handled by the RTE. As a further restriction, the current RTE Specification [13] does not support retrieving the activation reason for BswCalledEntitys even if they are triggered via the BSW Scheduler. This leads to the following constraint:

[constr_4070] Applicability of BswModuleEntity.activationReason ⌈ An activationReason shall not be set

- for instances of BswInterruptEntity
- for instances of BswCalledEntity

⌋()

6.9 BSW Communication Policy

The implementation of triggers, mode switches and sender-receiver-communication can follow various policies which have to be known by the generator of the RTE resp. BSW Scheduler in order to generate the correct "glue" code. The required attributes are shown in Figures 6.13 and 6.14 and are explained in the class tables below.

This kind of information is similar to what is represented by the so-called ComSpecs for VFB communication, see [6].

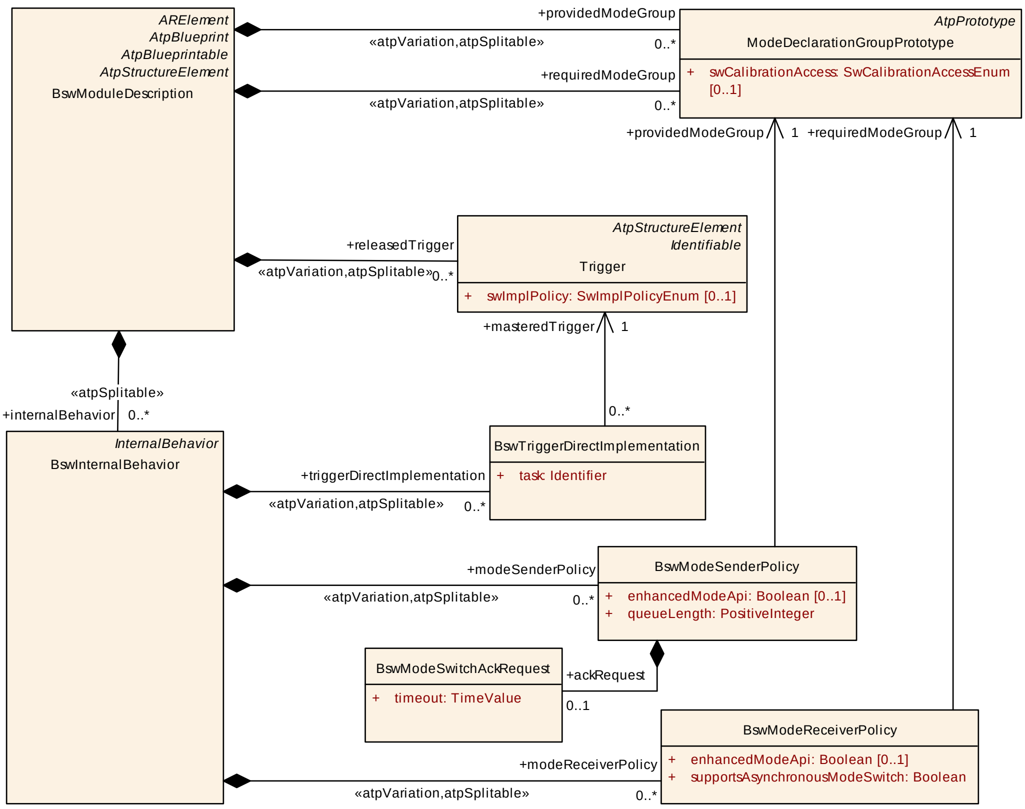

Figure 6.13: Special Implementation Policy for Modes and Triggers

Class	**BswTriggerDirectImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies a released trigger to be directly implemented via OS calls, for example in a Complex Driver module.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
masteredTrigger	Trigger	1	ref	The trigger which is directly mastered by this module. There may be several different BswTriggerDirectImplementations mastering the same Trigger. This may be required e.g. due to memory partitioning.
task	Identifier	1	attr	The name of the OS task, which is controlled by the referred trigger. This means, that the module uses the trigger condition to directly activate an OS task instead of calling an API of the BswScheduler. The task name is required by the RTE generator resp. BswScheduler to raise the appropriate events in components or modules receiving the trigger.

Table 6.36: BswTriggerDirectImplementation

Class	**BswModeSenderPolicy**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the details for the sending of a mode switch for the referred mode group.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
ackRequest	BswModeSwitchAckRequest	0..1	aggr	Request for acknowledgement
enhancedModeApi	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to TRUE the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
providedModeGroup	ModeDeclarationGroupPrototype	1	ref	The provided mode group for which the policy is specified.
queueLength	PositiveInteger	1	attr	Length of call queue on the sender side. The queue is implemented by the RTE resp.BswScheduler. The value must be greater or equal to 0. Setting the value of queueLength to 0 implies non-queued communication.

Table 6.37: BswModeSenderPolicy

Class	**BswModeSwitchAckRequest**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Requests acknowledgements that a mode switch has been processed successfully			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
timeout	TimeValue	1	attr	Number of seconds before an error is reported.

Table 6.38: BswModeSwitchAckRequest

Class	**BswModeReceiverPolicy**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the details for the reception of a mode switch for the referred mode group.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
enhancedMode Api	Boolean	0..1	attr	This controls the creation of the enhanced mode API that returns information about the previous mode and the next mode. If set to TRUE the enhanced mode API is supposed to be generated. For more details please refer to the SWS_RTE.
requiredMode Group	ModeDeclarationGroup Prototype	1	ref	The required mode group for which the policy is specified.
supports Asynchronous ModeSwitch	Boolean	1	attr	Specifies whether the module can handle the reception of an asynchronous mode switch (true) or not (false).

Table 6.39: BswModeReceiverPolicy

[TPS_BSWMDT_04107] Data reception policy ⌈ By aggregating a BswDataReceptionPolicy a BswInternalBehavior specifies the detailed reception policy of the referred VariableDataPrototype. Note the reception policy is the same for all reception points - defined via BswModuleEntity.dataReceivePoint - of the respective VariableDataPrototype in this module. ⌋*(RS_BSWMD_00067)*

Note that due to limitations of the sender-receiver communication mechanism in BSW (in contrast to VFB communication) it is only possible to specify queued reception. Furthermore, there are no communication attributes on the sender side.

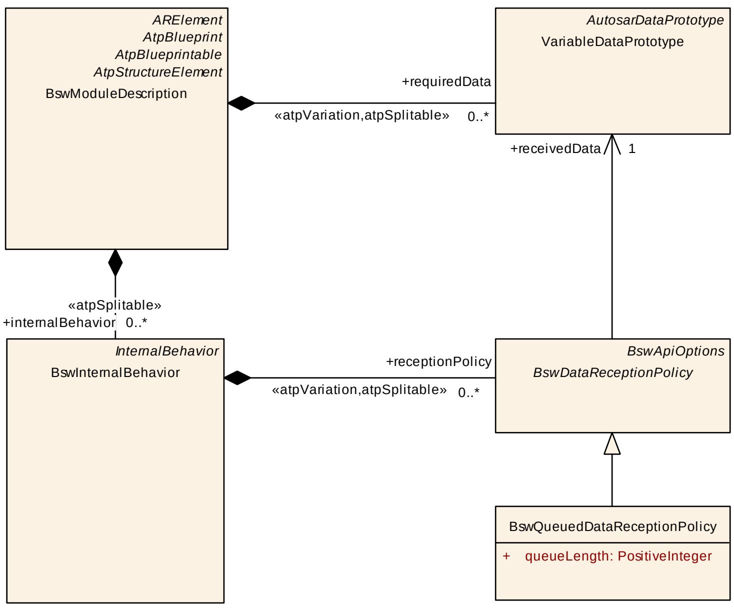

Figure 6.14: Implementation Policy for BSW Sender-Receiver Communication

— AUTOSAR CONFIDENTIAL —

Class	***BswDataReceptionPolicy*** (abstract)			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specifies the reception policy for the referred data in sender-receiver communication over the BSW Scheduler. To be used for inter-partition and/or inter-core communication.			
Base	*ARObject*, *BswApiOptions*			
Subclasses	BswQueuedDataReceptionPolicy			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
receivedData	VariableDataPrototype	1	ref	The data received over the BSW Scheduler using this policy.

Table 6.40: BswDataReceptionPolicy

Class	BswQueuedDataReceptionPolicy			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Reception policy attributes specific for queued receiving.			
Base	*ARObject*, *BswApiOptions*, *BswDataReceptionPolicy*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
queueLength	PositiveInteger	1	attr	Length of queue for received events.

Table 6.41: BswQueuedDataReceptionPolicy

[constr_4080] Existence of reception policy ⌈ If a `VariableDataPrototype` is referred from a `dataReceivePoint` of any `BswModuleEntity` in a given `BswInternalBehavior`, then exactly one corresponding `BswDataReceptionPolicy` must by aggregated by this `BswInternalBehavior`. ⌋*()*

6.10 BSW Local Data

A BSW module (or cluster) needs the ability to declare data in its BSWMD, for example

- in order to make them available for measurement and calibration tools (see chapter 10)
- in order to declare these data in relation to ServiceNeeds, e.g. as NvM blocks (see chapter 13)

[TPS_BSWMDT_04026] Local BSW data without RTE or BSW Scheduler support ⌈ In many cases such data in the context of a module (or cluster) do not need any support by the RTE resp. BSW Scheduler. They are simply allocated by the module's code but they still may be accessed from outside of the module for measurement, calibration or as NvM mirrors. These data are described by the following roles:

- `BswInternalBehavior.staticMemory` for variable data
- `BswInternalBehavior.constantMemory` for constant data

⌋*(RS_BSWMD_00045, RS_BSWMD_00052, RS_BSWMD_00062)*

[TPS_BSWMDT_04027] Local BSW data accessed via BSW Scheduler API ⌈ However it is also possible to have local data allocated by the BSW Scheduler. This is especially required in the case of calibration with software emulation. These kind of data are declared by:

- BswInternalBehavior.perInstanceParameter

⌋(RS_BSWMD_00030, RS_BSWMD_00062)

For compatibility reasons with the SWCT these various data are declared on the behavior level using the abstract class InternalBehavior as shown in figure 6.15. The class table for InternalBehavior has already been listed in chapter 6.1.

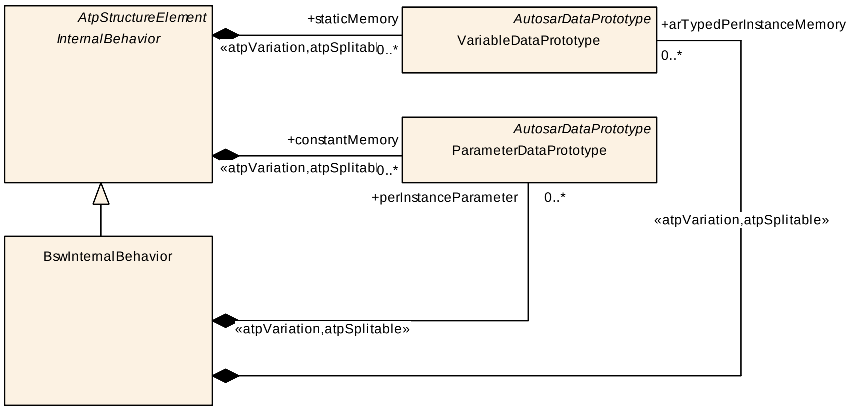

Figure 6.15: BSW Local Data

[TPS_BSWMDT_04128] BSW measurement data accessed via BSW Scheduler API ⌈ BSW measurement data accessed via BSW Scheduler API It is also possible to have local data allocated by the BSW Scheduler. This kind of data is declared by

- BswInternalBehavior.ArTypedPerInstanceMemory

⌋(RS_BSWMD_00030, RS_BSWMD_00062)

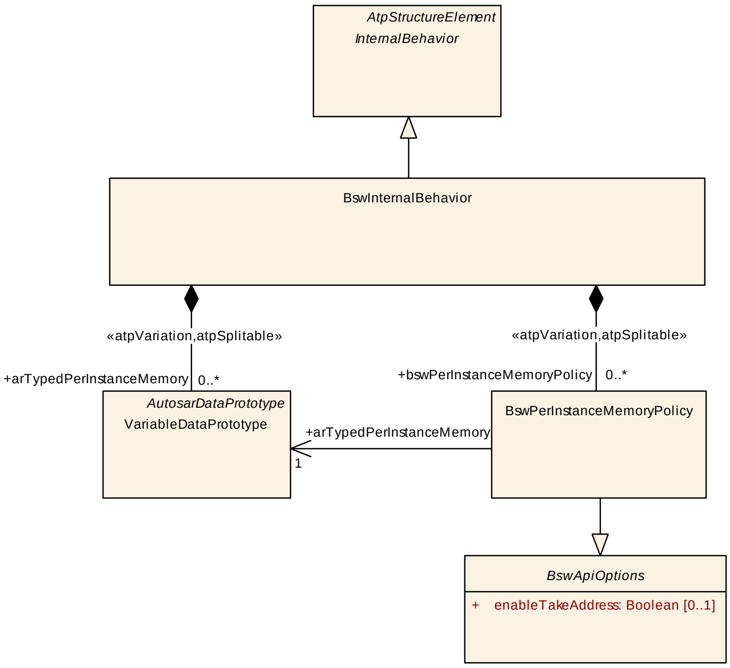

Figure 6.16: BSW Measurement Data

These data use the type system of AutosarDataPrototypes which is explained in more detail in [6]:

Class	**ParameterDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A parameter element used for parameter interface and internal behavior, supporting signal like parameter and characteristic value communication patterns and parameter and characteristic value definition.			
Base	*ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, Multilanguage Referrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype

Table 6.42: ParameterDataPrototype

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	*ARObject, AtpFeature, AtpPrototype, AutosarDataPrototype, DataPrototype, Identifiable, Multilanguage Referrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table 6.43: VariableDataPrototype

6.11 Synchronization with a Corresponding SWC

BSW modules which implement a ServiceSwComponentType, EcuAbstraction-SwComponentType or ComplexDeviceDriverSwComponentType require several mappings between their SWC description and BSWM description in order to generate the RTE resp. the BSW Scheduler.

One use case is as follows:

[TPS_BSWMDT_04074] Synchronization of mode switches or triggers ⌈ A BSW module which communicates via the RTE is able to provide triggers and mode switches within the basic software and toward SWCs above the RTE as well (for example a BSW module implementing an EcuAbstractionSwComponentType). It may happen, that a module wants to issue a mode switch or a trigger to both BSW and to SWCs "above the RTE" , i.e. a call via the BSW Scheduler API shall result in the same trigger resp. mode switch as a call via the RTE port-API (details are specified in [13]). In this case the Trigger resp. ModeDeclarationGroupPrototype provided within the BSW must be mapped to the Trigger resp. ModeDeclarationGroupPrototype provided by the port interface. This information is an input to configure the RTE accordingly. ⌋*(RS_BSWMD_00055, RS_BSWMD_00058)*

Another use case is the specification of a RunnableEntity in a BSW module in order to allow calls to or from the RTE via ports:

[TPS_BSWMDT_04075] RunnableEntity in BSW for RTE access ⌈ In this case, a BswModuleEntity should be specified in addition to allow for the BSW specific descriptions and the two elements have to be associated. This is e.g. required, if the RTE needs to find out whether a RunnableEntity runs in interrupt context. ⌋*()*

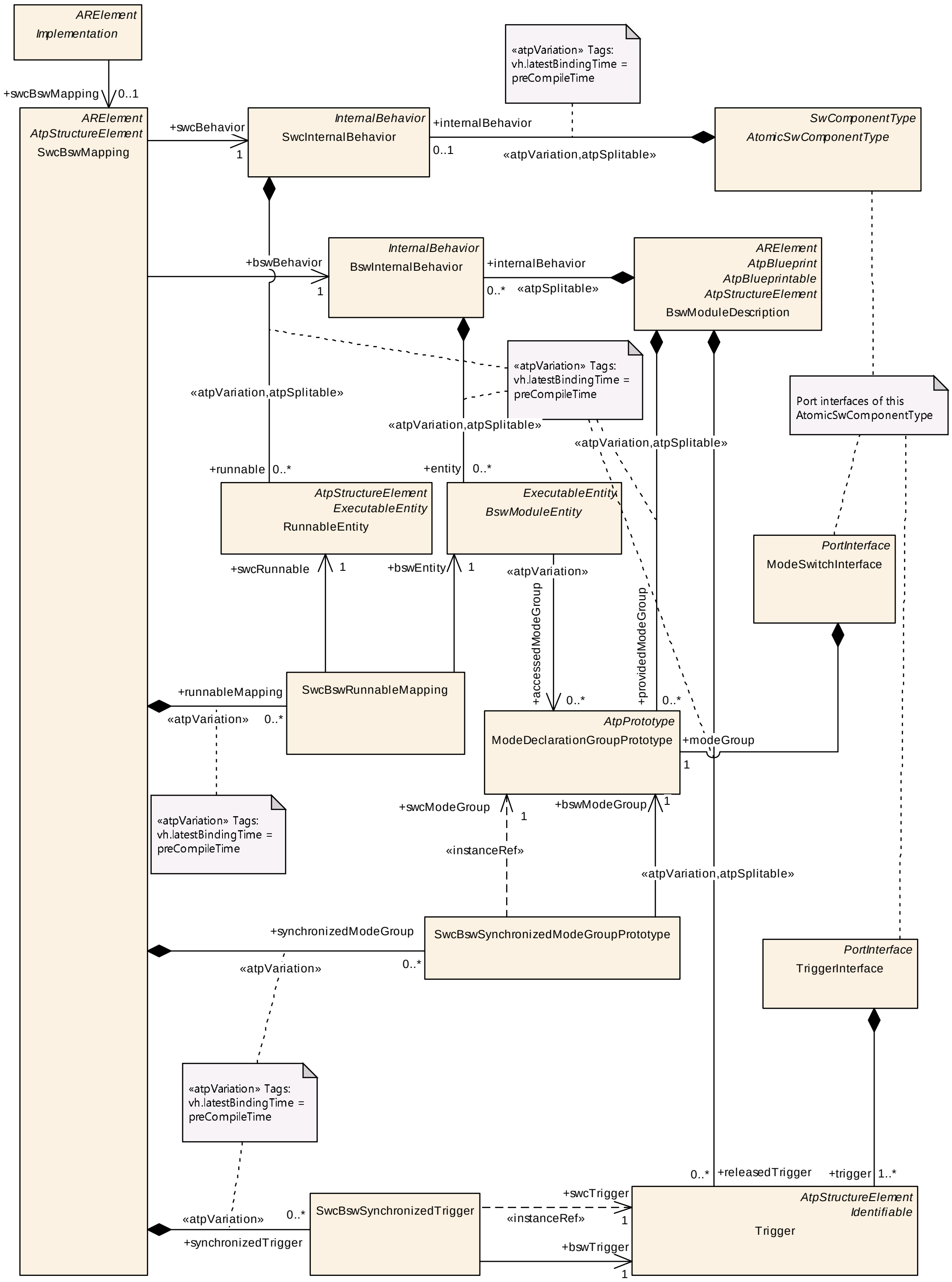

Figure 6.17: Mapping between an SWC and a BSW module.

Class	SwcBswMapping			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Maps an SwcInternalBehavior to an BswInternalBehavior. This is required to coordinate the API generation and the scheduling for AUTOSAR Service Components, ECU Abstraction Components and Complex Driver Components by the RTE and the BSW scheduling mechanisms. **Tags:** atp.recommendedPackage=SwcBswMappings			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
bswBehavior	BswInternalBehavior	1	ref	The mapped BswInternalBehavior
runnableMapping	SwcBswRunnableMapping	*	aggr	A mapping between a pair of SWC and BSW runnables. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swcBehavior	SwcInternalBehavior	1	ref	The mapped SwcInternalBehavior.
synchronizedModeGroup	SwcBswSynchronizedModeGroupPrototype	*	aggr	A pair of SWC and BSW mode group prototypes to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
synchronizedTrigger	SwcBswSynchronizedTrigger	*	aggr	A pair of SWC and BSW Triggers to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 6.44: SwcBswMapping

Class	SwcBswRunnableMapping			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Maps a BswModuleEntity to a RunnableEntity if it is implemented as part of a BSW module (in the case of an AUTOSAR Service, a Complex Driver or an ECU Abstraction). The mapping can be used by a tool to find relevant information on the behavior, e.g. whether the bswEntity shall be running in interrupt context.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
bswEntity	BswModuleEntity	1	ref	The mapped BswModuleEntity
swcRunnable	RunnableEntity	1	ref	The mapped SWC runnable.

Table 6.45: SwcBswRunnableMapping

Class	SwcBswSynchronizedModeGroupPrototype			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Synchronizes a mode group provided by a component via a port with a mode group provided by a BSW module or cluster.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
bswModeGroup	ModeDeclarationGroupPrototype	1	ref	The BSW mode group prototype.
swcModeGroup	ModeDeclarationGroupPrototype	1	iref	The SWC mode group prototype provided by a particular port.

Table 6.46: SwcBswSynchronizedModeGroupPrototype

Class	SwcBswSynchronizedTrigger			
Package	M2::AUTOSARTemplates::CommonStructure::SwcBswMapping			
Note	Synchronizes a Trigger provided by a component via a port with a Trigger provided by a BSW module or cluster.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
bswTrigger	Trigger	1	ref	The BSW Trigger.
swcTrigger	Trigger	1	iref	The SWC Trigger provided by a particular port.

Table 6.47: SwcBswSynchronizedTrigger

[TPS_BSWMDT_04028] Determination of argument names for BSW functions called via ports ⌈ In the case of functions calls via ports over the RTE, the RTE API generator shall determine the name of function arguments (for declaration purposes only) from the signature of the `BswModuleEntry` referred via the mapping.

The rule is:
The name of the function arguments shall be taken (in the given order) from
- the `shortName`s of the
- `SwServiceArg`s (according to the given order) defined in the
- `BswModuleEntry` referenced by the
- `BswModuleEntity` mapped in the
- `SwcBswRunnableMapping` to the
- `RunnableEntity` referenced by the
- `OperationInvokedEvent` that in turn references the
- `ClientServerOperation` that belongs to the
- `ClientServerInterface` that types the
- `PortPrototype` in question.
This rule applies to `PortDefinedArgumentValue` and "ordinary" port operation arguments as well.

If a `SwcBswRunnableMapping` exists, the above rule supersedes the definition of any argument identifiers by the attribute(s) `RunnableEntity.runnableEntityArgument`. ⌋*(RS_BSWMD_00039)*

The meta-model elements involved in this rule are shown in the following diagram.

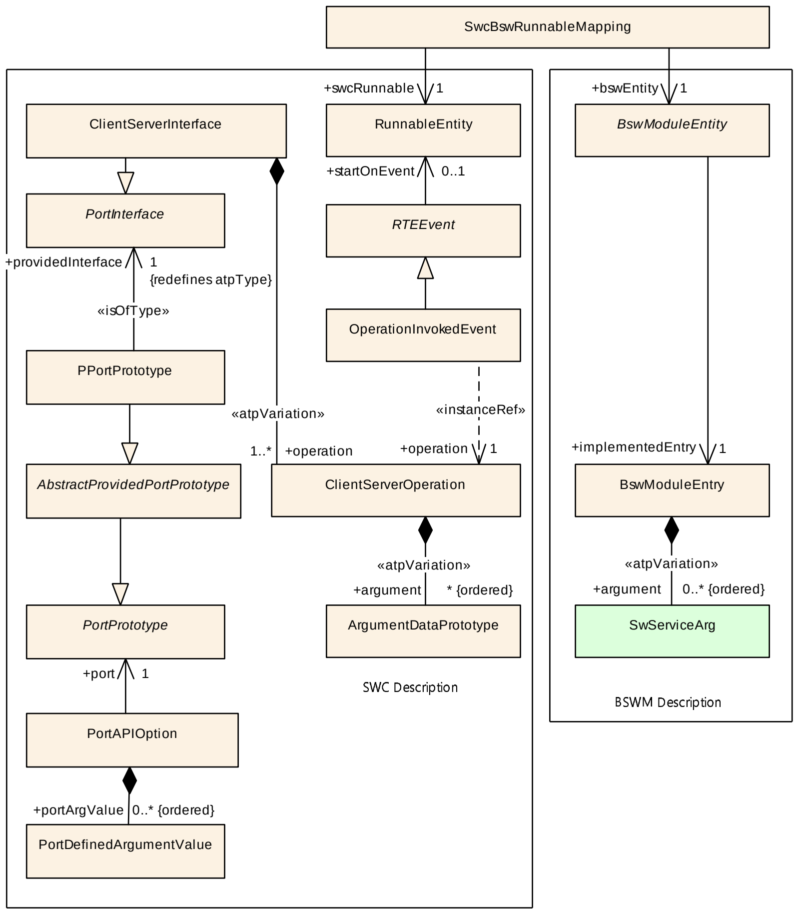

Figure 6.18: Mapping of function arguments between an SWC and a BSW module.

All mappings for one component/module are aggregated in SwcBswMapping which belongs to the CommonStructure of the meta-model. The mapping is considered as an add-on to the internal behavior (because it is mainly required to set up the RTE) but can be specified as a separate artifact which can be referred by the Implementation of the module. Therefore SwcBswMapping is derived from ARElement.

[TPS_BSWMDT_04138] Determination of the BswModuleEntry symbol ⌈ The symbol of the BswModuleEntry is composed as following: <bsnp>[_<vi>_<ai>]_<name> where:

<bsnp> the BswModuleDescription shortName if no BswSchedulerNamePrefix is defined or the value of the symbol attribute of the BswSchedulerNamePrefix of the BswModuleEntity if a BswSchedulerNamePrefix is defined,

<vi> is the vendorId of the BSW module,

<ai> is the vendorApiInfix of the BSW module,

<name> is the substring after "<bsnp>_" of the BswModuleEntry shortName referred as implementedEntry.

However if <bsnp>_ is not the prefix of the related BswModuleEntry shortName then <name> is identical to BswModuleEntry.shortName.

Please note also the SWS_RTE for further details. ⌋()

This synchronization mechanism between software components and BSW modules is limited to the relevant parts of the basic software:

[constr_4039] Semantics of SwcBswMapping ⌈ An SwcBswMapping is only valid, if the referred SwcInternalBehavior is aggregated by a ServiceSwComponentType, EcuAbstractionSwComponentType or ComplexDeviceDriverSwComponentType. ⌋()

[constr_4084] Consistency of references of InternalBehavior ⌈ The SwcInternalBehavior referenced by SwcBswMapping.SwcBehavior in the SwcBswMapping determined by SwcImplementation.swcBswMapping shall be identical to the SwcInternalBehavior referenced by SwcImplementation.behavior. ⌋()

[constr_4085] Consistency of references of InternalBehavior ⌈ The BswInternalBehavior referenced by SwcBswMapping.bswBehavior in the SwcBswMapping determined by BswImplementation.swcBswMapping shall be identical to the BswInternalBehavior referenced by BswImplementation.behavior. ⌋()

Further constraints are:

[constr_4071] Synchronized runnables and schedulable entities must be consistent ⌈ In the case that a RunnableEntity is mapped to a BswSchedulableEntity the RTE Generator emits an Entry Point Prototype only for the BswSchedulableEntity (depending on the specified events for SWC resp. BSW). The SwcBswRunnableMapping instance controlling this case is only valid if several attributes of the mapped RunnableEntity and BswSchedulableEntity are consistent, especially all of the following constraints apply to the attributes of the given instance of SwcBswRunnableMapping:

- swcRunnable.symbol must be identical to the symbol of bswEntity as defined in [TPS_BSWMDT_04138].
- swcRunnable.minimumStartInterval must be identical to bswEntity.minimumStartInterval.
- swcRunnable.canBeInvokedConcurrently must be identical to bswEntity.implementedEntry.isReentrant.
- swcRunnable.swAddrMethod must either be empty or must have identical attributes as the SwAddrMethod defined via bswEntity.swAddrMethod. This is required to ensure a unique configuration for the memory segment of the underlying code entity.
- swcRunnable.activationReason and bswEntity.activationReason must have identical shortName if they define the same bitPosition and must have identical bitPosition if they define the same shortName

Please note also the SWS_RTE for further details. ⌋()

[constr_4040] Synchronized mode groups must have same type ⌈ SwcBswSynchronizedModeGroupPrototype can only refer to equally typed ModeDeclarationGroupPrototypes, i.e. which have identical ModeDeclarationGroups. ⌋*()*

[constr_4041] Synchronized mode groups must have same context ⌈ The mapping defined by SwcBswSynchronizedModeGroupPrototype implies that the component providing the one mode group prototype is also mapped to the module which provides the other mode group prototype by means of synchronizing their respective behaviors in SwcBswMapping. ⌋*()*

[constr_4042] Synchronized triggers must have same context ⌈ The mapping defined by SwcBswSynchronizedTrigger implies that the component providing the one trigger is also mapped to the module which provides the other trigger by means of synchronizing their respective behaviors in SwcBswMapping. ⌋*()*

[constr_4064] Synchronized triggers must implement same policy ⌈ The mapping defined by SwcBswSynchronizedTrigger is only valid if the attribute SwcBswSynchronizedTrigger.swcTrigger.swImplPolicy has the same value as the attribute SwcBswSynchronizedTrigger.bswTrigger.swImplPolicy. ⌋*()*

The next constraint is to avoid conflicts in generated header files for the same reason as constraint [constr_4059] does within one module (see 5.2):

[constr_4058] Different mode groups in mapped BSWM and SWC must have different names ⌈ If an SwcInternalBehavior is mapped to a BswInternalBehavior the corresponding SWC and BSW module descriptions may not refer to different ModeDeclarationGroups having the same shortName but different elements. This holds especially if these mode groups are not synchronized but used independently. ⌋
()

6.12 BSW Behavior Distributed over Partitions

There a valid use cases in which parts of a given BSW module are executed on different partitions related to different processor cores[5] within one ECU (see [RS_BSWMD_00068] and [16]). This includes the case, that on a given ECU different services of the same module run within different partitions and also the case, that on the same ECU the same service is available within different partitions.

In a BSWMD there is no strict information on the association of software entities to partitions or processor cores. This information is added later in the ECU configuration phase through the mapping of BswEvents to OS tasks which in turn are mapped to OsApplications which are assigned to a partition and/or processor core (see [18]). The BswModuleEntity-s that are driven by these BswEvents are then indirectly mapped to partitions and cores.

[5]AUTOSAR currently supports at most one BSW partition per core. However, the rules outlined here are independent on this restriction.

Note that under certain circumstances (e.g. no memory protection, reentrancy) it is possible to use BswModuleEntity-s and BswOperationInvokedEvents that are not mapped to tasks but still can be accessed from several partitions (see [16] for details).

Likewise, the information whether a service is potentially called across partition boundaries is added via ECU configuration of the BSW Scheduler (in case of BSW communication) or via port connectors created at ECU configuration time (in case of AUTOSAR Services).

Nonetheless the BswInternalBehavior must be prepared for such a configuration because pieces of a module's code that potentially will run in different partitions and shall be explicitly mapped to different tasks must be driven by separate BswEvents. In addition, it is useful to distinguish the communication behavior of a BswModuleEntity per partition, for example if it sends out data when running on one processor core and receives them when running on another core. Such information may be needed for the fine grained configuration of the RTE and IOC as well as for documentation, timing and call tree analysis.[6]

In particular, the following rules can be stated:

[TPS_BSWMDT_04108] BswInternalBehavior containing BswModuleEntity-s executed on different partitions ⌈ If a module is designed to let the same code entities (after proper ECU configuration) run in different partitions, each code entity shall be described by only one BswModuleEntity. In other words, for a given code there shall be no separate BswModuleEntity-s per partition.

Furthermore, in case the behavior per partition shall be distinguished, the following elements shall be provided in the module's BswInternalBehavior:

- Each potential partition context in which some of the contained BswModuleEntity-s are able to run shall be modeled by an aggregation of an instance of meta-class BswDistinguishedPartition, see figure 6.19. Note that this is an abstract notation and the concrete partition must be defined later in the process as part of the configuration of the "virtual" module EcuC, see [11].

- The BswEvents starting the BswModuleEntitys of this BswInternalBehavior must be separate per potential partition and - in case there are limitations - shall indicate by the reference BswEvent.contextLimitation to which partition they are allowed to be mapped.

- The BswModuleCallPoints of this BswInternalBehavior shall - in case there are limitations - indicate by the reference BswModuleCallPoint.contextLimitation in which partitions they are used.

[6]The code has the possibility to retrieve information on which processor core it is running - see [16] and/or by which event it was started, see 6.8.

— AUTOSAR CONFIDENTIAL —

- The BswVariableAccess elements of this BswInternalBehavior shall - in case there are limitations - indicate by the reference BswVariableAccess.contextLimitation in which partitions they are accessed.

Note that no BswOperationInvokedEvent and no BswModuleClientServerEntry are needed for a function that is provided only for callers within one partition.

Furthermore, this rule is not applicable for BswCalledEntity-s that shall always run in the task context of the caller. ⌋*(RS_BSWMD_00068)*

[TPS_BSWMDT_04109] BswInternalBehavior for the same AUTOSAR Service provided on different partitions ⌈ If a module is designed to implement an AUTOSAR Service - represented as a particular ServiceSwComponentType - which shall run (after proper ECU configuration) by the same code on several different BSW partitions in explicitly mapped tasks, then it is enough to define for each RunnableEntity one SwcBswRunnableMapping and one mapped BswModuleEntity. However, the necessary RTEEvents must be different for each potential partition.

This rule does not apply for those RTEEvents and their corresponding RunnableEntity-s and BswModuleEntity-s which shall not be mapped to tasks.

Rule [TPS_BSWMDT_04108] applies in addition, if the behavior of the involved BswModuleEntity-s shall be distinguished per partition. ⌋*(RS_BSWMD_00068)*

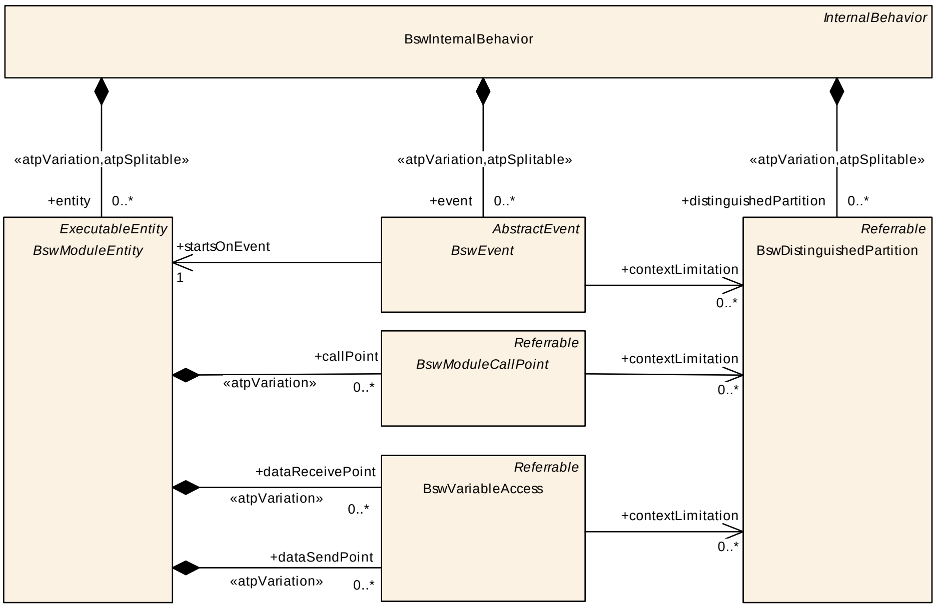

Figure 6.19: Usage of BswDistinguishedPartition.

Class	BswDistinguishedPartition			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Each instance of this meta-class represents an abstract partition in which context the code of the enclosing BswModuleBehavior can be executed. The intended use case is to distinguish between several partitions in order to implement different behavior per partition, for example to behave either as a master or satellite in a multicore ECU with shared BSW code.			
Base	*ARObject*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.48: BswDistinguishedPartition

[constr_4083] `BswDistinguishedPartition` shall be used only in the context of a particular `BswInternalBehavior` ⌈ All instances of `BswEvent`, `BswModuleCallPoint` and `BswVariableAccess` which refer to a `BswDistinguishedPartition` shall belong to the same `BswInternalBehavior` that also aggregates the referred `BswDistinguishedPartition`. ⌋*()*

7 BSW Implementation

7.1 Overview

The template elements to be used by the developer in order to document the actual implementation of a BSW module or cluster are very similar to what is needed for the same purpose in the case of SWCs. Therefore it is based on the `CommonStructure` part or the meta-model. This includes also the documentation of resource consumption. The generic classes of the meta-model used to document implementation and resource consumption are described in chapter 8 and chapter 9 in this document.

There are however some special features in describing the implementation of BSW. This is the purpose of the meta-class `BswImplementation` (see Figure 7.1 and the following class table).

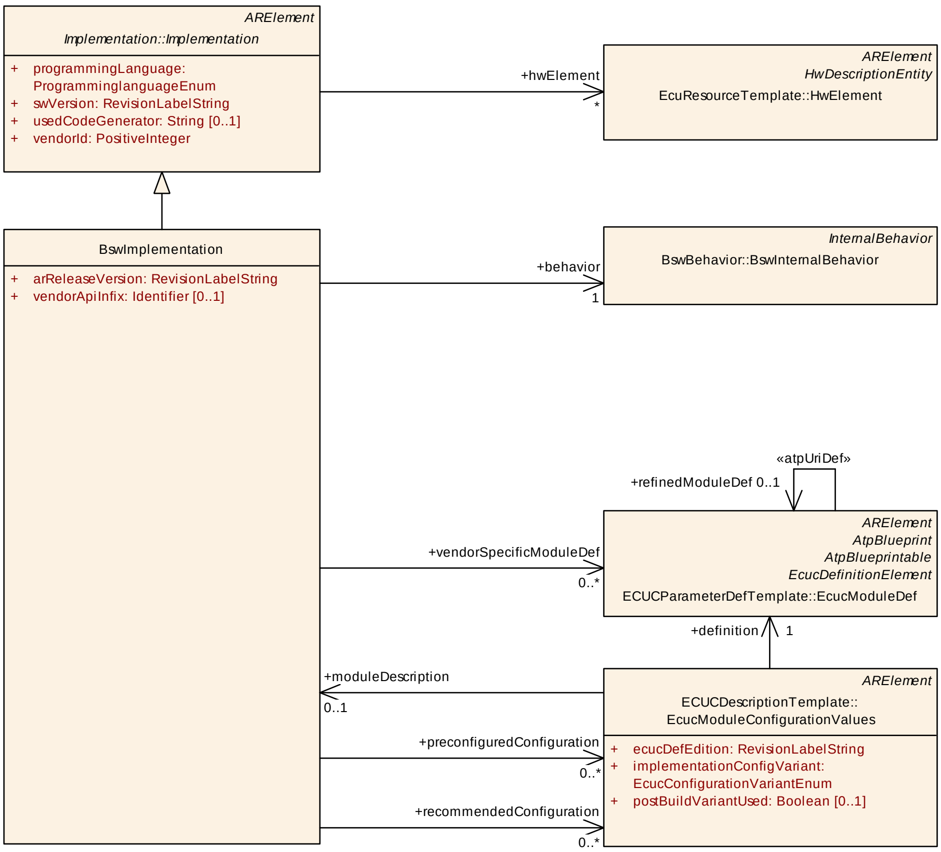

Figure 7.1: Overview of class BswImplementation

Class	**BswImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Contains the implementation specific information in addition to the generic specification (BswModule Description and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:** atp.recommendedPackage=BswImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
arRelease Version	RevisionLabelString	1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBehavior	1	ref	The behavior of this implementation. This relation is made as an association because • it follows the pattern of the SWCT • since ARElement cannot be splitted, but we want supply the implementation later, the Bsw Implementation is not aggregated in BswBehavior
preconfigured Configuration	EcucModule ConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:** xml.roleWrapperElement=true
recommended Configuration	EcucModule ConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiInfix	Identifier	0..1	attr	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <ModuleName>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecific ModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module ▽

▽

△

Class	BswImplementation			
				△ • several EcucModuleDefs used in this Bsw Implementation if it represents a cluster of modules • one or no EcucModuleDefs used in this Bsw Implementation if it represents a library **Tags:** xml.roleWrapperElement=true

Table 7.1: BswImplementation

[TPS_BSWMDT_04030] **BswImplementation.arReleaseVersion** ⌈ The inclusion of the AUTOSAR version information arReleaseVersion is specific for AUTOSAR BSW and specified per instance of BswImplementation. ⌋ *(RS_BSWMD_00001, RS_BSWMD_00025, RS_BSWMD_00043)*

[TPS_BSWMDT_04031] Instances of BswImplementation ⌈ Note that in case a BSW module can potentially be used in multiple implementations on the same ECU (which means, that the code has to be there multiple times with the exception of shared libraries), for each module implementation there has to be a separate instance of BswImplementation. This allows to define name expansions required for global symbols via the attribute vendorApiInfix. ⌋*(RS_BSWMD_00001, RS_BSWMD_00025, RS_BSWMD_00043)*

[constr_4099] Support of multiple instantiation ⌈ If a BSW Module supports multiple instantiation the attribute vendorApiInfix is mandatory. ⌋*()*

Note: If a standardized BSW Module shall support multiple instantiation is defined by AUTOSAR and described in the according STMD. For more information see [11]. It is the responsibility of a BSW Module vendor to apply unique vendorApiInfix values for its delivered modules.

[constr_4100] Uniqueness of module implementation prefixes ⌈ Inside one ECU the Module implementation prefixes (Mip) of BSW Modules shall be unique. ⌋*()*

Note: The definition of Mip is given in [SWS_BSW_00102]

The mechanism of vendorApiInfixes can be seen as a special method of resolving name conflicts. This aspect is further explained in [4] [TR_METH_03010].

The notation “Wayx” in Figure 7.2 and Figure 7.3 describes that a different HW mechanism (e.g. register set) can be used to achieve the same functionality (e.g. calculation of a PWM output).

Use-case for vendorApiInfixes would be that the microcontroller on chip and an off chip device provide the same functionality like e.g. CanDriver capabilities. Here the abstraction shall be done via the vendorApiInfixes.

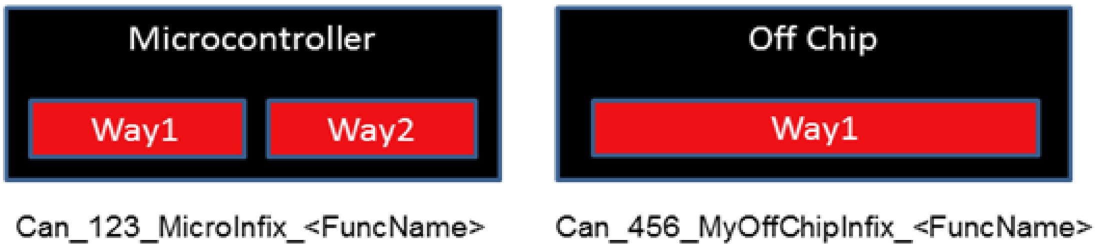

Figure 7.2: Example of a use case for `vendorApiInfix`

Non use-case for vendorApiInfixes would be that the microcontroller provides on chip for the calculation of a PWM different mechanisms for different channels. Here the abstraction shall be done via the handled ChannelNumber of the PWM.

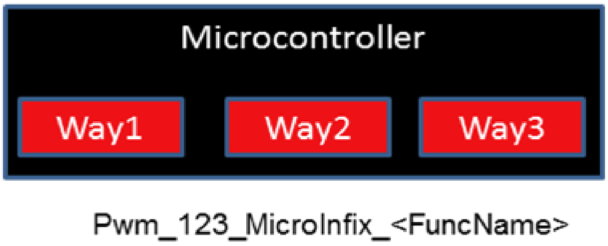

Figure 7.3: Example of a none use case for `vendorApiInfix`

With attribute `debugInfo` it is possible to specify information for the AUTOSAR BSW Debug Module.

[TPS_BSWMDT_04032] `Implementation.hwElement` ⌈ The attribute `hwElement` allows to document special hardware dependencies of a BSW module or cluster in addition to what can be expressed by the generic attribute `Implementation.resourceConsumption` ⌋*(RS_BSWMD_00009, RS_BSWMD_00026)* (see also chapter 9). The intended use case of this attribute is to document hardware dependencies of BSW modules or clusters namely in the layers MCAL, ECU Abstraction or Complex Drivers.

Finally it is possible to specify vendor specific configuration parameter definitions and predefined or recommended configuration parameter values within the scope of a BSW implementation and deliver them as part of a BSWMD. This is further explained in the next chapter.

7.2 Configuration Parameter Definitions and Values as Part of a BSWMD

[TPS_BSWMDT_04033] Reference to vendor specific configuration parameters ⌈ Vendor specific configuration parameters are expressed by an association from `BswImplementation` to `EcucModuleDef`. ⌋*(RS_BSWMD_00007, RS_BSWMD_00027, RS_BSWMD_00035, RS_BSWMD_00050)*

[TPS_BSWMDT_04034] Reference to predefined or recommended configuration values ⌈ Predefined or recommended configuration parameter values are expressed by associations from BswImplementation to EcucModuleConfigurationValues. ⌋*(RS_BSWMD_00007, RS_BSWMD_00032, RS_BSWMD_00033)*

The meta-classes EcucModuleDef and EcucModuleConfigurationValues are specified in the ECU Configuration Specification document [11].

Note that different implementations of the same BswModuleDescription can have different predefined or recommended parameter values and different sets of vendor specific configuration parameters. Of course it is also possible that different implementations of the same module refer to the same configuration parameter definitions resp. to the same predefined or recommended configuration parameter values.

A BswImplementation can either represent the implementation of a single module (or library) or the implementation of a cluster of modules. Therefore the following constraints hold for the multiplicities of the vendor specific configuration parameters and predefined configuration values:

[constr_4047] Multiplicity of vendor specific configuration parameters ⌈ The association BswImplementation.vendorSpecificModuleDef shall be implemented as reference to one or more instances of EcucModuleDef if the underlying BswModuleDescription has the category BSW_CLUSTER. In all other cases, it shall refer to exactly one instance of EcucModuleDef (the one belonging to this module). ⌋*()*

[constr_4048] Multiplicity of preconfigured values ⌈ The association BswImplementation.preconfiguredConfiguration shall be implemented as reference to zero or more different instances of EcucModuleConfigurationValues if the underlying BswModuleDescription has the category BSW_CLUSTER. In all other cases, it shall refer to at most one instance of EcucModuleConfigurationValues (the one belonging to this module). ⌋*()*

In order to specify the roles of predefined or recommended parameter values and distinguish them from the parameter value sets used finally in the ECU configuration, the following constraints hold for the enumeration attribute EcucModuleConfigurationValues.implementationConfigVariant (see [11] for definition and further usage of this attribute in the ECU configuration):

[constr_4045] implementationConfigVariant of preconfigured configuration ⌈ An EcucModuleConfigurationValues element with the implementationConfigVariant set to the value PreconfiguredConfiguration shall only be referenced in the role preconfiguredConfiguration and no other value for implementationConfigVariant is allowed in this role. ⌋*()*

[constr_4046] implementationConfigVariant of recommended configuration ⌈ An EcucModuleConfigurationValues element with the implementationConfigVariant set to the value RecommendedConfiguration shall only be referenced in the role recommendedConfiguration and no other value for implementationConfigVariant is allowed in this role. ⌋*()*

[TPS_BSWMDT_04035] Published parameter values ⌈Some AUTOSAR modules define so-called published parameters. A value of a published parameter cannot be set by the integrator, but has to be known. Thus the existence of published parameters always requires that their values have to be given as part of the preconfiguredConfiguration. ⌋*(RS_BSWMD_00024, RS_BSWMD_00033, RS_BSWMD_00043)*

[TPS_BSWMDT_04036] Back-reference from EcucModuleConfigurationValues ⌈ In addition the EcucModuleConfigurationValues from the ECU Configuration Template can refer to the BswImplementation for which it defines the configuration parameters. This relation is intended to be used by the integrator or tester to indicate for which BswImplementation an actual ECU configuration has been set up. ⌋*(RS_BSWMD_00001)*

8 Implementation

8.1 Introduction

This chapter explains, how the implementation details of AUTOSAR Software Components and Basic Software can be described. While AUTOSAR contains various component types, only Atomic Software Components and Basic Software Modules possess an Implementation. In the meta model this means that Implementation can be provided for AtomicSwComponentType or its derived classes and BswModuleDescription only.

On the other hand, compositions simply structure and encapsulate their contained components in a hierarchical manner, without adding any implementation relevant behavior or functionality. So they cannot be implemented directly. Instead, the leaf components in such a composition tree which by definition are again atomic, are implemented.

8.2 Implementation Description Overview

The Implementation class shown in Figure 8.1 serves the following main purposes:

- provide information about the resource consumption (chapter 9)
- link to code (source code, object code) (chapter 8.5)
- specify required and generated artifacts (chapter 8.6)
- specify the compiler (chapter 8.7)
- specify the linker (chapter 8.8)
- specify data to support measurement and calibration tools (chapter 10)

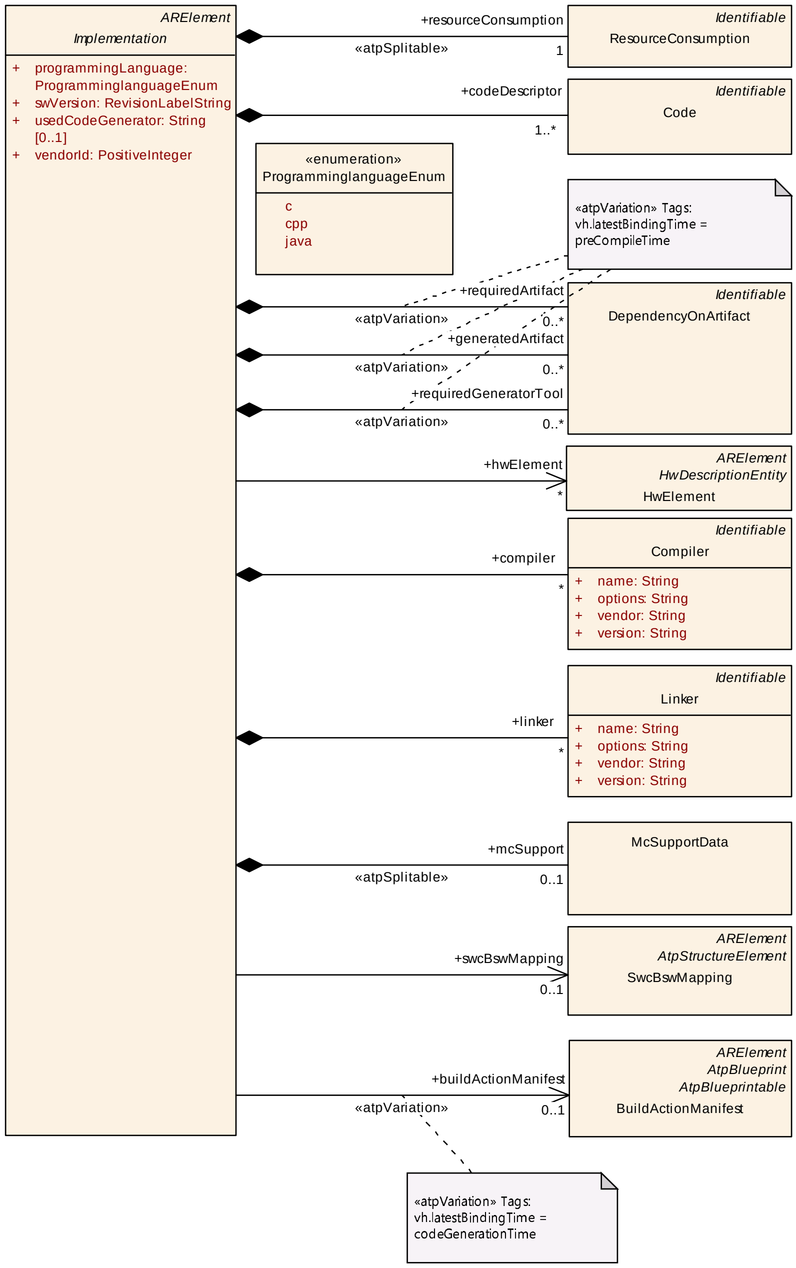

Figure 8.1: Overview of implementation description

As the figure shows, `Implementation` is derived from `ARElement`, i.e. it may be shipped as a separate engineering artifact, e.g. independent of the description of interfaces, ports and the component type.

The following table lists all attributes shown in Figure 8.1, thereby explaining the meaning of the remaining simple assertions and requirements of class `Implementation`.

Class	***Implementation*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Description of an implementation a single software component or module.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	BswImplementation, SwcImplementation			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
buildActionManifest	BuildActionManifest	0..1	ref	A manifest specifying the intended build actions for the software delivered with this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
codeDescriptor	Code	1..*	aggr	Specifies the provided implementation code.
compiler	Compiler	*	aggr	Specifies the compiler for which this implementation has been released
generatedArtifact	DependencyOnArtifact	*	aggr	Relates to an artifact that will be generated during the integration of this Implementation by an associated generator tool. Note that this is an optional information since it might not always be in the scope of a single module or component to provide this information. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
hwElement	HwElement	*	ref	The hardware elements (e.g. the processor) required for this implementation.
linker	Linker	*	aggr	Specifies the linker for which this implementation has been released.
mcSupport	McSupportData	0..1	aggr	The measurement & calibration support data belonging to this implementation. The aggregtion is «atpSplitable» because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcSupport
programmingLanguage	ProgramminglanguageEnum	1	attr	Programming language the implementation was created in.
requiredArtifact	DependencyOnArtifact	*	aggr	Specifies that this Implementation depends on the existance of another artifact (e.g. a library). This aggregation of DependencyOnArtifact is subject to variability with the purpose to support variability in the implementations. Different algorithms in the implementation might cause different dependencies, e.g. the number of used libraries. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
requiredGeneratorTool	DependencyOnArtifact	*	aggr	Relates this Implementation to a generator tool in order to generate additional artifacts during integration. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
resourceConsumption	ResourceConsumption	1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
swVersion	RevisionLabelString	1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.

▽

△

Class	***Implementation*** (abstract)			
swcBsw Mapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for AUTOSAR Service, ECU Abstraction and Complex Driver Components). It is up to the methodology to define whether this reference has to be set for the Swc- or Bsw Implementtion or for both.
usedCode Generator	String	0..1	attr	Optional: code generator used.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list

Table 8.1: Implementation

8.3 Assertions and Requirements

For some of the attributes mentioned below it is ambiguous whether they describe a requirement on the target environment or whether they are assertions made by the particular component implementation. The `Implementation` description's `compiler` attribute is an example for this: does it describe a requirement for source code to be compiled with the named compiler, or is this simply information which compiler was used in the process of creating an object file? The simple answer is: if possible, this is derived from the context. Otherwise the attribute needs to have proper documentation. For the `compiler` example just mentioned, the situation is straightforward: for source code, the attribute describes a requirement, for object code it is documented information. The same needs to be applied to all attributes in this section.

8.4 Implementation of a Software Component

[TPS_BSWMDT_04039] Association of an `Implementation` with a component or module ⌈Probably the most important information in `Implementation` is which Atomic Software Component or BSW Module is actually implemented. At first glance, this link seems to be missing in the overview in Figure 8.1. However, implementations are actually given for a particular component behavior, specified through the class `SwcInternalBehavior` respectively `BswInternalBehavior`. The contents of such a behavior are not of interest here, but as Figure 8.2 shows, it in turn is associated with a single `AtomicSwComponentType` or `BswModuleDescription`. ⌋ *(RS_BSWMD_00001)*

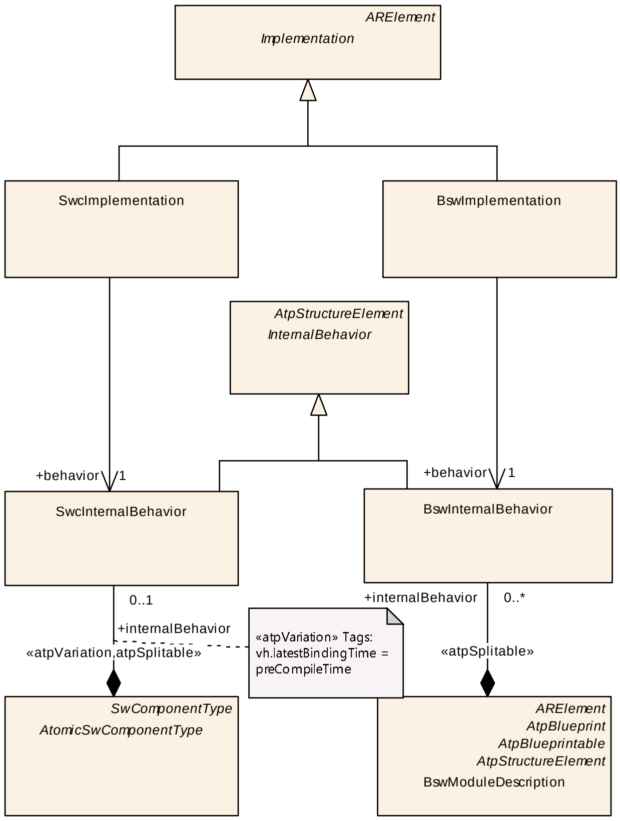

Figure 8.2: An implementation is associated with a single software component or module

8.5 Linking to Code

When a component is released the descriptions are accompanied by actual implementation code. This code can come in different ways: Source code in C, C++ or Java, object code or even executable code[1].

Figure 8.3 shows how an `Implementation` is linked to `Code`.

[TPS_BSWMDT_04040] **`Implementation.codeDescriptor`** ⌈For each available form of component code a `Code` element is used. For each `codeDescriptor`, all relevant artifacts are then referenced through the attribute `artifactDescriptor` (class `AutosarEngineeringObject`) which in turn references to a catalog of available files through a set of attributes as shown below. If for instance a component implementation is given as source code only, then the respective `Implementation` would contain exactly one `codeDescriptor`, whose `artifactDescriptor.category` attribute would denote the files to be source files. ⌋*(RS_BSWMD_00001, RS_BSWMD_00025)*

[1] Delivery of executable code is currently not supported by AUTOSAR.

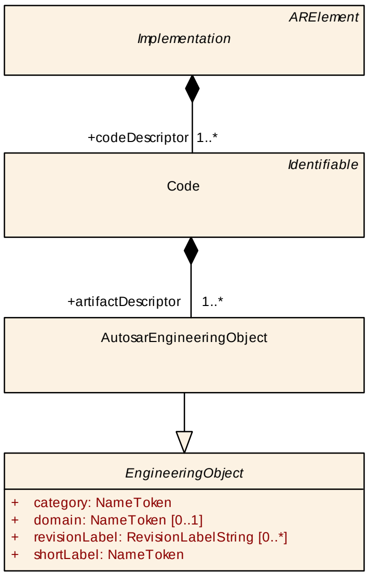

Figure 8.3: An `Implementation` references the code artifacts through the `Code` class

Class	**Code**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	A generic code descriptor. The type of the code (source or object) is defined via the category attribute of the associated engineering object.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
artifact Descriptor	AutosarEngineering Object	1..*	aggr	Refers to the artifact belonging to this code descriptor.
callbackHeader	ServiceNeeds	*	ref	The association callbackHeader describes in which header files the function declarations of callback functions are provided to a service module. With this information the service module can include the appropriate header files in its configuration files.

Table 8.2: Code

8.6 Dependencies

An implementation can generally depend on other artifacts, e.g. files. Such files could for example be required header, configuration or library files.

[TPS_BSWMDT_04041] **`DependencyOnArtifact`** ⌈This is described by the class `DependencyOnArtifact` which relates to meta-information via the class `AutosarEngineeringObject` as shown in Figure 8.4. ⌋*(RS_BSWMD_00034, RS_BSWMD_00037, RS_BSWMD_00044)*

[TPS_BSWMDT_04042] **Usage of `DependencyOnArtifact`** ⌈The class `DependencyOnArtifact` can be aggregated by `Implementation` in several different roles. By this it can also be used to specify that a certain generator tool is required to integrate a module and/or that a certain artifact is generated.

For libraries, like e.g. a `math.lib`, the desired version numbers can be specified via the attribute `revisionLabel`, therefore trying to ensure compatibility. Note that the specification of version numbers and other attributes is a meta-information about certain artifacts which must refer to a concrete catalog description. ⌋*(RS_BSWMD_00034, RS_BSWMD_00037, RS_BSWMD_00044)*This mechanism is described in more detail in the AUTOSAR Methodology, see [4].

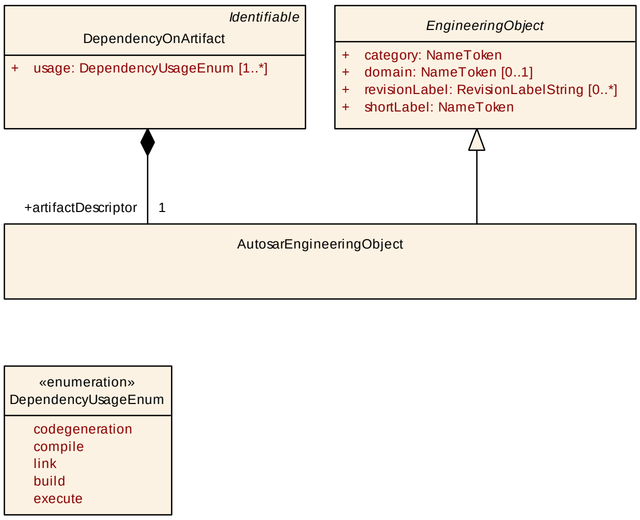

Figure 8.4: Dependencies of an `Implementation`

Class	**DependencyOnArtifact**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Dependency on the existence of another artifact, e.g. a library.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
artifact Descriptor	AutosarEngineering Object	1	aggr	The specified artifact needs to exist.
usage	DependencyUsage Enum	1..*	attr	Specification for which process step(s) this dependency is required.

Table 8.3: DependencyOnArtifact

Enumeration	**DependencyUsageEnum**
Package	M2::AUTOSARTemplates::CommonStructure::Implementation
Note	Enumeration describing the process steps a dependency is valid in.
Literal	***Description***
build	The object referred by the dependency is required during the build process. **Tags:** atp.EnumerationValue=0
codegeneration	The object referred by the dependency is required during code generation **Tags:** atp.EnumerationValue=1
compile	The object referred by the dependency is required during compilation. **Tags:** atp.EnumerationValue=2

▽

△

Enumeration	**DependencyUsageEnum**
execute	The object referred by the dependency is required at execution time. **Tags:** atp.EnumerationValue=3
link	The object referred by the dependency is required during linking. **Tags:** atp.EnumerationValue=4

Table 8.4: DependencyUsageEnum

Class	**AutosarEngineeringObject**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This denotes an engineering object being part of the process. It is a specialization of the abstract class EngineeringObject for usage within AUTOSAR.			
Base	*ARObject*, *EngineeringObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 8.5: AutosarEngineeringObject

Class	***EngineeringObject*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::EngineeringObject			
Note	This class specifies an engineering object. Usually such an object is represented by a file artifact. The properties of engineering object are such that the artifact can be found by querying an ASAM catalog file. The engineering object is uniquely identified by domain+category+shortLabel+revisionLabel.			
Base	*ARObject*			
Subclasses	AutosarEngineeringObject, BuildEngineeringObject, Graphic			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
category	NameToken	1	attr	This denotes the role of the engineering object in the development cycle. Categories are such as • SWSRC for source code • SWOBJ for object code • SWHDR for a C-header file Further roles need to be defined via Methodology. **Tags:** xml.sequenceOffset=20
domain	NameToken	0..1	attr	This denotes the domain in which the engineering object is stored. This allows to indicate various segments in the repository keeping the engineering objects. The domain may segregate companies, as well as automotive domains. Details need to be defined by the Methodology. Attribute is optional to support a default domain. **Tags:** xml.sequenceOffset=40
revisionLabel	RevisionLabelString	*	attr	This is a revision label denoting a particular version of the engineering object. **Tags:** xml.sequenceOffset=30

▽

△

Class	***EngineeringObject*** (abstract)			
shortLabel	NameToken	1	attr	This is the short name of the engineering object. Note that it is modeled as NameToken and not as Identifier since in ASAM-CC it is also a NameToken. **Tags:** xml.sequenceOffset=10

Table 8.6: EngineeringObject

8.7 Compiler

[TPS_BSWMDT_04043] **`Compiler`** ⌈For the specification of the used (or to be used) compiler the `Compiler` element shall be used: ⌋*(RS_BSWMD_00010)*

Class	**Compiler**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Specifies the compiler attributes. In case of source code this specifies requirements how the compiler shall be invoked. In case of object code this documents the used compiler settings.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
name	String	1	attr	Compiler name (like gcc).
options	String	1	attr	Specifies the compiler options.
vendor	String	1	attr	Vendor of compiler.
version	String	1	attr	Exact version of compiler executable.

Table 8.7: Compiler

8.8 Linker

[TPS_BSWMDT_04044] **`Linker`** ⌈For the specification of the to be used linker the `Linker` element shall be used: ⌋*()*

Class	**Linker**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Specifies the linker attributes used to describe how the linker shall be invoked.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
name	String	1	attr	Linker name.
options	String	1	attr	Specifies the linker options.
vendor	String	1	attr	Vendor of linker.
version	String	1	attr	Exact version of linker executable.

Table 8.8: Linker

8.9 Build Action Manifest

[TPS_BSWMDT_04085] `Implementation` refers to a `BuildActionManifest` ⌈An `Implementation` can optionally be linked to a `BuildActionManifest` in order to specify the intended build actions for the software delivered with this implementation. ⌋*(RS_BSWMD_00001, RS_BSWMD_00025)*

Class	BuildActionManifest			
Package	M2::AUTOSARTemplates::GenericStructure::BuildActionManifest			
Note	This meta-class represents the ability to specify a manifest for processing artifacts. An example use case is the processing of ECUC parameter values. **Tags:** atp.recommendedPackage=BuildActionManifests xml.globalElement=false			
Base	*ARElement, ARObject, AtpBlueprint, AtpBlueprintable, CollectableElement, Identifiable, Multilanguage Referrable, PackageableElement, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
buildAction	BuildAction	*	aggr	This represents a particular action in the build chain. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
buildAction Environment	BuildActionEnvironment	*	aggr	This represents a build action environment. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
dynamicAction	BuildAction	*	ref	This denots an Action which is to be executed as part of the dynamic action set.
startAction	BuildAction	*	ref	This specifies the list of actions to be performed at the beginning of the process. **Tags:** xml.sequenceOffset=-90
tearDownAction	BuildAction	*	ref	This specifies the set of action which shall be performed after all other actions in the manifest were performed. **Tags:** xml.sequenceOffset=-80

Table 8.9: BuildActionManifest

The setup of such a manifest is further explained in [1], see [TPS_GST_00294].

[TPS_BSWMDT_04086] Artifacts referred in `Implementation` and/or `BuildActionManifest` ⌈It should be noted that the `Implementation` instance as well as the `BuildActionManifest` instance can aggregate descriptive elements derived from meta-class `EngineeringObject` which eventually represent file artifacts to be used by the integrator. These two sets of artifacts may differ but are not necessarily exclusive, i.e. it shall be allowed to describe the same artifact under `Implementation` and under `BuildActionManifest` as well (of course not in contradiction).

Especially, the element `Implementation.codeDescriptor` is mandatory, so this element cannot be omitted even if an equivalent `EngineeringObject` describing the code file is part of the `BuildActionManifest`. ⌋*(RS_BSWMD_00001, RS_BSWMD_00025)*

9 ResourceConsumption

AUTOSAR software needs to be mapped on ECUs at some point during the development. Application Software Components can be basically mapped to any ECU available within the car. The mapping freedom is limited by the *System Constraints* [7] and the available resources on each ECU. BSW Modules are present in each ECU which provides the corresponding service. The ResourceConsumption element provides information about the needed resources concerning memory and execution time for each SwcImplementation or BswImplementation.

9.1 Static and Dynamic Resources

Resources can be divided into static and dynamic resources.

Static resources can only be allocated by one entity and stay with this entity. If the required amount of resources is bigger than the available resources the mapping does not fit physically. ROM is an example of a spare resource where obviously only the amount of data can be stored that is provided by the storage capacity.

Dynamic resources are shared and therefore can be allocated dynamically to different control threads over time. Processing time is a good example, where different tasks are given the processor for some time. If some runnable entity uses more processing time than originally planned, it can lead to functional failure. Also some sections of RAM can be seen as dynamic resources (e.g. stack, heap which grow and shrink dynamically).

9.2 Resource consumption overview

In Figure 9.1, the meta-model of the ResourceConsumption description is depicted.

[TPS_BSWMDT_04045] **Implementation.resourceConsumption** ⌈The ResourceConsumption is attached to an Implementation. For each Implementation, there is one ResourceConsumption description. ⌋*(RS_BSWMD_00001, RS_BSWMD_00005)*

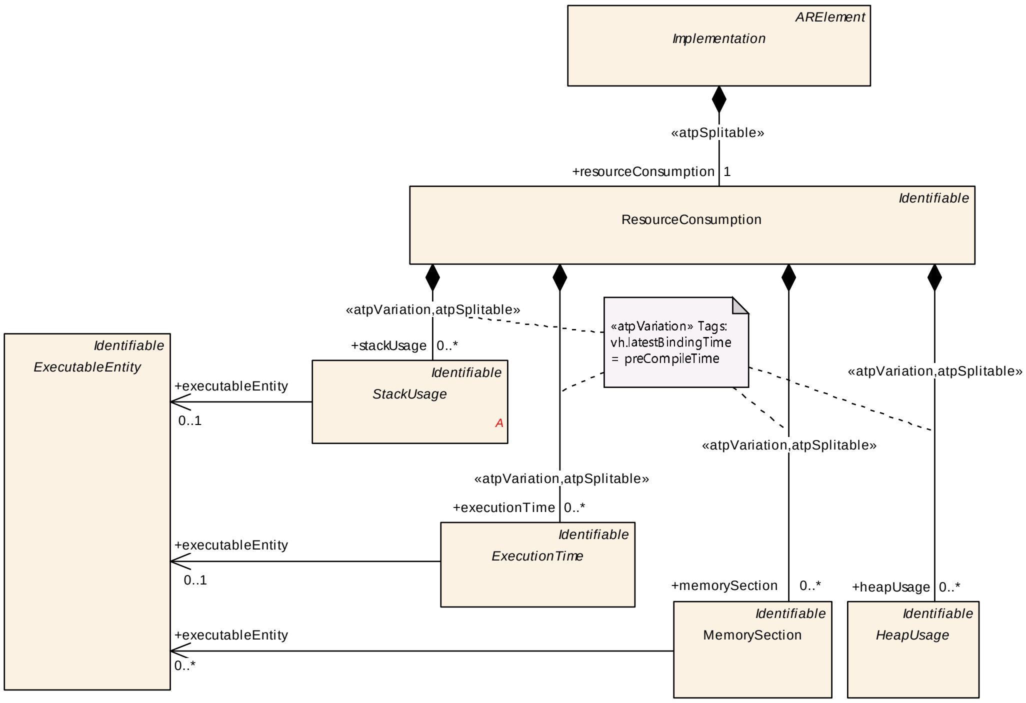

Figure 9.1: Resource consumption overview

As depicted by Figure 9.1, all resources are described within the `ResourceConsumption` meta-class.

`ExecutionTime` (chapter 9.5) and `StackUsage` (chapter 9.4.2) are used to provide information on the implementation specific resource usage of the `ExecutableEntity` defined in the `InternalBehavior` of SW-Component respectively in the `BswInternalBehavior` of BSW Module.

`MemorySection` (chapter 9.3.2) documents the resources needed to load the object file containing the implementation on the ECU.

`HeapUsage` (chapter 9.4.3) describes the dynamic memory usage of the software.

Class	**ResourceConsumption**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption			
Note	Description of consumed resources by one implementation of a software.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessCount Set	AccessCountSet	*	aggr	Set of access count values **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**ResourceConsumption**			
executionTime	ExecutionTime	*	aggr	Collection of the execution time descriptions for this implementation. The aggregation of executionTime is subject to variability with the purpose to support the conditional existence of runnable entities. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
heapUsage	HeapUsage	*	aggr	Collection of the heap memory allocated by this implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
memorySection	MemorySection	*	aggr	An abstract memory section required by this Implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
sectionName Prefix	SectionNamePrefix	*	aggr	A prefix to be used for the memory section symbol in the code. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
stackUsage	StackUsage	*	aggr	Collection of the stack memory usage for each runnable entity of this implementation. The aggregation of StackUsage is subject to variability with the purpose to support the conditional existence of runnable entities. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table 9.1: ResourceConsumption

9.3 Static Memory Needs

9.3.1 General

This sub-chapter describes how the static memory needs for the Implementation are specified. This includes all memory needs of software for code or data both at the class and at the instance level except for:

- stack space needed in the task that activates an ExecutableEntity of the implementation (see chapter 9.4.2)
- dynamic heap-behavior of the software (in case the software uses malloc/free to get/free buffers from the heap, see chapter 9.4.3[1])

9.3.2 Memory Sections

Memory will be needed to load the object-file containing an implementation of the software on an ECU. In which kind of memory the code and data of the software have to be allocated has to be defined in an abstract (i.e. platform and compiler independent) way in the source code of the software according to [19].

To support the integration and configuration of the software component or module the used (abstract) memory sections and their attributes have to be described also in XML via the MemorySection element from figure 9.2.

[1] This is often problematic in embedded and real-time systems: most software will only need static memory blocks and stack-size but will not require dynamic memory allocation

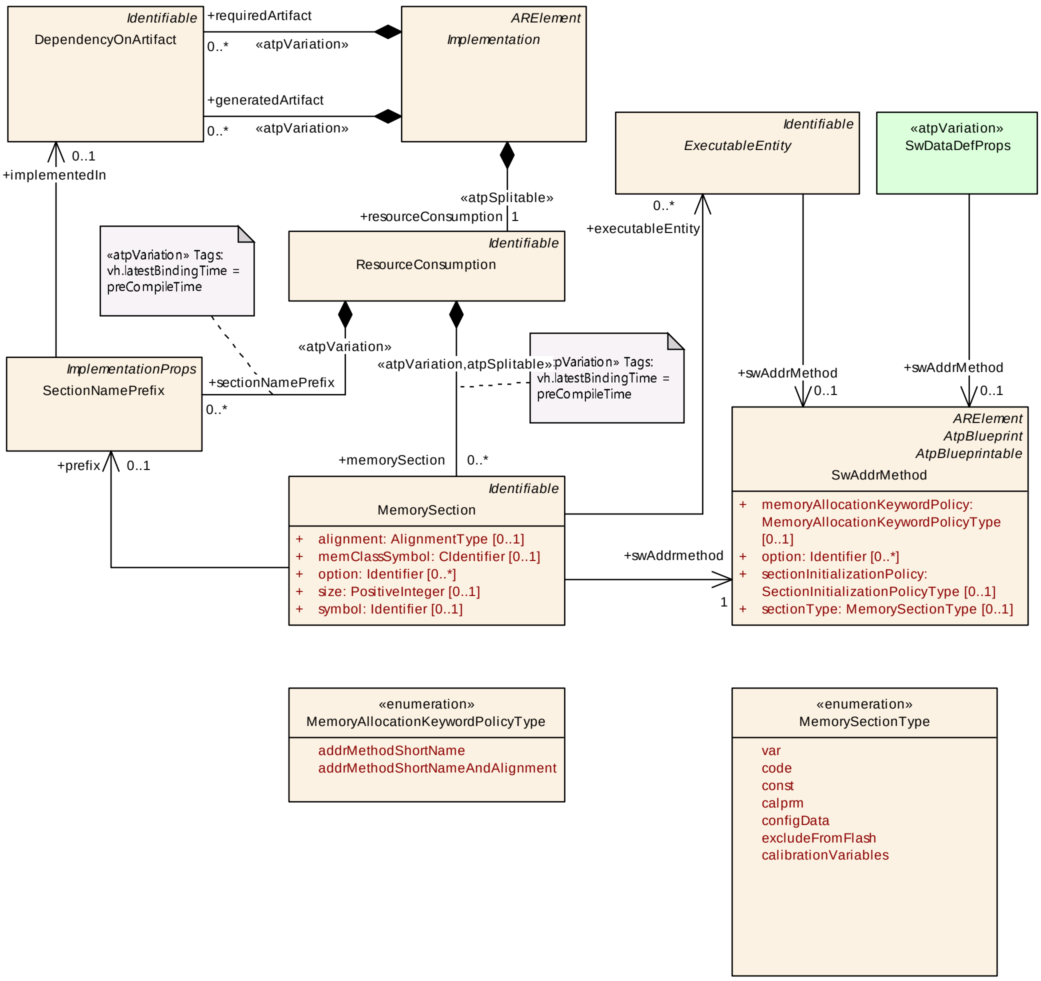

Figure 9.2: Meta-model related to the MemorySection

[TPS_BSWMDT_04046] Memory section name ⌈The actual section name is given by the MemorySection.symbol, if this attribute is missing the MemorySection.shortName is taken as default (this is for backwards compatibility reasons). The section name of each MemorySection instance shall be a part of the so-called memory allocation keyword used in preprocessor statements in the actual code.⌋ *(RS_BSWMD_00005, RS_BSWMD_00031)*

For example for a memory section entered by the macro RTE_START_SEC_VAR_FAST_8 the MemorySection.symbol shall be VAR_FAST_8.

The preprocessor macros contain in addition so-called prefixes which set up a kind of name space and by default are equal to the shortName of the enclosing BswModuleDescription or the AtomicSwComponentType (in the above example, the prefix is RTE).

[TPS_BSWMDT_04047] Memory section prefix ⌈It is possible to supersede these prefixes by more fine granular values using the meta-class SectionNamePrefix.⌋ *(RS_BSWMD_00031, RS_BSWMD_00014)*

There are basically two use cases to supersede the prefix of a memory allocation key word:

- A Basic Software Module Description provides the description for a ICC1 or ICC2 cluster which still has a sub granularity in its memory allocation implemented.
- A BSW module or software component is split into in allocatable memory parts. These memory parts are used to assign the sections (CODE, CONST, VAR) belonging to a certain functionality to a set of physical controller memories. For instance the interface code is put to memory which is rather fast accessibly from all interface users whereas the inner functionality is mapped to memory where the used hard ware can be accessed with less overhead.

[constr_4103] Name convention for `SectionNamePrefix` ⌈ In case a BSW module or Software Component is split into allocatable memory parts the `SectionNamePrefix` shall be set in the following form correspondingly:

- For BSW moduel: <MIP>_<feature>
- For Software Component: <software-component symbol name>_<feature>

where:

- <MIP>: is the capitalized Module Implementation Prefix
- <software-component symbol name>: is the symbol of the software component according to [TPS_SWCT_01000] in [6]
- <feature>: is the name of the sub-feature in the module or SWC denoting the allocatable memory part

⌋*()*

[constr_4104] Referencing of `MemorySection`s to `SectionNamePrefix` ⌈ In case a BSW module or Software Component is split into allocatable memory parts all `MemorySection`s belonging to the same allocatable memory part shall reference the identical `SectionNamePrefix` representing the allocatable memory part. ⌋*()*

The mapping of the allocation keywords to the compiler specific code is done via header files. It is possible to generate these header files from an ECU configuration description, which in turn is constrained by the `MemorySection`s and `SwAddrMethod`s used in the “upstream” descriptions of modules and components.

[TPS_BSWMDT_04092] Provide memory mapping header file names ⌈As a default rule, there is one memory mapping header file per BSW module or per SWC and the name of this file includes the `shortName` of the `BswModuleDescription` resp. the `AtomicSwComponentType` as a prefix.

However, for BSW modules or clusters it is possible to supersede the default rule by explicit reference to one or more files with specific names and granularity. This is specified by defining one or more `DependencyOnArtifact` elements aggregated by

BswImplementation in the role requiredArtifact and with DependencyOnArtifact.category set to the value MEMMAP.

The detailed rules on how these header file names are derived are given in [19]: [SWS_MemMap_00028], [SWS_MemMap_00029], [SWS_MemMap_00032], [SWS_MemMap_00035] ⌋*(RS_BSWMD_00025)*[2]

[TPS_BSWMDT_04097] Assigning different header files per section prefix ⌈In case more than one memory mapping header is referred by one BswImplementation according to [TPS_BSWMDT_04092], the different header files have to be assigned to individual memory section prefixes by setting the references SectionNamePrefix.implementedIn. ⌋*(RS_BSWMD_00025)*

[constr_4072] Constraints of SectionNamePrefix.implementedIn ⌈

- The SectionNamePrefix and the DependencyOnArtifact connected via this link must belong to the same BswImplementation.
- The DependencyOnArtifact referred by this link must be aggregated by BswImplementation in the role requiredArtifact.
- The DependencyOnArtifact referred by this link must have the category value set to MEMMAP.

⌋*()*

For a list of standardized allocation keywords, further explanation of the memory mapping header files and their configuration parameters see [19].

[TPS_BSWMDT_04048] Scope of declared memory sections ⌈It is further important to note, that a BSW module or an SWC shall declare only those sections which are actually part of its implemented code. ⌋*(RS_BSWMD_00005, RS_BSWMD_00052)*

That means in particular, if an SWC requires some data to be allocated by the RTE, for example shared calibration parameters or buffers for communication via ports, the memory sections of these data have to be declared via an BswImplementation which is generated by the RTE and represents the implementation of the module RTE.

Several different instances of MemorySection (also across module or component boundaries) can refer to the same SwAddrMethod, indicating that these abstract sections share a common means of being handled which is further characterized by SwAddrMethod.sectionType.

The attributes of SwAddrMethod (namely sectionType, memoryAllocationKeywordPolicy ,option and sectionInitializationPolicy) as well as MemorySection.alignment put constraints on the selection of appropriate allocation keywords resp. their configuration values. This is further explained in [19].

[2] Note that in any case the AUTOSAR memory mapping header files are considered as implementation of an own virtual BSW module MemMap, therefore other modules need to refer to these headers via the role requiredArtifact. In contrast, a BswImplementation representing the implementation of module MemMap would refer to these files via the role generatedArtifact.

Note that the shortName of SwAddrMethod also has some relationship to the allocation keyword and thus to the section name defined by MemorySection, which is an intended redundancy.

SwAddrMethod is also referred by the “upstream” specifications of the data or executable entities belonging to these sections, so that the section type can be predefined early in the process.

The attributes of MemorySection and SwAddrMethod are shown below:

Class	**MemorySection**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	Provides a description of an abstract memory section used in the Implementation for code or data. It shall be declared by the Implementation Description of the module or component, which actually allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the generated Implementation Description of the RTE shall contain the corresponding MemorySections. The attribute "symbol" (if symbol is missing: "shortName") defines the module or component specific section name used in the code. For details see the document "Specification of Memory Mapping". Typically the section name is build according the pattern: <SwAddrMethod shortName>[_<further specialization nominator>][_<alignment>] where • **[<SwAddrMethod shortName>]** is the shortName of the referenced SwAddrMethod • **[_<further specialization nominator>]** is an optional infix to indicate the specialization in the case that several MemorySections for different purpose of the same Implementation Description referring to the same or equally named SwAddrMethods. • **[_<alignment>]** is the alignment attributes value and is only applicable in the case that the memoryAllocationKeywordPolicy value of the referenced SwAddrMethod is set to addrMethodShortNameAndAlignment MemorySection used to Implement the code of RunnableEntitys and BswSchedulableEntitys shall have a symbol (if missing: shortName) identical to the referred SwAddrMethod to conform to the generated RTE header files. In addition to the section name described above, a prefix is used in the corresponding macro code in order to define a name space. This prefix is by default given by the shortName of the BswModuleDescription resp. the SwComponentType. It can be superseded by the prefix attribute.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
alignment	AlignmentType	0..1	attr	The attribute describes the alignment of objects within this memory section.
executableEntity	ExecutableEntity	*	ref	Reference to the ExecutableEntitites located in this section. This allows to locate different Executable Entitities in different sections even if the associated SwAddrmethod is the same. This is applicable to code sections only.
memClassSymbol	CIdentifier	0..1	attr	Defines a specific symbol in order to generate the compiler abstraction "memclass" code for this MemorySection. The existence of this attribute supersedes the usage of swAddrmethod.shortName for this purpose. The complete name of the "memclass" preprocessor symbol is constructed as <prefix>_<memClassSymbol> where prefix is defined in the same way as for the enclosing MemorySection. See also AUTOSAR_SWS_CompilerAbstraction SWS_COMPILER_00040.

▽

△

Class	**MemorySection**			
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of this MemorySection. The following two values are standardized (to be used for code sections only and exclusively to each other): • INLINE - The code section is declared with the compiler abstraction macro INLINE. • LOCAL_INLINE - The code section is declared with the compiler abstraction macro LOCAL_INLINE In both cases (INLINE and LOCAL_INLINE) the inline expansion depends on the compiler specific implementation of these macros. Depending on this, the code section either corresponds to an actual section in memory or is put into the section of the caller. See AUTOSAR_SWS_CompilerAbstraction for more details.
prefix	SectionNamePrefix	0..1	ref	The prefix used to set the memory section's namespace in the code. The existence of a prefix element supersedes rules for a default prefix (such as the Bsw ModuleDescription's shortName). This allows the user to define several name spaces for memory sections within the scope of one module, cluster or SWC.
size	PositiveInteger	0..1	attr	The size in bytes of the section.
swAddrmethod	SwAddrMethod	1	ref	This association indicates that this module specific (abstract) memory section is part of an overall SwAddr Method, referred by the upstream declarations (e.g. calibration parameters, data element prototypes, code entities) which share a common addressing strategy. This can be evaluated for the ECU configuration of the build support. This association shall always be declared by the Implementation description of the module or component, which allocates the memory in its code. This means in case of data prototypes which are allocated by the RTE, that the software components only declare the grouping of its data prototypes to SwAddrMethods, and the generated Implementation Description of the RTE actually sets up this association.
symbol	Identifier	0..1	attr	Defines the section name as explained in the main description. By using this attribute for code generation (instead of the shortName) it is possible to define several different MemorySections having the same name - e.g. symbol = CODE - but using different sectionName Prefixes.

Table 9.2: MemorySection

Primitive	**AlignmentType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes

▽

△

Primitive	**AlignmentType**
Note	This primitive represents the alignment of objects within a memory section. The value is in number of bits or UNKNOWN (deprecated), 8 , 16, 32 UNSPECIFIED, BOOLEAN, or PTR. Typical values for numbers are 8, 16, 32. **Tags:** xml.xsd.customType=ALIGNMENT-TYPE xml.xsd.pattern=[1-9][0-9]*|0[xX][0-9a-fA-F]*|0[bB][0-1]+|0[0-7]*|UNSPECIFIED|UNKNOWN|BOOLEAN|PTR xml.xsd.type=string

Table 9.3: AlignmentType

Class	**SwAddrMethod**			
Package	M2::MSR::DataDictionary::AuxillaryObjects			
Note	Used to assign a common addressing method, e.g. common memory section, to data or code objects. These objects could actually live in different modules or components. **Tags:** atp.recommendedPackage=SwAddrMethods			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
memory Allocation KeywordPolicy	MemoryAllocation KeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
option	Identifier	*	attr	This attribute introduces the ability to specify further intended properties of the MemorySection in with the related objects shall be placed. These properties are handled as to be selected. The intended options are mentioned in the list. In the Memory Mapping configuration, this option list is used to determine an appropriate MemMapAddressing ModeSet.
section Initialization Policy	SectionInitialization PolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableData Prototypes). Therefore this is an implementation constraint for initialization code of BSW modules (especially RTE) as well as the start-up code which initializes the memory segment to which the AutosarData Prototypes referring to the SwAddrMethod's are later on mapped. If the attribute is not defined it has the identical semantic as the attribute value "INIT"
sectionType	MemorySectionType	0..1	attr	Defines the type of memory sections which can be associated with this addresssing method.

Table 9.4: SwAddrMethod

Enumeration	**MemoryAllocationKeywordPolicyType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Literal	***Description***

▽

— AUTOSAR CONFIDENTIAL —

△

Enumeration	**MemoryAllocationKeywordPolicyType**
addrMethodShortName	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod. This is the default value if the attribute does not exist. **Tags:** atp.EnumerationValue=0
addrMethodShortNameAndAlignment	The MemorySection shortNames of referring MemorySections and therefore the belonging Memory Allocation Keywords in the code are build with the shortName of the SwAddrMethod and a variable alignment postfix. Thereby the alignment postfix needs to be consistent with the alignment attribute of the related MemorySection. **Tags:** atp.EnumerationValue=1

Table 9.5: MemoryAllocationKeywordPolicyType

Primitive	**SectionInitializationPolicyType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	SectionInitializationPolicyType describes the intended initialization of MemorySections. The following values are standardized in AUTOSAR Methodology: • **NO-INIT**: No initialization and no clearing is performed. Such data elements shall not be read before one has written a value into it. • **INIT**: To be used for data that are initialized by every reset to the specified value (initValue). • **POWER-ON-INIT**: To be used for data that are initialized by "Power On" to the specified value (initValue). Note: there might be several resets between power on resets. • **CLEARED**: To be used for data that are initialized by every reset to zero. • **POWER-ON-CLEARED**: To be used for data that are initialized by "Power On" to zero. Note: there might be several resets between power on resets. Please note that the values are defined similar to the representation of enumeration types in the XML schema to ensure backward compatibility. **Tags:** xml.xsd.customType=SECTION-INITIALIZATION-POLICY-TYPE xml.xsd.type=NMTOKEN

Table 9.6: SectionInitializationPolicyType

Enumeration	**MemorySectionType**
Package	M2::MSR::DataDictionary::AuxillaryObjects
Note	Enumeration to specify the essential nature of the data which can be allocated in a common memory class by the means of the AUTOSAR Memory Mapping.
Literal	***Description***
calibrationVariables	This memory section is reserved for "virtual variables" that are computed by an MCD system during a measurement session but do not exist in the ECU memory. **Tags:** atp.EnumerationValue=2
calprm	To be used for calibratable constants of ECU-functions. **Tags:** atp.EnumerationValue=3
code	To be used for mapping code to application block, boot block, external flash etc. **Tags:** atp.EnumerationValue=4
configData	Constants with attributes that show that they reside in one segment for module configuration. **Tags:** atp.EnumerationValue=5

▽

△

Enumeration	**MemorySectionType**
const	To be used for global or static constants. **Tags:** atp.EnumerationValue=6
excludeFromFlash	This memory section is reserved for "virtual parameters" that are taken for computing the values of so-called dependent parameter of an MCD system. Dependent Parameters that are not at the same time "virtual parameters" are allocated in the ECU memory. Virtual parameters, on the other hand, are not allocated in the ECU memory. Virtual parameters exist in the ECU Hex file for the purpose of being considered (for computing the values of dependent parameters) during an offline-calibration session. **Tags:** atp.EnumerationValue=7
var	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy. **Tags:** atp.EnumerationValue=9

Table 9.7: MemorySectionType

Class	**SectionNamePrefix**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::MemorySectionUsage			
Note	A prefix to be used for generated code artifacts defining a memory section name in the source code of the using module or SWC.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
implementedIn	DependencyOnArtifact	0..1	ref	Optional reference that allows to Indicate the code artifact (header file) containing the preprocessor implementation of memory sections with this prefix. The usage of this link supersedes the usage of a memory mapping header with the default name (derived from the BswModuleDescription's shortName).

Table 9.8: SectionNamePrefix

[constr_4028] Semantics of memory section type ⌈ `sectionType` must be semantically compatible to the usage of the enclosing `SwAddrMethod`, this means especially that if `SwAddrMethod` is associated by `ExecutableEntity`-s, the `sectionType` must be usable as code section, if it is associated by `SwDataDefProps`, `sectionType` must be usable as data section. ⌋*()*

In case `sectionType` has the value `userDefined`, additional documentation is needed to support the integrator in selecting the proper memory segment from the ECU.

Note: The section type `userDefined` is deprecated. Instead of this, user defined selection criteria shall be given by the attribute `SwAddrMethod.option`. This allows a more formal support for selecting the memory segment during integration. (see [19]).

Several values that can be used both for `SwAddrMethod.option` and `MemorySection.option` are predefined by AUTOSAR, see [TPS_SWCT_01456] in [6]. In addition to this, the following two values are standardized:

[TPS_BSWMDT_04080] Options for inline code sections ⌈For code sections the following two values of MemorySection.option are standardized (to be used exclusively to each other):

- INLINE - The code section is declared with the compiler abstraction macro INLINE.
- LOCAL_INLINE - The code section is declared with the compiler abstraction macro LOCAL_INLINE

In both cases the inline expansion depends on the compiler specific implementation of these macros. Depending on this, the code section either corresponds to an actual section in memory or is put into the section of the caller. See [20] for more details. ⌋ *(RS_BSWMD_00005, RS_BSWMD_00031)*

[constr_4054] Unambiguous links to addressing method ⌈ MemorySection.executableEntity must not be defined, if MemorySection.swAddrMethod represents a data section. MemorySection.executableEntity must not refer to an ExecutableEntity which is linked to a different SwAddrMethod than MemorySection.swAddrMethod. ⌋*()*

[TPS_BSWMDT_04049] Usage of MemorySection.executableEntity ⌈ It is in general not mandatory to define the relation MemorySection.executableEntity for code sections because this relationship might be sufficiently determined via the SwAddrMethod referred by both MemorySection and ExecutableEntity. However, if explicit name spaces are defined using the MemorySection.prefix attribute and if MemorySection.sectionType defines a code section, it is mandatory to assign all ExecutableEntity-s running in this section explicitly via MemorySection.executableEntity. Note that this is not a constraint that can be checked on ARXML level. ⌋*(RS_BSWMD_00005, RS_BSWMD_00014, RS_BSWMD_00031)*

The meta-classes described in this chapter are also used to predefine the so-called compiler abstraction memory class per memory section, so that the macro memclass can be generated as part of the AUTOSAR compiler abstraction header Compiler_Cfg.h:

[TPS_BSWMDT_04093] Memory classes for compiler abstraction ⌈As a default rule, the memclass symbols for basic software are constructed with a prefix defined in the same way as for the associated memory section plus the SwAddrMethod.shortName referred by the individual MemorySections. However, it is possible to supersede the rule for the 2nd part of the name (after the prefix) and define an individual memclass symbol by the value MemorySection.memClassSymbol. This is e.g. useful if many small callout code sections share a common SwAddrMethod.

For application software, the memclass symbols are always constructed from the AtomicSwComponentType.shortName plus the SwAddrMethod.shortName referred by the individual MemorySections.

For the detailed rule refer to [20], [SWS_COMPILER_00040]. ⌋*()*

9.4 Dynamic Memory Needs

9.4.1 General

The dynamic memory is mainly divided into two categories, the stack and the heap. While the stack is almost always used in embedded software, the heap is avoided as much as possible due to the complexity of its implementation, and fragmentation issues. The dynamic memory consumption of software has a much different quality than the static memory consumption. The amount of the static memory consumption can be retrieved from the compiler and is only dependent on the compiler and processor used as well as on the number of instances.

Dynamic memory consumption is heavily dependent on the actual code being executed which is dependent on the state of the software and the parameters. With the introduction of recursive concepts the uncertainty is even higher. Therefore the approach for dynamic memory consumption is far more related to the description of the execution time introduced in chapter 9.5.

9.4.2 Stack

The stack is an area in memory that is used to store temporary information like parameters and local variables of function calls. Therefore the stack usage is highly dependent on the calling hierarchy and the nesting level of function calls. The stack is organized in a LIFO (last in first out) manner. So each time a function is called the necessary stack memory is occupied. After leaving the function also the associated memory area is freed again and can be used for the next function call. Among tasks, that do not interrupt each other, fragmentation is not a problem for a stack. Only the available amount of stack memory is relevant from the software point of view. However, there can be several stacks in a concurrent task environment. Note that it is not in the scope of a module or component to define the number of stacks, only the amount of used stack memory can be given.

Different mechanisms can be used to describe the stack memory needs of software. Needed stack size can either be *calculated*, *measured* or *estimated*. This is shown in Figure 9.3.

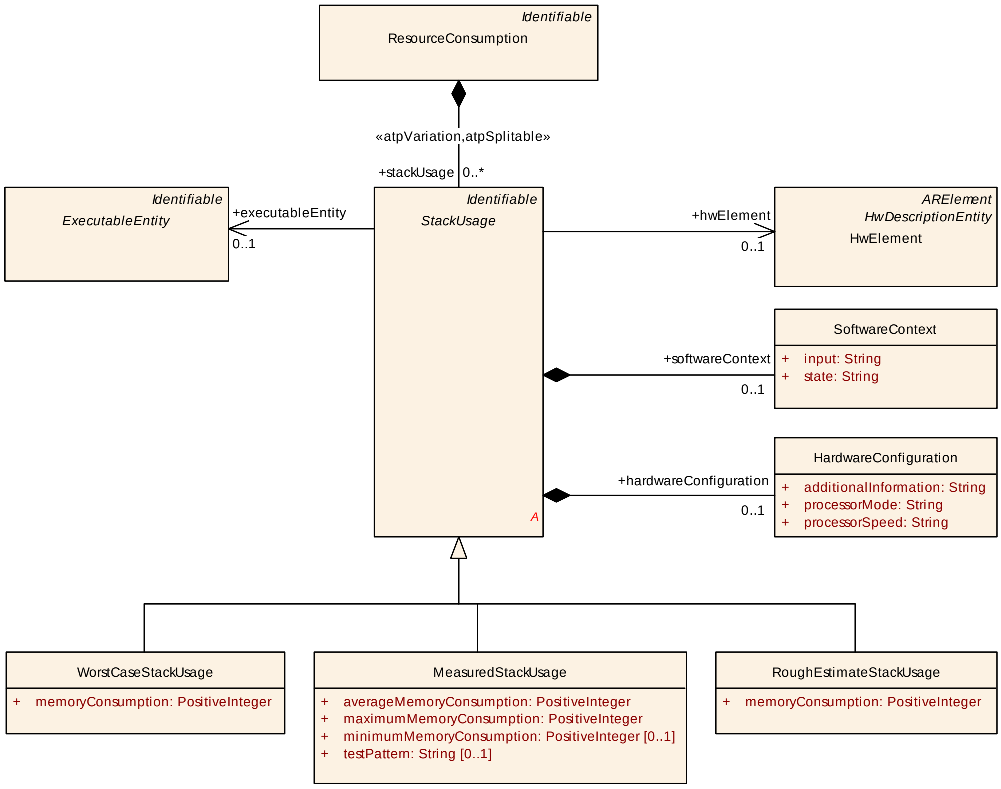

Figure 9.3: Stack Memory Consumption

The given stack memory consumption is dependent on the ECU, the software context and maybe also on the hardware configuration. The software context and the hardware configuration describe the state of the software and hardware under which the given stack usage was gathered. So for each given stack memory consumption these environmental descriptions have to be provided.

Class	***StackUsage*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::StackUsage			
Note	Describes the stack memory usage of a software.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	MeasuredStackUsage, RoughEstimateStackUsage, WorstCaseStackUsage			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
executableEntity	ExecutableEntity	0..1	ref	The executable entity for which this stack usage is described.
hardware Configuration	HardwareConfiguration	0..1	aggr	Contains information about the hardware context this stack usage is describing.
hwElement	HwElement	0..1	ref	Specifies for which hardware element (e.g. ECU) this stack usage is given.
softwareContext	SoftwareContext	0..1	aggr	Contains details about the software context this stack usage is provided for.

Table 9.9: StackUsage

Class	WorstCaseStackUsage			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::StackUsage			
Note	Provides a formal worst case stack usage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StackUsage*			
Attribute	Type	Mul.	Kind	Note
memory Consumption	PositiveInteger	1	attr	Worst case stack consumption. Unit: byte.

Table 9.10: WorstCaseStackUsage

Class	MeasuredStackUsage			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::StackUsage			
Note	The stack usage has been measured.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StackUsage*			
Attribute	Type	Mul.	Kind	Note
averageMemory Consumption	PositiveInteger	1	attr	The average stack usage measured. Unit: byte.
maximum Memory Consumption	PositiveInteger	1	attr	The maximum stack usage measured. Unit: byte.
minimum Memory Consumption	PositiveInteger	0..1	attr	The minimum stack usage measured. Unit: byte.
testPattern	String	0..1	attr	Description of the test pattern used to acquire the measured values.

Table 9.11: MeasuredStackUsage

[constr_4029] Measured stack usage ⌈ The attribute values of `MeasuredStackUsage` must fulfill:
`minimumMemoryConsumption` <= `averageMemoryConsumption` <= `maximumMemoryConsumption` ⌋*()*

Class	RoughEstimateStackUsage			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::StackUsage			
Note	Rough estimation of the stack usage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StackUsage*			
Attribute	Type	Mul.	Kind	Note
memory Consumption	PositiveInteger	1	attr	Rough estimate of the stack usage. Unit: byte.

Table 9.12: RoughEstimateStackUsage

9.4.3 Heap

Heap is the memory segment that is used to cover dynamic memory needs with explicit memory allocation and de-allocation. Since the allocation of the memory is controlled by the application program it also survives changes in the context of invocation from entering a function nesting level and leaving it again. So a memory block allocated in the subroutine can be used in the calling routine after the subroutine has returned. Also the allocated memory can be freed again in a different context.

Because of the independence of the heap consumption from processes and tasks only the whole software component or BSW Module heap consumption is provided in the description. The meta-model is shown in Figure 9.4.

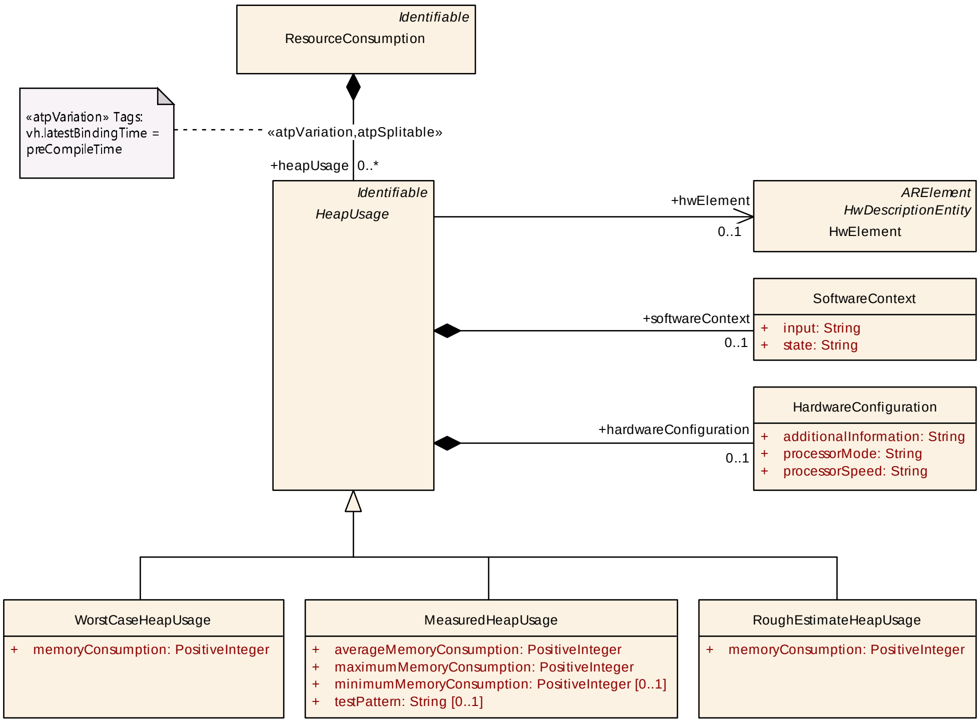

Figure 9.4: Heap Memory Consumption

The heap memory consumption also depends on the ECU, the software context and the hardware configuration.

Due to the highly dynamic nature of heap memory one problem is the fragmentation of the available memory area. So in some cases there can be not enough memory allocated, even though the total amount of free heap memory is big enough, because the available memory space is not available contiguously.

Class	***HeapUsage*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::HeapUsage			
Note	Describes the heap memory usage of a SW-Component.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	MeasuredHeapUsage, RoughEstimateHeapUsage, WorstCaseHeapUsage			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
hardware Configuration	HardwareConfiguration	0..1	aggr	Contains information about the hardware context this heap usage is describing.
hwElement	HwElement	0..1	ref	Specifies for which hardware element (e.g. ECU) this heap usage usage is given.

▽

△

Class	***HeapUsage*** (abstract)			
softwareContext	SoftwareContext	0..1	aggr	Contains details about the software context this heap usage is provided for.

Table 9.13: HeapUsage

Class	**WorstCaseHeapUsage**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::HeapUsage			
Note	Provides a formal worst case heap usage.			
Base	*ARObject*, *HeapUsage*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
memory Consumption	PositiveInteger	1	attr	Worst case heap consumption. Unit: byte.

Table 9.14: WorstCaseHeapUsage

Class	**MeasuredHeapUsage**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::HeapUsage			
Note	The heap usage has been measured.			
Base	*ARObject*, *HeapUsage*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
averageMemory Consumption	PositiveInteger	1	attr	The average heap usage measured. Unit: byte.
maximum Memory Consumption	PositiveInteger	1	attr	The maximum heap usage measured. Unit: byte.
minimum Memory Consumption	PositiveInteger	0..1	attr	The minimum heap usage measured. Unit: byte.
testPattern	String	0..1	attr	Description of the test pattern used to acquire the measured values.

Table 9.15: MeasuredHeapUsage

[constr_4030] Measured heap usage ⌈ The attribute values of `MeasuredHeapUsage` must fulfill:
`minimumMemoryConsumption` <= `averageMemoryConsumption` <= `maximumMemoryConsumption` ⌋*()*

Class	**RoughEstimateHeapUsage**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::HeapUsage			
Note	Rough estimation of the heap usage.			
Base	*ARObject*, *HeapUsage*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
memory Consumption	PositiveInteger	1	attr	Rough estimate of the heap usage. Unit: byte.

Table 9.16: RoughEstimateHeapUsage

— AUTOSAR CONFIDENTIAL —

9.5 Execution Time

9.5.1 General

This subsection defines a model to describe the ExecutionTime of a specific ExecutableEntity of a specific Implementation.

Chapter 9.5.3 describes the goals and scope of the ExecutionTime description proposed.

Chapter 9.5.4 lists all the thoughts and observations that lead to the actual model which is described in chapter 9.5.5.

9.5.2 Preliminaries

This subsection assumes that the reader is familiar with the definition of the following terminology (please see the AUTOSAR Glossary [5] for details):

- task
- thread
- process
- executable entity
- (worst case) execution time
- (worst case) response time

9.5.3 Scope

9.5.3.1 Assertions Versus Requirements

The ExecutionTime is an ASSERTION: a statement about the duration of the execution of a piece of code in a given situation. The execution time is NOT a REQUIREMENT on the software, on the hardware or on the scheduling policy.

9.5.3.2 In Scope

This section proposes a description of the ExecutionTime of an ExecutableEntity of an Implementation. Very roughly, this description includes:

- the nominal execution time ("0.000137 s") or a range of times
- a description of the entire context in which the execution time measurement or analysis has been made

- some indication of the quality of this measurement or estimation

The goal is to find a good compromise between flexibility and precision. The description must be flexible enough so that the entire range between analytic results ("worst-case execution time") and rough estimates can be described. The description should be precise enough so that it is entirely clear what the relevance or meaning of the stated execution time is. This implies that a large amount of context information needs to be provided. The following sections analyze what this context is and provide an appropriate structure for this information.

9.5.3.3 Out of Scope

It is however not in the scope of this section to specify how the execution time of a runnable entity can be or should be measured or analyzed. We will not discuss what tools or techniques can be used to find the execution time or worst-case execution time of a piece of software.

It also is not in the scope of this section to define how information about execution times is used when integrating various software onto one ECU. Similarly this section does not deal with the response time of the system to certain events. The response time does not only depend on the execution times of the involved software but also on the infrastructure overhead and on the scheduling policies which are used.

The focus also is on the description of the execution time of assembly instructions (typically generated out of compiled C or C++ code). The execution time of e.g. Java byte-code on a virtual machine has not been explicitly considered.

9.5.4 Background

This section provides some background to the proposed solution. Readers who want to skip to the result should go to chapter 9.5.5. The execution time can be described for a specific sequence of assembly instructions. It does not make sense to describe the execution time of a runnable provided as source-code unless a precise compiler (and compiler options) are also provided so that a unique set of assembly instructions can be generated out of the source-code. In addition, the execution time of such a sequence of assembly instructions depends on:

1. the hardware-platform
2. the hardware state
3. the logical (software) context
4. execution time of external pieces of code called from the software

These dependencies are discussed in detail in the following sections.

9.5.4.1 Dependency of the Execution Time on Hardware

The execution time depends both on the CPU-hardware and on certain parts of the peripheral hardware:

- The execution time depends on a complete description of the processor, including:
  - kind of processor (e.g. "PPC603")
  - the internal Processor frequency ("100 MHz")
  - amount of processor cache
  - configuration of CPU (e.g. power-mode)
- Aspects of the periphery that need to be described include:
  - external bus-speed
  - MMU (memory management unit)
  - configuration of the MMU (data-cache, code-cache, write-back,...)
  - external cache
  - memory (kind of RAM, RAM speed)

In addition, when other devices (I/O) are eventually accessed *as memory* by the I/O Hardware Abstraction, the speed of those devices potentially has a large influence on the execution time of software.

On top of this, the ECU might provide several ways to store the code and data that needs to be executed. This might also have a large influence on the execution time. For example:

- execution of assembly instructions stored in RAM versus execution out of ROM might have very different execution times
- when caching is present, the relative physical location of data accessed in memory might also influence the execution time

9.5.4.2 Dependency on Hardware State

In addition to the static configuration of the hardware and location of the code and data on this hardware, the dynamically changing state of the hardware might have a large influence on the execution time of a piece of code : some examples of this hardware state are:

- which parts of the code are available in the execution cache and what parts will need to be read from external RAM
- what part of the data is stored in data cache versus must be fetched from RAM

- potentially, the state of the processor pipeline

Although this influence is not relevant on simple or deterministic processors (without cache), the influence of the cache state on modern processors can be enormous (an order of magnitude difference is not impossible). Despite the potential importance of this initial hardware-state when caching is present, it is almost impossible and definitely impractical to describe this hardware state. Therefore it is important and clear that we will not provide explicit attributes for this purpose.

9.5.4.3 Dependency on Logical Context

This logical context includes:

1. the input parameters with which the runnable is called
2. also the logical "state" of the component to which the runnable belongs (or more precisely: the contents of all the memory that is used by the runnable)

While a description of the input-parameters is relatively straight-forward to specify, it might be very hard to describe the entire logical state that the software depends on.

In addition, in certain cases, one wants to provide a specific (e.g. measured or simulated) execution time for a very specific logical context; whereas in other cases, one wants to describe a *worst-case execution time* over all valid logical contexts or over a subset of logical contexts.

9.5.4.4 Dependency on External Code

Things get very complex when the piece of code whose execution time is described makes calls into ("jumps into") external libraries. To deal with this problem, we could take one of the following approaches:

1. Do not support this case at all: only code that does not rely on external libraries can be given an execution time
2. Support a description of the execution time for a very specific version (again at object-code level) of the libraries. The exact versions of external libraries used would be described together with the execution time. In addition, the relative location in memory of the runnable and the library, the HW-state with respect to the library (e.g. whether this code is in cache or not) and the logical state of the library might have an influence.
3. Conceptually, it might be possible to support a description of the software which explicitly describes the dependency on the execution times of the library. This description would include:
   (a) the execution time of the code provided by the software itself

(b) a specification of which external library-calls are made (with what parameters, how often, in what order, ...)

Option 3 is deemed unrealistic and impractical and is not supported. Option 2 however is important as many software might depend on very simple but very common external libraries (like a math-library that provides floating-point capability in software). Option 2 will therefore be supported for the case that the external library does not have an additional logical context which influences its execution time.

9.5.5 Description-Model for the Execution Time

9.5.5.1 Detailed Structure of an Execution-Time Description

Figure 9.5 shows how the `ExecutionTime` is part of the overall description of the `Implementation` and how it relates to various other model elements.

[TPS_BSWMDT_04050] `ExecutionTime` ⌈To each `ExecutableEntity` (of a specific `Implementation`) an arbitrary number of `ExecutionTime` descriptions can be related. Thereby this `ExecutionTime` description may also depend on code or data variant of the `Implementation`. ⌋*(RS_BSWMD_00016)*

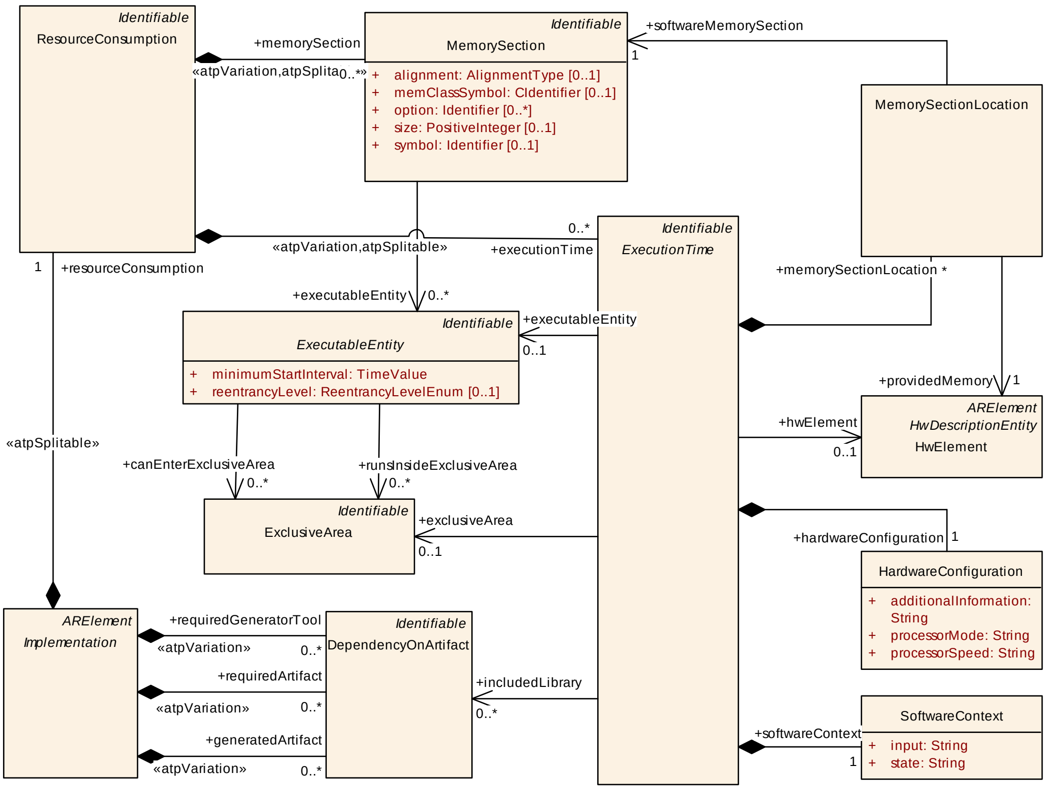

Figure 9.5: Detailed relations of an `ExecutionTime` description

It is expected that many `ExecutableEntity`-s will not have an associated `ExecutionTime` description. For `ExecutableEntity`-s that do have `ExecutionTime` descriptions, the software-implementor can provide several such descriptions with different scope: For example one per specific ECU on which the Implementation can run and on which the time was measured or estimated. Furthermore, even in a given ECU context it is possible to specify several different types of execution times, as will be explained below.

If an ExecutableEntity is defined to be running completely in an ExclusiveArea the related ExecutionTime can be considered as a constraint for configuring the data consistency mechanism in the RTE.

If an ExecutableEntity is defined to be able to enter an ExclusiveArea the ExecutionTime can be specified for each area. The time provided is the time consumed AFTER the call to enter the ExclusiveArea and BEFORE the call to leave the ExclusiveArea.

Figure 9.6 shows the various sub-classes of ExecutionTime. The following paragraphs describe the aspects of this model in more detail. For the definition of class TimeValue refer to the timing specification ([12]).

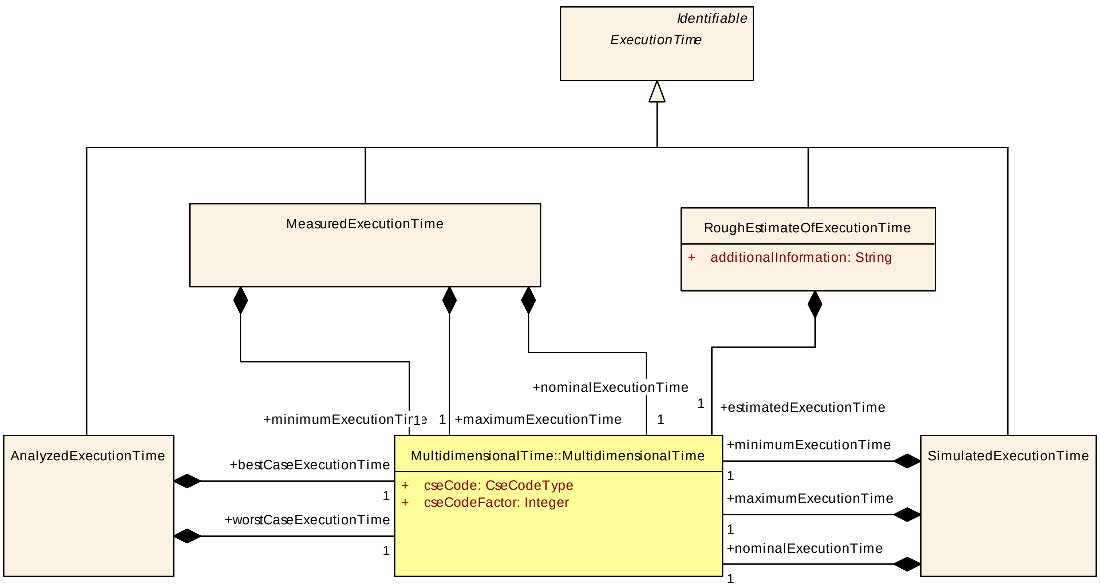

Figure 9.6: Sub-classes of ExecutionTime and their usage of TimeValue

The following shows the attributes of the ExecutionTime in tabular form:

Class	***ExecutionTime*** (abstract)
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime
Note	Base class for several means how to describe the ExecutionTime of software. The required context information is provided through this class.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*
Subclasses	AnalyzedExecutionTime, MeasuredExecutionTime, RoughEstimateOfExecutionTime, Simulated ExecutionTime

▽

△

Class	***ExecutionTime*** (abstract)			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
exclusiveArea	ExclusiveArea	0..1	ref	Reference to the ExclusiveArea this execution time is provided for.
executableEntity	ExecutableEntity	0..1	ref	The executable entity for which this execution time is described.
hardware Configuration	HardwareConfiguration	1	aggr	Provides information on the HardwareConfiguration used to specify this ExecutionTime.
hwElement	HwElement	0..1	ref	The hardware element (e.g. type of ECU) for which the execution time is specified.
includedLibrary	DependencyOnArtifact	*	ref	If this dependency is specified, the execution time of the library code is included in the execution time data for the runnable.
memorySection Location	MemorySectionLocation	*	aggr	Provides information on the MemorySectionLocation which is involved in the ExecutionTime description.
softwareContext	SoftwareContext	1	aggr	Provides information on the detailed SoftwareContext used to provide the ExecutionTime description.

Table 9.17: ExecutionTime

9.5.5.2 ExecutionTime References an "ECU"

[TPS_BSWMDT_04051] `ExecutionTime` references an ECU ⌈The `ExecutionTime` references an ECU (the concept ECU is defined by the ECU-Resource-Template [21]) via the attribute `hwElement`. This reference uniquely describes the hardware for which the `ExecutionTime` is provided. ⌋*(RS_BSWMD_00016)* This includes: the kind of processor, the type of MMU, the type of caches, type of memory available,...

Note that this reference to an `HwElement` has a different semantic than the attribute `processor` in the `Implementation`. The `processor` defines the family of processors on which the provided implementation may run (it is a requirement on the hardware on which the component may be deployed). The ECU on the other hand (of which the processor only is one part) is a statement on the context of the `ExecutionTime`. Of course, the processor of the ECU should be equal to the processor specified in the `Implementation`. Note that the ECU might include specific hardware that has no influence on the `ExecutionTime`. Despite of this, it seems better to specify a reference to the entire hardware-platform used rather than introduce another hardware sub-system that includes all hardware-elements that influence the `ExecutionTime` of software.

9.5.5.3 ExecutionTime Includes a HW-Configuration

[TPS_BSWMDT_04052] `ExecutionTime.hardwareConfiguration` ⌈The ECU described through the `hwElement` attribute can still run in several HW-modes. For example, many ECUs can run in several "speed"-modes (for example a normal fast-mode and a low-power slow mode). The goal of the HardwareConfiguration is to describe this. The attributes processorSpeed and processorMode should describe the specific mode of the ECU.

Because of the potential dependency on many other HW-Configuration settings (such as caching policy, MMU-settings, ...), a generic attribute additionalInformation is provided. Because the exact structure of the information seems to depend so much on the specific case, all attributes are unstructured text. ⌋*(RS_BSWMD_00016)*

Class	**HardwareConfiguration**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption			
Note	Describes in which mode the hardware is operating while needing this resource consumption.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
additional Information	String	1	attr	Specifies additional information on the Hardware Configuration.
processorMode	String	1	attr	Specifies in which mode the processor is operating.
processor Speed	String	1	attr	Specifies the speed the processor is operating.

Table 9.18: HardwareConfiguration

9.5.5.4 ExecutionTime Includes a MemorySectionLocation

[TPS_BSWMDT_04053] **ExecutionTime.memorySectionLocation** ⌈For each memorySection of the Implementation, the ExecutionTime must specify where this section was located on the physical memory of the ECU. The memorySections of the software are described in the resourceConsumption of the Implementation. The available memory-regions on the hardware are described inside the description of the ECU. The ExecutionTime contains descriptions of the location of the memory sections MemorySectionLocation which link a software memory section to a hardware memory section on the ECU. ⌋*(RS_BSWMD_00016)*

Class	**MemorySectionLocation**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime			
Note	Specifies in which hardware ProvidedMemorySegment the softwareMemorySection is located.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
provided Memory	HwElement	1	ref	Reference to the hardware ProvidedMemorySegment.
software MemorySection	MemorySection	1	ref	Reference to the MemorySection which is mapped on a certain hardware memory segment.

Table 9.19: MemorySectionLocation

9.5.5.5 ExecutionTime Includes a SoftwareContext

[TPS_BSWMDT_04054] **ExecutionTime.softwareContext** ⌈The Software-Context is the logical context for which the ExecutionTime is given. This includes two aspects:

1. the values of the input-parameters to the software
2. the state the logic of the runnable depends on

In the current form, both attributes are of type String and can contain free-form text describing this state. ⌋*(RS_BSWMD_00016)*

For the attribute input, it might be appropriate to refine this into a more formal description of the values of the parameters. For the attribute state, it is difficult to go beyond an informal text-field, because the state is a private matter of the component and there currently is no explicit mechanism in AUTOSAR to describe the value of this state.

Further, it is possible to provide several execution times of a runnable entity, for example, in case of different values of the input-parameters. This is one of the reasons why the template supports an arbitrary number of ExecutionTimes.

Class	**SoftwareContext**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption			
Note	Specifies the context of the software for this resource consumption.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
input	String	1	attr	Specifies the input vector which is used to provide the ExecutionTime.
state	String	1	attr	Specifies the state the software is in when the Execution Time is provided.

Table 9.20: SoftwareContext

9.5.5.6 Dependency on External Libraries

[TPS_BSWMDT_04055] **ExecutionTime.includedLibrary** ⌈The Execution-Time measurements can depend on the precise version of external libraries (such as a math-emulation library) that have been used. This information can be included by adding a reference to an object of type DependencyOnArtifact which must be aggregated by the corresponding Implementation.

If such a reference is specified, the ExecutionTime includes the execution time of that specific library version.

In case the Implementation aggregates attributes of type DependencyOnArtifact, to which the ExecutionTime does not refer, it means that the execution time

of the library code is NOT included in the execution time of the ExecutableEntity.
⌋*(RS_BSWMD_00016)*

9.5.5.7 Several Qualities of Execution Times

9.5.5.7.1 AnalyzedExecutionTime

The AnalyzedExecutionTime means that an "analytic" method was used to find guaranteed boundaries. These boundaries have a lower-limit (best case) and an upper-limit (worst case).

Considering the cache processor ECU, an execution time could be computed, and it depends on cache level. A bestCaseExecutionTime and a bestCaseExecutionTime have to be filled.

Class	**AnalyzedExecutionTime**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime			
Note	AnalyzedExecutionTime provides an analytic method for specifying the best and worst case execution time.			
Base	*ARObject, ExecutionTime, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bestCase ExecutionTime	MultidimensionalTime	1	aggr	The best case execution time (BCET) defines the minimum amount of time the related executable entity requires for its execution.
worstCase ExecutionTime	MultidimensionalTime	1	aggr	The worst case execution time (WCET) defines the maximum amount of time the related executable entity requires for its execution.

Table 9.21: AnalyzedExecutionTime

[constr_4031] Analyzed execution time ⌈ The attribute values of AnalyzedExecutionTime must fulfill:
bestCaseExecutionTime <= bestCaseExecutionTime ⌋*()*

9.5.5.7.2 MeasuredExecutionTime

The MeasuredExecutionTime describes the ExecutableEntity runtime on an ECU.

Class	**MeasuredExecutionTime**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime			
Note	Specifies the ExecutionTime which has been gathered using measurement means.			
Base	*ARObject, ExecutionTime, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***

▽

△

Class	**MeasuredExecutionTime**			
maximum ExecutionTime	MultidimensionalTime	1	aggr	The maximum measured execution time.
minimum ExecutionTime	MultidimensionalTime	1	aggr	The minimum measured execution time.
nominal ExecutionTime	MultidimensionalTime	1	aggr	The nominal measured execution time.

Table 9.22: MeasuredExecutionTime

[constr_4032] Measured execution time ⌈ The attribute values of MeasuredExecutionTime must fulfill:
minimumExecutionTime <= nominalExecutionTime <= maximumExecutionTime ⌋*()*

9.5.5.7.3 SimulatedExecutionTime

A SimulatedExecutionTime describes the time information which are coming from a simulation. Simulation could be based on:

- ExecutableEntity model on specific hardware with time weighting to simulate processor time behavior
- ExecutableEntity model before generation code

Class	**SimulatedExecutionTime**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime			
Note	Specifies the ExecutionTime which has been gathered using simulation means.			
Base	*ARObject*, *ExecutionTime*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
maximum ExecutionTime	MultidimensionalTime	1	aggr	The maximum simulated execution time.
minimum ExecutionTime	MultidimensionalTime	1	aggr	The minimum simulated execution time.
nominal ExecutionTime	MultidimensionalTime	1	aggr	The nominal simulated execution time.

Table 9.23: SimulatedExecutionTime

[constr_4033] Simulated execution time ⌈ The attribute values of SimulatedExecutionTime must fulfill:
minimumExecutionTime <= nominalExecutionTime <= maximumExecutionTime ⌋*()*

9.5.5.7.4 RoughEstimateOfExecutionTime

A RoughEstimateOfExecutionTime describes the time information which are based on some estimation.

Class	**RoughEstimateOfExecutionTime**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::ExecutionTime			
Note	Provides a description of a rough estimate on the ExecutionTime.			
Base	*ARObject*, *ExecutionTime*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
additional Information	String	1	attr	Provides description on the rough estimate of the ExecutionTime.
estimated ExecutionTime	MultidimensionalTime	1	aggr	The estimated execution time.

Table 9.24: RoughEstimateOfExecutionTime

10 Measurement and Calibration Support

10.1 Overview on McSupportData

AUTOSAR allows to declare data for measurement and calibration (MC-data) in the description of software components as a well as for basic software. Software components can declare MC-data which are handled locally, as well as MC-data for which the location and access (during normal execution) is implemented by the RTE, for example data elements in ports, data shared between instances or data requiring software emulation support. BSW modules usually have only local data, but for software emulation support they also may declare calibration data that are handled by the RTE (see also chapter 6.10 for the various data roles).

For the final configuration of the measurement and calibration tools another representation is needed (so-called “A2L”-file) which is not part of AUTOSAR (see [22]).

For a given RTE generator and ECU configuration, the data description part of the A2L-file could in principle be generated out of the “upstream” AUTOSAR descriptions of all involved components and modules (with additional address information from the linker). However, instead of this it has been decided for the AUTOSAR methodology to provide an additional intermediate ARXML work product, the so-called MC Support Data which is produced rather late in the ECU configuration process, out of which (with additional address information from the linker) the final A2L-file can be generated. The reasons for this approach are:

- For the MC data coded by the RTE generator, the actual C-symbols - which are needed to find the memory addresses - depend on the RTE implementation and are not available in the “upstream” descriptions.
- The names used for the data in the BSWM- and SWC-descriptions are not necessarily unique, due to the distributed development in AUTOSAR. In order to define unique names for display in the MC system (and also for other use cases) a so-called ECU Flat Map is provided (see [4] [TR_METH_03008] and [TR_METH_02003] for the method and [7] for the meta-model). These names shall be made available to the MC tools through the MC-support-data.
- The definition of data attributes - namely `SwDataDefProps` - is subject to additions or redefinitions in several artifacts which could be produced in different process steps (for more on this see [6]). In many cases this finally has to be evaluated by the RTE generator, therefore it is convenient, that the RTE generator also puts these final decisions on the `SwDataDefProps` into a generated set of MC support data.
- Information on the so-called calibration method has to be provided which is currently only available in the ECU configuration of the RTE.
- By making use of a dedicated support format, an external tool is less dependent on the overall AUTOSAR meta-model.

- By making use of a dedicated support format, it is possible to restrict the information given to the operator of the final A2L generation to what is actually required in this step.

It has further been decided, that the MC support format (i.e. its part of the meta-model) reuses already existing concepts of the meta-model like categories and SwDataDefProps, because these concepts are close to the “upstream” descriptions and to “A2L” concepts as well.

The resulting model is shown in an overview in figure 10.1, which illustrates also the placement in the context of an ECU configuration. As the figure shows, the root element of the MC support McSupportData is aggregated as splitable in an Implementation. This means, that one such element describes the calibration support for all data located in this implementation which could be a BSW module/cluster/library or an SWC as well. The splitable-stereotype allows, that the data can be defined as a separate artifact and at another point in time, than the Implementation itself. Especially, the support data for all calibration data located in the RTE shall be generated as part of the RTE's own BswImplementation.

In addition to the support for external MCD-tools, the MC-support-data produced by the RTE generator also can contain information which is needed to support the software emulation of calibration data inside the ECU. This is explained in more detail in chapter 10.3.

Furthermore, the MC-support-data produced by the RTE generator or a proprietary tool can contain information which is needed to support rapid prototyping. This is explained in chapter 10.6.

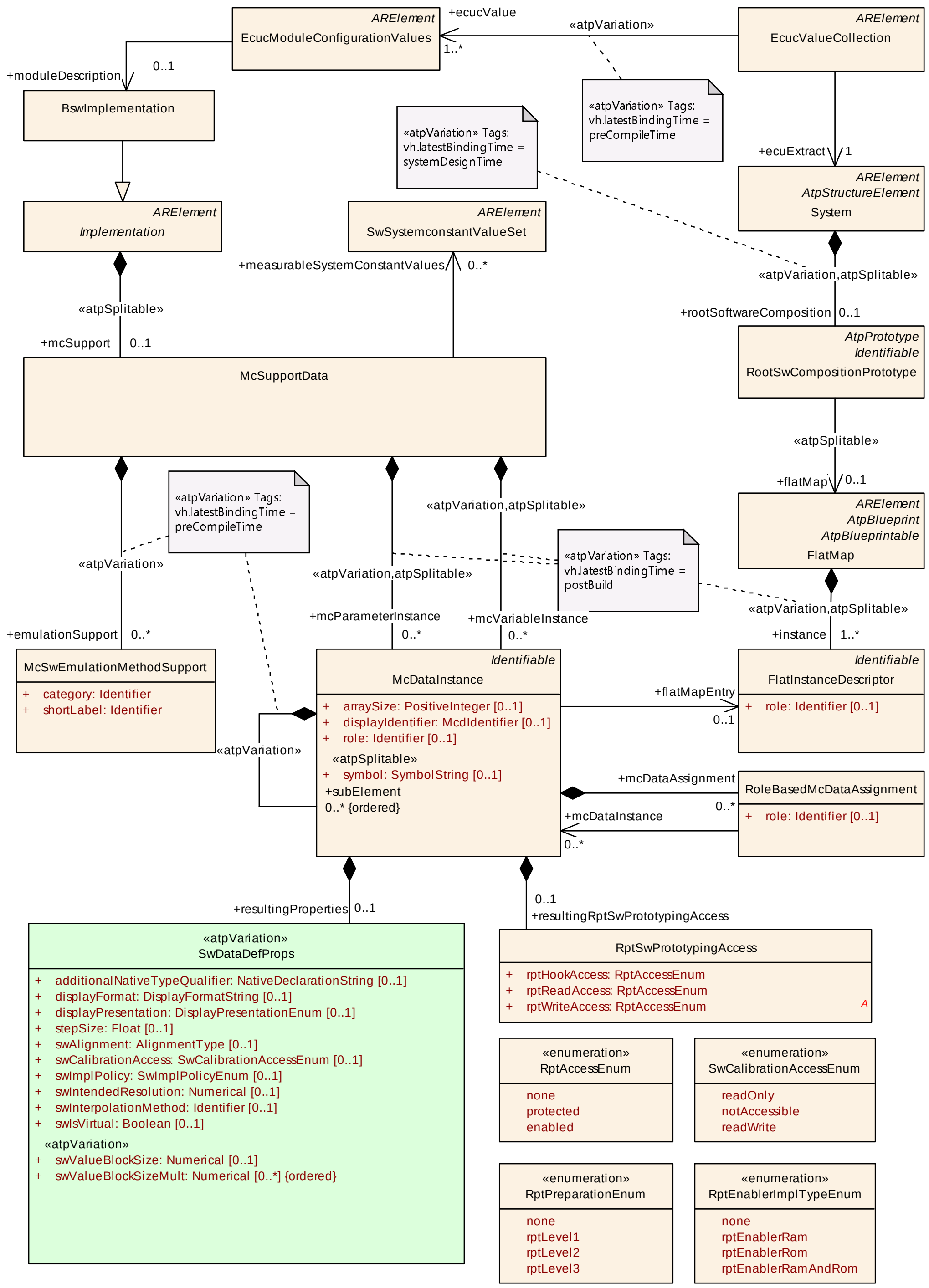

Figure 10.1: Calibration Support Data attached to Implementation

In general, MC support data must be generated for all data with measurement or calibration access in modules or components. For the methodology, we have to distinguish two cases:

- MC support data is generated by the RTE generator for those data, which are allocated also by the RTE (resp. the BSW Scheduler). For BSW modules, this means that those data need to be declared as `BswInternalBehavior.perInstanceMemory`. This is mandatory if calibration data need emulation support - note that for measurement data within basic software there is no use case requiring BSW data allocation by the RTE resp. the BSW Scheduler.

- MC support data are generated by any other tool if the data are allocated by the module or component itself, i.e. for InternalBehavior.staticMemory and InternalBehavior.constantMemory

[TPS_BSWMDT_04056] Multiplicity of McSupportData ⌈Thus in an ECU there will be at most one (generated) instance of McSupportData for each Implementation instance: ⌋*(RS_BSWMD_00062)*

Class	**McSupportData**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Root element for all measurement and calibration support data related to one Implementation artifact on an ECU. There shall be one such element related to the RTE implementation (if it owns MC data) and a separate one for each module or component, which owns private MC data.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
emulationSupport	McSwEmulationMethodSupport	*	aggr	Describes the calibration method used by the RTE. This information is not needed for A2L generation, but to setup software emulation in the ECU. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
mcParameterInstance	McDataInstance	*	aggr	A data instance to be used for calibration. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
mcVariableInstance	McDataInstance	*	aggr	A data instance to be used for measurement. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
measurableSystemConstantValues	SwSystemconstantValueSet	*	ref	Sets of system constant values to be transferred to the MCD system, because the system constants have been specified with "swCalibrationAccess" = readonly.
rptSupportData	RptSupportData	0..1	aggr	The rapid prototyping support data belonging to this implementation. The aggregtion is «atpSplitable» because in case of an already exisiting BSW Implementation model, this description will be added later in the process, namely at code generation time. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=rptSupportData

Table 10.1: McSupportData

[TPS_BSWMDT_04057] Self-contained MC support artifact ⌈It is important to understand, that the M1 model of an McSupportData element shall be a self-contained tree of XML elements witch can be given to an external tool without needing all the "upstream" descriptions. This rule cannot be expressed by the meta-model, it is part of the methodology. This means that all XML elements which are taken over from SWC and BSWM descriptions without change (e.g. data types) still have to be copied into an own artifact. Especially, the links to input variables of axis definitions must be modified as to point to the corresponding elements within the McSupportData.

There are several exceptions from this rule:

- The association to FlatMap shall be handled in a way that it points to the actual ECU Flat Map, in order to provide a backward link to the actual sources of the data for documentation purposes.
- In order to support software emulation of calibration data, a special reference to the description of the actual data in memory is needed (see 10.3). However, this is not relevant for A2L generation.
- As indicated in figure 10.1, the elements under McSupportData can still contain compile-time variation points. These need to be resolved in sync with the variants selected before compilation of the software, so that the generated A2L content corresponds to the actual code. Therefore, as long as the variants are not resolved, the variation points in the MC support artifact will depend on the system constants needed to resolve these variants.
- In order to support the functional modeling of measurement and calibration data, additional artifacts (based on meta-class McFunction) are (optionally) needed as input to the A2L generator, see 10.4.
- In order to support particular rapid prototyping solutions, references to the description of communication behavior of the involved software components are required, see chapter 10.6.

⌋*(RS_BSWMD_00062)*

[TPS_BSWMDT_04058] McSupportData.measurableSystemConstantValues ⌈In addition to variables and parameters, also names and values of system constants may need to be transferred to an MCD tool in order to be displayed. These are modeled by the role McSupportData.measurableSystemConstantValues. Note that the values of system constants are also possibly subject to compile-time variation (not visible in the figure). ⌋*(RS_BSWMD_00062)*

For details on variant handling refer to [1].

The final A2L-generation is not part of AUTOSAR, but in order to get the complete picture, it should be mentioned, that in addition to the MC support data some further information is required (see also [4]) :

- Output from the linker to find the actual memory addresses, as the MC support data will only contain the C-symbols. In addition, the actual (physical) memory segments must be found from the linker output in cases where the address is not global. Note that the abstract sections defined by MemorySection do not deliver this information.
- Driver specific access information (so called IF-DATA sections) needed by the MC system as part of the A2L-file. These are described in a special non-AUTOSAR data format and shall be generated by the driver modules, e.g. XCP.

- Via the AUTOSAR meta-class AliasNameSet (see [7]) one can provide alternative names as identifiers for the A2L data which could be used by the A2L generator to supersede names given by the MC support data. One possible use case is to resolve name conflicts of system constants which may happen if SwSystemconst names are to be copied to the A2L file out of different ARPackages (this kind of name conflict cannot be resolved by a FlatMap).
- Administrative data for the A2L-File which are nor delivered by AUTOSAR.
- It is up to the A2L generator (and possibly project specific configuration) how data types are converted into A2L which are coded as C-enums.[1]

10.2 Attributes for McSupportData

Figure 10.2 and the following class tables show the attributes which are to be attached to the McSupportData in order to support measurement and calibration by external tools.

[1] This is indicated by the string "enum" as part of the McDataInstance.resultingProperties.additionalNativeTypeQualifier.

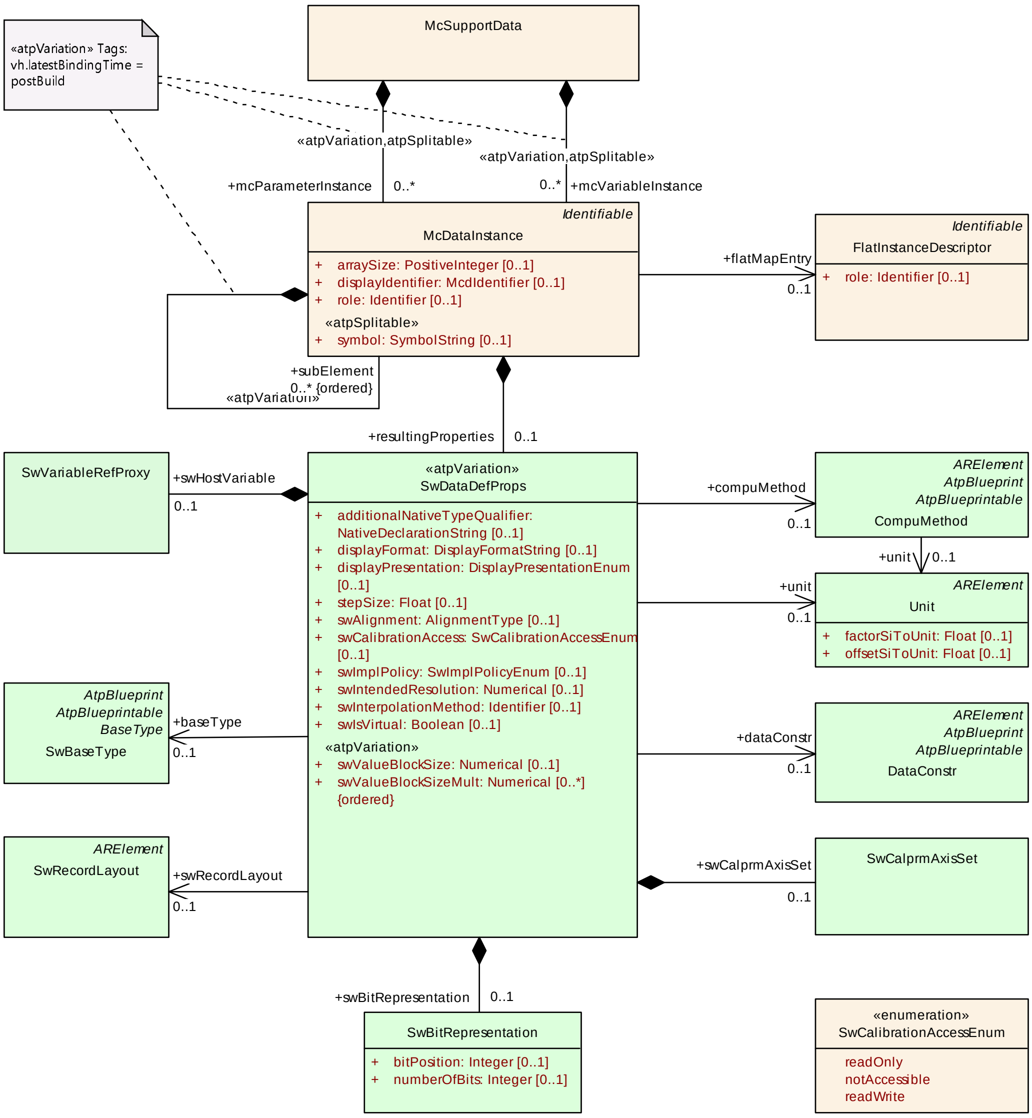

Figure 10.2: Attributes of MC Support Data

Note that McSupportData is a list of calibration elements (parameters) and measurement elements (variables) in which the component hierarchy has been removed. All elements of the list are described by meta-class McDataInstance. This meta-class allows to define arrays and structures, but is does not need a type-prototype-pattern, because it is not designed for reuse on M1:

Class	**McDataInstance**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport

▽

△

Class	**McDataInstance**			
Note	Describes the specific properties of one data instance in order to support measurement and/or calibration of this data instance. The most important attributes are: • Its shortName is copied from the ECU Flat map (if applicable) and will be used as identifier and for display by the MC system. • The category is copied from the corresponding data type (ApplicationDataType if defined, otherwise ImplementationDataType) as far as applicable. • The symbol is the one used in the programming language. It will be used to find out the actual memory address by the final generation tool with the help of linker generated information. It is assumed that in the M1 model this part and all the aggregated and referred elements (with the exception of the Flat Map and the references from ImplementationElementInParameterInstanceRef and McAccessDetails) are completely generated from "upstream" information. This means, that even if an element like e.g. a CompuMethod is only used via reference here, it will be copied into the M1 artifact which holds the complete McSupportData for a given Implementation.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
arraySize	PositiveInteger	0..1	attr	The existence of this attribute turns the data instance into an array of data. The attribute determines the size of the array in terms of number of elements.
displayIdentifier	McdIdentifier	0..1	attr	An optional attribute to be used to set the ASAM ASAP2 DISPLAY_IDENTIFIER attribute.
flatMapEntry	FlatInstanceDescriptor	0..1	ref	Reference to the corresponding entry in the ECU Flat Map. This allows to trace back to the original specification of the generated data instance. This link shall be added by the RTE generator mainly for documentation purposes. The reference is optional because • The McDataInstance may represent an array or struct in which only the subElements correspond to FlatMap entries. • The McDataInstance may represent a task local buffer for rapid prototyping access which is different from the "main instance" used for measurement access.
instanceIn Memory	ImplementationElement InParameterInstance Ref	0..1	aggr	Reference to the corresponding data instance in the description of calibration data structures published by the RTE generator. This is used to support emulation methods inside the ECU, it is not required for A2L generation.
mcDataAccess Details	McDataAccessDetails	0..1	aggr	Refers to "upstream" information on how the RTE uses this data instance. Use Case: Rapid Prototyping
mcData Assignment	RoleBasedMcData Assignment	*	aggr	An assignment between McDataInstances. This supports the indication of related McDataElement implementing the of "RP global buffer", "RP global measurement buffer", "RP enabler flag".
resulting Properties	SwDataDefProps	0..1	aggr	These are the generated properties resulting from decisions taken by the RTE generator for the actually implemented data instance. Only those properties are relevant here, which are needed for the measurement and calibration system.
resultingRptSw Prototyping Access	RptSwPrototyping Access	0..1	aggr	Describes the implemented accessibility of data and modes by the rapid prototyping tooling.

▽

△

Class	**McDataInstance**			
role	Identifier	0..1	attr	An optional attribute to be used for additional information on the role of this data instance, for example in the context of rapid prototyping.
rptImplPolicy	RptImplPolicy	0..1	aggr	Describes the implemented code preparation for rapid prototyping at data accesses for a hook based bypassing.
subElement (ordered)	McDataInstance	*	aggr	This relation indicates, that the target element is part of a "struct" which is given by the source element. This information will be used by the final generator to set up the correct addressing scheme. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbol	SymbolString	0..1	attr	This String is used to determine the memory address during final generation of the MC configuration data (e.g. "A2L" file) . It shall be the name of the element in the programming language such that it can be identified in linker generated information. In case the McDataInstance is part of composite data in the programming language, the symbol String may include parts denoting the element context, unless the context is given by the symbol attribute of an enclosing McDataInstance. This means in particular for the C language that the "." character shall be used as a separator between the name of a "struct" variable the name of one of its elements. The symbol can differ from the shortName in case of generated C data declarations. It is an optional attribute since it may be missing in case the instance represents an element (e.g. a single array element) which has no name in the linker map. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=symbol

Table 10.2: McDataInstance

An `McDataInstance` may represent the root of a nested composite of arrays and/or structs. This is modeled by adding appropriate `subElement`s. In this case, the attribute `McDataInstance.symbol` shall be set only for those elements which actually are visible in the linker map. This should be always the case for the the root element of such a composite (otherwise its address cannot be assigned via the linker map):

[constr_4062] Mandatory symbol for `McDataInstance` root ⌈`McDataInstance`s directly aggregated in `McSupportData` must have a valid `McDataInstance.symbol`. ⌋*()*

[TPS_BSWMDT_04059] Granularity of `McDataInstance.subElements` ⌈Note that it is possible to e.g. define single array elements or struct elements as to be measured or calibrated (the referencing mechanism used in the `FlatInstanceDescriptor` is capable of stating array indexes). In this case one needs to define one `McDataInstance` representing the globally visible C-array or -struct (and stating its symbol) and appropriate `subElement`s for the nested elements to be measured and link these elements to the individual `FlatInstanceDescriptor`s. ⌋*(RS_BSWMD_00062)*

[TPS_BSWMDT_04060] `McDataInstance.resultingProperties` ⌈The figure also shows the meta-classes of the typical elements which might be attached to an

McDataInstance via its SwDataDefProps. These elements (and their further detailing, which is not shown here) are used in the same way as in the SWCT (see [6]) though, as already mentioned, it is expected that the support data will contain copies of the elements found in the SWC- and BSWM-descriptions which refer to each other in a self-contained manner. ⌋*(RS_BSWMD_00062)*

[TPS_BSWMDT_04114] Using the hierarchical structuring of McDataInstance.subElements ⌈The structure of the subElements shall follow the structure of the corresponding ApplicationDataType respectively ImplementationDataType. The value of the symbol attribute of the subElements shall exist and it shall reflect the symbol of the subElement only (as opposed to reflecting the full combined symbol starting from the root element). ⌋*(RS_BSWMD_00062)*

[TPS_BSWMDT_04115] Use of indexing for array element of subElements ⌈McDataInstances have to be created for those array elements that are accessed by MCD in separate and these have to be put as subElements under an McDataInstance representing the whole array. The symbol of the subElement shall contain the array index in the C-notation, e.g [4]. ⌋*(RS_BSWMD_00062)*

10.3 Support for Software Emulation of Calibration Data

The RTE generator provides several methods to allocate calibration data in a way, that they can be emulated by software on the ECU during an online calibration procedure, see [13] for a more detailed description. If such an emulation is configured, the calibration data changed during online calibration are "emulated" by e.g. a Complex Driver, but the access to these data by the functional software is still handled by the RTE. In order to generate or configure the emulation code of e.g. the Complex Driver, the RTE generator has to publish a detailed description of the data structure of the calibration data and supporting elements which directly correspond to its C-code. This information is created by the RTE generator as part the BswInternalBehavior of its own BSWMD, namely by defining local data descriptions as had been shown earlier.

(Note: These local data descriptions should not be mixed up with the input defining the calibration data from the perspective of the module or component using the data. These are for example given as BswInternalBehavior.perInstanceMemory in the BSWMD of the using module, see figure 6.15.)

The generated data descriptions of the RTE are an M1 model of DataPrototypes based on ImplementationDataTypes using the "normal" meta-model elements. But in addition the RTE generator has to provide an information on the so-called calibration method which it actually uses and how this relates to the generated data structures (see [13] for details).

This is expressed by the meta-class McSwEmulationMethodSupport which for convenience is attached to the McSupportData as shown in figure 10.3 and the next two class tables.

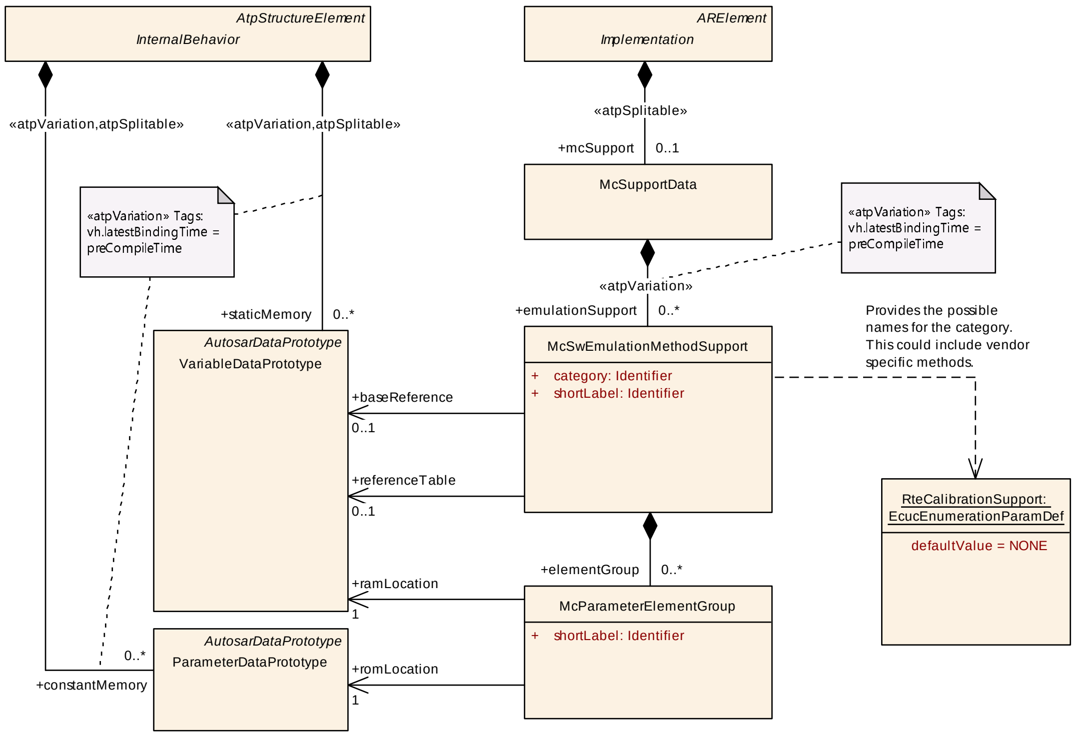

Figure 10.3: Describing the Software Emulation Method for Calibration Data

Class	**McSwEmulationMethodSupport**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	This denotes the method used by the RTE to handle the calibration data. It is published by the RTE generator and can be used e.g. to generate the corresponding emulation method in a Complex Driver. According to the actual method given by the category attribute, not all attributes are always needed: • double pointered method: only baseReference is mandatory • single pointered method: only referenceTable is mandatory • initRam method: only elementGroup(s) are mandatory Note: For single/double pointered method the group locations are implicitly accessed via the reference table and their location can be found from the initial values in the M1 model of the respective pointers. Therefore, the description of elementGroups is not needed in these cases. Likewise, for double pointered method the reference table description can be accessed via the M1 model under baseReference.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
category	Identifier	1	attr	Identifies the actual method. The possible names shall correspond to the symbols of the ECU configuration parameter for the calibration method of the RTE, and can include vendor specific methods. **Tags:** xml.sequenceOffset=-90
baseReference	VariableDataPrototype	0..1	ref	Refers to the base pointer in case of the double-pointered method.
elementGroup	McParameterElementGroup	*	aggr	Denotes the grouping of calibration parameters in the actual RTE code. Depending on the category, this information maybe required to set up the emulation code.
referenceTable	VariableDataPrototype	0..1	ref	Refers to the pointer table in case of the single-pointered method.

▽

△

Class	McSwEmulationMethodSupport			
shortLabel	Identifier	1	attr	Assigns a name to this element. **Tags:** xml.sequenceOffset=-100

Table 10.3: McSwEmulationMethodSupport

Class	McParameterElementGroup			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	Denotes a group of calibration parameters which are handled by the RTE as one data structure.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
ramLocation	VariableDataPrototype	1	ref	Refers to the RAM location of this parameter group. To be used for the init-RAM method.
romLocation	ParameterDataPrototype	1	ref	Refers to the ROM location of this parameter group. To be used for the init-RAM method.
shortLabel	Identifier	1	attr	Assigns a name to this element. **Tags:** xml.sequenceOffset=-100

Table 10.4: McParameterElementGroup

[TPS_BSWMDT_04061] McSwEmulationMethodSupport.category ⌈The value of McSwEmulationMethodSupport.category can either correspond to the enumeration value of the RTE configuration parameter RteCalibrationSupport (namely DOUBLE_POINTERED, SINGLE_POINTERED or INITIALIZED_RAM, see [13]), or it can be chosen differently in order to denote a vendor specific method. ⌋ *(RS_BSWMD_00062)*

[constr_4044] Content of McSwEmulationMethodSupport ⌈ The following constraints hold for the attributes of McSwEmulationMethodSupport:

- If category is DOUBLE_POINTERED, a baseReference must exist.
- If category is SINGLE_POINTERED, a referenceTable must exist.
- If category is INITIALIZED_RAM, one or more elementGroups must exist.

⌋*()*

[TPS_BSWMDT_04062] Upstream reference for emulation support ⌈For a full support of software emulation, we also need a relation between the “upstream” parameter description (represented by an entry in the ECU Flat Map) and the actually implemented code element. This is shown in figure 10.4. The required reference ImplementationElementInParameterInstanceRef is attached to McDataInstance. This is mainly done for convenience, as McDataInstance is generated in the same step and already refers to the Flat Map. This part of the meta-model assumes, that the RTE generator uses ImplementationDataTypes to describe the implemented data structures and that each implemented parameter element is part of a group, thus resulting in a ImplementationDataTypeElement as the target of the reference. ⌋ *(RS_BSWMD_00062)*

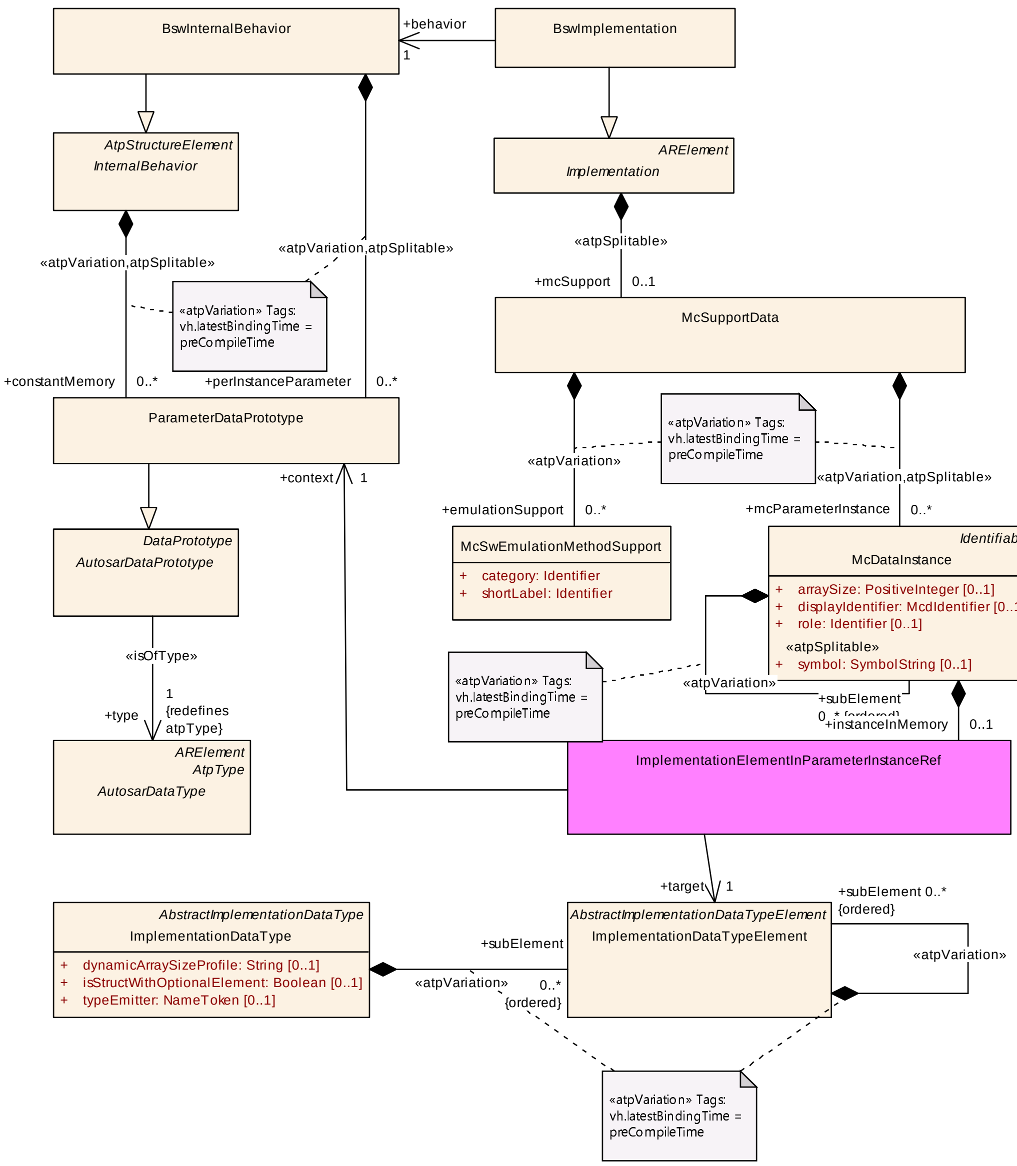

Figure 10.4: Reference to the Implemented Data needed for Emulation

Class	**ImplementationElementInParameterInstanceRef**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport
Note	Describes a reference to a particular ImplementationDataTypeElement instance in the context of a given ParameterDataPrototype. Thus it refers to a particular element in the implementation description of a software data structure. Use Case: The RTE generator publishes its generated structure of calibration parameters in its BSW module description using the "constantMemory" role of ParameterDataPrototypes. Each ParameterData Prototype describes a group of single calibration parameters. In order to point to these single parameters, this "instance ref" is needed. Note that this class follows the pattern of an InstanceRef but is not implemented based on the abstract classes because the ImplementationDataType isn't either, especially because ImplementationDataType Element isn't derived from AtpPrototype.

▽

△

Class	ImplementationElementInParameterInstanceRef			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
context	ParameterDataPrototype	1	ref	The context for the referred element. **Tags:** xml.sequenceOffset=20
target	ImplementationDataTypeElement	1	ref	The referred data element. **Tags:** xml.sequenceOffset=30

Table 10.5: ImplementationElementInParameterInstanceRef

[constr_4034] Target and context of MC emulation reference ⌈ Within one `ImplementationElementInParameterInstanceRef`, the `target` must refer to a sub-element of the `ParameterDataPrototype` which is referred as `context`. ⌋*()*

If the elements to be measured or calibrated are part of arrays or structs, it is important to define the references in a consistent and complete way for all sub-elements involved in order to avoid ambiguities. Since the `ImplementationElementInParameterInstanceRef` allows to define only one context element, we need the following constraint:

[constr_4061] Completeness of MC emulation reference ⌈ If an `McDataInstance` in the role of a `subElement` of another `McDataInstance` specifies an `instanceInMemory`, then the containing `McDataInstance` must also specify an `instanceInMemory`. The `target` of the latter (i.e. upper level) `instanceInMemory` must be identical (including array index, if defined) to the `context` of the first (i.e. lower level) `instanceInMemory`. ⌋*()*

Without this constraint, it would be possible to define a reference to an inner element of nested arrays/structs without that the corresponding global C variable could be identified.

10.4 Support for Functional Modeling of Measurement and Calibration

The “A2L” description format for measurement and calibration data allows to associate the data with so-called *functions* in order to guide the calibration engineer in handling a large number of such data (see description of the keyword FUNCTION in [22]).

Such functions are mainly logical constructs and do not necessarily match to software objects like modules or components in the sense of AUTOSAR. However, since it is the goal of measurement and calibration support of AUTOSAR to be able to generate A2L descriptions from AUTOSAR XML descriptions, the AUTOSAR meta-model also provides the means to define such functions in the sense of A2L.

[TPS_BSWMDT_04078] Semantics of McFunction ⌈ The meta-class McFunction together with associated McFunctionDataRefSets can be used to define the association of measurement and/or calibration data in a software system to a logical function in various roles. In addition, it allows to structure such functions hierarchically. ⌋ *(RS_BSWMD_00062)*

Note that McFunction is an ARElement so it can be used to define standalone artifacts which strictly speaking do not belong to any particular BSWMD. Nonetheless this part of the meta-model is described in this document because it belongs to the overall support for measurement and calibration.

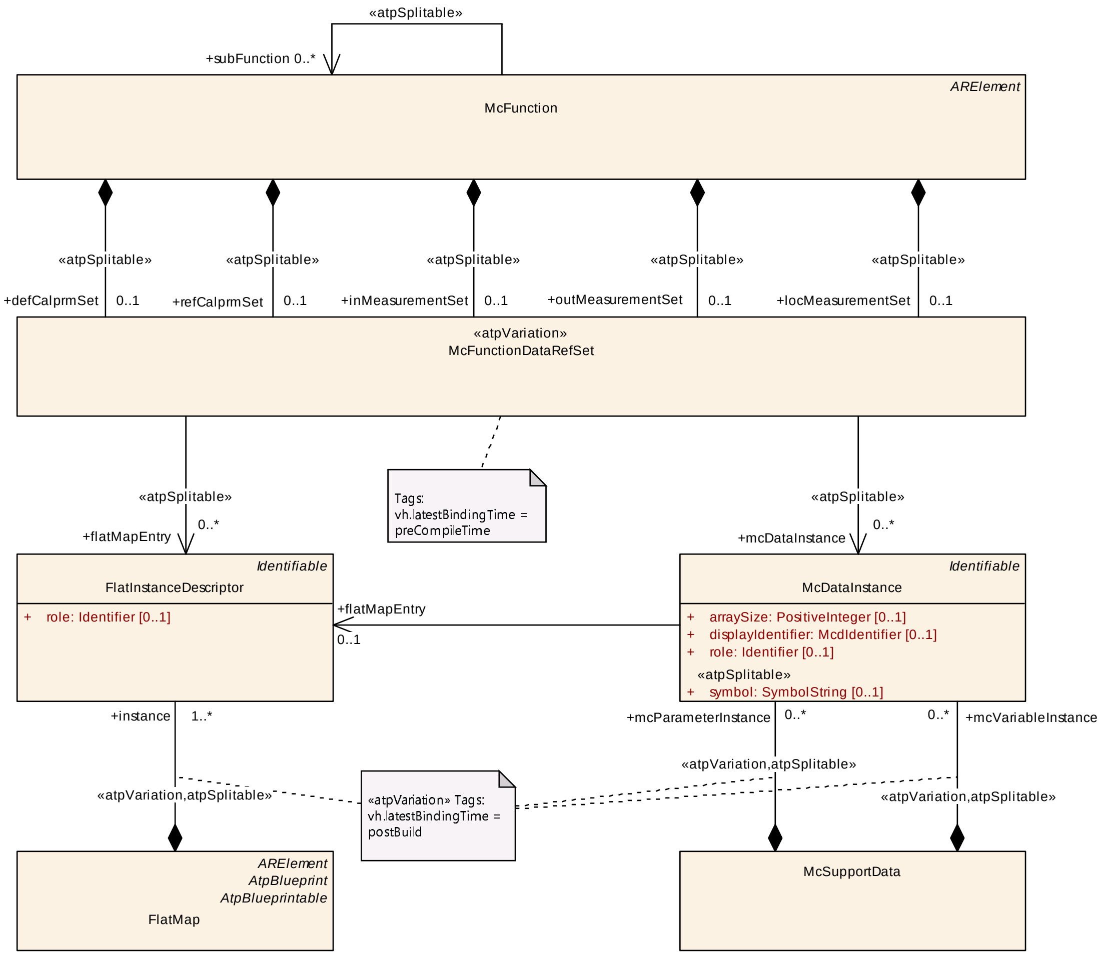

Figure 10.5: Meta-model for McFunction

Class	McFunction
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport

▽

△

Class	**McFunction**			
Note	Represents a functional element to be used as input to support measurement and calibration. It is used to • assign calibration parameters to a logical function • assign measurement variables to a logical function • structure functions hierarchically **Tags:** atp.recommendedPackage=McFunctions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
defCalprmSet	McFunctionDataRefSet	0..1	aggr	Refers to the set of adjustable data (= calibration parameters) defined in this function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=10
inMeasurement Set	McFunctionDataRefSet	0..1	aggr	Refers to the set of measurable input data for this function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=30
loc Measurement Set	McFunctionDataRefSet	0..1	aggr	Refers to the set of measurable local data in this function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=50
out Measurement Set	McFunctionDataRefSet	0..1	aggr	Refers to the set of measurable output data from this function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=60
refCalprmSet	McFunctionDataRefSet	0..1	aggr	Refers to the set of adjustable data (= calibration parameters) referred by this function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=20
subFunction	McFunction	*	ref	A sub-function that is seen as part of the enclosing function. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subFunction xml.sequenceOffset=70

Table 10.6: McFunction

Class	«atpVariation» **McFunctionDataRefSet**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport

▽

△

Class	«atpVariation» **McFunctionDataRefSet**			
Note	Refers to a set of data assigned to an McFunction in a particular role. The data are given • either by entries in a FlatMap • or by data instances that are part of MC support data. These two possibilities are exclusive within a given McFunctionDataRefSet. Which one to use depends on the process and tool environment. The set is subject to variability because the same functional model may be used with various representation of the data. **Tags:** vh.latestBindingTime=preCompileTime			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
flatMapEntry	FlatInstanceDescriptor	*	ref	Refers to an entry in a FlatMap that is part of the set, for example a calibration parameter or measured variable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=flatMapEntry xml.sequenceOffset=10
mcDataInstance	McDataInstance	*	ref	Refers to a data instance within MC support data that is part of the set, i.e. a calibration parameter or measured variable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcDataInstance xml.sequenceOffset=20

Table 10.7: McFunctionDataRefSet

[TPS_BSWMDT_04087] Scope of McFunctionDataRefSets ⌈It should be noted that McFunctionDataRefSets can refer to the data either via instances of FlatInstanceDescriptor or McDataInstance:

- The first possibility, i.e. the association via a FlatMap allows to define McFunctions rather early in the project on ECU or even System level before the actual McSupport has been generated.
- The second possibility, the association to McDataInstances allows to define (or transform) McFunctions for usage in a self-contained manner together with the McSupport data for A2L generation.

⌋*(RS_BSWMD_00062)*

[TPS_BSWMDT_04088] Usage of McFunction ⌈ Since the use cases for McFunction are considered as rather project specific and the specification how to generate A2L does not belong to AUTOSAR, not all possible constraints on the attributes and association owned by McFunction are specified in this document. Especially it is not standardized, how instances of McFunctions have to be derived from an M1 model of AUTOSAR software components or modules. ⌋*(RS_BSWMD_00062)*

Still some constraints are considered as mandatory:

[constr_4068] McFunctionDataRefSet.flatInstanceDescriptor's semantic ⌈

- An McFunctionDataRefSet aggregated in the role of McFunction.defCalprmSet or McFunction.refCalprmSet shall only refer to FlatInstanceDescriptors that
  - either can be traced down to a ParameterDataPrototype
  - or can be traced down to a VariableDataPrototype of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK

  and which are declared for calibration access i.e. have an associated SwDataDefProps.swCalibrationAccess set to readWrite or readOnly.
- An McFunctionDataRefSet aggregated in the role of McFunction.inMeasurementSet, McFunction.outMeasurementSet or McFunction.locMeasurementSet shall only refer to FlatInstanceDescriptors that can be traced down to either a VariableDataPrototype, an ArgumentDataPrototype or a ModeDeclarationGroupPrototype and are declared as measurable i.e. have an associated SwDataDefProps.swCalibrationAccess set to readOnly.

⌋*()*

[constr_4069] McFunctionDataRefSet.mcDataInstance's semantic ⌈

- An McFunctionDataRefSet aggregated in the role of McFunction.defCalprmSet or McFunction.refCalprmSet shall only refer to McDataInstances that are declared for calibration access i.e. are aggregated in the role McSupportData.mcParameterInstance or McSupportData.mcVariableInstance of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK.
- An McFunctionDataRefSet aggregated in the role of McFunction.inMeasurementSet, McFunction.outMeasurementSet or McFunction.locMeasurementSet shall only refer to McDataInstances that are declared as measurable i.e. are aggregated in the role McSupportData.mcVariableInstance.

⌋*()*

Please note, that VariableDataPrototypes - end corresponding McDataInstances - of category of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK are describing so called adaptive characteristics. Those are modifiable during the ECU run-time and therefore described as VariableDataPrototypes but are CHARACTERISTICs in the sense of A2L.

Older versions of the meta-model didn't contain the meta-class McFunction but there was already the possibility to specify the name of a function associated with a data object by the attribute SwDataDefProps.mcFunction. This had serious limitations as is was neither possible to define input data to a function, nor to define more than one function associated with some data, nor to define sub-functions. For backward compatibility reasons this possibility still exists but the attribute has been tagged as obsolete.

10.5 Support for Structuring of Measurement and Calibration

The “A2L” description format for measurement and calibration data allows to associate the data with so-called *groups* in order to support structuring of projects involving a very large number measurement and calibration data (see description of the keyword GROUP in [22]).

Such groups are used to structure measurement and calibration data as well as functions according user specific criteria, e.g. a structuring according the C file assignment or calibration components which describe the calibration engineers viewpoint. Therefor groups are mainly logical constructs and do not necessarily match to software objects like modules or components in the sense of AUTOSAR. However, since it is the goal of measurement and calibration support of AUTOSAR to be able to generate A2L descriptions from AUTOSAR XML descriptions, the AUTOSAR meta-model also provides the means to define such groups in the sense of A2L.

[TPS_BSWMDT_04168] Semantics of McGroup ⌈ The meta-class McGroup together with associated McGroupDataRefSets can be used to define the association of measurement and/or calibration data and/or functions in a software system to a logical structure in various roles. In addition, it allows to structure such groups hierarchically. ⌋*(RS_BSWMD_00062)*

Similar as McFunction the McGroup is an ARElement so it can be used to define standalone artifacts which strictly speaking do not belong to any particular BSWMD. Nonetheless this part of the meta-model is described in this document because it belongs to the overall support for measurement and calibration.

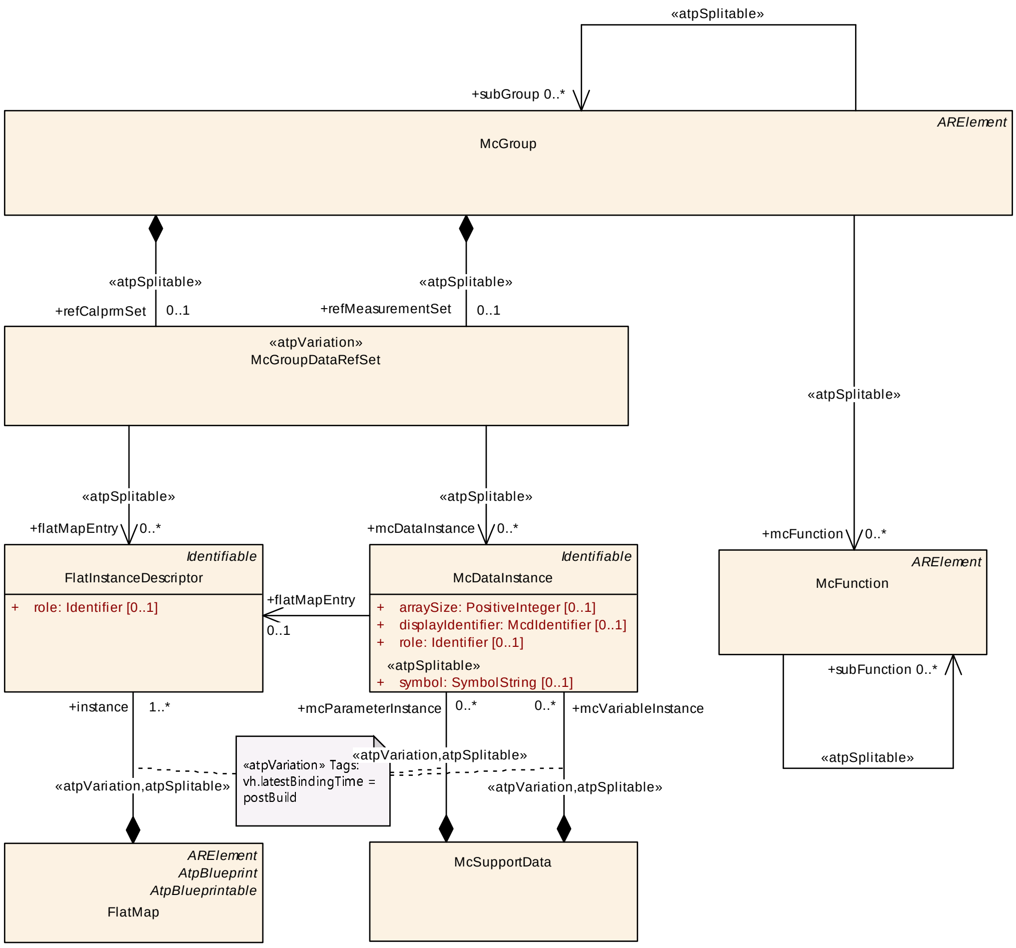

Figure 10.6: Meta-model for McGroup

Class	**McGroup**			
Package	M2::AUTOSARTemplates::CommonStructure::McGroups			
Note	Represents a group element to be used as input to support measurement and calibration. It is used to provide selection lists (groups) of calibration parameters, measurement variables, and functions in a hierarchical manner (subGroups). **Tags:** atp.recommendedPackage=McFunctions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
mcFunction	McFunction	*	ref	A McFunction that is seen as part of the enclosing group. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcFunction xml.sequenceOffset=40
refCalprmSet	McGroupDataRefSet	0..1	aggr	Refers to the set of adjustable data (= calibration parameters) referred by this McGroup. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=20
ref Measurement Set	McGroupDataRefSet	0..1	aggr	Refers to the set of measurable belonging to this Mc Group. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=variationPoint.shortLabel xml.sequenceOffset=30

▽

△

Class	McGroup			
subGroup	McGroup	*	ref	A sub-group that is seen as part of the enclosing group. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subGroup xml.sequenceOffset=10

Table 10.8: McGroup

Class	«atpVariation» **McGroupDataRefSet**			
Package	M2::AUTOSARTemplates::CommonStructure::McGroups			
Note	Refers to a set of data assigned to an McGroup in a particular role. The data are given • either by entries in a FlatMap • or by data instances that are part of MC support data. These two possibilities can be mixed within a given McGroupDataRefSet. Which one to use depends on the process and tool environment. The set is subject to variability because the same functional model may be used with various representation of the data. **Tags:** vh.latestBindingTime=preCompileTime			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
flatMapEntry	FlatInstanceDescriptor	*	ref	Refers to an entry in a FlatMap that is part of the set, for example a calibration parameter or measured variable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=flatMapEntry xml.sequenceOffset=50
mcDataInstance	McDataInstance	*	ref	Refers to a data instance within MC support data that is part of the set, i.e. a calibration parameter or measured variable. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=mcDataInstance xml.sequenceOffset=60

Table 10.9: McGroupDataRefSet

[TPS_BSWMDT_04169] Scope of `McGroupDataRefSets` ⌈ `McGroupDataRefSet`s can refer to the data either via instances of `FlatInstanceDescriptor` or `McDataInstance`:

- The first possibility, i.e. the association via a `FlatMap` allows to define `McGroup`s rather early in the project on ECU or even System level before the actual McSupport has been generated.
- The second possibility, the association to `McDataInstance`s allows to define (or transform) `McGroup`s for usage in a self-contained manner together with the McSupport data for A2L generation.

⌋ *(RS_BSWMD_00062)*

[TPS_BSWMDT_04170] Usage of `McGroup` ⌈ Since the use cases for `McGroup` are considered as rather project specific and the specification how to generate A2L does not belong to AUTOSAR, not all possible constraints on the attributes and association owned by `McGroup` are specified in this document. Especially it is not standardized, how instances of McGroups have to be derived from an M1 model of AUTOSAR software components or modules. ⌋*(RS_BSWMD_00062)*

Still some constraints are considered as mandatory:

[constr_4101] Semantics of McGroupDataRefSet.flatInstanceDescriptor ⌈

- An McGroupDataRefSet aggregated in the role of McGroup.refCalprmSet or McGroup.refCalprmSet shall only refer to FlatInstanceDescriptors that can either be traced down to a ParameterDataPrototype or can be traced down to a VariableDataPrototype of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK and which are declared for calibration access i.e. have an associated SwDataDefProps.swCalibrationAccess set to readWrite or readOnly.
- An McGroupDataRefSet aggregated in the role of McGroup.refMeasurementSet shall only refer to FlatInstanceDescriptors that can be traced down to either a VariableDataPrototype, an ArgumentDataPrototype or a ModeDeclarationGroupPrototype and are declared as measurable i.e. have an associated SwDataDefProps.swCalibrationAccess set to readOnly.

⌋*()*

[constr_4102] Semantics of McGroupDataRefSet.mcDataInstance ⌈

- An McGroupDataRefSet aggregated in the role of McGroup.refCalprmSet shall only refer to McDataInstances that are declared for calibration access i.e. are aggregated in the role McSupportData.mcParameterInstance or McSupportData.mcParameterInstance of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK.
- An McGroupDataRefSet aggregated in the role of McGroup.refMeasurementSet shall only refer to McDataInstances that are declared as measurable i.e. are aggregated in the role McSupportData.mcVariableInstance.

⌋*()*

Please note, that VariableDataPrototypes - end corresponding McDataInstances - of category of category COM_AXIS, RES_AXIS, CURVE, MAP, CUBOID, CUBE_4, CUBE_5 or VAL_BLK are describing so called adaptive characteristics. Those are modifiable during the ECU run-time and therefore described as VariableDataPrototypes but are CHARACTERISTICs in the sense of A2L.

10.6 McSupportData for Rapid Prototyping

The AUTOSAR meta-model supports the description of a software system that include rapid prototyping scenarios of Application Software Components. The high level part of such a description is done with the help of the meta-class RapidPrototypingScenario, see [6] for documentation.

So far this “high level” description of rapid prototyping is not a topic for the BSWMDT. However some special solutions for rapid prototyping require a direct access to RTE internal data buffers that are used to hold the data for communication between software components:

- The rapid prototyping implementation (which could run on an external ECU or as a Complex Driver on the same ECU) may directly[2] access the RTE data buffers in a similar way as it is done from an MCD system (e.g. via an XCP driver)
- The rapid prototyping functionality may be embedded in the RTE itself. In this case, external data access is needed to monitor the data as well as to switch between the “prototyping” and the “original” behavior of the RTE for a particular data access point.

In order to configure a rapid prototyping system that works according to the solutions outlined above, some knowledge on the RTE internal data buffers has to be provided to external tools in a similar way as for MCD access. Therefore the meta-classes below McSupportData are used for this purpose too. Several extensions to these meta-classes are needed for these use cases.

[2] “directly” means not via an RTE API or an RTE hook function

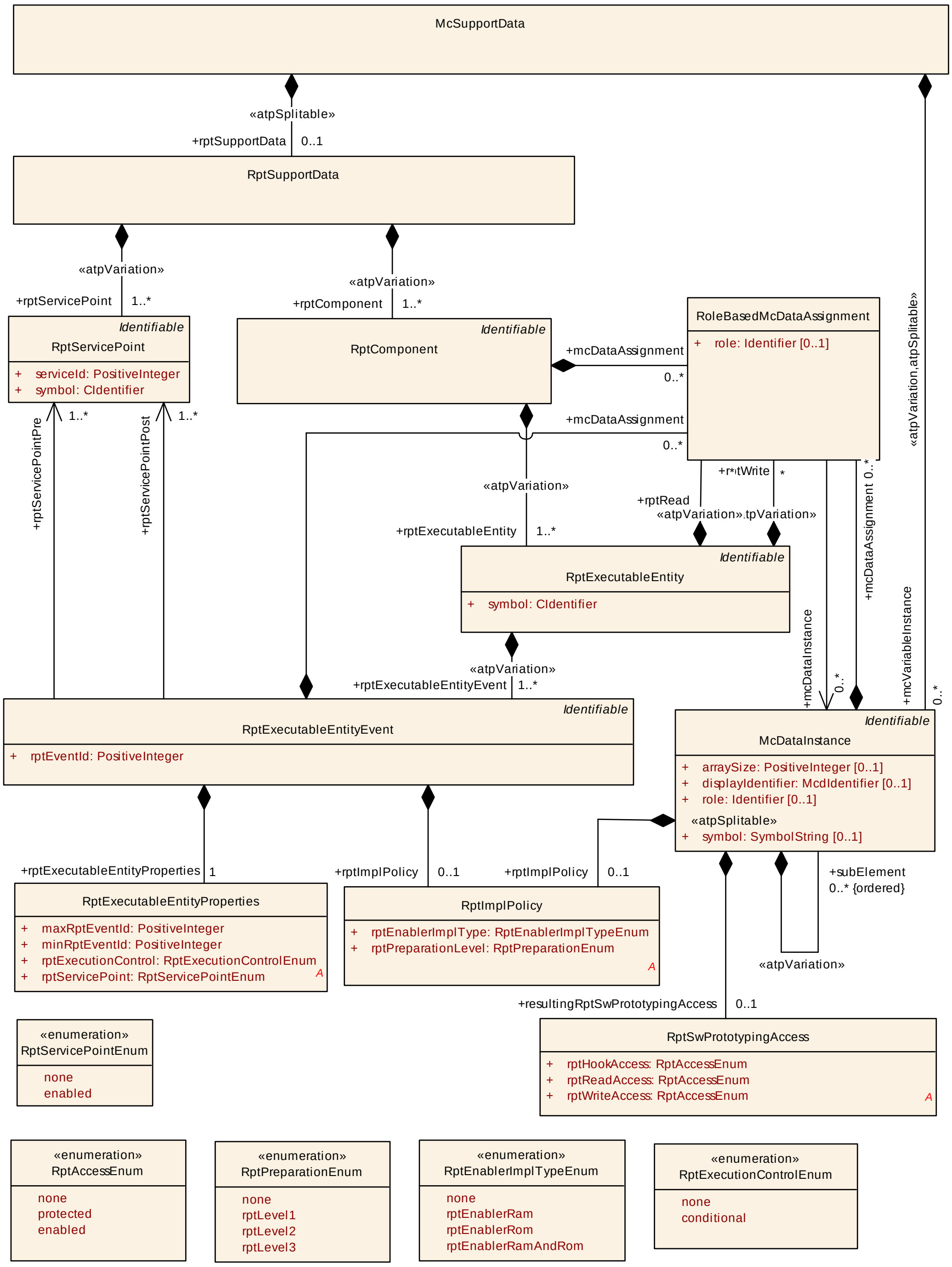

Figure 10.7: Extension of `McSupportData` for Rapid Prototyping

[TPS_BSWMDT_04094] Details of `McDataInstance` for rapid prototyping ⌈ Especially for the prototyping of a `RunnableEntity` with implicit communication, typically more than one RTE internal buffer needs to be accessed and it needs to be described what kind of data access and what RTE event is associated with each buffer.

This information can be provided (for example generated) by setting the references in `McDataInstance.mcDataAccessDetails`. The base of these references shall be the ECU Extract to which also the RTE implementation belongs for which the `McSupportData` is meant (see also constraint below).

In addition to this, the attribute McDataInstance.role may be used to add more information on the particular role of this data instance. Note the the content of this attribute is not standardized. ⌋(RS_BSWMD_00065)

[constr_4073] McDataAccessDetails shall refer to one ECU Extract ⌈ Within one given McDataAccessDetails, all instances of System referenced as the base of any McDataAccessDetails.roleMcDataAccessDetails or as the base of any McDataAccessDetails.roleMcDataAccessDetails shall be identical and of category ECU_EXTRACT. ⌋()

Class	McDataAccessDetails			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	This meta-class allows to attach detailed information about the usage of a data buffer by the RTE to a corresponding McDataInstance. Use Case: Direct memory access to RTE internal buffers for rapid prototyping. In case of implicit communication, the various task local buffers need to be identified in relation to RTE events and variable access points. Note that the SwComponentPrototype, the RunnableEntity and the VariableDataPrototype are implicitly given be the referred instances of RTEEvent and VariableAccess.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
rteEvent	RTEEvent	1..*	iref	The RTE event used to receive the data via this buffer.
variableAccess	VariableAccess	1..*	iref	The VariableAccess for which the data buffer is used.

Table 10.10: McDataAccessDetails

[TPS_BSWMDT_04095] Relationships between McDataInstances ⌈ In the case that rapid prototyping is embedded in the RTE, several McDataInstances are needed which have relationships to each other. For example, there could be a buffer holding the “original” data, a buffer holding the “replacement” data coming from a prototype implementation and a data instance holding the “switch” for switching between normal and replacement functionality.

The meta-class RoleBasedMcDataAssignment offers the possibility to express the relationships between such associated RTE data formally and use them as input to configure external software. Note that the meta-model is rather generic at this point in order to allow project specific use cases. Therefore the values of the attribute RoleBasedMcDataAssignment.role are not standardized except one:

- The value mainInstance of this attribute shall be used to characterize the relation to that particular McDataInstance that represent the main instance of this data buffer - i.e. the one that would be normally displayed in an MCD system.

⌋(RS_BSWMD_00065)

Class	**RoleBasedMcDataAssignment**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport			
Note	This meta-class allows to define links that specify logical relationships between single McDataInstances. The details on the existence and semantics of such links are not standardized. Possible Use Case: Rapid Prototyping solutions in which additional communication buffers and switches are implemented in the RTE that allow to switch between the usage of the original and the bypass buffers. The different buffers and the switch can be represented by McDataInstances (in order to be accessed by MC tools) which have relationships to each other.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
execution Context	RptExecutionContext	*	ref	Determines the executionContext in which the McData Instance describing a local (e.g Task-Local) buffer of a gobal buffer is valid.
mcDataInstance	McDataInstance	*	ref	The target of the assignment.
role	Identifier	0..1	attr	Shall be used to specify the role of the assigned data instance in relation to the instance that owns the assignment. The standardized roles of the RoleBasedMcData Assignment.role attribute are: • GlobalMeasurementBuffer • RpEnablerFlag • RpRunnableDisablerFlag • BufferOf

Table 10.11: RoleBasedMcDataAssignment

[TPS_BSWMDT_04096] Split between different use cases of McSupportData ⌈It should be noted that the aggregation of McDataInstance by McSupportData is splitable. This allows to keep the data description for MCD use cases and rapid prototyping use cases in separate artifacts and also to generate them at a different points in time. ⌋*(RS_BSWMD_00065)*

In the case that rapid prototyping is embedded in the RTE, different kinds of McDataInstances are needed. To describe the kind of the memory to which the McDataInstance relates, the attribute role is used. To describe the relationships between different kinds of McDataInstances or other elements the RoleBasedMcDataAssignment.role attribute is used. Basically the role values can be defined project specific but for the use case of rapid prototyping several role values and the according semantic are standardized.

[TPS_BSWMDT_04159] Standardized values of attribute RoleBasedMcDataAssignment.role ⌈For the use case of rapid prototyping several role values and the according semantic are standardized and described in the following table:10.12. ⌋*(RS_BSWMD_00065)*

role	*Explanation*
RptGlobalMeasurementBuffer	Specifies the relationship between a global buffer holding the data value used by ECU SW and the memory location which used by the MCD System to access the value for subsequent measurement purposes before replacement by the RP generated value.

RptGlobalMeasurementBufferIn	Specifies the relationship between a global buffer holding a inout argument of a ClientServerOperation and the data value used by ECU SW and the memory location which used by the RP tool or MCD System to access the originally IN value.
RptGlobalMeasurementBufferOut	Specifies the relationship between a global buffer holding a inout argument of a ClientServerOperation and the data value used by ECU SW and the memory location which used by the RP tool or MCD System to access the originally OUT value.
RptGlobalBuffer	Specifies the relationship between a rapid prototyping global buffer holding the data value written / read by the RP tool and the memory location which used by the RTE holding the value used for communication from/to other software component instances.
RptGlobalBufferIn	Specifies the relationship between a rapid prototyping global buffer holding the data value for a inout argument of a ClientServerOperation written / read by the RP tool for the IN direction and the memory location which used by the RTE holding the value used for communication from/to other software component instances.
RptGlobalBufferOut	Specifies the relationship between a rapid prototyping global buffer holding the data value for a inout argument of a ClientServerOperation written / read by the RP tool for the OUT direction and the memory location which used by the RTE holding the value used for communication from/to other software component instances.
RptRomEnablerFlag	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in ROM. This is used for run-time enabling/disabling the bypass.
RptRomEnablerFlagIn	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in ROM for the IN direction of an inout argument of a ClientServerOperation. This is used for runtime enabling/disabling the bypass.
RptRomEnablerFlagOut	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in ROM for the OUT direction of an inout argument of a ClientServerOperation. This is used for runtime enabling/disabling the bypass.
RptRamEnablerFlag	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in RAM. This is used for run-time enabling/disabling the bypass.
RptRamEnablerFlagIn	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in RAM for the IN direction of an inout argument of a ClientServerOperation. This is used for runtime enabling/disabling the bypass.
RptRamEnablerFlagOut	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag in RAM for the OUT direction of an inout argument of a ClientServerOperation. This is used for runtime enabling/disabling the bypass.
RptRunnableDisablerFlag	Specifies the relationship to the memory location implementing a rapid prototyping enabler flag controlling the execution of ExecutableEntitys
RptStimEnabler	Specifies the relationship to the memory location implementing the service point stimulation enabler flag. This is used for run-time enabling/disabling the stimulation by the service point.
ImplicitBuffer	Specifies the relationship from a McDataInstance describing a implicit communication buffer to the McDataInstance describing a global buffer.

Table 10.12: RoleBasedMcDataAssignment.role values

10.7 Rapid Prototyping support data

10.7.1 Rapid Prototyping support for software components or basic software modules

The meta class RptSupportData provides the infrastructure to describe the implemented Rapid Prototyping support in a software component or basic software module(s). Thereby it is possible, that the Rapid Prototyping is locally implemented in a software component or basic software module for the software entity itself or in case of RTE that the Rapid Prototyping support is implemented on the demand of the RapidPrototypingScenario for the integration of the respective software components or basic software modules.

Class	**RptSupportData**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Root element for rapid prototyping support data related to one Implementation artifact on an ECU, in particular the RTE. The rapid prototyping support data may reference to elements provided for McSupportData.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
executionContext	RptExecutionContext	1..*	aggr	Defines an environment for the execution of Executable Entites.
rptComponent	RptComponent	1..*	aggr	Description of components for which rapid prototyping support is implemented. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
rptServicePoint	RptServicePoint	1..*	aggr	This aggregation represents the collection of service points associated with the enclosing RptSuportData **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 10.13: RptSupportData

Class	**RptSwPrototypingAccess**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Describes the accessibility of data and modes by the rapid prototyping tooling.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
rptHookAccess	RptAccessEnum	1	attr	The related data element can be modified using a post-build hooking tool. An ENABLED VariableDataPrototype is implicitly READABLE/WRITABLE.
rptReadAccess	RptAccessEnum	1	attr	The related data element can be used as input for bypass functionality by RP tool. If rptImplPolicy is not specified then RTE generation must ensure at least suitable MC read points are created.

▽

△

Class	**RptSwPrototypingAccess**			
rptWriteAccess	RptAccessEnum	1	attr	The related data element can be used as output for bypass functionality by RP tool. The data element must be prepared to rptLevel2 and related write service points are present.

Table 10.14: RptSwPrototypingAccess

Class	**RptComponent**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Description of component instance for which rapid prototyping support is implemented.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
mcData Assignment	RoleBasedMcData Assignment	*	aggr	Reference to related McDataElement describing the implementation of "RP global buffer", "RP global measurement buffer", "RP enabler flag" and the "RP runnable disabler flag".
rpImplPolicy	RptImplPolicy	0..1	aggr	Describes the implemented code preparation for rapid prototyping at data accesses.
rptExecutable Entity	RptExecutableEntity	1..*	aggr	ExecutableEntity instance which can be bypassed. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 10.15: RptComponent

[TPS_BSWMDT_04160] `RptComponent` represents a software component or basic software module ⌈ `RptComponent` describes a software component or basic software module (e.g. a CDD) for which rapid prototyping support is implemented. ⌋ *(RS_BSWMD_00065)*

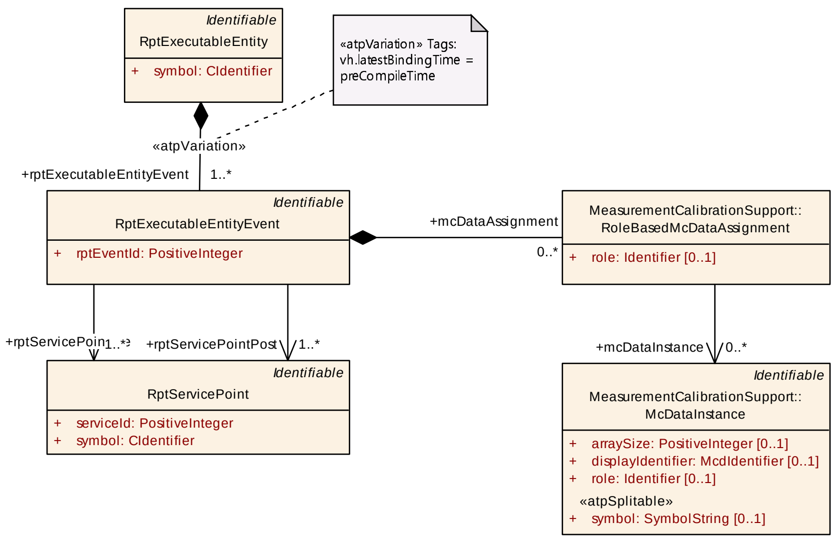

Figure 10.8: Meta-model for the usage of `RptServicePoint`

[TPS_BSWMDT_04161] `RptExecutableEntity` represents a `ExecutableEntity` with rapid prototyping support ⌈ The `RptExecutableEntity` describes a ExecutableEntity for which rapid prototyping support is implemented. ⌋ *(RS_BSWMD_00065)*

Class	**RptExecutableEntity**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	This describes a ExecutableEntity instance which can be bypassed.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
rptExecutableEntityEvent	RptExecutableEntityEvent	1..*	aggr	ExecutableEntity event instance activation the owning RptExecutableEntity. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
rptRead	RoleBasedMcDataAssignment	*	aggr	read access to a variable **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
rptWrite	RoleBasedMcDataAssignment	*	aggr	write access to a variable **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbol	CIdentifier	1	attr	The symbol describing this ExecutableEntity's entry point.

Table 10.16: RptExecutableEntity

[TPS_BSWMDT_04162] RptExecutableEntityEvent represents a RTEEvent or BswEvent for with rapid prototyping support ⌈ The RptExecutableEntityEvent describes a instance of a RTEEvent or BswEvent for which rapid prototyping support is implemented. This means typically that Service Function calls before and after the call of the ExecutableEntity implementing the activation by the RTEEvent or BswEvent. ⌋ *(RS_BSWMD_00065)*

Class	**RptExecutableEntityEvent**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	This describes an ExecutableEntity event instance which can be bypassed.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
executionContext	RptExecutionContext	1..*	ref	This describes the context in which the event of the executable entity is executed.
mcDataAssignment	RoleBasedMcDataAssignment	*	aggr	Reference to related McDataElements describing the implementation of "RP runnable disabler flag" and "stimulation enabler flag" The possible roles of the RoleBasedMcDataAssignment.role attribute are: • RpRunnableDisablerFlag"
rptEventId	PositiveInteger	1	attr	RPT event id used for service points call.
rptExecutableEntityProperties	RptExecutableEntityProperties	1	aggr	Describes the implemented code preparation for rapid prototyping at ExecutableEntity invocation.
rptImplPolicy	RptImplPolicy	0..1	aggr	Describes the RptImplPolicy of a RptExecutableEvent for service based bypassing.
rptServicePointPost	RptServicePoint	1..*	ref	This describes the applicable Post Service Points for a RTEEvent / BswEvent of a bypassed ExecutableEntity.

▽

△

Class	**RptExecutableEntityEvent**			
rptServicePoint Pre	RptServicePoint	1..*	ref	This describes the applicable Pre Service Points for a RTEEvent / BswEvent of a bypassed ExecutableEntity.

Table 10.17: RptExecutableEntityEvent

Class	**RptImplPolicy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	Describes the code preparation for rapid prototyping at data accesses.			
Base	*ARObject*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
rptEnablerImpl Type	RptEnablerImplType Enum	1	attr	For Level 2 or Level3 this property determines how the RTE implements the additional "RP enabler" flag.
rptPreparation Level	RptPreparationEnum	1	attr	Mandates RP preparation level for access to VariableData Prototype within generated RTE implementation.

Table 10.18: RptImplPolicy

Enumeration	**RptEnablerImplTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Describes the required / implemented usage of enabler flags for data access in the code.
Literal	*Description*
none	No "RP enabler" is implemented. **Tags:** atp.EnumerationValue=0
rptEnablerRam	"RP enabler" is implemented as a RAM variable **Tags:** atp.EnumerationValue=1
rptEnablerRamAnd Rom	The RTE generator implements both the RAM and ROM "RP enabler". **Tags:** atp.EnumerationValue=3
rptEnablerRom	"RP enabler" is implemented as a calibrateable ROM variable. **Tags:** atp.EnumerationValue=2

Table 10.19: RptEnablerImplTypeEnum

Enumeration	**RptPreparationEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines the RP preparation level for access to VariableDataPrototypes within the generated RTE implementation.
Literal	*Description*
none	No RP preparation for VariableDataPrototype. **Tags:** atp.EnumerationValue=0
rptLevel1	The RTE implementation uses an "RP global buffer" for measurement and post-build hooking purposes. **Tags:** atp.EnumerationValue=1

▽

△

Enumeration	**RptPreparationEnum**
rptLevel2	As rpLevel1 but the RTE implementation also uses both "RP enabler flag" to permit RP overwrite at run-time. **Tags:** atp.EnumerationValue=2
rptLevel3	As rpLevel2 but the RTE implementation also uses "RP global measurement buffer" to record the original ECU-generated value in addition to the RP value. **Tags:** atp.EnumerationValue=3

Table 10.20: RptPreparationEnum

Class	**RptExecutableEntityProperties**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	Describes the code preparation for rapid prototyping at ExecutableEntity invocation.			
Base	*ARObject*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
maxRptEventId	PositiveInteger	1	attr	Highest RPT event id useable for RTE generated service points. This attribute is relevant, if dedicated id range shall be applied to the ExecutableEntitys of a software component or specific ExecutableEntitys.
minRptEventId	PositiveInteger	1	attr	Lowest RPT event id useable for RTE generated service points. This attribute is relevant, if dedicated id range shall be applied to the ExecutableEntitys of a software component or specific ExecutableEntitys.
rptExecutionControl	RptExecutionControlEnum	1	attr	This attribute specifies the rapid prototyping control of the executable
rptServicePoint	RptServicePointEnum	1	attr	Enables generation of service points by the RTE generator.

Table 10.21: RptExecutableEntityProperties

Enumeration	**RptExecutionControlEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines rapid prototyping preparation of an ExecutableEntity.
Literal	*Description*
conditional	The ExecutableEntity is only executed when the rapid prototyping disable flag is NOT set. **Tags:** atp.EnumerationValue=0
none	The ExecutableEntity is executed without specific rapid prototyping condition. **Tags:** atp.EnumerationValue=1

Table 10.22: RptExecutionControlEnum

Enumeration	**RptServicePointEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario
Note	Specifies whether the invocation of ExecutableEntitys due to activation of specific RteEvents/Bsw Events requires the insertion of Service Points.
Literal	**Description**
enabled	Enables generation of service points by the RTE generator. **Tags:** atp.EnumerationValue=0
none	No Service Points are requested. **Tags:** atp.EnumerationValue=1

Table 10.23: RptServicePointEnum

10.7.2 Differentiation of execution contexts

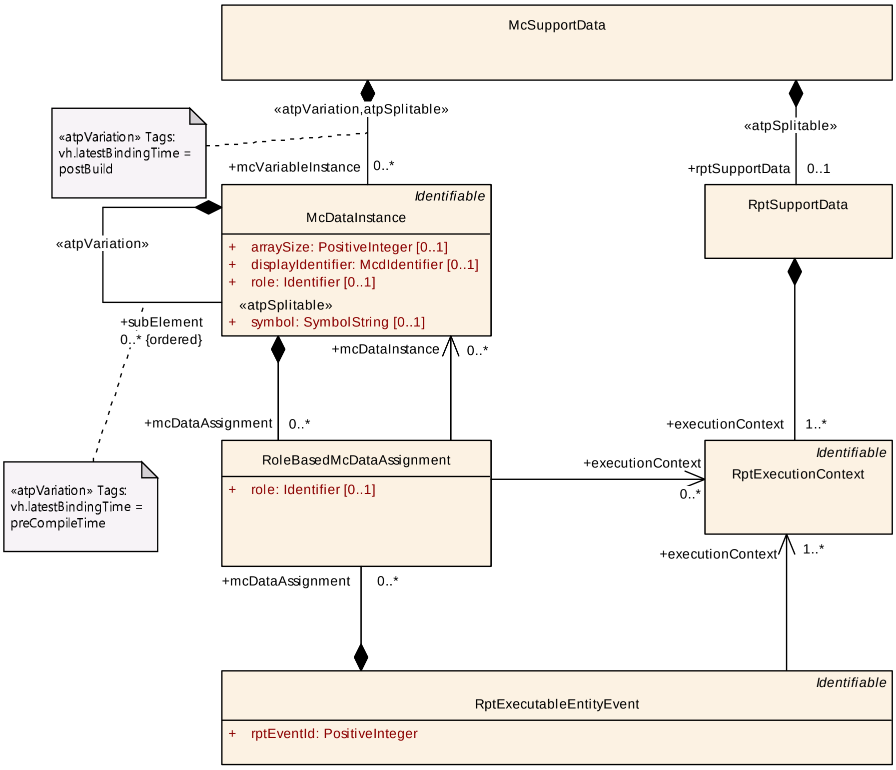

Figure 10.9: Meta-model for RptExecutionContext

[TPS_BSWMDT_04163] RptExecutionContext represents a common environment for a set of RptExecutableEntitys or McDataInstances ⌈ The RptExecutionContext represents a common environment for a set of RptExecutableEntitys or McDataInstances. This common environment is qualified by the identical OSTask context and a identical set of implicit communication buffers. ⌋ *(RS_BSWMD_00065)*

Class	**RptExecutionContext**			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Defines an environment for the execution of ExecutableEntites which is qualified by • OSTask • communication buffer usage			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 10.24: RptExecutionContext

With the means of RptExecutionContexts its possible to denote the temporary validity of McDataInstances describing implicit communication buffers. This is important if the identical implicit communication buffer is reused during a sequence of RunnableEntitys. In this case the McDataInstances describing implicit communication buffers holds the value of different global buffers at different point of times. For example the same OSTask can be split into several sub-sequences where the usage of the implicit communication buffers changes between the sub-sequences. This is the case when the content of the implicit buffer (previously used by a RunnableEntity is written back to the global buffer and after wards fill by the value of an other global buffer in order to be used by a successor RunnableEntity. Please note that the validity of RptExecutionContexts can even overlap (with respect to execution time) since not necessarily the whole implicit communication buffers set used for sub-sequence in a OSTask changes at such a point.

[TPS_BSWMDT_04164] Description of implicit communication buffers ⌈ The Mc-DataInstance describing a implicit communication buffers shall reference the Mc-DataInstance describing the global buffer with a RoleBasedMcDataAssignment where the role attribute is set to ImplicitBuffer. ⌋*(RS_BSWMD_00065)*

Enumeration	**RptAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport
Note	Determines the access rights to a data object with respect to rapid prototyping.
Literal	***Description***
enabled	The related data element is accessible by RP tool. **Tags:** atp.EnumerationValue=0
none	The related data element is not accessible by RP tool. **Tags:** atp.EnumerationValue=1
protected	The data element is known to the RP tool however its usage for RP can be restricted. Use case: limitation based on access rights **Tags:** atp.EnumerationValue=2

Table 10.25: RptAccessEnum

Class	RptServicePoint			
Package	M2::AUTOSARTemplates::CommonStructure::MeasurementCalibrationSupport::RptSupport			
Note	Description of a Service Point implemented for rapid prototyping.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
serviceId	PositiveInteger	1	attr	Unique ID (Range: 0 ... 65535) representing the service point.
symbol	CIdentifier	1	attr	Complete symbol of the function implementing the service point. This symbol is used for post-build hooking purposes.

Table 10.26: RptServicePoint

The following examples illustrate the usage of the McDataInstances and the RoleBasedMcDataAssignments with the role attribute values according [TPS_BSWMDT_04159] to describe the different locations in memory with their relationships and specific meaning for an rapid prototyping tooling.

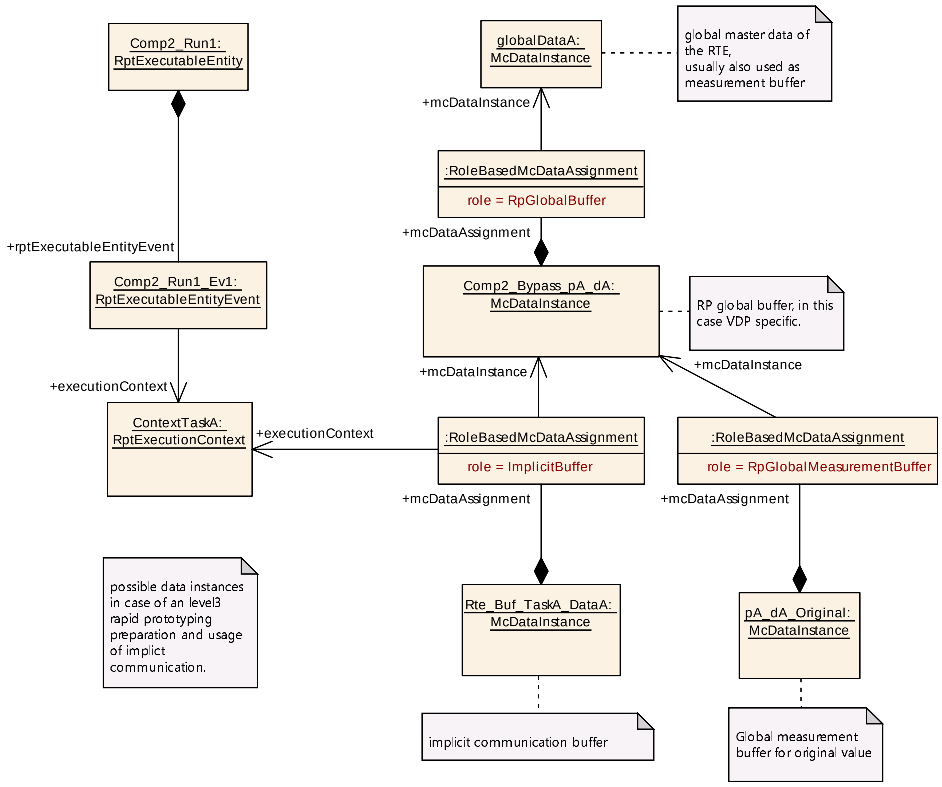

Figure 10.10: Example about Level3 RPT support implementation

Figure 10.10 shows the description of the rapid prototyping support created for the RunnableEntity "'Comp2_Run1"' which had original a dataReadAccess and a dataWriteAccess to dataElement "'dA"' in PRPortPrototype "'pA"'. The requested rapid prototype support was rptLevel3. For the communication of the data value to other components the RTE implements the variable "'globalDataA"' and describes it as McDataInstance. Typically this is also the normal buffer used for measurement. The RunnableEntity is described by RptExecutableEntity

Comp2_Run1 and this references the McDataInstance "'globalDataA"' in the roles rptRead and rptWrite to document the dataReadAccess and dataWriteAccess of the original RunnableEntity.

The access for the rapid prototype tooling is provided by the RP global buffer "'Comp2_Bypass_pA_dA"' which his as well described as McDataInstance referencing the McDataInstance "'globalDataA"' with the RoleBasedMcDataAssignment.role = RptGlobalBuffer.

In this example the RTE uses distinct implicit communication buffers and the according buffer is described as well by an McDataInstance "'Rte_Buf_TaskA_DataA"' with the RoleBasedMcDataAssignment.role = ImplicitBuffer to indicate that this buffer the RP global buffer. For the rptLevel3 support it's required that the RTE provides an additional location in memory, where the original value produced by the RunnableEntity can be accessed. This RP global measurement buffer is described by a McDataInstance pA_dA_Original and linked by RoleBasedMcDataAssignment.role = RpGlobalMeasurementBuffer to the RP global buffer.

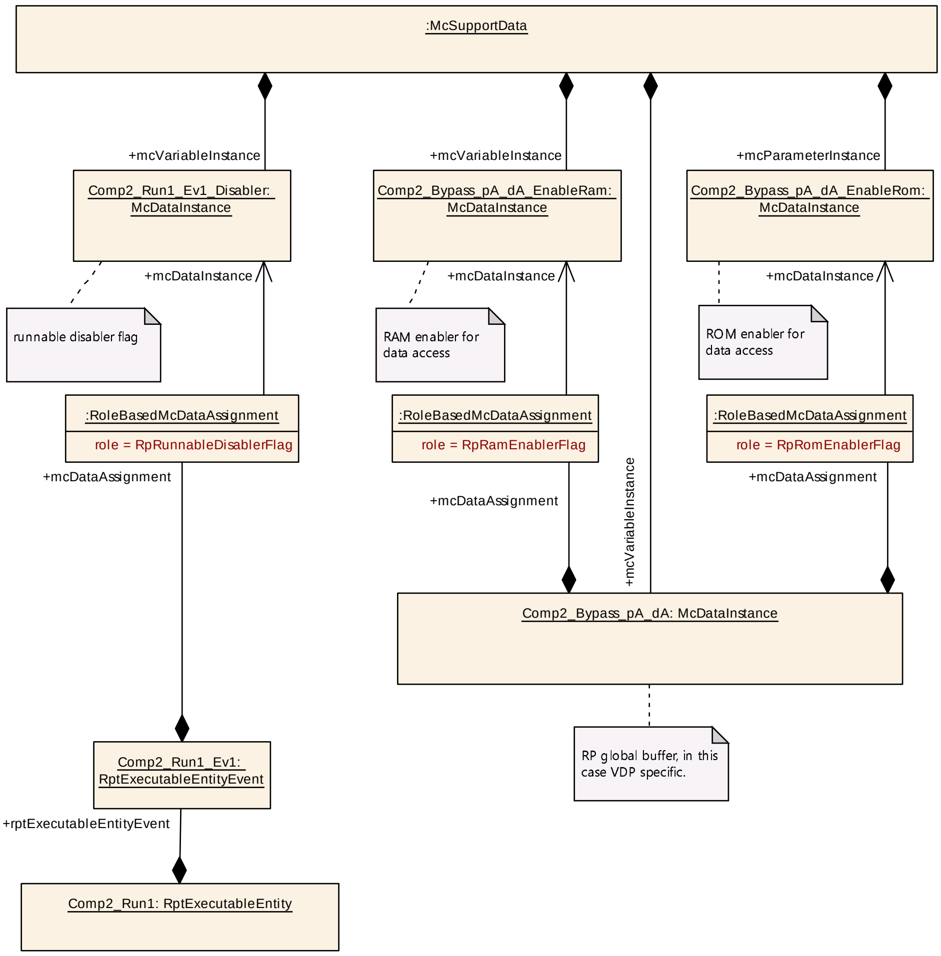

Figure 10.11: Example about Level3 enabler

Figure 10.11 shows the according enabler flags required for the `rptLevel3` rapid prototyping support. Thereby the the `McDataInstance` describing the `RP global buffer` is referencing the

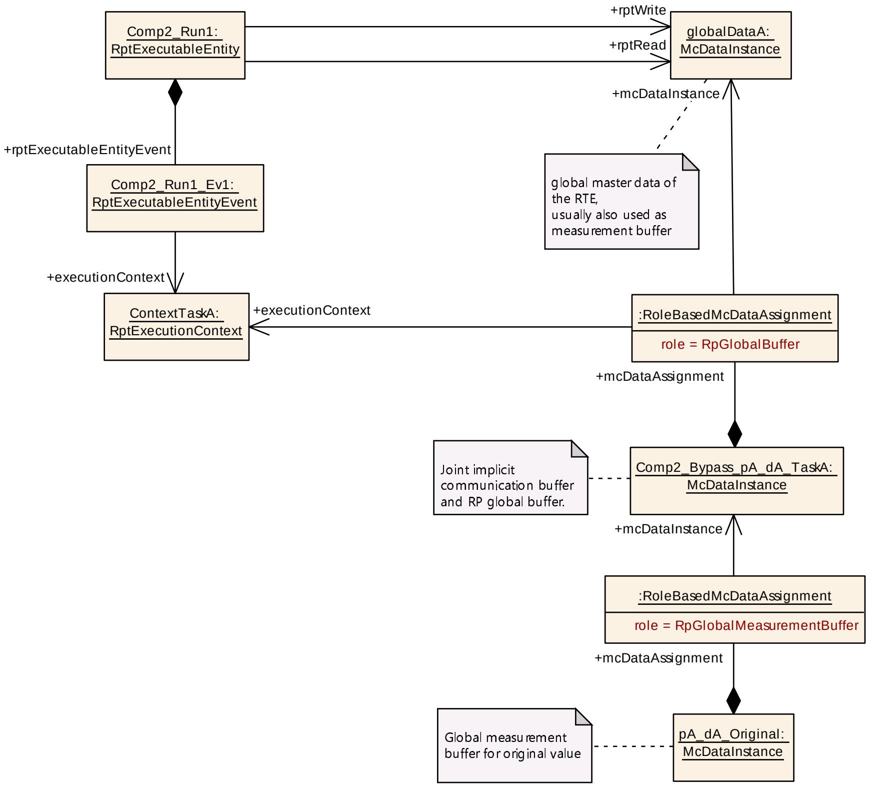

Figure 10.12: Example about optimized RPT support implementation

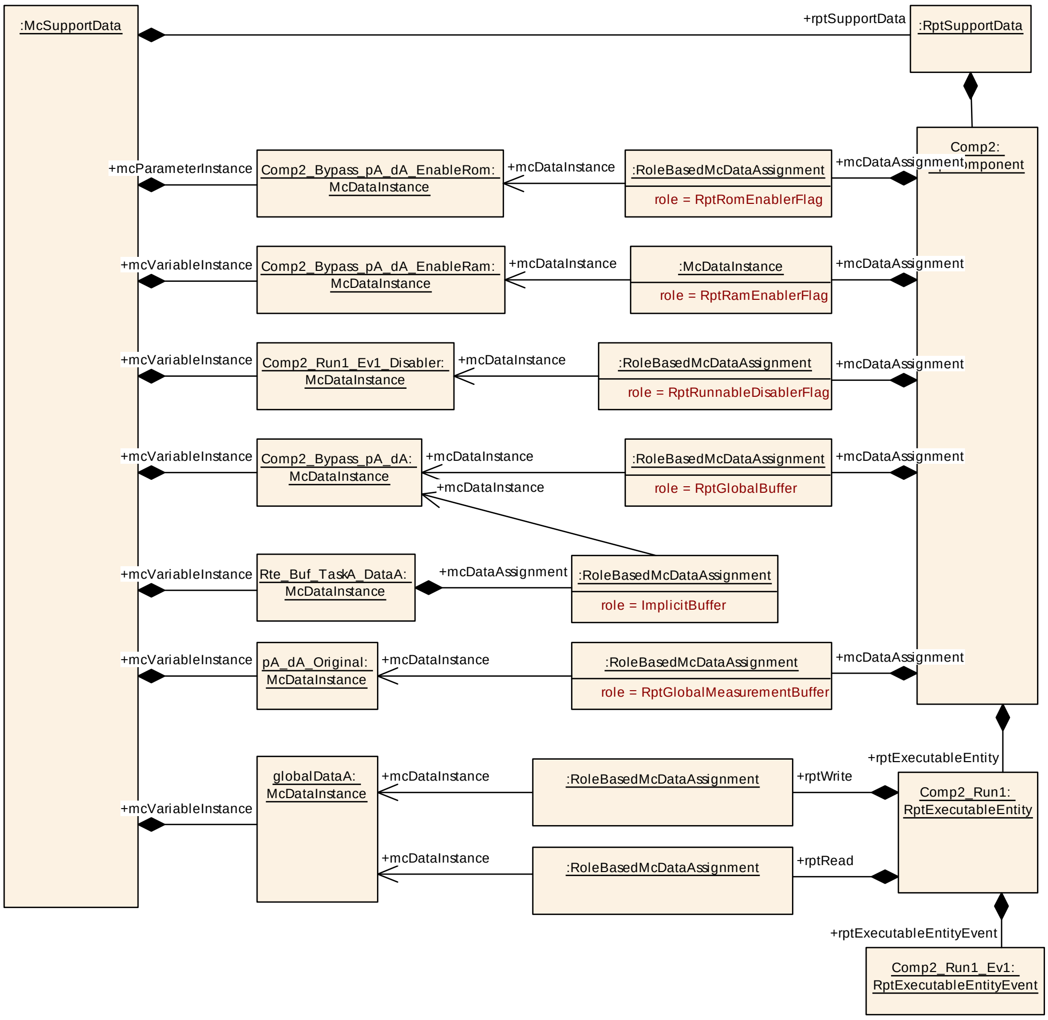

Figure 10.13: Example about RptComponent usage

11 BSW Variant Handling

The BSWMDT includes variation points which allow to describe a set of variants of a BSW module or cluster by a single XML artifact (for general information on variant handling in AUTOSAR see [1]).

Variation points are provided at all three levels of the template.

11.1 BSW Interface Variation Points

[TPS_BSWMDT_04063] BSW Interface Variation Points ⌈The variation points in the scope of BswModuleDescription with latestBindingTime = preCompileTime allow to declare variable sets of optional documentation, communication interfaces, dependencies, triggers and mode groups as part of one BSW module description, see figures 11.1 and 11.2. Further variation points in this hierarchy with can be bound at compile-time are not allowed in order to keep the meta-model and the resulting M1 models maintainable. ⌋*(RS_BSWMD_00049)*

If for example one wants to specify two variants of a module which handles a certain C-function argument either as a 16 bit or as a 32 bit type respectively and this needs to be bound at compile-time, this is possible by variation of the associations to BswModuleEntry, but is is not possible to declare a single BswModuleEntry with two compile-time variants just for a single argument.

However, at an earlier stage of development it is possible to include this kind of additional variability into Blueprints of BswModuleEntry-s (see [9]). This is especially useful if a BSWMD is used to represent an SWS of the AUTOSAR standard, since interfaces are specified here on the level of Blueprints, i.e. they still contain optional or alternative function arguments:

[TPS_BSWMDT_04090] Variation Points for BswModuleEntry arguments ⌈It is possible to declare a BswModuleEntry.argument as a variation point but its binding time must not be later than blueprintDerivationTime, see figure 11.1 ⌋*(RS_BSWMD_00049)*

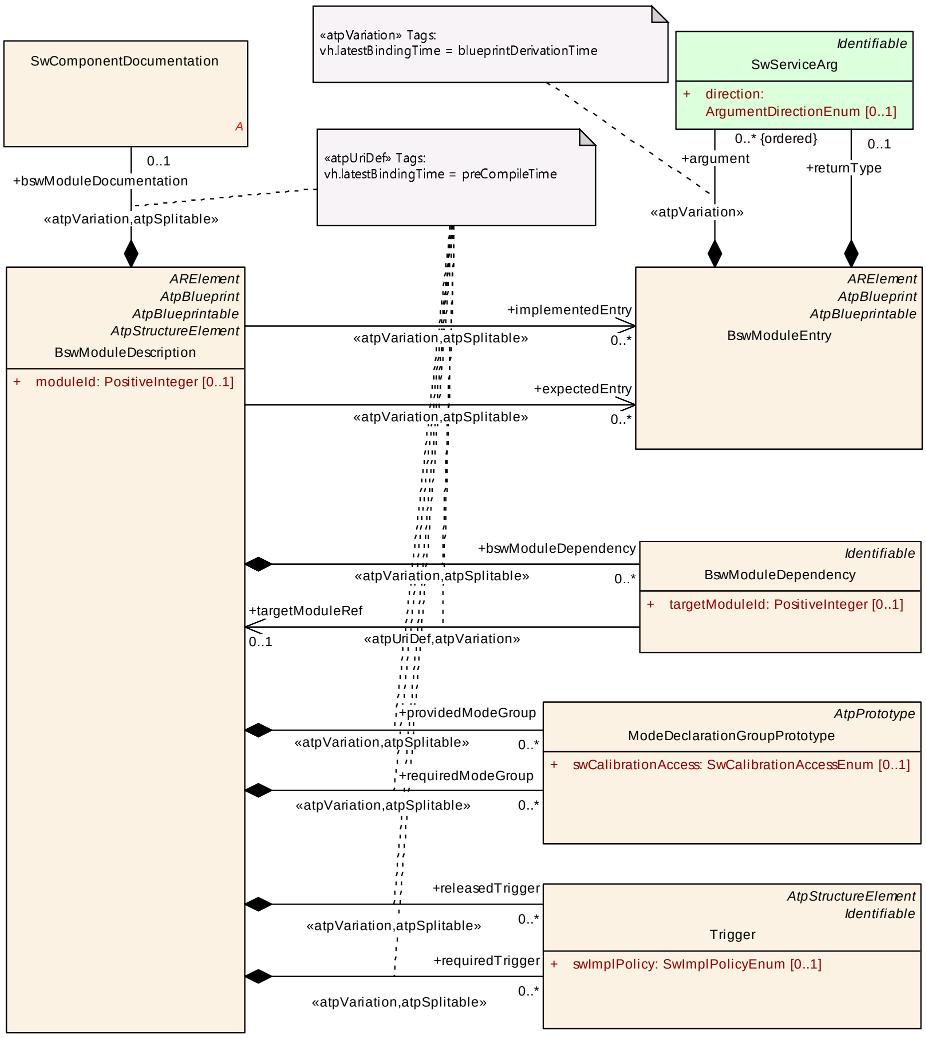

Figure 11.1: Variation points under `BswModuleDescription`, Part 1

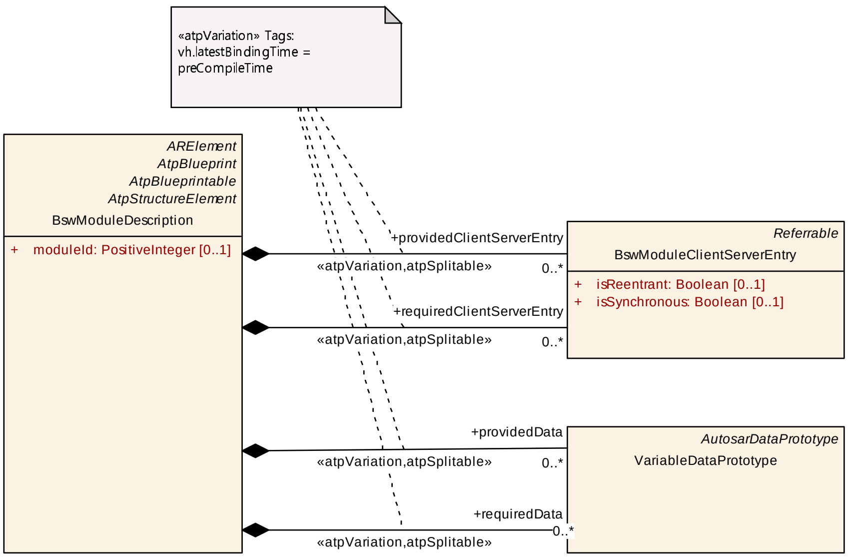

Figure 11.2: Variation points under `BswModuleDescription`, Part 2

One use case is to maintain a specification which includes optional or alternative interfaces/dependencies for a module at design time. For example, as already mentioned above, it is possible to provide one BSWMD (as an XML artifact) which describes the AUTOSAR standard for the C-interfaces of a standardized AUTOSAR module including specification of the optional parts as variants. These variants will be selected in the BSWMD of a module which is actually implemented against such a specification.

Another use case is to deliver a BSWMD still including some variation points to the integrator, which means in this case the variants will be selected by the integrator. Since most of the variation points described in this section influence the executable code, this use case requires that the relevant parts of the code are regenerated and/or recompiled at integration time. Due to this reason, the latest possible binding time of most variation points described here is set to to `preCompileTime`.

The second use case may require that the actual selection of a variation points will constraint the ECU configuration parameter values of the module (for example, if a configuration parameter configures the existence/non-existence of a callback function this will be constrained by deselecting a variant of the attributes `expectedEntry` or `implementedEntry`. This could simply be done by delivering sets of preconfigured parameter values which obey to the same variant conditions as the corresponding elements referred/aggregated by `BswModuleDescription`. However, a more elegant solution will be to derive the parameter definition in question "automatically" (.e. via its definition) from the condition which is implicitly defined in the M1 model with each variant selection (see [1]).

11.2 BSW Behavior Variation Points

[TPS_BSWMDT_04064] BSW Behavior Variation Points ⌈In a similar way, variation points underneath `BswInternalBehavior` allow to declare variants in the aggregation of `BswModuleEntity`-s, `BswEvent`s and further elements, see figure 11.3.

Likewise, several references and aggregations owned by `BswModuleEntity` are variation points, see figure 11.4.

The figure 11.3 also shows the variation point in the aggregation of local data for calibration and measurement and of `ExclusiveArea` by the base class `InternalBehavior`. ⌋*(RS_BSWMD_00049)*

The use cases are similar to the ones described above (chapter 11.1). For the same reasons, the latest possible binding time for these variation points is defined as `PreCompileTime`.

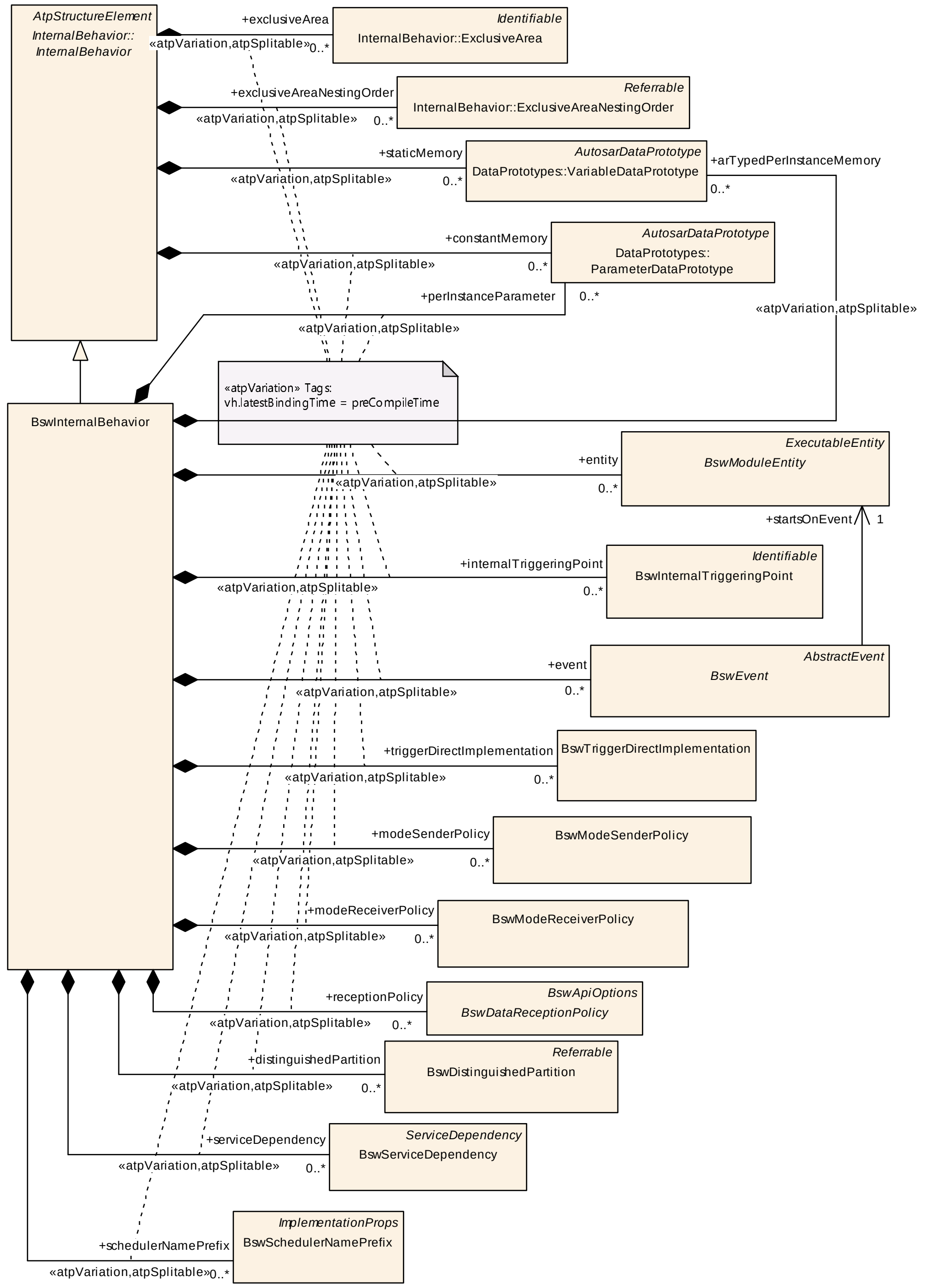

Figure 11.3: Variation points under BswInternalBehavior

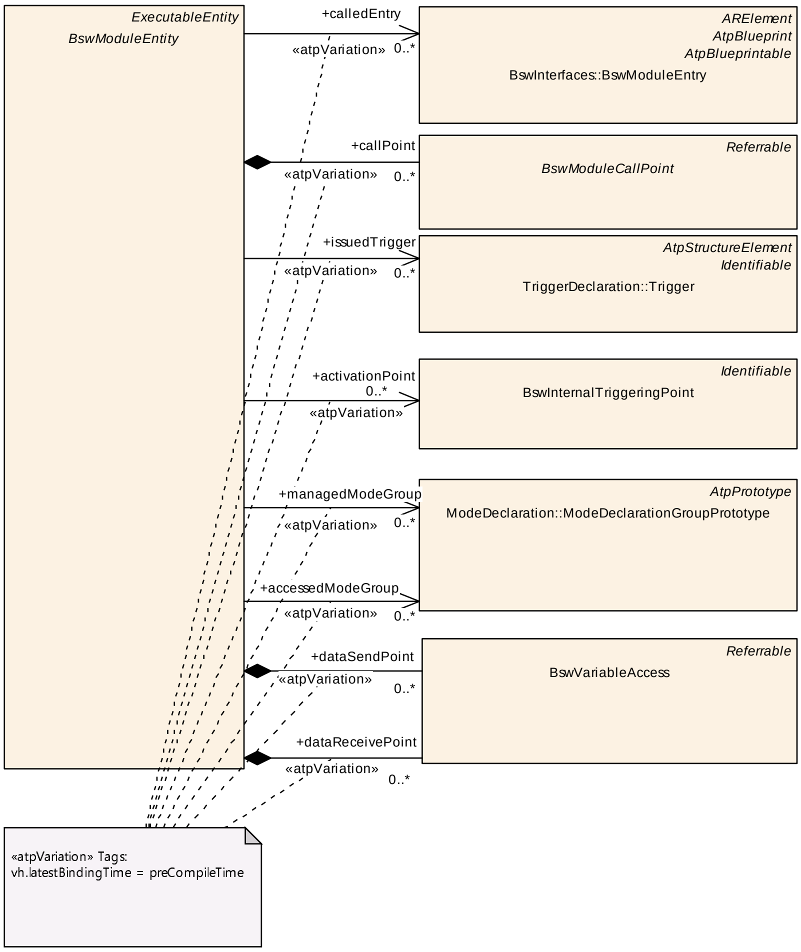

Figure 11.4: Variation points under BswModuleEntity

11.3 BSW Implementation Variation Points

[TPS_BSWMDT_04065] BSW Implementation Variation Points ⌈Figure 11.5 shows the only variation point below meta-class BswImplementation which is the aggregation debugInfo. Also for this variation point the latest possible binding time is preCompileTime.

In addition, there are several variation points in the base class Implementation and the elements aggregated from there. These are visible in the respective figures of chapter 8. They are usable for BSW and SWC descriptions as well. They all support the use case, that a module or component is delivered as source code leading to several implementation variants.

Furthermore, if an Implementation contains McSupportData, these can also have variation points, as explained in chapter 10.1. ⌋*(RS_BSWMD_00049)*

The associations to vendorSpecificModuleDef and preconfiguredConfiguration are not considered as variation points, since they correspond to artifacts which are supposed to be fixed at the time a module is delivered. Also recommendedConfiguration corresponds to a fixed set of artifacts at delivery time.

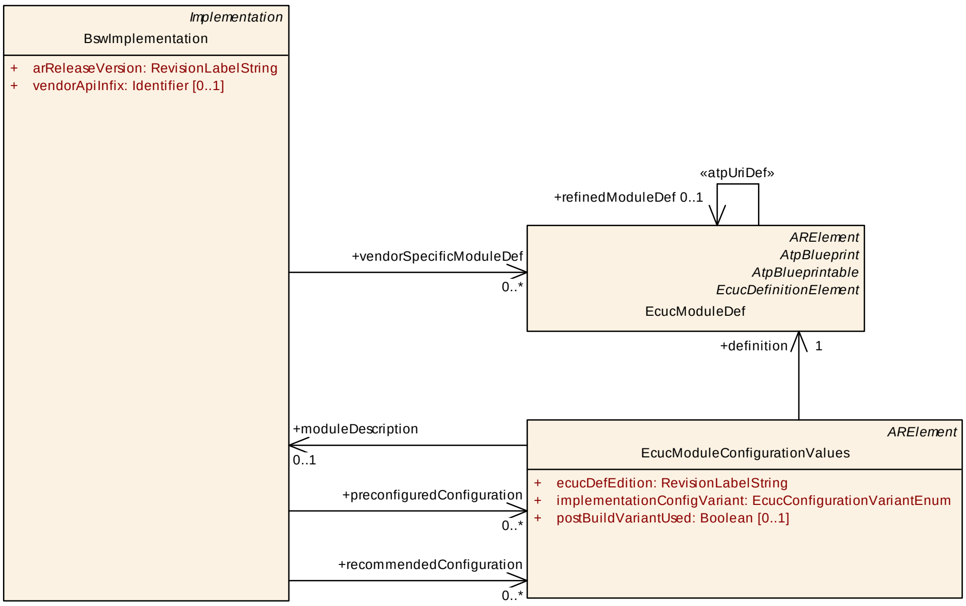

Figure 11.5: Variation points under BswImplementation

12 Implementation Conformance Statement

12.1 Background

This chapter describes, which elements of the BSWMDT have to be used to specify the delivery of a BSW module for the purpose of an AUTOSAR conformance test. For the background on conformance tests refer to [23].

The use case assumed in this chapter is as follows:

- The test is done for an ICC3 module.
- The code to be tested is delivered as fully configured object code. Note that this could be more than one file, e.g. core code + separately compiled configuration data.
- The tester has no means to change the configuration. This implies that, if AUTOSAR has specified tests for several different sets of configuration values, corresponding sets of object code files must be delivered.
- In addition to the object code, header files and ARXML-descriptions are delivered as far as needed to declare the conformity and to set up the test.

Especially, the BSWMD (and the attached configuration parameter definitions and configuration values) shall contain the Implementation Conformance Statement (ICS). The purpose of the ICS is to declare the extent to which the module covers the relevant AUTOSAR specification. See also [5] for the overall definition of the ICS.

The ARXML model elements that form an Implementation Conformance Statement shall be aggregated under a `ARPackage` with the category **ICS**. It is not required (but possible) that sub-packages below this package also have the category **ICS**, but they may not have the category **BLUEPRINT**. See [1] for formal constraints on the package categories.

Note that in the current AUTOSAR release, the standardized specification elements (i.e. the content of an SWS) for an ICC3 module are published by AUTOASAR not in the format of ARXML, but as pdf-Document. Therefore, the mechanism how to trace between a given BSWMD and the corresponding SWS is currently not standardized.

12.2 Interface Level

[TPS_BSWMDT_04066] Relevant elements for ICS on Interface level ⌈On the Interface level of the BSWMDT, the following elements are relevant for the Conformance Test:

- `BswModuleDescription.moduleId`

  This identifies the ICC3 module and its specification.

- BswModuleDescription.implementedEntry
  BswModuleDescription.expectedEntry

  These elements are required to describe the name and signature of standardized provided functions resp. outgoing callbacks which are actually present in the tested code (mandatory as well as optional ones). Vendor specific functions/callbacks shall not be included.

  Note: If the names of callbacks are configurable, the respective configuration values must also be delivered.

- BswModuleDescription.bswModuleDependency.targetModuleId

  These elements are required as far as they describe the dependency on standardized elements of other standardized ICC3 modules (identified by the targetModuleId).

  Note: Conformance test cases on standardized functions must be executable without any dependency on non-standardized functions/modules. Therefore the test setup must be possible by knowing only the dependencies of the module on other standardized elements.

- BswModuleEntry.shortName
  BswModuleEntry. - all attributes of this meta-class
  BswModuleEntry.argument.swDataDefProps
  BswModuleEntry.returnType.swDataDefProps

  Here, BswModuleEntry stands for the root element for a function signature referred by the function declarations - e.g. implementedEntry - listed above. The major amount of the aggregated or referred elements below SwDataDefProps are not required for the ICS. Only those parts of SwDataDefProps are needed, which uniquely specify the C data type of the arguments and the returnType. Please refer to chapter "Implementation Data Type" of [6] for example how to describe C data types in this way.

⌋*(RS_BSWMD_00039, RS_BSWMD_00040, RS_BSWMD_00041, RS_BSWMD_00042)*

The rest of the elements on the Interface level of the BSWMDT are not relevant for the conformance test. They are listed here for completeness:

- BswModuleDescription.providedModeGroup
  BswModuleDescription.requiredModeGroup
  BswModuleDescription.releasedTrigger
  BswModuleDescription.requiredTrigger

  These elements are used to support the delegation of mode switching or triggering to the BSW Scheduler. These mechanisms are currently not referred by any standardized ICC3 specification; they are mainly targeted at Complex Drivers or IO HW Abstraction. Therefore is its currently not required to use these elements within the ICS.

12.3 Internal Behavior Level

[TPS_BSWMDT_04067] No relevant elements for ICS on Internal Behavior level ⌈On the Internal Behavior level of the BSWMDT, there are no elements relevant for the conformance test ⌋*(RS_BSWMD_00030)* as the following overview shows:

- BswInternalBehavior.entity
  BswInternalBehavior.event
  BswInternalBehavior.triggeringPoint
  BswInternalBehavior.bswTriggerDirectImplementation
  BswInternalBehavior.modeSenderPolicy

  The main use case of these elements is to provide input for configuring the Basic Software Scheduler (part of the RTE). In addition, they provide information for timing or call-chain analysis. These elements are neither relevant for the ICS nor otherwise needed for the conformance test, since the conformance test does not need this information to call single C-functions.

- BswInternalBehavior.constantMemory
  BswInternalBehavior.staticMemory

  These elements are used to declare data that are local to the module, main use case is for measurement and calibration and for data needed to set up the configuration of the NVRAM Manager. They need not to be declared for the conformance test.

- BswInternalBehavior.serviceDependency

  This element (and further elements aggregated by it) are used to declare requirements on the configuration of other standardized service modules like NVRAM Manager or DEM. It is not considered as relevant for the conformance test, since the conformance test environment does not have to simulate the behavior of these service modules in such detail, that is needs to be configured in response to ServiceNeeds (see chapter 13).

12.4 Implementation Level

[TPS_BSWMDT_04068] Relevant elements for ICS on Implementation level ⌈On the Implementation level of the BSWMDT, a couple of elements are relevant for the Conformance Test. Though not part of the ICS in a strict sense, they are required for administrative reasons and to set up the test environment. The following Elements are relevant on the implementation level of the BSWMDT:

- BswImplementation.programmingLanguage
  BswImplementation.swVersion
  BswImplementation.arRelaseVersion
  BswImplementation.vendorId
  BswImplementation.vendorApiInfix

BswImplementation.codeDescriptor
BswImplementation.compiler
BswImplementation.linker

Defining the programming language, version information, identifiers to expand the API names (in case of multiple instantiation), code files attached to the delivery, compiler and linker settings. For details see chapters 7 and 8.

- BswImplementation.hwElement

  This may be added in case there is a formal description of hardware dependency, especially for MCAL modules. However, the details and the amount of this information are not standardized.

⌋*(RS_BSWMD_00010, RS_BSWMD_00025, RS_BSWMD_00026)*

The rest of the elements on the Implementation level of the BSWMDT are not relevant for the conformance test. They are listed here for completeness:

- BswImplementation.usedCodeGenerator
  BswImplementation.requiredArtifact
  BswImplementation.requiredGeneratorTool
  BswImplementation.generatedArtifact

  Since only object code is delivered, information on code generation is not needed. Also as far as the test cases is concerned, there should be no dependencies on other artifacts except on other ICC3 modules, but the latter are already defined via bswModuleDependency on the interface level.

- BswImplementation.resourceConsumption
  BswImplementation.mcSupport
  BswImplementation.debugInfo

  Information about resource consumption, measurement, calibration and data for debugging is not relevant for the conformance test.

- BswImplementation.swcBswMapping

  This is not relevant to test the conformity of the "naked" ICC3 module. The additional specification of Ports on top of a BSW module does not change its code. They are relevant to generate the RTE but not to set up the test environment

12.5 Configuration and Variants

[TPS_BSWMDT_04069] Configuration in ICS ⌈Configuration parameters and configuration values also form part of the ICS. They shall be attached to the BSWMD as follows:

- BswImplementation.vendorSpecificModuleDef

This is needed for two reasons:

1. It must be possible to run the ICC3 test cases without knowledge of non-standardized vendor specific configuration parameters. However, copies of the supported standardized parameter definitions is also part of the vendorSpecificModuleDef (as usual) and is needed here, because the preconfiguredConfiguration references them.
2. Vendor specific parameter definitions which are "derived" from standardized ones have to be included for static test (i.e. whether they are derived according to the standard). Parameters should also declare the value range that is supported by the given release of the module - even if only some of the values are actually pre-configured and tested (see below).

However, it is not required to include completely new vendor specific parameter definitions (no "origin" in the standardized configuration parameters), because in this case there is nothing to be tested for conformity.

- BswImplementation.preconfiguredConfiguration

  Since each delivered implementation is a fully configured object code, for each such implementation a complete set of pre-configured values (i.e. values for all of the parameters given in the above vendorSpecificModuleDef) must be attached. Of course, if more than one configuration set shall be tested, there will be several such preconfiguredConfigurations (and likewise several BswImplementations and object files) but only one vendorSpecificModuleDef (the one belonging to the release of this module).

⌋*(RS_BSWMD_00024, RS_BSWMD_00027, RS_BSWMD_00035)*

The following is obviously not relevant for the conformance test, because the tester cannot change the configuration:

- BswImplementation.recommendedConfiguration

[TPS_BSWMDT_04070] No variants in ICS ⌈A BSWMD that describes an actual product can contain variation points (see chapter 11). But since the conformance tester gets fully configured object code, this means also, that the ICS-version of a BSWMD must be free of any variation points, because the tester has no means to resolve the variants.

If several variants of such a module shall be tested for conformance, for each variant a separate extract of the BSWMD (representing the ICS) plus object code must be delivered to the tester⌋*(RS_BSWMD_00049)*.

13 BSW Service Needs

13.1 Overview

The mechanism of so-called Service Dependencies and Service Needs is used by Software Components above the RTE to express their needs on the configuration of AUTOSAR Services. The same mechanism can be used also in the basic software in order to have a uniform approach, if an AUTOSAR Service has to be configured per ECU for the needs of both BSW and SWCs.

Figure 13.1 shows the various meta-classes which can be used on the behavior level of BSW modules and SWCs in order to express these dependencies.

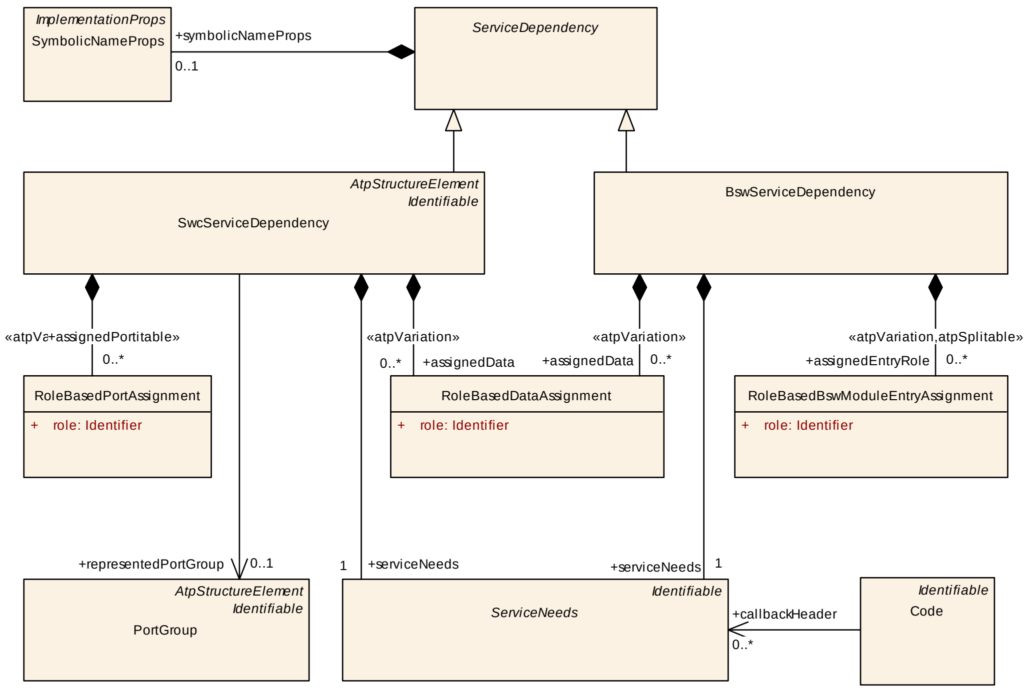

Figure 13.1: Concept of ServiceDependency for BSW and SWC

[TPS_BSWMDT_04029] Usage of BswServiceDependency ⌈In figure 13.2 the set of BswServiceDependency-s represents the requirements of the module or cluster on the configuration of AUTOSAR Services like NVRAM Manager or Watchdog Manager. These requirements include not only the specific ServiceNeeds attributes, but can optionally include references to local data (for example to declare RAM mirror or ROM default data for the NVRAM Manager) or to BswModuleEntry-s (for example to declare which expected callbacks belong to a specific NvM block).⌋ *(RS_BSWMD_00045)*

Further explanation could be found in the class tables below.

[TPS_BSWMDT_04127] Callback header declarations ⌈When a service configures callback functions the header files providing the callback function declarations needs to be identified. The reference callbackHeader describes in which header files the function declarations of callback functions are provided for the AUTOSAR service implementing the ServiceNeeds.⌋*(RS_BSWMD_00045)*

[constr_4089] Association callbackHeader is only applicable for BSW modules ⌈ The association callbackHeader is only supported for codeDescriptors of BswImplementation and only permitted to reference ServiceNeeds owned by BswServiceDependency. ⌋*()*

[constr_4090] The callbackHeader reference has to be consistent with behavior reference ⌈ The reference callbackHeader is only allowed to reference ServiceNeeds in the context of the BswServiceDependency which in turn is referenced by the BswImplementation behavior of the BswImplementation owning the codeDescriptor. ⌋*()*

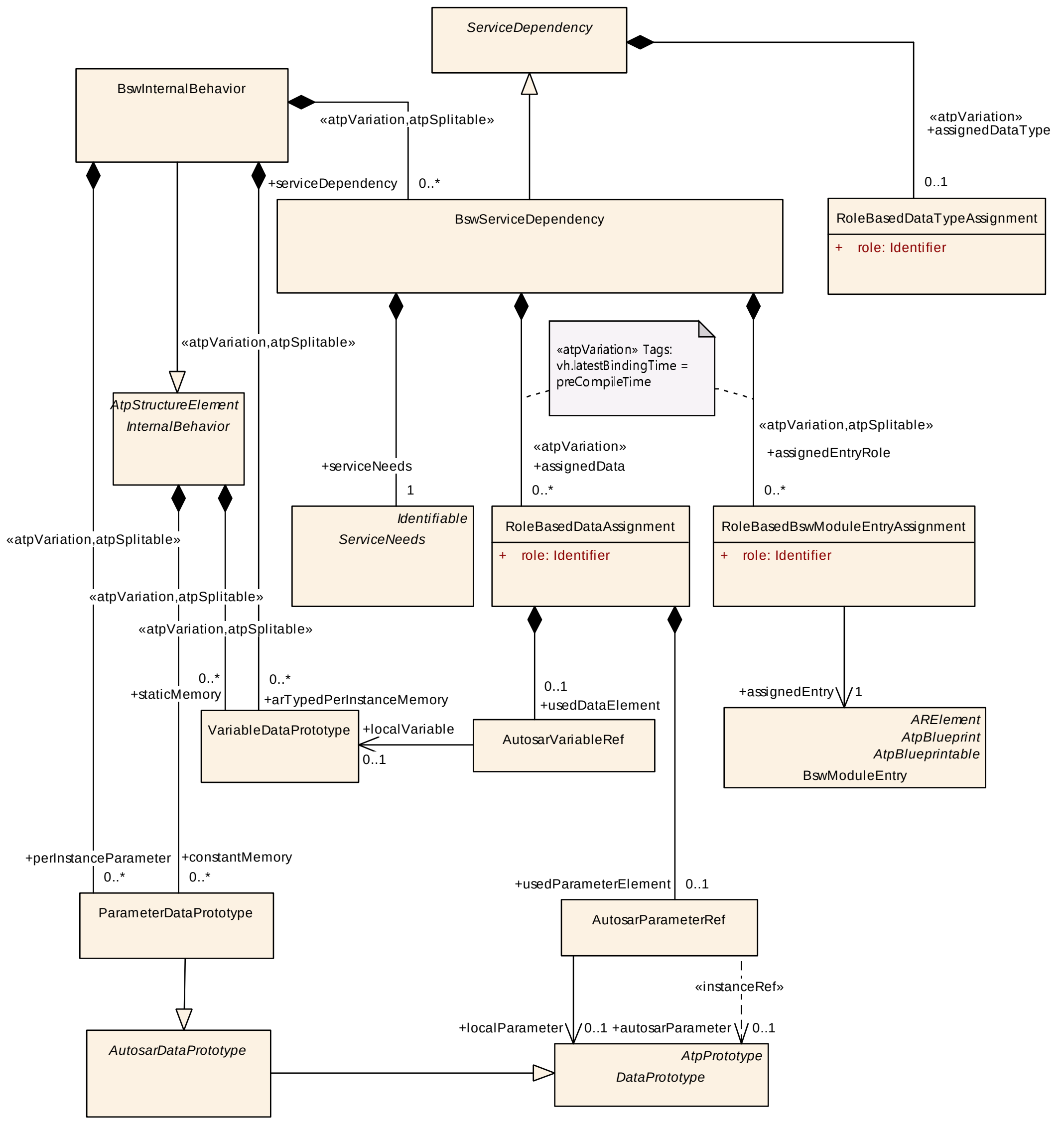

Figure 13.2: BswServiceDependency attached to a BswInternalBehavior

Class	***ServiceDependency*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Collects all dependencies of a software module or component on an AUTOSAR Service related to a specific item (e.g. an NVRAM Block, a diagnostic event etc.). It defines the quality of service (Service Needs) of this item as well as (optionally) references to additional elements. This information is required for tools in order to generate the related basic software configuration and ServiceSwComponentTypes.			
Base	*ARObject*			
Subclasses	BswServiceDependency, SwcServiceDependency			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
assignedDataType	RoleBasedDataTypeAssignment	0..1	aggr	This is the role of the assignment data type in the given context. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolicNameProps	SymbolicNameProps	0..1	aggr	This attribute can be taken to contribute to the creation of symbolic name values.

Table 13.1: ServiceDependency

Class	**BswServiceDependency**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Specialization of ServiceDependency in the context of an BswInternalBehavior. It allows to associate BswModuleEntries and data defined for a BSW module or cluster to a given ServiceNeeds element.			
Base	*ARObject*, *ServiceDependency*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
assignedData	RoleBasedDataAssignment	*	aggr	Defines the role of an associated data object (owned by this module or cluster) in the context of the ServiceNeeds element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
assignedEntryRole	RoleBasedBswModuleEntryAssignment	*	aggr	Defines the role of an associated BswModuleEntry in the context of the ServiceNeeds element. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=assignedEntryRole, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
ident	BswServiceDependencyIdent	0..1	aggr	This adds the ability to become referrable to BswServiceDependency. **Tags:** atp.Status=shallBecomeMandatory xml.sequenceOffset=-100
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table 13.2: BswServiceDependency

Class	**RoleBasedBswModuleEntryAssignment**
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior
Note	This class specifies an assignment of a role to a particular BswModuleEntry (usually a configurable callback). With this assignment, the role of the callback is mapped to a specific ServiceNeeds element, so that a tool is able to create appropriate configuration values for the module that implements the AUTOSAR Service.

▽

△

Class	**RoleBasedBswModuleEntryAssignment**			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
assignedEntry	BswModuleEntry	1	ref	The assigned entry. It should be an implementedEntry or expectedEntry of the module or cluster that requires the ServiceNeeds.
role	Identifier	1	attr	This is the role of the assigned BswModuleEntry in the given context. The attribute is required (for example) because different kind of callbacks may be associated with the same ServiceNeeds (e.g. end-notification vs. error-notification). The value must be the role name of a configurable function call (usually a callback) as standardized in the Software Specification of the related AUTOSAR Service.

Table 13.3: RoleBasedBswModuleEntryAssignment

Class	RoleBasedDataAssignment			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class specifies an assignment of a role to a particular data object in the SwcInternalBehavior of a software component (or in the BswModuleBehavior of a module or cluster) in the context of an AUTOSAR Service. With this assignment, the role of the data can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct access.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
role	Identifier	1	attr	This is the role of the assigned data in the given context, for example for an NVRAM Block it is used to distinguish between an mirror block and a ROM default block. Possible values need to be specified on M1 level. This also is intended to support the so called "Signal based Approach" of the DCM. In this use case the name of the involved data element is required. This name shall be taken from the DataElement referenced by the property usedDataElement. The following values are standardized: • **ramBlock** indicates data to be used as a mirror for an NVRAM Block. • **defaultValue** indicates constant data to be used as default in the context of this ServiceNeeds, e.g. for an NVRAM Block. • **signalBasedDiagnostics** indicates the Role BasedDataAssignment shall be used for signal based diagnostics.
usedData Element	AutosarVariableRef	0..1	aggr	The VariableDataPrototype used in this role, e.g. • Permanent RAM Block of an NVRAM Block which shall belong to the same SwcInternal Behavior or BswInternalBehavior. • In the role signalBasedDiagnostics it has to refer to a VariableDataPrototype in a SenderReceiver Interface or a NvDataInterface.
usedParameter Element	AutosarParameterRef	0..1	aggr	The ParameterDataPrototype used in this role, e.g. • ROM Block of an NVRAM Block. It shall belong to the same SwcInternalBehavior or Bsw Internalbehavior. • In the role signalBasedDiagnostics it has to refer to a ParameterDataPrototype in a Parameter Interface.

▽

△

Class	RoleBasedDataAssignment			
usedPim	PerInstanceMemory	0..1	ref	The (untyped) PerInstanceMemory used in this role (e.g. as a Permanent RAM Block for an NVRAM Block).

Table 13.4: RoleBasedDataAssignment

Class	RoleBasedDataTypeAssignment			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular data type of a software component (or in the BswModuleBehavior of a module or cluster) in the context of an AUTOSAR Service. With this assignment, the role of the data type can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct access.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
role	Identifier	1	attr	This is the role of the associated data type in the given context.
used Implementation DataType	ImplementationData Type	1	ref	This represents the associated ImplementationDataType.

Table 13.5: RoleBasedDataTypeAssignment

Class	***ServiceNeeds*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This expresses the abstract needs that a Software Component or Basic Software Module has on the configuration of an AUTOSAR Service to which it will be connected. "Abstract needs" means that the model abstracts from the Configuration Parameters of the underlying Basic Software.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	BswMgrNeeds, ComMgrUserNeeds, CryptoServiceJobNeeds, CryptoServiceNeeds, *Diagnostic CapabilityElement*, DltUserNeeds, *DoIpServiceNeeds*, EcuStateMgrUserNeeds, ErrorTracerNeeds, FunctionInhibitionAvailabilityNeeds, FunctionInhibitionNeeds, GlobalSupervisionNeeds, HardwareTest Needs, IndicatorStatusNeeds, J1939RmIncomingRequestServiceNeeds, J1939RmOutgoingRequest ServiceNeeds, NvBlockNeeds, SecureOnBoardCommunicationNeeds, SupervisedEntityCheckpoint Needs, SupervisedEntityNeeds, SyncTimeBaseMgrUserNeeds, V2xFacUserNeeds, V2xMUserNeeds, VendorSpecificServiceNeeds			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 13.6: ServiceNeeds

Note that several kinds of data assignments are restricted to be used within an SWC because they need RTE support:

RoleBasedDataTypeAssignment

This denotes the type of a temporary Ram Block and used internal data structure in case of explicit synchronization with NvMReadRamBlockFromNvM and NvMWriteRamBlockToNvM interface respectively. The type information can be used to calculate the NvBlock size and minimum Ram Mirror size.

- `temporaryRamBlock` [0..1]

[constr_4051] RoleBasedDataAssignment in BSW ⌈ When used in the context of BswServiceDependency, the following restriction hold for date references described by RoleBasedDataAssignment:

- Within RoleBasedDataAssignment.usedDataElement, only the reference AutosarVariableRef.localVariable is applicable.
- Within RoleBasedDataAssignment.usedParameterElement, only the reference AutosarParameterRef.localParameter is applicable.
- The reference RoleBasedDataAssignment.usedPim shall not be set.

⌋*()*

[TPS_BSWMDT_04113] Rule for setting RoleBasedBswModuleEntryAssignment.role ⌈ The value of RoleBasedBswModuleEntryAssignment.role cannot arbitrarily set but shall to equal to the shortName of the applicable BswModuleEntry taken from the standardized AUTOSAR BswModuleEntry model (this implies that the category of the ARPackage that owns the BswModuleEntry is set to BLUEPRINT[1] and the top-most ARPackage.shortName is set to AUTOSAR, see also [24]). ⌋ *(RS_BSWMD_00045)*

13.2 Specific Service Needs

The abstract meta-class ServiceNeeds and its more specific child classes are defined in the CommonStructure package of the meta-model. This class hierarchy is shown in the three figures (13.3, 13.4 and 13.5).

The subsequent tables show those specialized ServiceNeeds which are of interest for the basic software.

Note that several detailed meta-classes for diagnostic capabilities (derived from DiagnosticCapabilityElement) and for diagnostic over IP (derived from DoIpServiceNeeds) are not shown here, because they are mainly of interest for application software. For a detailed description of those refer to [6].

Note that the ServiceNeeds describes only the source data of an abstract dependency. How this is actually traced down to the configuration parameters is specified by the configuration parameters of the dependent modules itself. For a description of this mechanism see [TPS_ECUC_02047] under topic "Derived Parameter Definition" in [11]. To get the complete picture, it should be noted that also other templates can define source data for dependencies, for example the configuration of the COM stack depends on information defined via the AUTOSAR System Template.

This information as defined by AUTOSAR for standardized configuration parameters is also called “Upstream Mapping”. The Upstream Mapping relevant for BSWMDT is listed in this document in appendix C.

[1] see [TPS_STDT_00033]

If a BSW module implements an AUTOSAR Service, it is possible that parts of its own ServiceNeeds are in turn influenced by the ServiceNeeds of the SWCs and BSW modules integrated on an ECU. In this case, the ServiceNeeds of that module must be adjusted at ECU integration time before the initial ECU configuration is set up. For example, the NvBlockNeeds of the Diagnostic Event Manager will be determined in response to the number of diagnostic events on an ECU which are given by the DiagnosticEventNeeds of all integrated SWCs and BSW modules. Since parts of the XML-description of AUTOSAR Services (namely the SWC-part) are generated at integration time anyway, the adjustment of ServiceNeeds can be done in the same step.

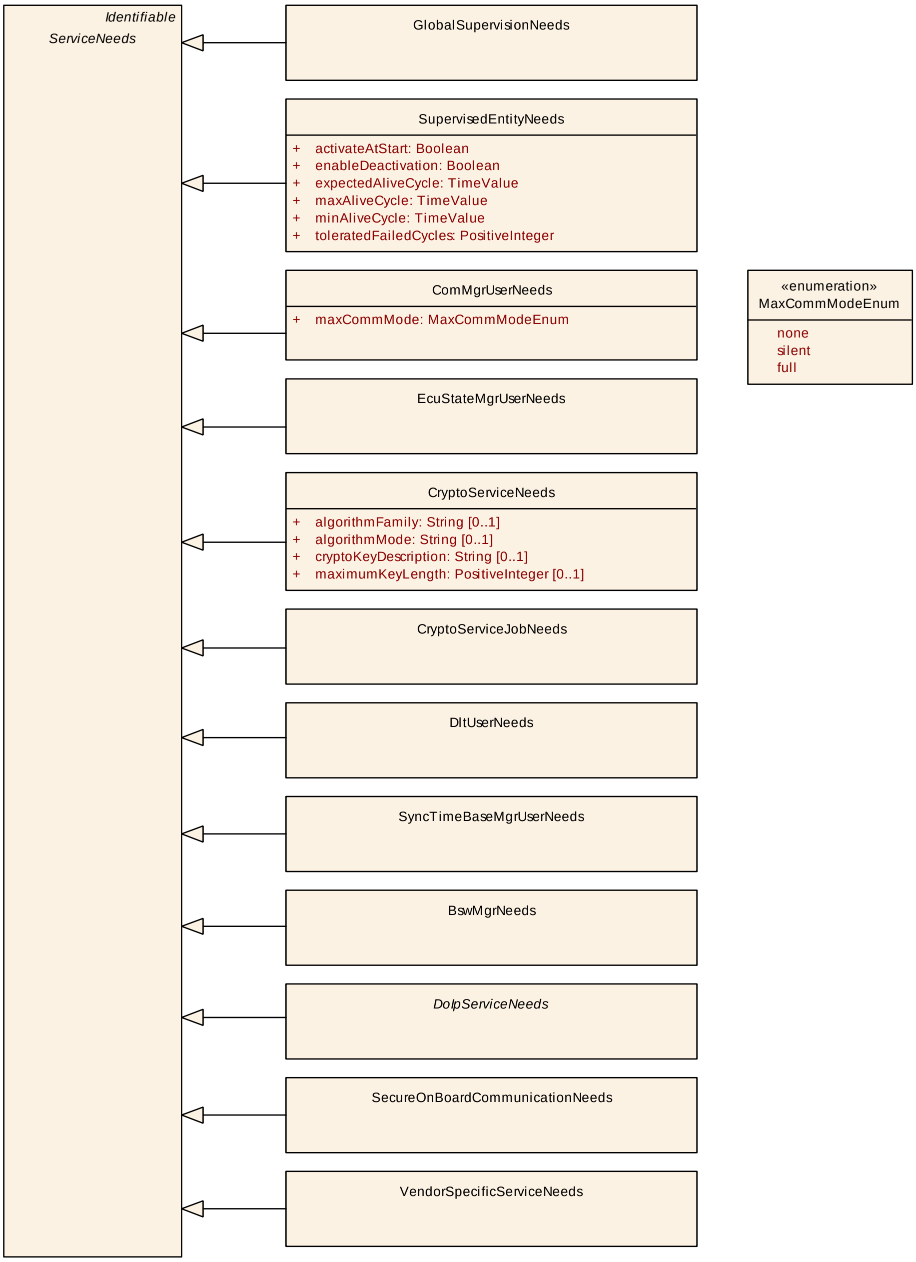

Figure 13.3: Class ServiceNeeds from CommonStructure and some derived classes

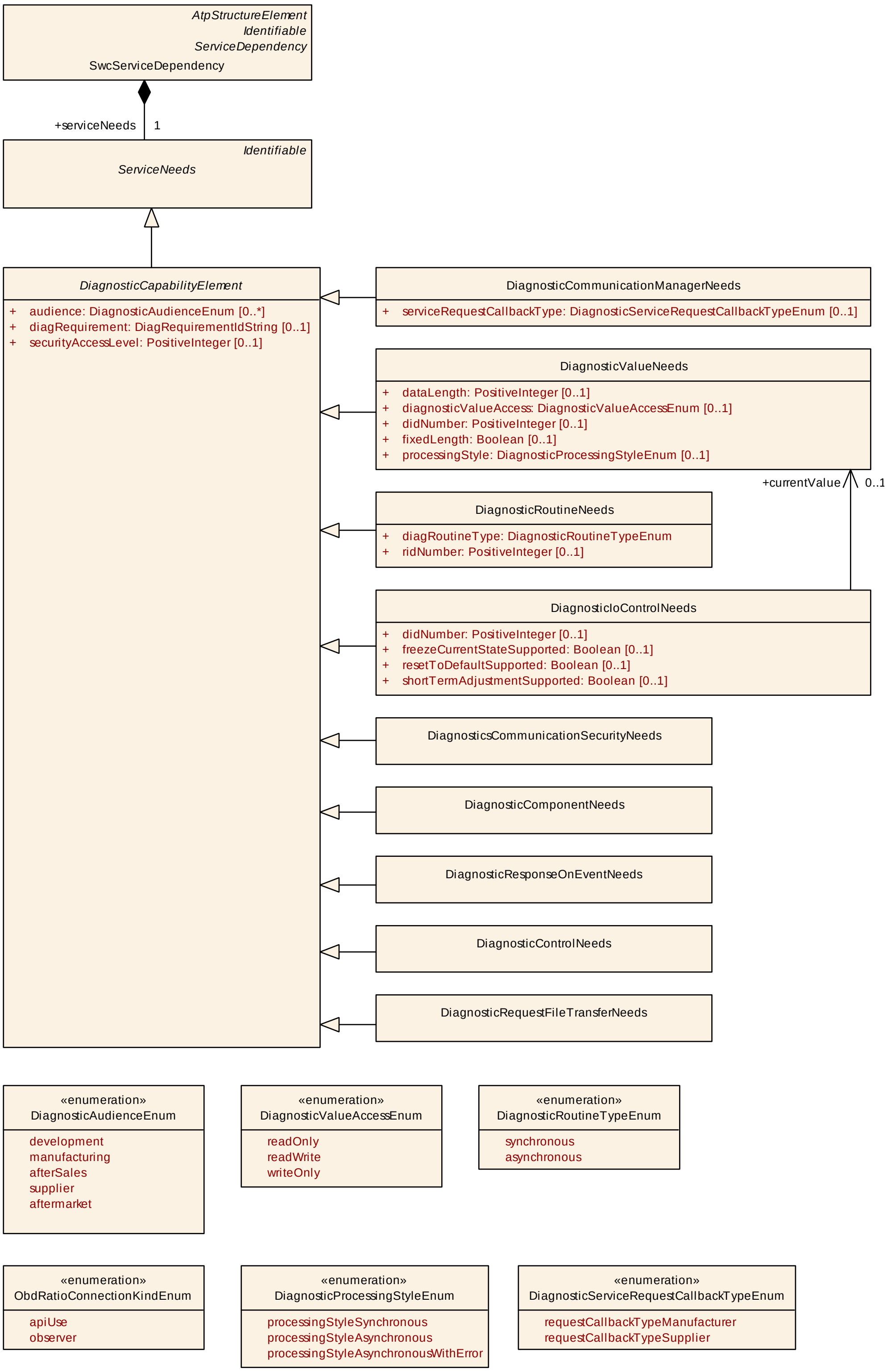

Figure 13.4: Class `ServiceNeeds` from `CommonStructure` and derived classes for diagnosis use cases

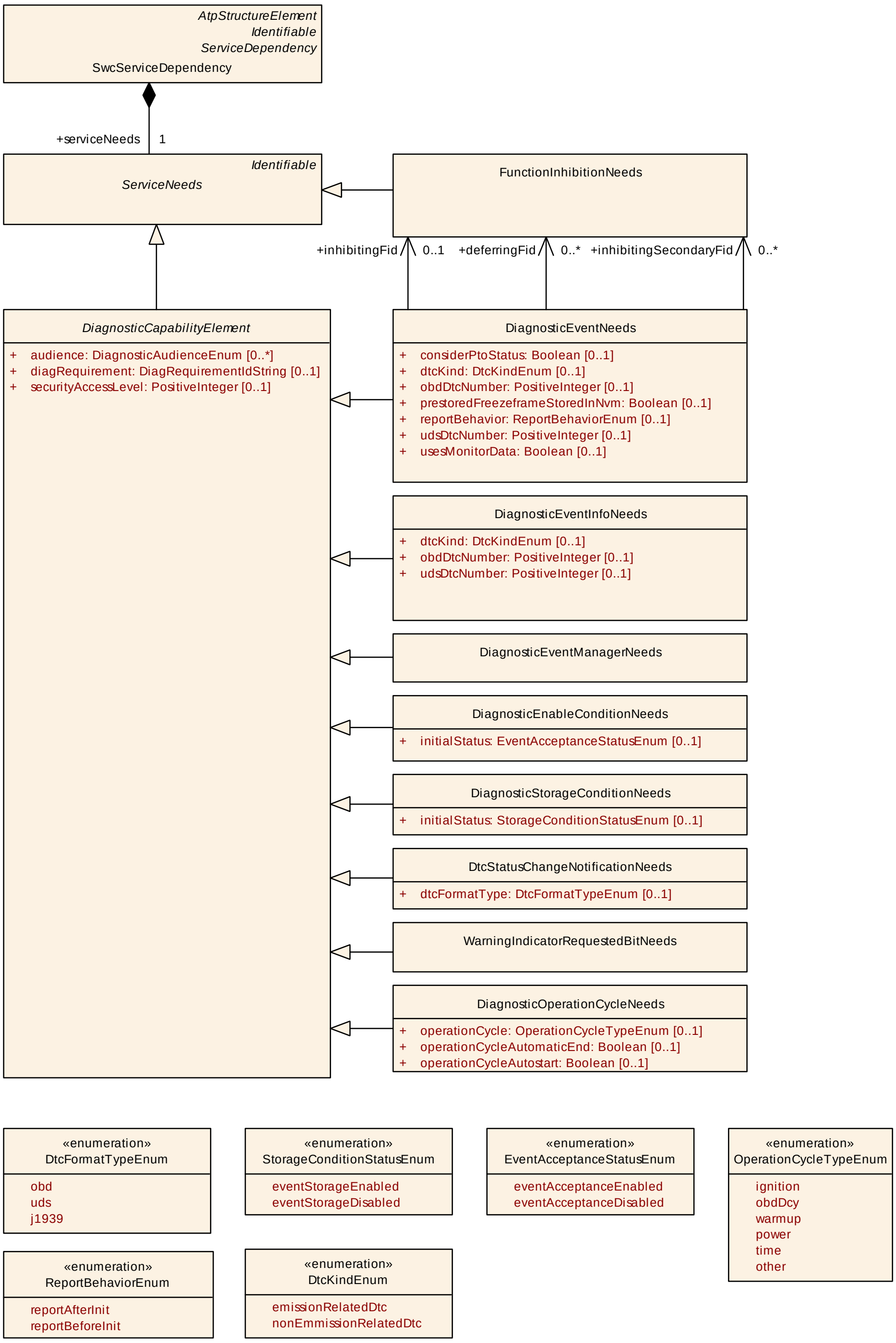

Figure 13.5: Class `ServiceNeeds` from `CommonStructure` and derived classes for diagnosis use cases

Class	NvBlockNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of a single NVRAM Block.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
calcRamBlockCrc	Boolean	0..1	attr	Defines if CRC (re)calculation for the permanent RAM Block is required.
checkStaticBlockId	Boolean	0..1	attr	Defines if the Static Block Id check shall be enabled.
cyclicWritingPeriod	TimeValue	0..1	attr	This represents the period for cyclic writing of NvData to store the associated RAM Block.
nDataSets	PositiveInteger	0..1	attr	Number of data sets to be provided by the NVRAM manager for this block. This is the total number of ROM Blocks and RAM Blocks.
nRomBlocks	PositiveInteger	0..1	attr	Number of ROM Blocks to be provided by the NVRAM manager for this block. Please note that these multiple ROM Blocks are given in a contiguous area.
ramBlockStatusControl	RamBlockStatusControlEnum	0..1	attr	This attribute defines how the management of the RAM Block status is controlled.
readonly	Boolean	0..1	attr	True: data of this NVRAM Block are write protected for normal operation (but protection can be disabled) false: no restriction
reliability	NvBlockNeedsReliabilityEnum	0..1	attr	Reliability against data loss on the non-volatile medium.
resistantToChangedSw	Boolean	0..1	attr	Defines whether an NVRAM Block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, please refer to the NVRAM specification.
restoreAtStart	Boolean	0..1	attr	Defines whether the associated RAM Block shall be implicitly restored during startup by the basic software.
selectBlockForFirstInitAll	Boolean	0..1	attr	If this attribute is set to true the NvM shall process this block in the NvM_FirstInitAll() function.
storeAtShutdown	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored during shutdown by the basic software.
storeCyclic	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored periodically by the basic software.
storeEmergency	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored in case of ECU failure (e.g. loss of power) by the basic software. If the attribute storeEmergency is set to true the associated RAM Block shall be configured to have immediate priority.
storeImmediate	Boolean	0..1	attr	Defines whether or not the associated RAM Block shall be implicitly stored immediately during or after execution of the according SW-C RunnableEntity by the basic software.
useAutoValidationAtShutDown	Boolean	0..1	attr	If set to true the RAM Block shall be auto validated during shutdown phase.
useCRCCompMechanism	Boolean	0..1	attr	If set to true the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job in order to skip unnecessary NVRAM writings.
writeOnlyOnce	Boolean	0..1	attr	Defines write protection after first write: true: This block is prevented from being changed/erased or being replaced with the default ROM data after first initialization by the software-component. false: No such restriction.

▽

△

Class	**NvBlockNeeds**			
writeVerification	Boolean	0..1	attr	Defines if Write Verification shall be enabled for this NVRAM Block.
writing Frequency	PositiveInteger	0..1	attr	Provides the amount of updates to this block from the application point of view. It has to be provided in "number of write access per year".
writingPriority	NvBlockNeedsWriting PriorityEnum	0..1	attr	Requires the priority of writing this block in case of concurrent requests to write other blocks.

Table 13.7: NvBlockNeeds

Enumeration	**NvBlockNeedsReliabilityEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Reliability against data loss on the non-volatile medium. These requirements give only a relative indication, for example on the required degree of redundancy for storage. They do, however, not specify by which means (e.g. software or hardware) the reliability is actually achieved.
Literal	*Description*
errorCorrection	Errors shall be corrected **Tags:** atp.EnumerationValue=0
errorDetection	Errors shall be detected **Tags:** atp.EnumerationValue=1
noProtection	Data need not to be handled with protection **Tags:** atp.EnumerationValue=2

Table 13.8: NvBlockNeedsReliabilityEnum

Enumeration	**NvBlockNeedsWritingPriorityEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Specifies the priority of writing this block in case of concurrent requests to write other blocks.
Literal	*Description*
high	Writing priority is high. **Tags:** atp.EnumerationValue=0
low	Writing priority is low. **Tags:** atp.EnumerationValue=1
medium	Writing priority is medium. **Tags:** atp.EnumerationValue=2

Table 13.9: NvBlockNeedsWritingPriorityEnum

Enumeration	**RamBlockStatusControlEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent
Note	This enumeration type defines options for how the management of the ramBlock status is controlled.
Literal	*Description*
api	The ramBlock status is controlled via service interface by usage of the SetRamBlockStatus operation. **Tags:** atp.EnumerationValue=0

▽

— AUTOSAR CONFIDENTIAL —

△

Enumeration	**RamBlockStatusControlEnum**
nvRamManager	The ramBlock status is controlled exclusively by the Nv Ram Manager. **Tags:** atp.EnumerationValue=1

Table 13.10: RamBlockStatusControlEnum

Enumeration	**MaxCommModeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Maximum bus communication mode required by a user of the Communication Manager Service.
Literal	*Description*
full	Full communication is requested. **Tags:** atp.EnumerationValue=0
none	No communication is requested. **Tags:** atp.EnumerationValue=1
silent	Silent communication is requested: Only listening but not "talking". **Tags:** atp.EnumerationValue=2

Table 13.11: MaxCommModeEnum

Class	**SupervisedEntityNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager for one specific Supervised Entity.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
activateAtStart	Boolean	1	attr	True/false: supervision activation status of Supervised Entity shall be enabled/disabled at start.
checkpoints	SupervisedEntity CheckpointNeeds	*	ref	This reference indicates the checkpoints belonging to the Supervised Entity. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
enable Deactivation	Boolean	1	attr	True: software-component shall be allowed to deactivate supervision of this SupervisedEntity false: software-component shall be not allowed to deactivate supervision of this SupervisedEntity
expectedAlive Cycle	TimeValue	1	attr	Expected cycle time of alive trigger of this Supervised Entity (in seconds).
maxAliveCycle	TimeValue	1	attr	Maximum cycle time of alive trigger of this Supervised Entity (in seconds).
minAliveCycle	TimeValue	1	attr	Minimum cycle time of alive trigger of this Supervised Entity (in seconds).

▽

△

Class	SupervisedEntityNeeds			
toleratedFailedCycles	PositiveInteger	1	attr	Number of consecutive failed alive cycles for this SupervisedEntity which shall be tolerated until the supervision status of the SupervisedEntity is set to WDGM_ALIVE_EXPIRED (see SWS WdgM for more details). Note that this value has to be recalculated with respect to the WdgM's own cycle time for ECU configuration.

Table 13.12: SupervisedEntityNeeds

Class	**ComMgrUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Communication Manager for one "user".			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
maxCommMode	MaxCommModeEnum	1	attr	Maximum communication mode requested by this ComM user.

Table 13.13: ComMgrUserNeeds

Class	**EcuStateMgrUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the ECU State Manager for one "user". This class currently contains no attributes. Its name can be regarded as a symbol identifying the user from the viewpoint of the component or module which owns this class.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 13.14: EcuStateMgrUserNeeds

Class	**CryptoServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the needs on the configuration of the CryptoServiceManager for one ConfigID (see Specification AUTOSAR_SWS_CSM.doc). An instance of this class is used to find out which ports of a software-component belong to this ConfigID.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
algorithmFamily	String	0..1	attr	This attribute represents a description of the family (e.g. AES) of crypto algorithm implemented by the crypto service use case.
algorithmMode	String	0..1	attr	This meta-class has the ability to represent a crypto service use case.
cryptoKeyDescription	String	0..1	attr	This attribute allows for a verbal description of the applicable cryptographic key. The goal is to pass a hint for the integrator about how to treat the corresponding service use case.
maximumKeyLength	PositiveInteger	0..1	attr	The maximum length of a cryptographic key, that is used by the software-component or module for this configuration. Unit: bit.

Table 13.15: CryptoServiceNeeds

Class	DltUserNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class specifies the needs on the configuration of the Diagnostic Log and Trace module for one SessionId. This class currently contains no attributes. An instance of this class is used to find out which PortPrototypes of an AtomicSwComponentType belong to this SessionId in order to group the request and response PortPrototypes of the same SessionId. The actual SessionId value is stored in the PortDefinedArgumentValue of the respective PortPrototype specification.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 13.16: DltUserNeeds

Class	SyncTimeBaseMgrUserNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the needs on the configuration of the Synchronized Time-base Manager for one time-base. This class currently contains no attributes. An instance of this class is used to find out which ports of a software-component belong to this time-base in order to group the request and response ports of the same time-base. The actual time-base value is stored in the PortDefinedArgumentValue of the respective port specification.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 13.17: SyncTimeBaseMgrUserNeeds

Class	*DiagnosticCapabilityElement* (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class identifies the capability to provide generic information about diagnostic capabilities			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Subclasses	DiagnosticCommunicationManagerNeeds, DiagnosticComponentNeeds, DiagnosticControlNeeds, DiagnosticEnableConditionNeeds, DiagnosticEventInfoNeeds, DiagnosticEventManagerNeeds, DiagnosticEventNeeds, DiagnosticIoControlNeeds, DiagnosticOperationCycleNeeds, DiagnosticRequestFileTransferNeeds, DiagnosticResponseOnEventNeeds, DiagnosticRoutineNeeds, DiagnosticStorageConditionNeeds, DiagnosticUploadDownloadNeeds, DiagnosticValueNeeds, DiagnosticsCommunicationSecurityNeeds, DtcStatusChangeNotificationNeeds, ObdControlServiceNeeds, ObdInfoServiceNeeds, ObdMonitorServiceNeeds, ObdPidServiceNeeds, ObdRatioDenominatorNeeds, ObdRatioServiceNeeds, WarningIndicatorRequestedBitNeeds			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
audience	DiagnosticAudienceEnum	*	attr	This specifies the intended audience for the diagnostic object. Note that this is not only for the documentation but also subsequent audience specific implementation.
diagRequirement	DiagRequirementIdString	0..1	attr	This denotes the requirement identifier to which the object can be linked to. Note that with the implementation of a generic tracing concept in AUTOSAR this attribute might become obsolete.

▽

△

Class	***DiagnosticCapabilityElement*** (abstract)			
securityAccess Level	PositiveInteger	0..1	attr	This attribute denotes the level of security which is touched by the diagnostic object. The higher the level the more relevance for the security exists. This level shall be mapped to the security level in the ECU.

Table 13.18: DiagnosticCapabilityElement

Class	**FunctionInhibitionNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Function Inhibition Manager for one Function Identifier (FID). This class currently contains no attributes. Its name can be regarded as a symbol identifying the FID from the viewpoint of the component or module which owns this class.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 13.19: FunctionInhibitionNeeds

Class	***DoIpServiceNeeds*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This represents an abstract base class for ServiceNeeds related to DoIP.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Subclasses	DoIpActivationLineNeeds, DoIpGidNeeds, DoIpGidSynchronizationNeeds, DoIpPowerModeStatus Needs, DoIpRoutingActivationAuthenticationNeeds, DoIpRoutingActivationConfirmationNeeds, Further ActionByteNeeds			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 13.20: DoIpServiceNeeds

13.2.1 NvM Service Dependencies

This chapter describes the usage of the specific meta-classes derived from `ServiceNeeds` within a `Basic Software Module`. The meta-class `NvBlockNeeds` is used to define requirements to configure the NVRAM Manager Service. There are several use cases how a `Basic Software Module` can interact with the NVRAM Manager service. Each use case is discussed in a separate sub-chapter.

13.2.1.1 `Nvm` Use Case: Permanent RAM Block

Scenario: a `Basic Software Module` is using an an `NvBlock` with a `Permanent RAM Block`.

[TPS_BSWMDT_04116] Setup for `Nvm` Use Case: `Permanent RAM Block` ⌈

ServiceNeeds kind `NvBlockNeeds`

RoleBasedBswModuleEntryAssignment

For every used BswModuleEntry provided by the NvM it is necessary to create a RoleBasedBswModuleEntryAssignment and set the value of the attribute role of the RoleBasedBswModuleEntryAssignment to the name of the used standardized BswModuleEntry. The following BswModuleEntrys shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvM_ReadBlock [0..1]
- NvM_WriteBlock [0..1]
- NvM_RestoreBlockDefaults [0..1]
- NvM_EraseNvBlock [0..1]
- NvM_InvalidateNvBlock [0..1]
- NvM_ReadPRAMBlock [0..1]
- NvM_WritePRAMBlock [0..1]
- NvM_RestorePRAMBlockDefaults [0..1]
- NvM_SetDataIndex [0..1]
- NvM_GetDataIndex [0..1]
- NvM_SetBlockProtection [0..1]
- NvM_GetErrorStatus [0..1]
- NvM_SetRamBlockStatus [0..1]
- NvM_SetBlockLockStatus [0..1]
- NvM_CancelJobs [0..1]
- NvM_SingleBlockCallbackFunction [0..1]
- InitBlockCallbackFunction [0..1]

RoleBasedDataAssignment

RoleBasedDataAssignment shall be created that refers to the Variable-DataPrototype in the role usedDataElement. The value of the attribute role of the RoleBasedDataAssignment shall be set to ramBlock.

Optionally, it is possible to create an additional RoleBasedDataAssignment to a ParameterDataPrototype in the role usedParameterElement. The value of the ParameterDataPrototype is then taken as the initial or default value for the NvBlock. In this case the value of the attribute role of the Role-BasedDataAssignment shall be set to defaultValue.
Therefore, the following roles are applicable:

- ramBlock [1]

- defaultValue [0 .. 1]

RoleBasedDataTypeAssignment
N/A

⌋()

For more information please refer to [SWS_NvM_00734], [SWS_NvM_00735], [SWS_NvM_00736], and [SWS_NvM_00737].

13.2.1.2 Nvm Use Case: Temporary RAM Block

Scenario: a Basic Software Module is using some NV blocks with a Temporary RAM Block.

[TPS_BSWMDT_04117] Setup for Nvm Use Case: Temporary RAM Block ⌈

ServiceNeeds kind NvBlockNeeds

RoleBasedBswModuleEntryAssignment valid roles:
For every used BswModuleEntry provided by the NvM it is necessary to create a RoleBasedBswModuleEntryAssignment and set the value of the attribute role of the RoleBasedBswModuleEntryAssignment to the name of the used standardized BswModuleEntry. The following BswModuleEntrys shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvM_ReadBlock [0..1]
- NvM_WriteBlock [0..1]
- NvM_RestoreBlockDefaults [0..1]
- NvM_EraseNvBlock [0..1]
- NvM_InvalidateNvBlock [0..1]
- NvM_SetDataIndex [0..1]
- NvM_GetDataIndex [0..1]
- NvM_SetBlockProtection [0..1]
- NvM_GetErrorStatus [0..1]
- NvM_SetRamBlockStatus [0..1]
- NvM_SetBlockLockStatus [0..1]
- NvM_CancelJobs [0..1]
- NvM_SingleBlockCallbackFunction [0..1]
- InitBlockCallbackFunction [0..1]

RoleBasedDataAssignment
RoleBasedDataAssignment may be created that refers to a ParameterDataPrototype in the role usedParameterElement. The value of the ParameterDataPrototype is then taken as the initial or default value for the NvBlock. In this case the value of the attribute role of the RoleBasedDataAssignment shall be set to defaultValue.
Therefore, the following roles are applicable:

- defaultValue [0 .. 1]

RoleBasedDataTypeAssignment
This denotes the type of the temporary Ram Block. The type information can be used to calculate the NVRAM block. [constr_4088] applies.

- temporaryRamBlock [0 .. 1]

⌋*()*

[constr_4088] Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role ⌈The usage of a RoleBasedDataTypeAssignment with attribute role set to the value temporaryRamBlock is only allowed if **no** RoleBasedDataAssignment defined with attribute role set to value defaultValue exists in the owning BswServiceDependency. ⌋*()*

The rationale for [constr_4088] is that the existence of a RoleBasedDataAssignment would already provide sufficient information for the intended purpose. The parallel existence of a RoleBasedDataTypeAssignment is therefore fully redundant and could only lead to potential inconsistencies.

13.2.1.3 Nvm Use Case: RAM Block with explicit synchronization

Scenario: a Basic Software Module is using some NV blocks where the RAM Block is synchronized by means of explicit synchronizatin using the mirror interfaces.

[TPS_BSWMDT_04118] Setup for Nvm Use Case: RAM Block synchronised with explicit synchronization ⌈

RoleBasedBswModuleEntryAssignment valid roles:
For every used BswModuleEntry provided by the NvM it is necessary to create a RoleBasedBswModuleEntryAssignment and set the value of the attribute role of the RoleBasedBswModuleEntryAssignment to the name of the used standardized BswModuleEntry. The following BswModuleEntrys shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvM_ReadBlock [0..1]
- NvM_WriteBlock [0..1]

- NvM_RestoreBlockDefaults [0..1]
- NvM_EraseNvBlock [0..1]
- NvM_InvalidateNvBlock [0..1]
- NvM_ReadPRAMBlock [0..1]
- NvM_WritePRAMBlock [0..1]
- NvM_RestorePRAMBlockDefaults [0..1]
- NvM_SetDataIndex [0..1]
- NvM_GetDataIndex [0..1]
- NvM_SetBlockProtection [0..1]
- NvM_GetErrorStatus [0..1]
- NvM_SetRamBlockStatus [0..1]
- NvM_SetBlockLockStatus [0..1]
- NvM_CancelJobs [0..1]
- NvM_SingleBlockCallbackFunction [0..1]
- InitBlockCallbackFunction [0..1]
- NvM_ReadRamBlockFromNvm [1]
- NvM_WriteRamBlockToNvm [1]

RoleBasedDataAssignment

RoleBasedDataAssignment may be created that refers to a ParameterDataPrototype in the role usedParameterElement. The value of the ParameterDataPrototype is then taken as the initial or default value for the NvBlock. In this case the value of the attribute role of the RoleBasedDataAssignment shall be set to defaultValue.
Therefore, the following roles are applicable:

- defaultValue [0 .. 1]

RoleBasedDataTypeAssignment

This denotes the type of the internal data structure synchronized with NvMReadRamBlockFromNvM and NvMWriteRamBlockToNvM interface. The type information can be used to calculate the NVRAM block size and minimum RAM Mirror size. [constr_4088] applies.

- temporaryRamBlock [0 .. 1]

⌋()

13.2.2 Diagnostic Service Dependency

This chapter describes the usage of the specific diagnostic meta-classes derived from ServiceNeeds within a Basic Software Module.

13.2.2.1 Function Inhibition Needs

The meta-class FunctionInhibitionNeeds is used to define requirements in order to configure the Function Inhibition Manager.

A BswInternalBehavior may provide several FunctionInhibitionNeeds elements, each defines the requirements related to one function inhibition ID (for the terms related to the AUTOSAR Function Inhibition Manager, see [25]).

13.2.2.1.1 Function Inhibition Manager Service use Case: read function permission

[TPS_BSWMDT_04119] Setup for Function Inhibition Manager Service use Case: read function permission ⌈

Scenario: a Basic Software Module reads the function permission from FiM in order to enable or disable a functionality. In this case the following setup apply:

ServiceNeeds kind FunctionInhibitionNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- FiM_GetFunctionPermission [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

13.2.2.1.2 Function Inhibition Manager Service use Case: react on suppressed or unavailable events

[TPS_BSWMDT_04167] Setup for Function Inhibition Manager Service use Case: read function permission ⌈

Scenario: a Basic Software Module wants to react on suppressed or unavailable events and disable the permission to run for a FID. In this case, the following setup applies:

ServiceNeeds kind FunctionInhibitionAvailabilityNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- FiM_SetFunctionAvailable [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

Note: for variant coding ClientServerInterface, ControlFunctionAvailable is used to deactivate a certain functionality (e.g. to set the FID to not available).

For more information please refer to [SWS_Fim_00106].

13.2.2.2 Diagnostic Event Needs

The meta-classes DiagnosticEventNeeds is used to define requirements in order to configure the Diagnostic Event Manager Service.

An BswInternalBehavior may provide several DiagnosticEventNeeds elements that each defines the requirements related to one diagnostic monitor. (For the terms related to the AUTOSAR Diagnostic Event Manager see [26]).

13.2.2.2.1 Dem Service Use Case: diagnostic monitor, debouncing by Dem

Scenario: a Basic Software Module implements a Diagnostic Monitor. The debouncing of the failure condition shall be configured and processed by the Dem. In this case the following setup apply:

[TPS_BSWMDT_04120] Basic Software Module implements a Diagnostic Monitor ⌈

ServiceNeeds kind DiagnosticEventNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Dem_SetEventStatus [1]
- Dem_ResetEventDebounceStatus [0..1]
- Dem_GetEventStatus [0..1]

- `Dem_GetEventFailed` [0..1]
- `Dem_GetEventTested` [0..1]
- `Dem_GetDTCOfEvent` [0..1]
- `Dem_SetEventDisabled` [0..1]
- `InitMonitorForEvent` [0..1]
- `DemTriggerOnEventStatus` [0..1]
- `DemClearEventAllowed` [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.2.2.2 Service Use Case: `Basic Software Module` implements a Hardware Shutdown

Scenario: when a hardware component is detected as being defective, the Dem shall inform the Bsw Module which is responsible for executing a hardware-shutdown.

[TPS_BSWMDT_04139] Dem Use Case: Bsw Module implements a hardware shutdown ⌈

ServiceNeeds kind `DiagnosticComponentNeeds`

RoleBasedPortAssignment valid roles:

- `DemTriggerOnComponentStatus` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*()*

13.2.2.3 Diagnostic Communication Needs

The meta-class `DiagnosticValueNeeds` is used to define requirements in order to configure the Diagnostic Communication Manager Service as well as the Diagnostic Event Manager Service. The DcM and Dem can access local values via callback functions.

The attribute DiagnosticValueNeeds.diagnosticValueAccess of type DiagnosticValueAccessEnum allows for distinguishing between current values to read diagnostic information (readOnly) and data elements which are additionally classified as configurable (readWrite).

Class	**DiagnosticValueNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the DCM which are not related to a particular item. In the case of using a sender receiver communicated value, the related value shall be taken via assigned Data in the role "signalBasedDiagnostics". In case of using a client/server communicated value, the related value shall be communicated via the port referenced by asssignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the related software specifications (SWS).			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeed is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this particular PID signal.
diagnosticValue Access	DiagnosticValueAccess Enum	0..1	attr	This attribute controls whether the data can be read and written or whether it is to be handled read-only.
didNumber	PositiveInteger	0..1	attr	This represents a Data identifier for the diagnostic value. This allows to predefine the DID number if the responsible function developer has received a particular requirement from the OEM or from a standardization body.
fixedLength	Boolean	0..1	attr	This attribute controls whether the data length of the data is fixed.
processingStyle	DiagnosticProcessing StyleEnum	0..1	attr	This attribute controls whether interaction requires the software-component to react synchronously on a request or whether it processes the request in background but still the DCM has to issue the call again to eventually obtain the result of the request.

Table 13.21: DiagnosticValueNeeds

Enumeration	**DiagnosticValueAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Defines the access of the configured diagnostic current values which will be used by the Dem or Dcm module.
Literal	***Description***
readOnly	The access to the data element is limited to read-only. This is typically used to read-out diagnostic information (e.g. current values). **Tags:** atp.EnumerationValue=0
readWrite	The value of the diagnostic data element is classified as configurable (read and write access is possible). **Tags:** atp.EnumerationValue=1
writeOnly	The access to the data element is limited to write-only. This supports the use case where the Dcm just writes data to the application software without the intention to read it back, **Tags:** atp.EnumerationValue=2

Table 13.22: DiagnosticValueAccessEnum

Enumeration	**DiagnosticProcessingStyleEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This meta-class represents the ability to define the processing style of diagnostic requests.
Literal	*Description*
processingStyleAsynchronous	The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request. **Tags:** atp.EnumerationValue=0
processingStyleAsynchronousWithError	The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request or handle error code. **Tags:** atp.EnumerationValue=1
processingStyleSynchronous	The software-component is supposed to react synchronously on the request. **Tags:** atp.EnumerationValue=2

Table 13.23: DiagnosticProcessingStyleEnum

The meta-class DiagnosticRoutineNeeds is used to define requirements to configure the Diagnostic Communication Manager Service. A Basic Software Module may provide BswModuleEntrys (for example, “start”, “stop”, and “RequestResults”). The BswModuleEntrys correspond to the diagnostic service RoutineControl for one routine identifier. The enumeration parameter DiagnosticRoutineTypeEnum is used to define whether the diagnostic server or client is responsible for stopping the routine.

Class	**DiagnosticRoutineNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (Dcm) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
diagRoutineType	DiagnosticRoutineTypeEnum	1	attr	This denotes the type of diagnostic routine which is implemented by the referenced server port.
ridNumber	PositiveInteger	0..1	attr	This represents a routine identifier for the diagnostic routine. This allows to predefine the RID number if the a function developer has received a particular requirement from the OEM or from a standardization body.

Table 13.24: DiagnosticRoutineNeeds

Enumeration	**DiagnosticRoutineTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumerator specifies the different types of diagnostic routines.
Literal	*Description*
asynchronous	This indicates that the diagnostic server is not blocked while the diagnostic routine is running. **Tags:** atp.EnumerationValue=0

▽

△

Enumeration	**DiagnosticRoutineTypeEnum**
synchronous	This indicates that the diagnostic routine blocks the diagnostic server in the ECU while the routine is running. **Tags:** atp.EnumerationValue=1

Table 13.25: DiagnosticRoutineTypeEnum

The meta-class `DiagnosticIoControlNeeds` is used to define requirements to configure the Diagnostic Communication Manager Service.

Class	**DiagnosticIoControlNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
currentValue	DiagnosticValueNeeds	0..1	ref	Reference to the DiagnosticValueNeeds indicating the access to the current value via signalBasedDiagnostics.
didNumber	PositiveInteger	0..1	attr	This represents a Data identifier for the diagnostic value. This allows to predefine the DID number if the a function developer has received a particular requirement from the OEM or from a standardization body.
freezeCurrent StateSupported	Boolean	0..1	attr	This attribute determines, if the referenced port supports temporary freezing of I/O value.
resetToDefault Supported	Boolean	0..1	attr	This represents a flag for the existence of the ResetTo Default operation in the service interface.
shortTerm Adjustment Supported	Boolean	0..1	attr	This attribute determines, if the referenced port supports temporarily setting of I/O value to a specific value provided by the diagnostic tester.

Table 13.26: DiagnosticIoControlNeeds

Class	**DiagnosticsCommunicationSecurityNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the needs of a software-component to verify the access to security level via diagnostic services.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *Service Needs*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 13.27: DiagnosticsCommunicationSecurityNeeds

Class	DiagnosticCommunicationManagerNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (Dcm) which are not related to a particular item (e.g. a PID or DiagnosticRoutineNeeds). The main use case is the mapping of service ports to the Dcm which are not related to a particular item.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
serviceRequestCallbackType	DiagnosticServiceRequestCallbackTypeEnum	0..1	attr	This represents the ability to define whether the usage of PortInterface ServiceRequestNotification has the characteristics of being initiated by a manufacturer or by a supplier.

Table 13.28: DiagnosticCommunicationManagerNeeds

13.2.2.3.1 Dcm Service Use Case: read/write current values by BswModuleEntrys

Scenario: a Basic Software Module offers a BswModuleEntrys to read/write current value via diagnostic services.

[TPS_BSWMDT_04121] Basic Software Module offers BswModuleEntrys to read/write current value via diagnostic services ⌈

ServiceNeeds kind DiagnosticValueNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Xxx_ReadData [0..1] (1 in case read is supported)
- Xxx_WriteData [0..1] (1 in case write is supported)
- Xxx_ReadDataLength [0..1] (1 in case of variable length)
- Xxx_ConditionCheckRead [0..1]](1 in case the read condition is provided by the BSW module)

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.2.3.2 Dcm Service Use Case: start/stop or request routine results

Scenario: a Basic Software Module offers a BswModuleEntrys to start/stop or request routines via diagnostic services.

[TPS_BSWMDT_04122] `Basic Software Module` offers `BswModuleEntry`s to start/stop or request routines via diagnostic services ⌈

ServiceNeeds kind `DiagnosticRoutineNeeds`

RoleBasedBswModuleEntryAssignment valid roles:

- `Xxx_Start` [1]
- `Xxx_Stop` [0..1]
- `Xxx_RequestResults` [0..1]
- `Xxx_StartConfirmation` [0..1]
- `Xxx_StopConfirmation` [0..1]
- `Xxx_RequestResultsConfirmation` [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.2.3.3 Dcm Service Use Case: IO control

Scenario: a `Basic Software Module` offers a `BswModuleEntry`s BswModuleEntrys to adjust the IO signal via diagnostic services.

[TPS_BSWMDT_04123] `Basic Software Module` offers `BswModuleEntry`s to adjust the IO signal via diagnostic services ⌈

ServiceNeeds kind `DiagnosticIoControlNeeds`

RoleBasedBswModuleEntryAssignment valid roles:

- `Xxx_ReadData` [1]
- `Xxx_ReturnControlToECU` [0..1]
- `Xxx_ResetToDefault` [0..1]
- `Xxx_FreezeCurrentState` [0..1]
- `Xxx_ShortTermAdjustment` [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

13.2.2.3.4 Dcm Service Use Case: Access to protocol, session and security Information

Scenario: a Basic Software Module offers a BswModuleEntrys to access protocol, session and security information.

[TPS_BSWMDT_04124] Basic Software Module offers BswModuleEntrys to access protocol, session and security information ⌈

ServiceNeeds kind DiagnosticCommunicationManagerNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Dcm_ResetToDefaultSession [0..1]
- Dcm_GetSecurityLevel [0..1]
- Dcm_GetSesCtrlTypel [0..1]
- Dcm_GetActiveProtocol [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

13.2.2.3.5 Dcm Service Use Case: Seed / Key handling for security level access and the optional security attempt counter handling

Scenario: a Basic Software Module offers BswModuleEntrys for the Seed and Key handling for security level access and the optional security attempt counter handling.

[TPS_BSWMDT_04125] Basic Software Module offers BswModuleEntrys for the Seed adn Key handling for security level access and the optional security attempt counter handling ⌈

ServiceNeeds kind DiagnosticsCommunicationSecurityNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Xxx_CompareKey [1]
- Xxx_GetSeed [1]
- Xxx_GetSecurityAttemptCounter [0..1]

- Xxx_SetSecurityAttemptCounter [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

13.2.2.3.6 Dcm Service Use Case: Upload and download of data

Scenario: a Basic Software Module implements the ability to accept data for upload and/or provide data for download. For this purpose the Basic Software Module provides a BswModuleEntry that connects to the Dcm service component.

[TPS_BSWMDT_04172] Basic Software Module implements the ability to accept data for upload and/or provide data for download. For this purpose the Basic Software Module provides a BswModuleEntry that connects to the Dcm service component. ⌈

ServiceNeeds kind DiagnosticUploadDownloadNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- UploadDownloadServices [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

Class	DiagnosticUploadDownloadNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to specify needs regarding upload and download by means of diagnostic services.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	Type	Mul.	Kind	Note
–	–	–	–	–

Table 13.29: DiagnosticUploadDownloadNeeds

13.2.2.4 OBD Service Needs

The meta-class ObdPidServiceNeeds is used to define requirements to configure OBD Services in relation to a particular PID (parameter identifier).

13.2.2.4.1 OBD Service Use Case: Read value via OBD services

Scenario: a Basic Software Module offers a BswModuleEntrys BswModuleEntrys to read value via OBD services.

[TPS_BSWMDT_04165] Basic Software Module offers BswModuleEntrys to read value via OBD services ⌈

ServiceNeeds kind ObdPidServiceNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Xxx_ReadData [1] (1 in case read is supported)

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

The meta-class ObdInfoServiceNeeds is used to define requirements to configure OBD Services in relation to a given InfoType (OBD Service 09).

13.2.2.4.2 OBD Service Use Case: Read vehicle information via OBD services

Scenario: a Basic Software Module offers a BswModuleEntrys BswModuleEntrys to read vehicle information via OBD services.

[TPS_BSWMDT_04166] Basic Software Module offers BswModuleEntrys to read vehicle information via OBD services ⌈

ServiceNeeds kind ObdInfoServiceNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- Xxx_GetInfoTypeValueData [1] (1 in case read is supported)

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.3 Watchdog Service Dependencies

The meta-class SupervisedEntityNeeds is used to define requirements to configure the Watchdog Service. For the terms related to the AUTOSAR Watchdog Manager see [27].

13.2.4 Watchdog Service use Case: Local Supervision

The service interaction with the *Watchdog Manager* consists of two aspects:

- supervised entity
- checkpoint

For each of the two aspects a separated ServiceNeeds is defined. However, the BswServiceDependencys that own these ServiceNeeds are semantically bound and cannot be used independently of each other.

In other words, the usage of two kinds of BswServiceDependency in concert creates a higher-level semantics. Of course, in order to express this higher-level semantics a reference between the BswServiceDependencys has to be available.

However, since the BswServiceDependency represents a generic concept the actual reference needs to be implemented on the level of specific subclass of ServiceNeeds, in this case the SupervisedEntityNeeds and the SupervisedEntityCheckpointNeeds.

The former refers to the latter in order to express the relation of a supervised entity to its checkpoints.

Class	**SupervisedEntityCheckpointNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager to support a Checkpoint for a Supervised Entity.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 13.30: SupervisedEntityCheckpointNeeds

[TPS_BSWMDT_04129] Definition a Supervised Entity in a `Basic Software Module` ⌈

ServiceNeeds kind : SupervisedEntityNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- WdgM_GetLocalStatus [0..1]
- WdgM_LocalMode [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

For more information please refer to [SWS_WdgM_00333], and [SWS_WdgM_00335].

Please note that an BswInternalBehavior may provide several SupervisedEntityNeeds elements where each defines the requirements in relation to one supervised entity.

[TPS_BSWMDT_04157] Definition of Checkpoints for a Supervised Entity in a Basic Software Module ⌈

ServiceNeeds kind : SupervisedEntityCheckpointNeeds

RoleBasedBswModuleEntryAssignment valid roles:

- WdgM_CheckpointReached [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

For more information please refer to [SWS_WdgM_00333], and [SWS_WdgM_00335].

Please note that an BswInternalBehavior may provide several SupervisedEntityCheckpointNeeds elements where each defines the relation to one SupervisedEntityNeeds .

13.2.5 Watchdog Service use Case: Control global supervision or get global supervision status

Scenario: a Basic Software Module either controls the global operation of the watchdog manager or gets information about the current operations status requiring at least one of the following use cases:

- Sets the current mode of Watchdog Manager
- Gets the current mode of the Watchdog Manager
- Gets the global supervision status of the Watchdog Manager
- Identifier of the supervised entity that first reached the expired state
- Instructs the Watchdog Manager to cause a watchdog reset

For instance the Basic Software Module sets the current mode of the Watchdog Manager according the operational state of the ECU or polls the global supervision status.

In this case the following setup applies:

[TPS_BSWMDT_04158] Setup for a Basic Software Module which sets or gets Global Supervision Status ⌈

ServiceNeeds kind : GlobalSupervisionNeeds

RoleBasedPortAssignment valid roles:

- WdgM_GetFirstExpiredSEID [0..1]
- WdgM_GetGlobalStatus [0..1]
- WdgM_GetLocalStatus [0..1]
- WdgM_GetMode [0..1]
- WdgM_PerformReset [0..1]
- WdgM_SetMode [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.6 ECU State Manager Service Needs

The meta-class EcuStateMgrUserNeeds is used to define the requirements to configure the ECU State Manager Service. There are actually two variants of AUTOSAR ECU management: flexible and fixed. An BswInternalBehavior may provide several EcuStateMgrUserNeeds elements where each defines the requirements from one "user" of the EcuM Service (for the terms related to the AUTOSAR ECU State Manager see [28]).

13.2.6.1 EcuM Flex Use Case: select Shutdown Target

Scenario: a Basic Software Module wants to select a shutdown target. This corresponds to the "select shutdown target" use case of the fix EcuM.

In this case the following rules apply:

[TPS_BSWMDT_04135] `Basic Software Module` wants to select a shutdown target (flexible variant) ⌈

RoleBasedBswModuleEntryAssignment valid roles:

- `EcuM_GetShutdownTarget` [1]
- `EcuM_SelectShutdownTarget` [1]
- `EcuM_GetLastShutdownTarget` [1]
- `EcuM_GetShutdownCause` [1]
- `EcuM_SelectShutdownCause` [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.6.2 EcuM Flex Use Case: select Boot Target

Scenario: a `Basic Software Module` wants to select a boot target.

In this case the following rules apply:

[TPS_BSWMDT_04136] `Basic Software Module` wants to select a boot target (flexible variant) ⌈

RoleBasedBswModuleEntryAssignment valid roles:

- `EcuM_GetBootTarget` [1]
- `EcuM_SelectBootTarget` [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.2.6.3 EcuM Flex Use Case: use Alarm Clock

Scenario: a `Basic Software Module` wants to use an alarm clock.

In this case the following rules apply:

[TPS_BSWMDT_04137] **`Basic Software Module` wants to use an alarm clock (flexible variant)** ⌈

RoleBasedBswModuleEntryAssignment valid roles:

- `EcuM_SetRelWakeupAlarm` [1]
- `EcuM_SetAbsWakeupAlarm` [1]
- `EcuM_AbortWakeupAlarm` [1]
- `EcuM_SetClock` [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

13.3 Basic Software Production Errors

The meta-class `DiagnosticEventNeeds` is used to specify production errors in a BSWMD.

Class	**DiagnosticEventNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Diagnostic Event Manager for one diagnostic event. Its shortName can be regarded as a symbol identifying the diagnostic event from the viewpoint of the component or module which owns this element. In case the diagnostic event specifies a production error, the shortName shall be the name of the production error.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
considerPtoStatus	Boolean	0..1	attr	PTO (Power Take Off) has an impact on the respective emission-related event (OBD). This information shall be provided by SW-C description in order to consider the PTO relevance e.g. for readiness (PID $01) computation. For events with dtcKind set to 'nonEmmissionRelatedDtc' this attribute is typically false.
deferringFid	FunctionInhibitionNeeds	*	ref	This reference contains the link to a function identifier within the FiM which is used by the monitor before delivering a result.
diagEventDebounceAlgorithm	DiagEventDebounceAlgorithm	0..1	aggr	Specifies the abstract need on the Debounce Algorithm applied by the Diagnostic Event Manager.
dtcKind	DtcKindEnum	0..1	attr	This attribute indicates the kind of the diagnostic monitor according to the SWS Diagnostic Event Manger. This attribute applies for the UDS diagnostics use case.

▽

△

Class	**DiagnosticEventNeeds**			
inhibitingFid	FunctionInhibitionNeeds	0..1	ref	This represents the primary Function Inhibition Identifier used for inhibition of the diagnostic monitor. The FID might either inhibit the monitoring of a symptom or the reporting of detected faults.
inhibiting SecondaryFid	FunctionInhibitionNeeds	*	ref	This represents the secondary Function Inhibition Identifier used for inhibition of the diagnostic monitor. Any of the FID inhibitions leads to an inhibition of the monitoring of a symptom or the reporting of detected faults.
obdDtcNumber	PositiveInteger	0..1	attr	This represents a reasonable Diagnostic Trouble Code. This allows to predefine the Diagnostic Trouble Code, e.g. if the a function developer has received a particular requirement from the OEM or from a standardization body. This attribute applies for the OBD diagnostics use case.
prestored Freezeframe StoredInNvm	Boolean	0..1	attr	If the Event uses a prestored freeze-frame (using the operations PrestoreFreezeFrame and ClearPrestored FreezeFrame of the service interface DiagnosticMonitor) this attribute indicates if the Event requires the data to be stored in non-volatile memory. TRUE = Dem shall store the prestored data in non-volatile memory, FALSE = Data can be lost at shutdown (not stored in Nvm).
reportBehavior	ReportBehaviorEnum	0..1	attr	This switch indicates whether or not the BSW module is allowed to report the related Events before Dem_Init().
udsDtcNumber	PositiveInteger	0..1	attr	This represents a reasonable Diagnostic Trouble Code. This allows to predefine the Diagnostic Trouble Code, e.g. if the a function developer has received a particular requirement from the OEM or from a standardization body. This attribute applies for the UDS diagnostics use case.
usesMonitor Data	Boolean	0..1	attr	This attribute defines whether additional monitor data shall be added to the reporting of events.

Table 13.31: DiagnosticEventNeeds

Class	***DiagEventDebounceAlgorithm*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class represents the ability to specify the pre-debounce algorithm which is selected and/or required by the particular monitor. This class inherits from Identifiable in order to allow further documentation of the expected or implemented debouncing and to use the category for the identification of the expected / implemented debouncing.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DiagEventDebounceCounterBased, DiagEventDebounceMonitorInternal, DiagEventDebounceTime Based			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 13.32: DiagEventDebounceAlgorithm

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemDebounceAlgorithmClass to DemDebounce CounterBased.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
counterBased FdcThreshold StorageValue	Integer	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
counter DecrementStep Size	Integer	1	attr	This value shall be taken to decrement the internal debounce counter.
counterFailed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "failed" counter status.
counter IncrementStep Size	Integer	1	attr	This value shall be taken to increment the internal debounce counter.
counterJump Down	Boolean	1	attr	This value activates or deactivates the counter jump-down behavior.
counterJump DownValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from incrementing to decrementing.
counterJumpUp	Boolean	1	attr	This value activates or deactivates the counter jump-up behavior.
counterJumpUp Value	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing.
counterPassed Threshold	Integer	1	attr	This value defines the event-specific limit that indicates the "passed" counter status.

Table 13.33: DiagEventDebounceCounterBased

Class	**DiagEventDebounceTimeBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to set the EcuC choice container DemDebounceAlgorithmClass to DemDebounceTime Base.			
Base	*ARObject*, *DiagEventDebounceAlgorithm*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
timeBasedFdc Threshold StorageValue	TimeValue	0..1	attr	Threshold to allocate an event memory entry and to capture the Freeze Frame.
timeFailed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "failed" status.
timePassed Threshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "passed" status.

Table 13.34: DiagEventDebounceTimeBased

Class	**DiagEventDebounceMonitorInternal**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the pre-debounce algorithm shall be used by the Dem for this diagnostic monitor. This is related to setting the EcuC choice container DemDebounceAlgorithmClass to DemDebounce MonitorInternal. If the FaultDetectionAlogrithm is already known to be implemented by a specific BswModuleEntry the reference bswModuleEntry points to the function specification. If the FaultDetectionCounter value is accessible at a PortPrototype this PortPrototype shall be referenced by an assignedPort.			
Base	*ARObject, DiagEventDebounceAlgorithm, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 13.35: DiagEventDebounceMonitorInternal

Enumeration	**DtcKindEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration defines the possible kinds of diagnostic monitors regarding the OBD relevance.
Literal	**Description**
emissionRelatedDtc	This indicates that the monitor reports a OBD-relevant malfunction. **Tags:** atp.EnumerationValue=0
nonEmmission RelatedDtc	This indicates that the monitor reports a non-OBD-relevant malfunction. **Tags:** atp.EnumerationValue=1

Table 13.36: DtcKindEnum

Enumeration	**ReportBehaviorEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumeration specifies the report status of related events before or after diagnostic event initialization.
Literal	**Description**
reportAfterInit	This allows reporting related events after initialization **Tags:** atp.EnumerationValue=0
reportBeforeInit	This allows reporting related events before initialization **Tags:** atp.EnumerationValue=1

Table 13.37: ReportBehaviorEnum

[TPS_BSWMDT_04110] Declaration of production errors ⌈ If a BSW module reports diagnostic events to the module DEM (= Diagnostic Event Manager ,see [26]), its `BswInternalBehavior` shall contain for each kind of diagnostic event one `ServiceDependency` element in the role `serviceDependency`.

This diagnostic event is further characterized by the element `ServiceDependency.serviceNeeds` which shall be an instance of meta-class `DiagnosticEventNeeds`. If the diagnostic event describes a production error, its `DiagnosticEventNeeds.category` attribute shall have one of the following values:

- **PRODUCTION_ERROR** if it represents a production error.

- **EXTENDED_PRODUCTION_ERROR** if it represents an extended production error.

Its DiagnosticEventNeeds.shortName shall be equal to the error symbol defined in the AUTOSAR SWS of the respective module if the production error is standardized. ⌋*(RS_BSWMD_00045, RS_BSWMD_00069)*

For further information on production error reporting refer to [29].

Production errors and extended production errors are reported to the DEM via the C-function Dem_SetEventStatus(). This scenario shall be specified in the following way:

[TPS_BSWMDT_04111] BswServiceDependency refers to Dem_SetEventStatus() ⌈ A BswModuleEntry representing the signature of the C-function Dem_SetEventStatus() shall be specified. According to the rules [TPS_BSWMDT_04008] and [TPS_BSWMDT_04016] defined earlier in this document, its shortName shall have the value Dem_SetEventStatus and the package location in XML shall be:

```
AUTOSAR_Dem/BswModuleEntrys/
```

Each BswServiceDependency representing a production error in a BSDWMD shall refer to this BswModuleEntry via an aggregated assignedEntryRole which has its role attribute set to the value ReportErrorStatus. ⌋*(RS_BSWMD_00045, RS_BSWMD_00069)*

Note that in order to model the complete picture, the module in question should also have a BswModuleDescription.bswModuleDependency.expectedEntry[2] referring to

```
AUTOSAR_Dem/BswModuleEntrys/Dem_SetEventStatus
```

and one more BswModuleCallPoints representing the calls into Dem_SetEventStatus(). This additional information is not mandatory to configure the DEM, but it can be used for documentation and call tree or timing analysis.

If the diagnostic event is associated with a callback routine to be called by the DEM and implemented by the module in question, this shall also be modeled by a BswModuleEntry which is referred as BswServiceDependency.assignedEntryRole. This holds namely for the standardized callback InitMonitorForEvent specified in [SWS_Dem_00256]:

[TPS_BSWMDT_04112] BswServiceDependency refers to InitMonitorForEvent ⌈ If a module implements the callback InitMonitorForEvent, a BswModuleEntry shall be defined with

- shortName = Service name as defined in [SWS_Dem_00256]

[2]This must be modeled differently, if the call crosses partition boundaries, see 5.6.2

The BswServiceDependency representing this diagnostic event shall refer to this BswModuleEntry via its assignedEntry and its assignedEntryRole shall have the value InitMonitorForEvent. ⌋*(RS_BSWMD_00045, RS_BSWMD_00069)*

Note that in order to model the complete picture for such a callback, the module in question should also have a BswModuleDescription.bswModuleDependency.implementedEntry[3] referring to the BswModuleEntry that describes the callback signature and a BswModuleEntity representing the implementation of the callback. This additional information is not mandatory to configure the DEM, but it can be used for documentation and call tree or timing analysis.

13.4 Error Tracer Needs

The meta-class ErrorTracerNeeds is used to define requirements in order to configure the Default Error Tracer and to implement the according transient fault handler.

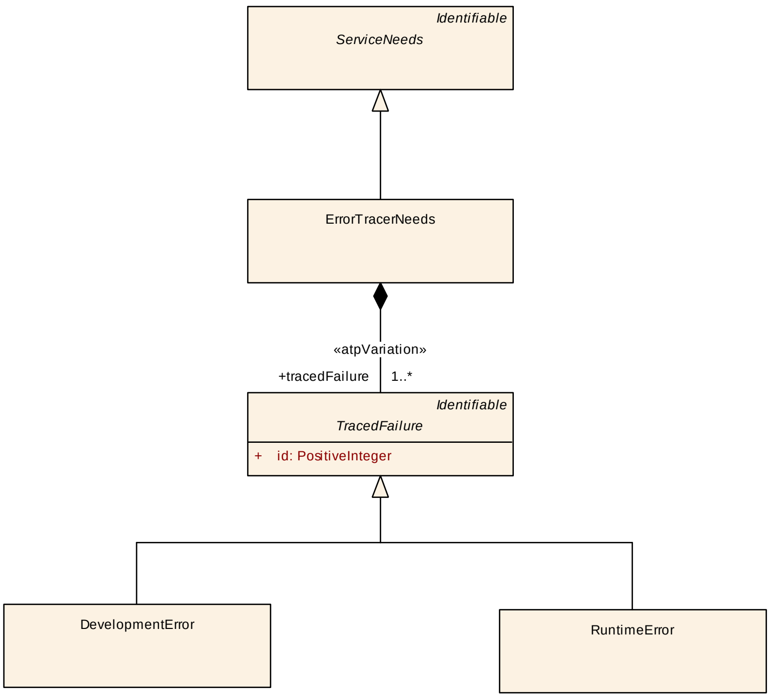

Figure 13.6: Modeling of ErrorTracerNeeds

[constr_4092] Number of ErrorTracerNeeds in BswInternalBehavior ⌈ A BswInternalBehavior shall provide at most one ErrorTracerNeeds element. ⌋*()*

This ErrorTracerNeeds element provides the exhaustive list of all tracedFailures implemented in the BSW module. Each tracedFailure relates to one ID. For more suggestion see Specification of Default Error Tracer [30].

[3]This must be modeled differently, if the call crosses partition boundaries, see 5.6.2

Class	**ErrorTracerNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the need to report failures to the error tracer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
tracedFailure	TracedFailure	1..*	aggr	list of traced failures **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 13.38: ErrorTracerNeeds

Class	***TracedFailure*** (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the ability to report a specific failure to the error tracer. The short name specifies the literal applicable for the Default Error Tracer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DevelopmentError, RuntimeError, TransientFault			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
id	PositiveInteger	1	attr	ID of detected failure used in reporting API as error or fault id.

Table 13.39: TracedFailure

Class	**DevelopmentError**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The reported failure is classified as development error.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TracedFailure*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 13.40: DevelopmentError

Class	**RuntimeError**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	The reported failure is classified as runtime error.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TracedFailure*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 13.41: RuntimeError

13.4.1 Default Error Tracer Service use Case: report failure

[TPS_BSWMDT_04152] Setup for Default Error Tracer Service use Case: report failure: ⌈ Scenario: a `Basic Software Module` reports a failure to the Default Error Tracer. In this case the following setup apply:

ServiceNeeds kind `ErrorTracerNeeds`

RoleBasedBswModuleEntryAssignment valid roles:

- Det_ReportError [0..1]
- Det_ReportRuntimeError [0..1]
- Det_ReportTransientFault [0..1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋()

13.5 Hardware Test Manager

This meta-class represents the ability to indicate that a Basic Software Module is interested in the results of the hardware test and will establish a BswModuleEntry to query the hardware test manager HtssM.

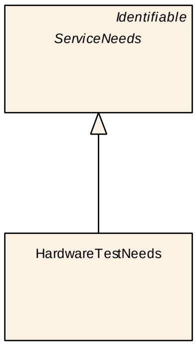

Figure 13.7: Modeling of HardwareTestNeeds

Class	**HardwareTestNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that a software-component is interested in the results of the hardware test and will establish a PortPrototype to query the hardware test manager.			
Base	*ARObject, Identifiable, MultilanguageReferrable, Referrable, ServiceNeeds*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 13.42: HardwareTestNeeds

13.5.1 HtssM Service Use Case: Query results of hardware tests

Scenario: a Basic Software Module offers a BswModuleEntry to query the results of hardware tests conducted by the HtssM.

[TPS_BSWMDT_04171] HtssM Service Use Case: Query results of hardware tests ⌈

ServiceNeeds kind : `HardwareTestNeeds`

RoleBasedBswModuleEntryAssignment valid roles:

- `GetTestStatus` [1]

RoleBasedDataAssignment
N/A

RoleBasedDataTypeAssignment
N/A

⌋*()*

A Constraint and Specification History

A.1 Constraint History of this Document according to AUTOSAR R4.0.1

A.1.1 Changed Constraints in R4.0.1

N/A

A.1.2 Added Constraints in R4.0.1

Number	Heading
[constr_4013]	BSW service identifier
[constr_4014]	Call type and execution context
[constr_4015]	calledEntry constraints
[constr_4016]	BswCalledEntity constraints
[constr_4017]	BswSchedulableEntity constraints
[constr_4018]	BswInterruptEntity constraints
[constr_4019]	BSW module identifier
[constr_4020]	Categories of BswModuleDescription
[constr_4021]	Implementation policy of function pointer target[1]
[constr_4022]	BswModuleEntry only uses the module's interface
[constr_4023]	External trigger must belong to the interface
[constr_4024]	Semantics of BSW mode switch event
[constr_4025]	Modes used by BSW mode switch event
[constr_4026]	Mode group used by BSW mode switch acknowledge event
[constr_4028]	Semantics of memory section type
[constr_4029]	Measured stack usage
[constr_4030]	Measured heap usage
[constr_4031]	Analyzed execution time
[constr_4032]	Measured execution time
[constr_4033]	Simulated execution time
[constr_4034]	Target and context of MC emulation reference
[constr_4036]	Entries linked to BswModuleDescription
[constr_4037]	Entries linked to BswModuleDependency
[constr_4038]	bswModuleDependency must refer to a different module
[constr_4039]	Semantics of SwcBswMapping
[constr_4040]	Synchronized mode groups must have same type
[constr_4041]	Synchronized mode groups must have same context
[constr_4042]	Synchronized triggers must have same context
[constr_4043]	Period of BswTimingEvent
[constr_4044]	Content of McSwEmulationMethodSupport
[constr_4045]	implementationConfigVariant of preconfigured configuration
[constr_4046]	implementationConfigVariant of recommended configuration

Table A.1: Added Constraints in R4.0.1

[1] this constraint was by mistake named **Bsw service identifier** in R4.0.1 and R4.0.2

A.1.3 Deleted Constraints

N/A

A.2 Constraint History of this Document according to AUTOSAR R4.0.2

A.2.1 Changed Constraints in R4.0.2

N/A

A.2.2 Added Constraints in R4.0.2

Number	Heading
[constr_4047]	Multiplicity of vendor specific configuration parameters
[constr_4048]	Multiplicity of preconfigured values

Table A.2: Added Constraints in R4.0.2

A.2.3 Deleted Constraints in R4.0.2

N/A

A.3 Constraint and Specification History of this Document according to AUTOSAR R4.0.3

A.3.1 Changed Constraints in R4.0.3

N/A

A.3.2 Added Specification Items in R4.0.3

Number	Heading
[TPS_BSWMDT_04000]	BSW modules with AUTOSAR Interfaces
[TPS_BSWMDT_04001]	Attaching SwComponentDocumentation to a BSWMD
[TPS_BSWMDT_04002]	Usage of BswModuleEntry
[TPS_BSWMDT_04003]	BswModuleDependency
[TPS_BSWMDT_04004]	BswModuleDescription.providedModeGroup
[TPS_BSWMDT_04005]	BswModuleDescription.releasedTrigger
[TPS_BSWMDT_04006]	BswModuleDescription.internalBehavior
[TPS_BSWMDT_04007]	BswModuleEntry

[TPS_BSWMDT_04008]	C-symbol of BswModuleEntry
[TPS_BSWMDT_04009]	Usage of SwServiceArg
[TPS_BSWMDT_04010]	SwServiceArg.swDataDefProps.implementationDataType
[TPS_BSWMDT_04011]	SwServiceArg.swDataDefProps.swPointerTargetProps
[TPS_BSWMDT_04012]	SwServiceArg.direction
[TPS_BSWMDT_04014]	ModeRequestTypeMap in BSW
[TPS_BSWMDT_04015]	Usage of Trigger in BSW
[TPS_BSWMDT_04016]	Location of standardized BswModuleEntrys
[TPS_BSWMDT_04017]	Reference to standardized BswModuleEntry-s
[TPS_BSWMDT_04018]	BswDirectCallPoint.calledEntry
[TPS_BSWMDT_04019]	BswModuleEntity attributes
[TPS_BSWMDT_04020]	Usage of BswSchedulerNamePrefix
[TPS_BSWMDT_04021]	Usage of BswEvent
[TPS_BSWMDT_04022]	Timing and background events for BSW
[TPS_BSWMDT_04023]	Internal trigger and timing events for BSW
[TPS_BSWMDT_04024]	External trigger event for BSW
[TPS_BSWMDT_04025]	Mode switches and events in BSW
[TPS_BSWMDT_04026]	Local BSW data without RTE or BSW Scheduler support
[TPS_BSWMDT_04027]	Local BSW data accessed via BSW Scheduler API
[TPS_BSWMDT_04028]	Determination of argument names for BSW functions called via ports
[TPS_BSWMDT_04029]	Usage of BswServiceDependency
[TPS_BSWMDT_04030]	BswImplementation.arReleaseVersion
[TPS_BSWMDT_04031]	Instances of BswImplementation
[TPS_BSWMDT_04032]	Implementation.hwElement
[TPS_BSWMDT_04033]	Reference to vendor specific configuration parameters
[TPS_BSWMDT_04034]	Reference to predefined or recommended configuration values
[TPS_BSWMDT_04035]	Published parameter values
[TPS_BSWMDT_04036]	Back-reference from EcucModuleConfigurationValues
[TPS_BSWMDT_04039]	Association of an Implementation with a component or module
[TPS_BSWMDT_04040]	Implementation.codeDescriptor
[TPS_BSWMDT_04041]	DependencyOnArtifact
[TPS_BSWMDT_04042]	Usage of DependencyOnArtifact
[TPS_BSWMDT_04043]	Compiler
[TPS_BSWMDT_04044]	Linker
[TPS_BSWMDT_04045]	Implementation.resourceConsumption
[TPS_BSWMDT_04046]	Memory section name
[TPS_BSWMDT_04047]	Memory section prefix
[TPS_BSWMDT_04048]	Scope of declared memory sections
[TPS_BSWMDT_04049]	Usage of MemorySection.executableEntity
[TPS_BSWMDT_04050]	ExecutionTime
[TPS_BSWMDT_04051]	ExecutionTime references an ECU
[TPS_BSWMDT_04052]	ExecutionTime.hardwareConfiguration
[TPS_BSWMDT_04053]	ExecutionTime.memorySectionLocation
[TPS_BSWMDT_04054]	ExecutionTime.softwareContext
[TPS_BSWMDT_04055]	ExecutionTime.includedLibrary
[TPS_BSWMDT_04056]	Multiplicity of McSupportData
[TPS_BSWMDT_04057]	Self-contained MC support artifact
[TPS_BSWMDT_04058]	McSupportData.measurableSystemConstantValues
[TPS_BSWMDT_04059]	Granularity of McDataInstance.subElements
[TPS_BSWMDT_04060]	McDataInstance.resultingProperties
[TPS_BSWMDT_04061]	McSwEmulationMethodSupport.category

[TPS_BSWMDT_04062]	Upstream reference for emulation support
[TPS_BSWMDT_04063]	BSW Interface Variation Points
[TPS_BSWMDT_04064]	BSW Behavior Variation Points
[TPS_BSWMDT_04065]	BSW Implementation Variation Points
[TPS_BSWMDT_04066]	Relevant elements for ICS on Interface level
[TPS_BSWMDT_04067]	No relevant elements for ICS on Internal Behavior level
[TPS_BSWMDT_04068]	Relevant elements for ICS on Implementation level
[TPS_BSWMDT_04069]	Configuration in ICS
[TPS_BSWMDT_04070]	No variants in ICS

Table A.3: Added Specification Items in 4.0.3

A.3.3 Added Constraints in R4.0.3

Number	Heading
[constr_4051]	RoleBasedDataAssignment in BSW
[constr_4052]	BswModuleEntry returnType direction
[constr_4053]	BswModuleEntry argument direction
[constr_4054]	Unambiguous links to addressing method
[constr_4056]	BswModuleEntry with no returnType
[constr_4057]	BswModuleEntry with no argument
[constr_4058]	Different mode groups in mapped BSWM and SWC must have different names
[constr_4059]	Different mode groups referred by a BSWM must have different names
[constr_4060]	Allowed values of Trigger.swImplPolicy for BSW
[constr_4061]	Completeness of MC emulation reference
[constr_4062]	Mandatory symbol for McDataInstance root
[constr_4063]	Restrictions of ModeRequestTypeMap in BSW
[constr_4064]	Synchronized triggers must implement same policy
[constr_4065]	Allowed values of BswInternalTriggeringPoint.swImplPolicy

Table A.4: Added Constraints in R4.0.3

A.3.4 Deleted Constraints in R4.0.3

N/A

A.4 Constraint and Specification History of this Document according to AUTOSAR R4.1.1

A.4.1 Changed Specification Items in R4.1.1

Number	Heading
[TPS_BSWMDT_04021]	Usage of BswEvent
[TPS_BSWMDT_04025]	Mode switches and events in BSW
[TPS_BSWMDT_04057]	Self-contained MC support artifact
[TPS_BSWMDT_04063]	BSW Interface Variation Points
[TPS_BSWMDT_04064]	BSW Behavior Variation Points

Table A.5: Changed Specification Items in 4.1.1

A.4.2 Changed Constraints in R4.1.1

Number	Heading
[constr_4015]	calledEntry constraints for direct calls
[constr_4022]	BswModuleEntry only uses the module's interface

Table A.6: Changed Constraints in R4.1.1

A.4.3 Added Specification Items in R4.1.1

Number	Heading
[TPS_BSWMDT_04071]	Usage of module identifier and category
[TPS_BSWMDT_04072]	Executable entity in BSW
[TPS_BSWMDT_04073]	Exclusive area in BSW
[TPS_BSWMDT_04074]	Synchronization of mode switches or triggers
[TPS_BSWMDT_04075]	RunnableEntity in BSW for RTE access
[TPS_BSWMDT_04126]	General meta-model methodology
[TPS_BSWMDT_04076]	ECUC features
[TPS_BSWMDT_04077]	Timing requirements and guarantees
[TPS_BSWMDT_04078]	Semantics of McFunction
[TPS_BSWMDT_04079]	Usage of module shortName
[TPS_BSWMDT_04080]	Options for inline code sections
[TPS_BSWMDT_04081]	ExclusiveAreaNestingOrder
[TPS_BSWMDT_04082]	Indicate that the locking behavior is fully described for BswModuleEntity
[TPS_BSWMDT_04083]	Locking behavior is not described for BswModuleEntity-s
[TPS_BSWMDT_04084]	Relation of BswModuleCallPoint to ExclusiveAreaNestingOrder
[TPS_BSWMDT_04085]	Implementation refers to a BuildActionManifest
[TPS_BSWMDT_04086]	Artifacts referred in Implementation and/or BuildActionManifest
[TPS_BSWMDT_04087]	Scope of McFunctionDataRefSets
[TPS_BSWMDT_04088]	Usage of McFunction
[TPS_BSWMDT_04089]	Access to activation reason
[TPS_BSWMDT_04090]	Variation Points for BswModuleEntry arguments
[TPS_BSWMDT_04091]	Function signature containing the keyword **enum** in C
[TPS_BSWMDT_04092]	Provide memory mapping header file names
[TPS_BSWMDT_04093]	Memory classes for compiler abstraction
[TPS_BSWMDT_04094]	Details of McDataInstance for rapid prototyping
[TPS_BSWMDT_04095]	Relationships between McDataInstances
[TPS_BSWMDT_04096]	Split between different use cases of McSupportData
[TPS_BSWMDT_04097]	Assigning different header files per section prefix
[TPS_BSWMDT_04098]	Declaration of BswModuleClientServerEntry
[TPS_BSWMDT_04099]	Semantics of BswModuleClientServerEntry attributes
[TPS_BSWMDT_04100]	Different ways of referring BswModuleEntry
[TPS_BSWMDT_04101]	Declaration of providedData and requiredData
[TPS_BSWMDT_04102]	Usage of BswSynchronousServerCallPoint
[TPS_BSWMDT_04103]	BswModuleEntity reentrancy level
[TPS_BSWMDT_04104]	Usage of BswAsynchronousServerCallPoint
[TPS_BSWMDT_04105]	Usage of BswAsynchronousServerCallResultPoint
[TPS_BSWMDT_04106]	BswModuleEntity attributes for sender-receiver data exchange
[TPS_BSWMDT_04107]	Data reception policy

[TPS_BSWMDT_04108]	BswInternalBehavior containing BswModuleEntity-s executed on different partitions
[TPS_BSWMDT_04109]	BswInternalBehavior for the same AUTOSAR Service provided on different partitions
[TPS_BSWMDT_04110]	Declaration of production errors
[TPS_BSWMDT_04111]	BswServiceDependency refers to Dem_SetEventStatus()
[TPS_BSWMDT_04112]	BswServiceDependency refers to InitMonitorForEvent
[TPS_BSWMDT_04113]	Rule for setting RoleBasedPortAssignment.role
[TPS_BSWMDT_04114]	Use the hierarchical structuring of McDataInstance.subElements
[TPS_BSWMDT_04115]	Use of indexing for array element of subElements

Table A.7: Added Specification Items in 4.1.1

A.4.4 Added Constraints in R4.1.1

Number	Heading
[constr_1275]	Applicability of reference startsOnEvent for BswScheduleEvent
[constr_1276]	Applicability of reference startsOnEvent for BswOperationInvokedEvent
[constr_4066]	BswModeSwitchEvent and the definition of ModeTransition
[constr_4067]	Exclusive usage of data references in McFunctionDataRefSet
[constr_4068]	Semantics of McFunctionDataRefSet.flatInstanceDescriptor
[constr_4069]	Semantics of McFunctionDataRefSet.mcDataInstance
[constr_4070]	Applicability of BswModuleEntity.activationReason
[constr_4071]	Synchronized runnables and schedulable entities must be consistent
[constr_4072]	Constraints of SectionNamePrefix.implementedIn
[constr_4073]	McDataAccessDetails shall refer to one ECU Extract
[constr_4074]	Compatibility of BswModuleClientServerEntry-s
[constr_4075]	Constraints for providedData and requiredData
[constr_4076]	Constraints on BswModuleEntry used for Client-Server
[constr_4077]	Constraints for BswModuleEntity.reentrancyLevel
[constr_4078]	Consistent usage of BswOperationInvokedEvent
[constr_4079]	calledEntry constraints for client-server calls
[constr_4080]	Existence of reception policy
[constr_4081]	Mode group used by BSW mode manager error event
[constr_4083]	BswDistinguishedPartition shall be used only in the context of a particular BswInternalBehavior
[constr_4084]	Consistency of references of InternalBehavior
[constr_4085]	Consistency of references of InternalBehavior

Table A.8: Added Constraints in R4.1.1

A.4.5 Deleted Specification Items in R4.1.1

N/A

A.4.6 Deleted Constraints in R4.1.1

N/A

A.5 Constraint History of this Document according to AUTOSAR R4.2.1

A.5.1 Changed Constraints in R4.2.1

N/A

A.5.2 Added Constraints in R4.2.1

Number	Heading
[constr_4086]	invocation of ExecutableEntitys by direct function call dependent from BswExecutionContext
[constr_4087]	Usage of category "MACRO"
[constr_4088]	Existence of RoleBasedDataTypeAssignment.role vs. RoleBasedDataAssignment.role

Table A.9: Added Constraints in R4.2.1

A.5.3 Deleted Constraints in R4.2.1

N/A

A.5.4 Changed Specification Items in R4.2.1

Number	Heading
[TPS_BSWMDT_04113]	Rule for setting RoleBasedBswModuleEntryAssignment.role

Table A.10: Changed Specification Items in 4.2.1

A.5.5 Added Specification Items in R4.2.1

Number	Heading
[TPS_ BSWMDT_04116]	Setup for Nvm Use Case: Permanent RAM Block
[TPS_ BSWMDT_04117]	Setup for Nvm Use Case: Temporary RAM Block
[TPS_ BSWMDT_04118]	Setup for Nvm Use Case: RAM Block synchronised with explicit synchronisation
[TPS_BSWMDT_04119]	Setup for Function Inhibition Manager Service use Case: read function permission
[TPS_BSWMDT_04120]	Basic Software Module implements a Diagnostic Monitor
[TPS_BSWMDT_04121]	Basic Software Module offers BswModuleEntrys to read/write current value via diagnostic services
[TPS_BSWMDT_04122]	Basic Software Module offers BswModuleEntrys to start/stop or request routines via diagnostic services
[TPS_BSWMDT_04123]	Basic Software Module offers BswModuleEntrys BswModuleEntrys to adjust the IO signal via diagnostic services

[TPS_BSWMDT_04124]	Basic Software Module offers BswModuleEntrys to access protocol, session and security information
[TPS_BSWMDT_04125]	Basic Software Module offers BswModuleEntrys for the Seed adn Key handling for security level access

Table A.11: Added Specification Items in 4.2.1

A.5.6 Deleted Specification Items in R4.2.1

N/A

A.6 Constraint History of this Document according to AUTOSAR R4.2.2

A.6.1 Added Traceables in 4.2.2

Id	Heading
[TPS_BSWMDT_04076]	ECUC features
[TPS_BSWMDT_04077]	Timing requirements and guarantees
[TPS_BSWMDT_04116]	Setup for Nvm Use Case: Permanent RAM Block
[TPS_BSWMDT_04117]	Setup for Nvm Use Case: Temporary RAM Block
[TPS_BSWMDT_04118]	Setup for Nvm Use Case: RAM Block synchronised with explicit synchronization
[TPS_BSWMDT_04126]	General meta-model methodology
[TPS_BSWMDT_04127]	Callback header declarations
[TPS_BSWMDT_04128]	BSW measurement data accessed via BSW Scheduler API

Table A.12: Added Traceables in 4.2.2

A.6.2 Changed Traceables in 4.2.2

Id	Heading
[TPS_BSWMDT_04027]	Local BSW data accessed via BSW Scheduler API

Table A.13: Changed Traceables in 4.2.2

A.6.3 Deleted Traceables in 4.2.2

Id	Heading
[TPS_ BSWMDT_04116]	Setup for Nvm Use Case: Permanent RAM Block
[TPS_ BSWMDT_04117]	Setup for Nvm Use Case: Temporary RAM Block
[TPS_ BSWMDT_04118]	Setup for Nvm Use Case: RAM Block synchronised with explicit synchronization
[TPS_BSWMDT_GEN]	General meta-model methodology
[TPS_BSWMDT_GEN_04076]	ECUC features
[TPS_BSWMDT_GEN_04077]	Timing requirements and guarantees

Table A.14: Deleted Traceables in 4.2.2

A.6.4 Added Constraints in 4.2.2

Id	Heading
[constr_4089]	Association callbackHeader is only applicable for BSW modules
[constr_4090]	The callbackHeader reference has to be consistent with behavior reference

Table A.15: Added Constraints in 4.2.2

A.6.5 Changed Constraints in 4.2.2

none

A.6.6 Deleted Constraints in 4.2.2

none

A.7 Constraint History of this Document according to AUTOSAR R4.3.0

A.7.1 Added Traceables in 4.3.0

Id	Heading
[TPS_BSWMDT_04129]	Definition a Supervised Entity in a Basic Software Module
[TPS_BSWMDT_04130]	Linkage of BswModuleEntry
[TPS_BSWMDT_04131]	Basic Software Module reads the current ECU mode (fixed variant)
[TPS_BSWMDT_04132]	Basic Software Module shall keep the ECU alive (fixed variant)
[TPS_BSWMDT_04133]	Basic Software Module wants to select a shutdown target (fixed variant)
[TPS_BSWMDT_04134]	Basic Software Module wants to select a boot target (fixed variant)
[TPS_BSWMDT_04135]	Basic Software Module wants to select a shutdown target (flexible variant)
[TPS_BSWMDT_04136]	Basic Software Module wants to select a boot target (flexible variant)
[TPS_BSWMDT_04137]	Basic Software Module wants to use an alarm clock (flexible variant)
[TPS_BSWMDT_04138]	Determination of the BswModuleEntry symbol
[TPS_BSWMDT_04139]	Dem Use Case: Bsw Module implements a hardware shutdown
[TPS_BSWMDT_04140]	AccessCount.value describes an intrinsic property
[TPS_BSWMDT_04141]	The attribute countProfile denotes the counting rules
[TPS_BSWMDT_04142]	Standardized values of attribute countProfile

[TPS_BSWMDT_04143]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Explicit Communication, single access
[TPS_BSWMDT_04144]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Explicit Communication, multiple accesses
[TPS_BSWMDT_04145]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Implicit Communication and parameter accesses, single access
[TPS_BSWMDT_04146]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Implicit Communication and parameter accesses, multiple accesses
[TPS_BSWMDT_04147]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Server calls, issued Triggers, Mode Switch Notifications, single access
[TPS_BSWMDT_04148]	countProfile DISTINGUISH_SINGULAR_ACCESSES, Server calls, issued Triggers, Mode Switch Notifications, multiple accesses
[TPS_BSWMDT_04149]	Structuring according ExecutableEntitys
[TPS_BSWMDT_04150]	Structuring according Variants
[TPS_BSWMDT_04151]	Structuring according different countProfile definitions
[TPS_BSWMDT_04152]	Setup for Default Error Tracer Service use Case: report failure:
[TPS_BSWMDT_04153]	Usage of BswModuleEntry
[TPS_BSWMDT_04154]	ExclusiveArea is entered and exit by common set of API
[TPS_BSWMDT_04155]	ExclusiveArea is entered and exit by individual set of API
[TPS_BSWMDT_04156]	Usage of functionPrototypeEmitter
[TPS_BSWMDT_04157]	Definition of Checkpoints for a Supervised Entity in a Basic Software Module
[TPS_BSWMDT_04158]	Setup for a Basic Software Module which sets or gets Global Supervision Status
[TPS_BSWMDT_04159]	Standardized values of attribute RoleBasedMcDataAssignment.role
[TPS_BSWMDT_04160]	RptComponent represents a software component or basic software module
[TPS_BSWMDT_04161]	RptExecutableEntity represents a ExecutableEntity with rapid prototyping support
[TPS_BSWMDT_04162]	RptExecutableEntityEvent represents a RTEEvent or BswEvent for with rapid prototyping support
[TPS_BSWMDT_04163]	RptExecutionContext represents a common environment for a set of RptExecutableEntitys or McDatainstances
[TPS_BSWMDT_04164]	Description of implicit communication buffers

Table A.16: Added Traceables in 4.3.0

A.7.2 Changed Traceables in 4.3.0

Id	Heading
[TPS_BSWMDT_04002]	Provision of BswModuleEntry
[TPS_BSWMDT_04010]	SwServiceArg.swDataDefProps.implementationDataType
[TPS_BSWMDT_04011]	SwServiceArg.swDataDefProps.swPointerTargetProps
[TPS_BSWMDT_04016]	Location of standardized BswModuleEntry-s
[TPS_BSWMDT_04017]	Reference to standardized BswModuleEntry-s
[TPS_BSWMDT_04025]	Mode switches and events in BSW
[TPS_BSWMDT_04026]	Local BSW data without RTE or BSW Scheduler support
[TPS_BSWMDT_04066]	Relevant elements for ICS on Interface level
[TPS_BSWMDT_04087]	Scope of McFunctionDataRefSets
[TPS_BSWMDT_04100]	Different ways of referring BswModuleEntry
[TPS_BSWMDT_04111]	BswServiceDependency refers to Dem_SetEventStatus()
[TPS_BSWMDT_04120]	Basic Software Module implements a Diagnostic Monitor

[TPS_BSWMDT_04122]	Basic Software Module offers BswModuleEntrys to start/stop or request routines via diagnostic services
[TPS_BSWMDT_04128]	BSW measurement data accessed via BSW Scheduler API

Table A.17: Changed Traceables in 4.3.0

A.7.3 Deleted Traceables in 4.3.0

Id	Heading
[TPS_BSWMDT_04003]	BswModuleDependency
[TPS_BSWMDT_04037]	BswDebugInfo
[TPS_BSWMDT_04038]	Data types for debug data

Table A.18: Deleted Traceables in 4.3.0

A.7.4 Added Constraints in 4.3.0

Id	Heading
[constr_4091]	AccessCount.value needs to be unambiguous
[constr_4092]	Number of ErrorTracerNeeds in BswInternalBehavior
[constr_4093]	Entries linked to BswModuleEntrys shall have compatible signature
[constr_4094]	compatibility of SwServiceArg in role returnType
[constr_4095]	Compatibility of SwServiceArg in role argument
[constr_4096]	Matching BswModuleEntrys should have compatible attributes
[constr_4097]	Limitation on the number of BswExclusiveAreaPolicys

Table A.19: Added Constraints in 4.3.0

A.7.5 Changed Constraints in 4.3.0

Id	Heading
[constr_4015]	calledEntry constraints for direct calls
[constr_4020]	Categories of BswModuleDescription
[constr_4021]	Implementation policy of function pointer target
[constr_4022]	BswModuleEntity only uses the module's interface
[constr_4071]	Synchronized runnables and schedulable entities must be consistent
[constr_4077]	Constraints for BswModuleEntity.reentrancyLevel
[constr_4079]	calledEntry constraints for client-server calls
[constr_4086]	invocation of ExecutableEntitys by direct function call dependent from BswExecutionContext

Table A.20: Changed Constraints in 4.3.0

A.7.6 Deleted Constraints in 4.3.0

Id	Heading
[constr_4036]	Entries linked to BswModuleDescription
[constr_4037]	Entries linked to ARMetaClassBswModuleDependency

Table A.21: Deleted Constraints in 4.3.0

— AUTOSAR CONFIDENTIAL —

A.8 Constraint History of this Document according to AUTOSAR R4.3.1

A.8.1 Added Traceables in 4.3.1

Number	Heading
[TPS_BSWMDT_04165]	Basic Software Module offers BswModuleEntrys to read value via OBD services
[TPS_BSWMDT_04166]	Basic Software Module offers BswModuleEntrys to read vehicle information via OBD services
[TPS_BSWMDT_04167]	Setup for Function Inhibition Manager Service use Case: read function permission

Table A.22: Added Traceables in 4.3.1

A.8.2 Changed Traceables in 4.3.1

Number	Heading
[TPS_BSWMDT_04125]	Basic Software Module offers BswModuleEntrys for the Seed adn Key handling for security level access and the optional security attempt counter handling

Table A.23: Changed Traceables in 4.3.1

A.8.3 Deleted Traceables in 4.3.1

none

A.8.4 Added Constraints in 4.3.1

Number	Heading
[constr_4098]	No mode disabling for BswOperationInvokedEvent

Table A.24: Added Constraints in 4.3.1

A.8.5 Changed Constraints in 4.3.1

none

A.8.6 Deleted Constraints in 4.3.1

none

A.9 Constraint History of this Document according to AUTOSAR R4.4.0

A.9.1 Added Traceables in 4.4.0

Number	Heading
[TPS_BSWMDT_04168]	Semantics of McGroup
[TPS_BSWMDT_04169]	Scope of McGroupDataRefSets
[TPS_BSWMDT_04170]	Usage of McGroup
[TPS_BSWMDT_04171]	HtssM Service Use Case: Query results of hardware tests
[TPS_BSWMDT_04172]	Basic Software Module implements the ability to accept data for upload and/or provide data for download. For this purpose the Basic Software Module provides a BswModuleEntry that connects to the Dcm service component.

Table A.25: Added Traceables in 4.4.0

A.9.2 Changed Traceables in 4.4.0

Number	Heading
[TPS_BSWMDT_04032]	Implementation.hwElement

Table A.26: Changed Traceables in 4.4.0

A.9.3 Deleted Traceables in 4.4.0

Number	Heading
[TPS_BSWMDT_04131]	Basic Software Module reads the current ECU mode (fixed variant)
[TPS_BSWMDT_04132]	Basic Software Module shall keep the ECU alive (fixed variant)
[TPS_BSWMDT_04133]	Basic Software Module wants to select a shutdown target (fixed variant)
[TPS_BSWMDT_04134]	Basic Software Module wants to select a boot target (fixed variant)

Table A.27: Deleted Traceables in 4.4.0

— AUTOSAR CONFIDENTIAL —

A.9.4 Added Constraints in 4.4.0

Number	Heading
[constr_4099]	Support of multiple instantiation
[constr_4100]	Uniqueness of module implementation prefixes
[constr_4101]	Semantics of McGroupDataRefSet.flatInstanceDescriptor
[constr_4102]	Semantics of McGroupDataRefSet.mcDataInstance
[constr_4103]	Name convention for SectionNamePrefix
[constr_4104]	Referencing of MemorySections to SectionNamePrefix

Table A.28: Added Constraints in 4.4.0

A.9.5 Changed Constraints in 4.4.0

Number	Heading
[constr_4068]	McFunctionDataRefSet.flatInstanceDescriptor's semantic
[constr_4069]	McFunctionDataRefSet.mcDataInstance's semantic
[constr_4071]	Synchronized runnables and schedulable entities must be consistent

Table A.29: Changed Constraints in 4.4.0

A.9.6 Deleted Constraints in 4.4.0

Number	Heading
[constr_4067]	Exclusive usage of data references in McFunctionDataRefSet

Table A.30: Deleted Constraints in 4.4.0

B Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	***ARElement*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	*ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, ApplicationPartition, *AutosarDataType*, *BaseType*, BlueprintMappingSet, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, BuildActionManifest, CalibrationParameterValueSet, ClientIdDefinitionSet, ClientServerInterfaceToBswModuleEntryBlueprintMapping, Collection, CompuMethod, ConsistencyNeedsBlueprintSet, ConstantSpecification, ConstantSpecificationMappingSet, CryptoServiceCertificate, CryptoServiceKey, CryptoServicePrimitive, DataConstr, DataExchangePoint, DataTransformationSet, DataTypeMappingSet, *DiagnosticCommonElement*, DiagnosticConnection, DiagnosticContributionSet, DiagnosticMasterToSlaveEventMappingSet, Documentation, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleConfigurationValues, EcucModuleDef, EcucValueCollection, EndToEndProtectionSet, EvaluatedVariantSet, FMFeature, FMFeatureMap, FMFeatureModel, FMFeatureSelectionSet, FlatMap, GeneralPurposeConnection, HwCategory, HwElement, HwType, IPv6ExtHeaderFilterSet, *Implementation*, InterpolationRoutineMappingSet, J1939ControllerApplication, KeywordSet, LifeCycleInfoSet, LifeCycleStateDefinitionGroup, McFunction, McGroup, ModeDeclarationGroup, ModeDeclarationMappingSet, PhysicalDimension, PhysicalDimensionMappingSet, *PortInterface*, PortInterfaceMappingSet, PortPrototypeBlueprint, PostBuildVariantCriterion, PostBuildVariantCriterionValueSet, PredefinedVariant, RapidPrototypingScenario, SdgDef, SwAddrMethod, SwAxisType, *SwComponentType*, SwRecordLayout, SwSystemconst, SwSystemconstantValueSet, SwcBswMapping, System, SystemSignal, SystemSignalGroup, TcpOptionFilterSet, *TimingExtension*, TransformationPropsSet, Unit, UnitGroup, ViewMapSet			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table B.1: ARElement

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	*ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30

▽

△

Class	**ARPackage**			
element	PackageableElement	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortLabel xml.sequenceOffset=10

Table B.2: ARPackage

Enumeration	**AdditionalBindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumeration specifies the additional binding times applicable for vh.latestBindingTime of variation points.
Literal	***Description***
blueprintDerivationTime	The point in time when an object is created from a blueprint. **Tags:** atp.EnumerationValue=0
postBuild	After the executable has been built. **Tags:** atp.EnumerationValue=1

Table B.3: AdditionalBindingTimeEnum

Class	**AliasNameSet**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	This meta-class represents a set of AliasNames. The AliasNameSet can for example be an input to the A2L-Generator. **Tags:** atp.recommendedPackage=AliasNameSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
aliasName	AliasNameAssignment	1..*	aggr	AliasNames contained in the AliasNameSet. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortLabel vh.latestBindingTime=preCompileTime

Table B.4: AliasNameSet

Class	***ApplicationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationData Types only.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*ApplicationCompositeDataType*, ApplicationPrimitiveDataType			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table B.5: ApplicationDataType

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
direction	ArgumentDirection Enum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgument ImplPolicy	ServerArgumentImpl PolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures.

Table B.6: ArgumentDataPrototype

Class	**AsynchronousServerCallResultPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a AsynchronousServerCallResultPoint it is entitled to get the result of the referenced AsynchronousServerCallPoint. If it is associated with AsynchronousServerCallReturnsEvent, this RTEEvent notifies the completion of the required ClientServerOperation or a timeout. The occurrence of this event can either unblock a Wait Point or can lead to the invocation of a RunnableEntity.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
asynchronous ServerCallPoint	AsynchronousServer CallPoint	1	ref	The referenced Asynchronous Server Call Point defines the asynchronous server call from which the results are returned.

Table B.7: AsynchronousServerCallResultPoint

Class	***AtomicSwComponentType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Subclasses	ApplicationSwComponentType, ComplexDeviceDriverSwComponentType, EcuAbstractionSwComponentType, NvBlockSwComponentType, SensorActuatorSwComponentType, ServiceProxySwComponentType, ServiceSwComponentType			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=internalBehavior, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table B.8: AtomicSwComponentType

Class	***AtpBlueprint*** (abstract)			
Package	M2::AUTOSARTemplates::StandardizationTemplate::AbstractBlueprintStructure			
Note	This meta-class represents the ability to act as a Blueprint. As this class is an abstract one, particular blueprint meta-classes inherit from this one.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractImplementationDataType*, AclObjectSet, AclOperation, AclPermission, AclRole, AliasNameSet, *ApplicationDataType*, BswEntryRelationshipSet, BswModuleDescription, BswModuleEntry, *BuildActionEntity*, BuildActionEnvironment, BuildActionManifest, ClientServerInterfaceToBswModuleEntryBlueprintMapping, CompuMethod, ConsistencyNeeds, DataConstr, DataTypeMappingSet, EcucDefinitionCollection, EcucDestinationUriDefSet, EcucModuleDef, FlatMap, KeywordSet, LifeCycleState, LifeCycleStateDefinitionGroup, ModeDeclarationGroup, *PortInterface*, *PortInterfaceMapping*, PortInterfaceMappingSet, PortPrototypeBlueprint, SwAddrMethod, SwBaseType, *SwComponentType*, VfbTiming			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
blueprintPolicy	BlueprintPolicy	*	aggr	This role indicates whether the blueprintable element will be modifiable or not motifiable.

Table B.9: AtpBlueprint

Class	***AutosarDataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ArgumentDataPrototype, ParameterDataPrototype, VariableDataPrototype			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
type	AutosarDataType	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table B.10: AutosarDataPrototype

Class	AutosarParameterRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents a reference to a parameter within AUTOSAR which can be one of the following use cases: localParameter: • localParameter which is used as whole (e.g. sharedAxis for curve) autosarVariable: • a parameter provided via PortPrototype which is used as whole (e.g. parameterAccess) • an element inside of a composite local parameter typed by ApplicationDatatype (e.g. sharedAxis for a curve) • an element inside of a composite parameter provided via Port and typed by ApplicationDatatype (e.g. sharedAxis for a curve) autosarParameterInImplDatatype: • an element inside of a composite local parameter typed by ImplementationDatatype • an element inside of a composite parameter provided via PortPrototype and typed by ImplementationDatatype			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
autosar Parameter	DataPrototype	0..1	iref	This instance reference is used if the callibration parameter is either imported via a port or is part of a composite data structure.
localParameter	DataPrototype	0..1	ref	In the majority of cases this reference goes to Parameter DataPrototypes rather than VariableDataPrototypes. Pointing the reference to a VariableDataPrototype is limited to special use cases, e.g. if the AutosarParameter Ref is used in the context of an SwAxisGrouped. This reference is used if the arParameter is local to the current component. Of course, it would technically also be feasible to use an InstanceRef for this case. However, the InstanceRef would not have a contextElement (because the current instance is the context). Hence, the local instance is a special case which may provide further optimization. Therefore an explicit reference is provided for this case.

Table B.11: AutosarParameterRef

Class	AutosarVariableRef
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements
Note	This class represents a reference to a variable within AUTOSAR which can be one of the following use cases: localVariable: • localVariable which is used as whole (e.g. InterRunnableVariable, inputValue for curve) autosarVariable: • a variable provided via Port which is used as whole (e.g. dataAccesspoints) • an element inside of a composite local variable typed by ApplicationDatatype (e.g. inputValue for a curve) ▽

▽

△

Class	**AutosarVariableRef**			
	△ • an element inside of a composite variable provided via Port and typed by ApplicationDatatype (e.g. inputValue for a curve) autosarVariableInImplDatatype: • an element inside of a composite local variable typed by ImplementationDatatype (e.g. nvram Data mapping) • an element inside of a composite variable provided via Port and typed by Implementation Datatype (e.g. inputValue for a curve)			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
autosarVariable	DataPrototype	0..1	iref	This references a variable which is provided by a port and/or which is part of a CompositeDataType.
autosarVariable InImplDatatype	ArVariableIn ImplementationData InstanceRef	0..1	aggr	This is used if the target variable is inside of variableData Prototype typed by an ImplementationDataType.
localVariable	VariableDataPrototype	0..1	ref	This reference is used if the variable is local to the current component. It would also be possible to use the instance refence here. Such an instance ref would not have a contextElement, since the current instance is the context. But the local instance is a special case which may provide further optimization. Therefore an expclicit reference is provided for this case.

Table B.12: AutosarVariableRef

Enumeration	**BindingTimeEnum**
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling
Note	This enumerator specifies the applicable binding times for the pre build variation points.
Literal	***Description***
codeGeneration Time	• Coding by hand, based on requirements document. • Tool based code generation, e.g. from a model. • The model may contain variants. • Only code for the selected variant(s) is actually generated. **Tags:** atp.EnumerationValue=0
linkTime	Configure what is included in object code, and what is omitted Based on which variant(s) are selected E.g. for modules that are delivered as object code (as opposed to those that are delivered as source code) **Tags:** atp.EnumerationValue=1
preCompileTime	This is typically the C-Preprocessor. Exclude parts of the code from the compilation process, e.g., because they are not required for the selected variant, because they are incompatible with the selected variant, because they require resources that are not present in the selected variant. Object code is only generated for the selected variant(s). The code that is excluded at this stage code will not be available at later stages. **Tags:** atp.EnumerationValue=2
systemDesignTime	• Designing the VFB. • Software Component types (PortInterfaces). • SWC Prototypes and the Connections between SWCprototypes. • Designing the Topology • ECUs and interconnecting Networks • Designing the Communication Matrix and Data Mapping **Tags:** atp.EnumerationValue=3

Table B.13: BindingTimeEnum

Primitive	**Boolean**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	A Boolean value denotes a logical condition that is either 'true' or 'false'. It can be one of "0", "1", "true", "false" **Tags:** xml.xsd.customType=BOOLEAN xml.xsd.pattern=0|1|true|false xml.xsd.type=string

Table B.14: Boolean

Class	**ClientServerInterface**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:** atp.recommendedPackage=PortInterfaces
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*

▽

△

Class	**ClientServerInterface**			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1..*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table B.15: ClientServerInterface

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=blueprintDerivationTime
possibleError	ApplicationError	*	ref	Possible errors that may by raised by the referring operation.

Table B.16: ClientServerOperation

Class	**ComplexDeviceDriverSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ComplexDeviceDriverSwComponentType is a special AtomicSwComponentType that has direct access to hardware on an ECU and which is therefore linked to a specific ECU or specific hardware. The ComplexDeviceDriverSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
hardwareElement	HwDescriptionEntity	*	ref	Reference from the ComplexDeviceDriverSwComponentType to the description of the used HwElements.

Table B.17: ComplexDeviceDriverSwComponentType

Class	**CompuMethod**
Package	M2::MSR::AsamHdo::ComputationMethod
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:** atp.recommendedPackage=CompuMethods
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*

▽

△

Class	**CompuMethod**			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
compuInternal ToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** xml.sequenceOffset=80
compuPhysTo Internal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** xml.sequenceOffset=90
displayFormat	DisplayFormatString	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** xml.sequenceOffset=30

Table B.18: CompuMethod

Class	***DataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level.

Table B.19: DataPrototype

Class	**DataTypeMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents a list of mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups. **Tags:** atp.recommendedPackage=DataTypeMappingSets			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an Application DataType and its AbstractImplementationDataType.
modeRequest TypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an Mode DeclarationGroup and its AbstractImplementationData Type.

Table B.20: DataTypeMappingSet

Class	DiagnosticComponentNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to specify the service needs for the configuration of component events.			
Base	*ARObject, DiagnosticCapabilityElement, Identifiable, MultilanguageReferrable, Referrable, Service Needs*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table B.21: DiagnosticComponentNeeds

Class	EcuAbstractionSwComponentType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ECUAbstraction is a special AtomicSwComponentType that resides between a software-component that wants to access ECU periphery and the Microcontroller Abstraction. The EcuAbstractionSw ComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement, ARObject, AtomicSwComponentType, AtpBlueprint, AtpBlueprintable, AtpClassifier, Atp Type, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, Sw ComponentType*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
hardware Element	HwDescriptionEntity	*	ref	Reference from the EcuAbstractionComponentType to the description of the used HwElements.

Table B.22: EcuAbstractionSwComponentType

Class	EcucModuleConfigurationValues			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:** atp.recommendedPackage=EcucModuleConfigurationValuess			
Base	*ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, Packageable Element, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
container	EcucContainerValue	1..*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, shortName, variation Point.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10

▽

△

Class	**EcucModuleConfigurationValues**			
definition	EcucModuleDef	1	ref	Reference to the definition of this EcucModuleConfigurationValues element. Typically, this is a vendor specific module configuration. **Tags:** xml.sequenceOffset=-10
ecucDefEdition	RevisionLabelString	1	attr	This is the version info of the ModuleDef ECUC Parameter definition to which this values conform to / are based on. For the Definition of ModuleDef ECUC Parameters the AdminData shall be used to express the semantic changes. The compatibility rules between the definition and value revision labels is up to the module's vendor.
implementationConfigVariant	EcucConfigurationVariantEnum	1	attr	Specifies the kind of deliverable this EcucModuleConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfigured Configuration or recommendedConfiguration) then the value must be one of VariantPreCompile, VariantLinkTime, VariantPostBuild.
moduleDescription	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfigurationValues element is also used to configure the ECU infrastructure (memory map) or Application SW-Cs. However in case the EcucModuleConfigurationValues are used to configure the module, the reference is mandatory in order to fetch module specific "common" published information.
postBuildVariantUsed	Boolean	0..1	attr	Indicates whether a module implementation has or plans to have (i.e., introduced at link or post-build time) new post-build variation points. TRUE means yes, FALSE means no. If the attribute is not defined, FALSE semantics shall be assumed.

Table B.23: EcucModuleConfigurationValues

Class	**EcucModuleDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used as the top-level element for configuration definition for Software Modules, including BSW and RTE as well as ECU Infrastructure. **Tags:** atp.recommendedPackage=EcucModuleDefs			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpDefinition*, *CollectableElement*, *EcucDefinitionElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
apiServicePrefix	CIdentifier	0..1	attr	For CDD modules this attribute holds the apiServicePrefix. The shortName of the module definition of a Complex Driver is always "Cdd". Therefore for CDD modules the module apiServicePrefix is described with this attribute.
container	EcucContainerDef	1..*	aggr	Aggregates the top-level container definitions of this specific module definition. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName xml.sequenceOffset=11
postBuildVariantSupport	Boolean	0..1	attr	Indicates if a module supports different post-build variants (previously known as post-build selectable configuration sets). TRUE means yes, FALSE means no.

▽

△

Class	**EcucModuleDef**			
refinedModule Def	EcucModuleDef	0..1	ref	Optional reference from the Vendor Specific Module Definition to the Standardized Module Definition it refines. In case this EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION this reference shall not be provided. In case this Ecuc ModuleDef has the category VENDOR_SPECIFIC_MODULE_DEFINITION this reference is mandatory. **Stereotypes:** atpUriDef
supported ConfigVariant	EcucConfiguration VariantEnum	*	attr	Specifies which ConfigurationVariants are supported by this software module. This attribute is optional if the EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_MODULE_DEFINITION then this attribute is mandatory.

Table B.24: EcucModuleDef

Class	**ExecutableEntityActivationReason**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	This meta-class represents the ability to define the reason for the activation of the enclosing Executable Entity.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
bitPosition	PositiveInteger	1	attr	This attribute allows for defining the position of the enclosing ExecutableEntityActivationReason in the activation vector.

Table B.25: ExecutableEntityActivationReason

Class	**ExternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns an ExternalTriggeringPoint it is entitled to raise an ExternalTriggerOccurred Event.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
ident	ExternalTriggeringPoint Ident	0..1	aggr	The aggregation in the role ident provides the ability to make the ExternalTriggeringPoint identifiable. From the semantical point of view, the ExternalTriggering Point is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Tags:** atp.Status=shallBecomeMandatory xml.sequenceOffset=-100

▽

△

Class	**ExternalTriggeringPoint**			
trigger	Trigger	0..1	iref	The trigger taken for the ExternalTriggeringPoint. **Tags:** xml.namePlural=TRIGGER-IREF xml.roleElement=false xml.roleWrapperElement=true xml.typeElement=true xml.typeWrapperElement=false

Table B.26: ExternalTriggeringPoint

Class	**FlatInstanceDescriptor**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	Represents exactly one node (e.g. a component instance or data element) of the instance tree of a software system. The purpose of this element is to map the various nested representations of this instance to a flat representation and assign a unique name (shortName) to it. Use cases: • Specify unique names of measurable data to be used by MCD tools • Specify unique names of calibration data to be used by MCD tool • Specify a unique name for an instance of a component prototype in the ECU extract of the system description Note that in addition it is possible to assign alias names via AliasNameAssignment.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
ecuExtract Reference	AtpFeature	0..1	iref	Refers to the instance in the ECU extract. This is valid only, if the FlatMap is used in the context of an ECU extract. The reference shall be such that it uniquely defines the object instance. For example, if a data prototype is declared as a role within an SwcInternalBehavior, it is not enough to state the SwcInternalBehavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying instance of the component prototype and the Atomic SoftwareComponentType, which is refered by the particular SwcInternalBehavior. **Tags:** xml.sequenceOffset=40
role	Identifier	0..1	attr	The role denotes the particular role of the downstream memory location described by this FlatInstanceDescriptor. It applies to use case where one upstream object results in multiple downstream objects, e.g. ModeDeclaration GroupPrototypes which are measurable. In this case the RTE will provide locations for current mode, previous mode and next mode.
rtePluginProps	RtePluginProps	0..1	aggr	The properties of a communication graph with respect to the utilization of RTE Implementation Plug-in. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=rtePluginProps
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this FlatInstanceDescriptor.

▽

△

Class	**FlatInstanceDescriptor**			
upstream Reference	AtpFeature	0..1	iref	Refers to the instance in the context of an "upstream" descriptions, wich could be the system or system extract description, the basic software module description or (if a flat map is used in preliminary context) a description of an atomic component or composition. This reference is optional in case the flat map is used in ECU context. The reference shall be such that it uniquely defines the object instance in the given context. For example, if a data prototype is declared as a role within an SwcInternal Behavior, it is not enough to state the SwcInternal Behavior as context and the aggregated data prototype as target. In addition, the reference shall also include the complete path identifying the instance of the component prototype that contains the particular instance of Swc InternalBehavior. **Tags:** xml.sequenceOffset=20

Table B.27: FlatInstanceDescriptor

Class	**FlatMap**			
Package	M2::AUTOSARTemplates::CommonStructure::FlatMap			
Note	Contains a flat list of references to software objects. This list is used to identify instances and to resolve name conflicts. The scope is given by the RootSwCompositionPrototype for which it is used, i.e. it can be applied to a system, system extract or ECU-extract. An instance of FlatMap may also be used in a preliminary context, e.g. in the scope of a software component before integration into a system. In this case it is not referred by a RootSwComposition Prototype. **Tags:** atp.recommendedPackage=FlatMaps			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
instance	FlatInstanceDescriptor	1..*	aggr	A descriptor instance aggregated in the flat map. The variation point accounts for the fact, that the system in scope can be subject to variability, and thus the existence of some instances is variable. The aggregation has been made splitable because the content might be contributed by different stakeholders at different times in the workflow. Plus, the overall size might be so big that eventually it becomes more manageable if it is distributed over several files. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild

Table B.28: FlatMap

Class	FunctionInhibitionAvailabilityNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Function Inhibition Manager to provide the control function for one Function Identifier (FID).			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
controlledFid	FunctionInhibitionNeeds	0..1	ref	This reference represents the controlled FID

Table B.29: FunctionInhibitionAvailabilityNeeds

Class	GlobalSupervisionNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager to get access on the Global Supervision control and status interface.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table B.30: GlobalSupervisionNeeds

Class	HwElement			
Package	M2::AUTOSARTemplates::EcuResourceTemplate			
Note	This represents the ability to describe Hardware Elements on an instance level. The particular types of hardware are distinguished by the category. This category determines the applicable attributes. The possible categories and attributes are defined in HwCategory. **Tags:** atp.recommendedPackage=HwElements			
Base	*ARElement*, *ARObject*, *CollectableElement*, *HwDescriptionEntity*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
hwElement Connection	HwElementConnector	*	aggr	This represents one particular connection between two hardware elements. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=110
hwPinGroup	HwPinGroup	*	aggr	This aggregation is used to describe the connection facilities of a hardware element. Note that hardware element has no pins but only pingroups. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=90
nestedElement	HwElement	*	ref	This association is used to establish hierarchies of hw elements. Note that one particular HwElement can be target of this association only once. I.e. multiple instantiation of the same HwElement is not supported (at any hierarchy level). **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=70

Table B.31: HwElement

Class	*Identifiable* (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ARPackage, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractServiceInstance*, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticFunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, *DiagnosticRoutineSubfunction*, DoIpLogicAddress, ECUMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinitionElement*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, EndToEndProtection, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, MemorySection, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, NmEcu, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ResourceConsumption, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecureCommunicationAuthenticationProps, SecureCommunicationFreshnessProps, *ServerCallPoint*, *ServiceNeeds*, SocketAddress, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StructuredReq, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcToApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, TcpOptionFilterList, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModelInstance, TlsCryptoCipherSuite, Topic1, TpAddress, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
desc	MultiLanguageOverviewParagraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40

▽

△

Class	***Identifiable*** (abstract)			
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table B.32: Identifiable

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow in case this data type is a variable size array.
isStructWith Optional Element	Boolean	0..1	attr	This attribute is only valid if the attribute category is set to STRUCTURE. If set to True, this attribute indicates that the ImplementationDataType has been created with the intention to define at least one element of the structure as optional. **Tags:** atp.Status=draft

▽

△

Class	ImplementationDataType			
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an array, struct, or union data type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the Implementation DataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table B.33: ImplementationDataType

Class	ImplementationDataTypeElement			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this Implementation DataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
arraySize Handling	ArraySizeHandling Enum	0..1	attr	The way how the size of the array is handled in case of a variable size array.
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the meaning of the value of the array size.
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ImplementationDataTypeElement as optional. This means that, at runtime, the ImplementationDataType Element may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the CppImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end. **Tags:** atp.Status=draft

▽

△

Class	ImplementationDataTypeElement			
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Element of an array, struct, or union in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a Implementation DataType representing a structure. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this ImplementationDataTypeElement.

Table B.34: ImplementationDataTypeElement

Class	InternalTriggeringPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns an InternalTriggeringPoint it is entitled to trigger the execution of Runnable Entities of the corresponding software-component.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.

Table B.35: InternalTriggeringPoint

Class	ModeAccessPoint			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeAccessPoint is required by a RunnableEntity owned by a Mode Manager or Mode User. Its semantics implies the ability to access the current mode (provided by the RTE) of a ModeDeclaration GroupPrototype's ModeDeclarationGroup.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
ident	ModeAccessPointIdent	0..1	aggr	The aggregation in the role ident provides the ability to make the ModeAccessPoint identifiable. From the semantical point of view, the ModeAccessPoint is considered a first-class Identifiable and therefore the aggregation in the role ident shall always exist (until it may be possible to let ModeAccessPoint directly inherit from Identifiable). **Tags:** atp.Status=shallBecomeMandatory xml.sequenceOffset=-100
modeGroup	ModeDeclarationGroupPrototype	0..1	iref	The mode declaration group that is accessed by this runnable. **Tags:** xml.typeElement=true

Table B.36: ModeAccessPoint

Class	**ModeSwitchPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeSwitchPoint is required by a RunnableEntity owned a Mode Manager. Its semantics implies the ability to initiate a mode switch.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
modeGroup	ModeDeclarationGroupPrototype	0..1	iref	The mode declaration group that is switched by this runnable.

Table B.37: ModeSwitchPoint

Class	**ObdInfoServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a given InfoType (OBD Service 09) which is supported by this component or module.			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeeds is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this InfoType.
infoType	PositiveInteger	0..1	attr	The InfoType according to ISO 15031-5

Table B.38: ObdInfoServiceNeeds

Class	**ObdPidServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular PID (parameter identifier) which is supported by this component or module. In case of using a client/server communicated value, the related value shall be communicated via the port referenced by asssignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the related software specifications (SWS).			
Base	*ARObject*, *DiagnosticCapabilityElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceNeeds*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeeds is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this particular PID signal.
parameterId	PositiveInteger	0..1	attr	Standardized parameter identifier (PID) according to the OBD standard specified in attribute "standard".
standard	String	0..1	attr	Annotates the standard according to which the PID is given, e.g. "ISO15031-5" or "SAE J1979 Rev May 2007".

Table B.39: ObdPidServiceNeeds

Class	OperationInvokedEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The OperationInvokedEvent references the ClientServerOperation invoked by the client.			
Base	*ARObject, AbstractEvent, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, Multilanguage Referrable, RTEEvent, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
operation	ClientServerOperation	0..1	iref	The operation to be executed as the consequence of the event.

Table B.40: OperationInvokedEvent

Class	PRPortPrototype			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This kind of PortPrototype can take the role of both a required and a provided PortPrototype.			
Base	*ARObject, AbstractProvidedPortPrototype, AbstractRequiredPortPrototype, AtpBlueprintable, Atp Feature, AtpPrototype, Identifiable, MultilanguageReferrable, PortPrototype, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
provided Required Interface	PortInterface	1	tref	This represents the PortInterface used to type the PRPort Prototype **Stereotypes:** isOfType

Table B.41: PRPortPrototype

Class	ParameterAccess			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a ParameterAccess implies that a RunnableEntity needs access to a ParameterData Prototype.			
Base	*ARObject, AbstractAccessPoint, AtpClassifier, AtpFeature, AtpStructureElement, Identifiable, MultilanguageReferrable, Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
accessed Parameter	AutosarParameterRef	1	aggr	Refernce to the accessed calibration parameter.
swDataDef Props	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis.

Table B.42: ParameterAccess

Class	PortDefinedArgumentValue			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PPortPrototypes of a ClientServer Interface.			
Base	*ARObject*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
value	ValueSpecification	1	aggr	Specifies the actual value.

▽

△

Class	PortDefinedArgumentValue			
valueType	ImplementationData Type	1	tref	The implementation type of this argument value. It should not be composite type or a pointer. **Stereotypes:** isOfType

Table B.43: PortDefinedArgumentValue

Class	***PortPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	*ARObject*, *AtpBlueprintable*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*AbstractProvidedPortPrototype*, *AbstractRequiredPortPrototype*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
clientServer Annotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPort Annotation	DelegatedPort Annotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstraction Server Annotation	IoHwAbstractionServer Annotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePort Annotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPort Annotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPort Annotation	ParameterPort Annotation	*	aggr	Annotations on this parameter port.
senderReceiver Annotation	SenderReceiver Annotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPort Annotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table B.44: PortPrototype

Class	***RTEEvent*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	Abstract base class for all RTE-related events			
Base	*ARObject*, *AbstractEvent*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Subclasses	AsynchronousServerCallReturnsEvent, BackgroundEvent, DataReceiveErrorEvent, DataReceivedEvent, DataSendCompletedEvent, DataWriteCompletedEvent, ExternalTriggerOccurredEvent, InitEvent, InternalTriggerOccurredEvent, ModeSwitchedAckEvent, OperationInvokedEvent, SwcModeManagerError Event, SwcModeSwitchEvent, TimingEvent, TransformerHardErrorEvent			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
disabledMode	ModeDeclaration	*	iref	Reference to the Modes that disable the Event. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=contextPort, contextModeDeclaration GroupPrototype, targetModeDeclaration
startOnEvent	RunnableEntity	0..1	ref	RunnableEntity starts when the corresponding RTEEvent occurs.

Table B.45: RTEEvent

Class	**RapidPrototypingScenario**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::RPTScenario			
Note	This meta class provides the ability to describe a Rapid Prototyping Scenario. Such a Rapid Prototyping Scenario consist out of two main aspects, the description of the byPassPoints and the relation to an rpt Hook. **Tags:** atp.recommendedPackage=RapidPrototypingScenarios			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
hostSystem	System	1	ref	System which describes the software components of the host ECU.
rptContainer	RptContainer	1..*	aggr	Top-level rptContainer definitions of this specific rapid prototyping scenario. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
rptProfile	RptProfile	*	aggr	Defiens the applicable Rapid Prototyping profils which are especially defining the smbol of the service functions and the valid id range. The order of the RptProfiles determines the order of the service function invocation by RTE. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
rptSystem	System	0..1	ref	System which describes the rapid prototyping algorithm in the format of AUTOSAR Software Components. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=rptSystem

Table B.46: RapidPrototypingScenario

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, Bsw VariableAccess, CouplingPortTrafficClassAssignment, DiagnosticDebounceAlgorithmProps, *Diagnostic EnvModeElement*, EthernetPriorityRegeneration, EventHandler, ExclusiveAreaNestingOrder, *Hw DescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, PncMappingIdent, *SingleLanguageReferrable*, SocketConnectionBundle, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table B.47: Referrable

Class	**RoleBasedPortAssignment**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular service port (RPortPrototype or PPort Prototype) of an AtomicSwComponentType. With this assignment, the role of the service port can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct connector.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
portPrototype	PortPrototype	1	ref	Service PortPrototype used in the assigned role. This PortPrototype shall either belong to the same AtomicSw ComponentType as the SwcInternalBehavior which owns the ServiceDependency or to the same NvBlockSw ComponentType as the NvBlockDescriptor.
role	Identifier	1	attr	This is the role of the assigned Port in the given context. The value shall be a shortName of the Blueprint of a Port Interface as standardized in the Software Specification of the related AUTOSAR Service.

Table B.48: RoleBasedPortAssignment

Class	**RunnableEntity**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponent Type and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *ExecutableEntity*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.
asynchronous ServerCall ResultPoint	AsynchronousServer CallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
canBeInvoked Concurrently	Boolean	1	attr	If the value of this attribute is set to "true" the enclosing RunnableEntity can be invoked concurrently (even for one instance of the corresponding AtomicSwComponent Type). This implies that it is the responsibility of the implementation of the RunnableEntity to take care of this form of concurrency. Note that the default value of this attribute is set to "false".
dataRead Access	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional ▽

▽

△

Class	**RunnableEntity**			
				△ existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceivePointByArgument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataReceivePointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointByValue is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
dataWriteAccess	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	RunnableEntity			
external TriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=externalTriggeringPoint, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
internal TriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeAccess Point	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeAccessPoint, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
modeSwitch Point	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
parameter Access	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a ParameterDataPrototype which may either be local or within a PortPrototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of ParameterAccess (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableDataPrototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit ▽

▽

△

Class	**RunnableEntity**			
				△ InterRunnableVariable or the variant existence of read LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
serverCallPoint	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
symbol	CIdentifier	1	attr	The symbol describing this RunnableEntity's entry point. This is considered the API of the RunnableEntity and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The WaitPoint associated with the RunnableEntity.
writtenLocal Variable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableData Prototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicit InterRunnableVariable or the variant existence of written LocalVariable (points) in the implementation. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table B.49: RunnableEntity

Class	**SenderReceiverInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *DataInterface*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	*Type*	*Mul.*	*Kind*	*Note*
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidation Policy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

Table B.50: SenderReceiverInterface

Class	***ServerCallPoint*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a ServerCallPoint it is entitled to invoke a particular ClientServerOperation of a specific RPortPrototype of the corresponding AtomicSwComponentType			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	AsynchronousServerCallPoint, SynchronousServerCallPoint			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	0..1	iref	The operation that is called by this runnable.
timeout	TimeValue	1	attr	Time in seconds before the server call times out and returns with an error message. It depends on the call type (synchronous or asynchronous) how this is reported.

Table B.51: ServerCallPoint

Class	**ServiceSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	ServiceSwComponentType is used for configuring services for a given ECU. Instances of this class are only to be created in ECU Configuration phase for the specific purpose of the service configuration. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	*ARElement*, *ARObject*, *AtomicSwComponentType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *SwComponentType*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table B.52: ServiceSwComponentType

Primitive	**String**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This represents a String in which white-space must be normalized before processing. For example: in order to compare two Strings: • leading and trailing white-space needs to be removed • consecutive white-space (blank, cr, lf, tab) needs to be replaced by one blank. **Tags:** xml.xsd.customType=STRING xml.xsd.type=string

Table B.53: String

Class	**SwBaseType**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *BaseType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table B.54: SwBaseType

Enumeration	**SwCalibrationAccessEnum**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Literal	***Description***
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file. **Tags:** atp.EnumerationValue=0
readOnly	The element will only appear as read-only in an ASAP file. **Tags:** atp.EnumerationValue=1
readWrite	The element will appear in the ASAP file with both read and write access. **Tags:** atp.EnumerationValue=2

Table B.55: SwCalibrationAccessEnum

Class	**SwComponentDocumentation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SoftwareComponentDocumentation			
Note	This class specifies the ability to write dedicated documentation to a component type according to ASAM FSX.			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
chapter	Chapter	*	aggr	These chapters provide additional information about the software component that do not fit in the other chapters. Note that this is subject to variation because Chapter aggregations in the role chapter are variant within the documentation in general. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild xml.roleElement=true xml.roleWrapperElement=false xml.sequenceOffset=100 xml.typeElement=false
swCalibration Notes	Chapter	0..1	aggr	This element contains calibration instructions and hints for a calibration engineer. **Tags:** xml.roleElement=true xml.sequenceOffset=60 xml.typeElement=false
swCarbDoc	Chapter	0..1	aggr	This element records the documentation requested by CARB. **Tags:** xml.roleElement=true xml.sequenceOffset=80 xml.typeElement=false
swDiagnostics Notes	Chapter	0..1	aggr	This element contains general information about diagnostics issues within the component. **Tags:** xml.roleElement=true xml.sequenceOffset=75 xml.typeElement=false
swFeatureDef	Chapter	0..1	aggr	This element contains the definition of the physical functionality of this software component. This definition is more or less formal and is intended to be delivered from modeling tools. **Tags:** xml.roleElement=true xml.sequenceOffset=20 xml.typeElement=false

▽

— AUTOSAR CONFIDENTIAL —

△

Class	**SwComponentDocumentation**			
swFeatureDesc	Chapter	0..1	aggr	This element contains the textual description of the software functionality of this software component. Expert should write this description. **Tags:** xml.roleElement=true xml.sequenceOffset=30 xml.typeElement=false
swMaintenance Notes	Chapter	0..1	aggr	This element contains information regarding the software maintenance of the component. **Tags:** xml.roleElement=true xml.sequenceOffset=70 xml.typeElement=false
swTestDesc	Chapter	0..1	aggr	This element contains suggestions and hints for the test of the software functionality of this software component. **Tags:** xml.roleElement=true xml.sequenceOffset=50 xml.typeElement=false

Table B.56: SwComponentDocumentation

Class	«atpVariation» **SwDataDefProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235

▽

△

Class	«atpVariation» **SwDataDefProps**			
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30

▽

△

Class	«atpVariation» **SwDataDefProps**			
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method. **Tags:** xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250

▽

△

Class	«atpVariation» **SwDataDefProps**			
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock Size Mult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table B.57: SwDataDefProps

Enumeration	**SwImplPolicyEnum**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	Specifies the implementation strategy with respect to consistency mechanisms of variables.
Literal	***Description***
const	forced implementation such that the running software within the ECU shall not modify it. For example implemented with the "const" modifier in C. This can be applied for parameters (not for those in NVRAM) as well as argument data prototypes. **Tags:** atp.EnumerationValue=0
fixed	This data element is fixed. In particular this indicates, that it might also be implemented e.g. as in place data, (#DEFINE). **Tags:** atp.EnumerationValue=1
measurementPoint	The data element is created for measurement purposes only. The data element is never read directly within the ECU software. In contrast to a "standard" data element in an unconnected provide port is, this unconnection is guaranteed for measurementPoint data elements. **Tags:** atp.EnumerationValue=2
queued	The content of the data element is queued and the data element has 'event' semantics, i.e. data elements are stored in a queue and all data elements are processed in 'first in first out' order. The queuing is intended to be implemented by RTE Generator. This value is not applicable for parameters. **Tags:** atp.EnumerationValue=3
standard	This is applicable for all kinds of data elements. For variable data prototypes the 'last is best' semantics applies. For parameter there is no specific implementation directive. **Tags:** atp.EnumerationValue=4

Table B.58: SwImplPolicyEnum

Class	**SwSystemconst**			
Package	M2::MSR::DataDictionary::SystemConstant			
Note	This element defines a system constant which serves an input to select a particular variation point. In particular a system constant serves as an operand of the binding function (swSyscond) in a Variation point. Note that the binding process can only happen if a value was assigned to to the referenced system constants. **Tags:** atp.recommendedPackage=SwSystemconsts			
Base	*ARElement*, *ARObject*, *AtpDefinition*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This denotes the data defintion properties of the system constant. This supports to express the limits and optionally a conversion within the internal to physical values by a compu method. **Tags:** xml.sequenceOffset=40

Table B.59: SwSystemconst

Class	SwcImplementation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software. **Tags:** atp.recommendedPackage=SwcImplementations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *Implementation*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
behavior	SwcInternalBehavior	1	ref	The internal behavior implemented by this Implementation.
perInstance MemorySize	PerInstanceMemory Size	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstanceMemorySize is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects, in this case PerInstance Memory. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
required RTEVendor	String	0..1	attr	Identify a specific RTE vendor. This information is potentially important at the time of integrating (in particular: linking) the application code with the RTE. The semantics is that (if the association exists) the corresponding code has been created to fit to the vendor-mode RTE provided by this specific vendor. Attempting to integrate the code with another RTE generated in vendor mode is in general not possible.

Table B.60: SwcImplementation

Class	SwcInternalBehavior			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *InternalBehavior*, *Multilanguage Referrable*, *Referrable*			
Attribute	**Type**	**Mul.**	**Kind**	**Note**
arTypedPer Instance Memory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is set to "true" or if the component defines NVRAM access via permanent blocks. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the software component's implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime

▽

△

Class	**SwcInternalBehavior**			
event	RTEEvent	*	aggr	This is a RTEEvent specified for the particular Swc InternalBehavior. The aggregation of RTEEvent is subject to variability with the purpose to support the conditional existence of RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using Data ReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
exclusiveArea Policy	SwcExclusiveArea Policy	*	aggr	Options how to generate the ExclusiveArea related APIs. When no SwcExclusiveAreaPolicy is specified for an ExclusiveArea the default values apply. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=exclusiveAreaPolicy vh.latestBindingTime=preCompileTime
explicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnableVariable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
handle TerminationAnd Restart	HandleTerminationAnd RestartEnum	1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSw ComponentType may either not support stop and restart, or support only stop, or support both stop and restart.
implicitInter Runnable Variable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnableVariable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
includedData TypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=includedDataTypeSet
includedMode Declaration GroupSet	IncludedMode DeclarationGroupSet	*	aggr	This aggregation represents the included Mode DeclarationGroups **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=includedModeDeclarationGroupSet

▽

△

Class	**SwcInternalBehavior**			
instantiation DataDefProps	InstantiationDataDef Props	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes and component local memories like "perInstanceParameter" or "arTypedPer InstanceMemory". **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=instantiationDataDefProps, variation Point.shortLabel vh.latestBindingTime=preCompileTime
perInstance Memory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstanceMemory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
perInstance Parameter	ParameterData Prototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is set to "true". The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=portAPIOption, variationPoint.short Label vh.latestBindingTime=preCompileTime
runnable	RunnableEntity	*	aggr	This is a RunnableEntity specified for the particular Swc InternalBehavior. The aggregation of RunnableEntity is subject to variability with the purpose to support the conditional existence of RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of Port Prototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
service Dependency	SwcService Dependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternal Behavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different expert domain (e.g OBD expert for OBD-related Service Needs) tools. Therefore the aggregation is «atpSplitable».

△

Class	**SwcInternalBehavior**			
sharedParameter	ParameterDataPrototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same SwComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=preCompileTime
supportsMultipleInstantiation	Boolean	1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPointProxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table B.61: SwcInternalBehavior

Class	**System**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The top level element of the System Description. The System description defines five major elements: Topology, Software, Communication, Mapping and Mapping Constraints. The System element directly aggregates the elements describing the Software, Mapping and Mapping Constraints; it contains a reference to an ASAM FIBEX description specifying Communication and Topology. **Tags:** atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
clientIdDefinitionSet	ClientIdDefinitionSet	*	ref	Set of Client Identifiers that are used for inter-ECU client-server communication in the System.
containerIPduHeaderByteOrder	ByteOrderEnum	0..1	attr	Defines the byteOrder of the header in ContainerIPdus.
ecuExtractVersion	RevisionLabelString	0..1	attr	Version number of the Ecu Extract.
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

▽

△

Class	**System**			
j1939Shared AddressCluster	J1939SharedAddress Cluster	*	aggr	Collection of J1939Clusters that share a common address space for the routing of messages. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints). In order to support OEM / Tier 1 interaction and shared development for one common System this aggregation is atpSplitable and atpVariation. The content of System Mapping can be provided by several parties using different names for the SystemMapping. This element is not required when the System description is used for a network-only use-case. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild
pncVector Length	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
system Documentation	Chapter	*	aggr	Possibility to provide additional documentation while defining the System. The System documentation can be composed of several chapters. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variationPoint.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=-10
systemVersion	RevisionLabelString	1	attr	Version number of the System Description.

Table B.62: System

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE xml.xsd.type=double

Table B.63: TimeValue

Class	**VariableAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableData Prototype. The kind of access is specified by the role in which the class is used.			
Base	*ARObject*, *AbstractAccessPoint*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mul.***	***Kind***	***Note***
accessed Variable	AutosarVariableRef	1	aggr	This denotes the accessed variable.
scope	VariableAccessScope Enum	0..1	attr	This attribute allows for constraining the scope of the corresponding communication. For example, it possible to express whether the communication is intended to cross the boundary of an ECU or whether it is intended not to cross the boundary of a single partition.

Table B.64: VariableAccess

C Upstream Mapping

C.1 Introduction

This chapter describes the mapping of the ECU Configuration parameters (M1 model) onto the meta-classes and attributes of the BSWMDT.

The relationships between BSWMDT and ECU Configuration are described in order to answer typical questions like:

- How shall a supplier use the information in a System Description in order to fulfill the needs defined by the systems engineer?
- How is a tool vendor supposed to generate an ECU Configuration Description out of an ECU Extract of System Description and BSWMDs delivered for an ECU?

Please note that the tables contain the following columns:

BSW Module: Name of BSW module

BSW Context: Reference to parameter container

BSW Parameter: Name of the BSW parameter

BSW Type: Type of parameter

BSW Description: Description from the configuration document

M2 Template: The upstream templates

M2 Description: Description from the upstream template definition

M2 Parameter: Name of the upstream template parameter

Mapping Rule: Textual description on how to transform between M2 and BSW domains

Mapping Type:

- local: no mapping needed since parameter local to BSW
- partial: some data can be automatically mapped but not all
- full: all data can be automatically mapped

C.2 NvM

BSW Module	**BSW Context**	
NvM	NvM	
BSW Parameter		**BSW Type**
NvMBlockDescriptor		EcucParamConfContainerDef
BSW Description		

Container for a management structure to configure the composition of a given NVRAM Block Management Type. Its multiplicity describes the number of configured NVRAM blocks, one block is required to be configured. The NVRAM block descriptors are condensed in the NVRAM block descriptor table.		
Template Description		
Specifies the abstract needs on the configuration of a single NVRAM Block.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds		
Mapping Rule		**Mapping Type**
In case the owner of the NvBlockNeeds is a BSW module then the NvMBlock Descriptor.shortName = {capitalizedMip}_{ServiceDependency.symbolicName Props.symbol}.		full
Mapping Status		**Mapping ID**
valid		up_NvM_00002

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockJobPriority		EcucIntegerParamDef
BSW Description		
Defines the job priority for a NVRAM block (0 = Immediate priority).		
Template Description		
NvBlockNeeds.writingPriority: Requires the priority of writing this block in case of concurrent requests to write other blocks. **NvBlockNeeds.storeEmergency**: Defines whether or not the associated RAM Block shall be implicitly stored in case of ECU failure (e.g. loss of power) by the basic software. If the attribute storeEmergency is set to true the associated RAM Block shall be configured to have immediate priority.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.writingPriority, CommonStructure::ServiceNeeds::NvBlockNeeds.storeEmergency		
Mapping Rule		**Mapping Type**
It is the integrators job to secure the value-monotonic assignment of writing Priority to NvMBlockJobPriority. This means that the lowest assigned value of writingPriority=MEDIUM shall be greater than highest assigned value of writingPriority=HIGH etc.If NvBlockNeeds.storeEmergency is set to True then NvM BlockJobPriority shall be 0 (Immediate priority). If NvBlockNeeds.storeEmergency is set to False then the value of NvMBlockJob Priority depends on the value of NvBlockNeeds.writingPriority.		full
Mapping Status		**Mapping ID**
valid		up_NvM_00016

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockManagementType		EcucEnumerationParamDef
BSW Description		
Defines the block management type for the NVRAM block.[NVM137]		
Template Description		
Reliability against data loss on the non-volatile medium.		
M2 Parameter		

CommonStructure::ServiceNeeds::NvBlockNeeds.reliability	
Mapping Rule	**Mapping Type**
if (reliability == errorDetection | noProtection) && nDataSets==0 then NvmBlockManagementType = NVM_BLOCK_NATIVE. if reliability == errorCorrection then NvmBlockManagementType = NVM_BLOCK_REDUNDANT. [constr_1095] applies.	full
Mapping Status	**Mapping ID**
valid	up_NvM_00009

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockUseAutoValidation		EcucBooleanParamDef
BSW Description		
Defines whether the RAM Block shall be auto validated during shutdown phase. true: if auto validation mechanism is used, false: otherwise		
Template Description		
If set to true the RAM Block shall be auto validated during shutdown phase.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.useAutoValidationAtShutDown		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00018

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockUseCRCCompMechanism		EcucBooleanParamDef
BSW Description		
Defines whether the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job. true: if compare mechanism is used, false: otherwise		
Template Description		
If set to true the CRC of the RAM Block shall be compared during a write job with the CRC which was calculated during the last successful read or write job in order to skip unnecessary NVRAM writings.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.useCRCCompMechanism		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00019

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**

NvMBlockUseCrc	EcucBooleanParamDef	
BSW Description		
Defines CRC usage for the NVRAM block, i.e. memory space for CRC is reserved in RAM and NV memory. true: CRC will be used for this NVRAM block. false: CRC will not be used for this NVRAM block.		
Template Description		
Reliability against data loss on the non-volatile medium.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.reliability		
Mapping Rule		**Mapping Type**
reliability == errorCorrection | errorDetection means that NvmBlockUseCrc shall bet set to true, else NvmBlockUseCrc = false		full
Mapping Status		**Mapping ID**
valid		up_NvM_00003

BSW Module	**BSW Context**		
NvM	NvM/NvMBlockDescriptor		
BSW Parameter		**BSW Type**	
NvMBlockUseSetRamBlockStatus		EcucBooleanParamDef	
BSW Description			
Defines if NvMSetRamBlockStatusApi shall be used for this block or not. Note: If NvMSetRamBlockStatusApi is disabled this configuration parameter shall be ignored. true: calling of NvMSetRamBlockStatus for this RAM block shall set the status of the RAM block. false: calling of NvMSetRamBlockStatus for this RAM block shall be ignored.			
Template Description			
This attribute defines how the management of the RAM Block status is controlled.			
M2 Parameter			
CommonStructure::ServiceNeeds::NvBlockNeeds.ramBlockStatusControl			
Mapping Rule			**Mapping Type**
If the value of NvBlockNeeds.ramBlockStatusControl is set to RamBlockStatusControlEnum.api the parameter shall be set to true. If the value of NvBlockNeeds.ramBlockStatusControl is set to RamBlockStatusControlEnum.nvRamManager it shall be set to false.			full
Mapping Status			**Mapping ID**
valid			up_NvM_00017

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMBlockWriteProt		EcucBooleanParamDef
BSW Description		
Defines an initial write protection of the NV block true: Initial block write protection is enabled. false: Initial block write protection is disabled.		
Template Description		

True: data of this NVRAM Block are write protected for normal operation (but protection can be disabled) false: no restriction	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.readonly	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**Mapping ID**
valid	up_NvM_00005

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMCalcRamBlockCrc		EcucBooleanParamDef
BSW Description		
Defines CRC (re)calculation for the permanent RAM block or NVRAM blocks which are configured to use explicit synchronization mechanism. true: CRC will be (re)calculated for this permanent RAM block. false: CRC will not be (re)calculated for this permanent RAM block.		
Template Description		
Defines if CRC (re)calculation for the permanent RAM Block is required.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.calcRamBlockCrc		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00007

BSW Module	**BSW Context**	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMNvBlockNum		EcucIntegerParamDef
BSW Description		
Defines the number of multiple NV blocks in a contiguous area according to the given block management type. 1-255 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to SWS_NvM_00444. 1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 2 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT		
Template Description		
NvBlockNeeds.nDataSets: Number of data sets to be provided by the NVRAM manager for this block. This is the total number of ROM Blocks and RAM Blocks. **NvBlockNeeds.reliability**: Reliability against data loss on the non-volatile medium.		
M2 Parameter		

CommonStructure::ServiceNeeds::NvBlockNeeds.nDataSets,CommonStructure::ServiceNeeds::NvBlockNeeds.reliability	
Mapping Rule	**Mapping Type**
if (nDataSets == 0 && reliability ==noProtection | errorDetection) then NvMNvBlockNum = 1. if (nDataSets >0 && reliability ==noProtection | errorDetection) then NvMNvBlockNum = nDataSets.	full
Mapping Status	**Mapping ID**
valid	up_NvM_00011

BSW Module	**BSW Context**
NvM	NvM/NvMBlockDescriptor
BSW Parameter	**BSW Type**
NvMResistantToChangedSw	EcucBooleanParamDef
BSW Description	
Defines whether a NVRAM block shall be treated resistant to configuration changes or not. If there is no default data available at configuration time then the application shall be responsible for providing the default initialization data. In this case the application has to use NvM_GetErrorStatus()to be able to distinguish between first initialization and corrupted data. true: NVRAM block is resistant to changed software. false: NVRAM block is not resistant to changed software.	
Template Description	
Defines whether an NVRAM Block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, please refer to the NVRAM specification.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.resistantToChangedSw	
Mapping Rule	**Mapping Type**
1:1 Mapping	full
Mapping Status	**Mapping ID**
valid	up_NvM_00006

BSW Module	**BSW Context**
NvM	NvM/NvMBlockDescriptor
BSW Parameter	**BSW Type**
NvMRomBlockNum	EcucIntegerParamDef
BSW Description	
Defines the number of multiple ROM blocks in a contiguous area according to the given block management type. 0-254 For NVRAM blocks to be configured of block management type NVM_BLOCK_DATASET. The actual range is limited according to SWS_NvM_00444. 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_NATIVE 0-1 For NVRAM blocks to be configured of block management type NVM_BLOCK_REDUNDANT	
Template Description	
Number of ROM Blocks to be provided by the NVRAM manager for this block. Please note that these multiple ROM Blocks are given in a contiguous area.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.nRomBlocks	

Mapping Rule	Mapping Type
1:1 mapping	full
Mapping Status	**Mapping ID**
valid	up_NvM_00008

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMSelectBlockForReadAll		EcucBooleanParamDef
BSW Description		
Defines whether a NVRAM block shall be processed during NvM_ReadAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_ReadAll false: NVRAM block shall not be processed by NvM_ReadAll		
Template Description		
Defines whether the associated RAM Block shall be implicitly restored during startup by the basic software.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.restoreAtStart		
Mapping Rule		**Mapping Type**
1:1 Mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00013

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMSelectBlockForWriteAll		EcucBooleanParamDef
BSW Description		
Defines whether a NVRAM block shall be processed during NvM_WriteAll or not. This configuration parameter has only influence on those NVRAM blocks which are configured to have a permanent RAM block or which are configured to use explicit synchronization mechanism. true: NVRAM block shall be processed by NvM_WriteAll false: NVRAM block shall not be processed by NvM_WriteAll		
Template Description		
Defines whether or not the associated RAM Block shall be implicitly stored during shutdown by the basic software.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.storeAtShutdown		
Mapping Rule		**Mapping Type**
1:1 Mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00014

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMStaticBlockIDCheck		EcucBooleanParamDef

BSW Description	
Defines if the Static Block ID check is enabled. false: Static Block ID check is disabled. true: Static Block ID check is enabled.	
Template Description	
Defines if the Static Block Id check shall be enabled.	
M2 Parameter	
CommonStructure::ServiceNeeds::NvBlockNeeds.checkStaticBlockId	
Mapping Rule	**Mapping Type**
1:1 mapping	full
Mapping Status	**Mapping ID**
valid	up_NvM_00012

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMWriteBlockOnce		EcucBooleanParamDef
BSW Description		
Defines write protection after first write. The NVRAM manager sets the write protection bit either after the NV block was written the first time or if the block was already written and it is detected as valid and consistent during a read for it. [NVM276]. true: Defines write protection after first write is enabled. false: Defines write protection after first write is disabled.		
Template Description		
Defines write protection after first write: true: This block is prevented from being changed/erased or being replaced with the default ROM data after first initialization by the software-component. false: No such restriction.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.writeOnlyOnce		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**Mapping ID**
valid		up_NvM_00015

BSW Module	BSW Context	
NvM	NvM/NvMBlockDescriptor	
BSW Parameter		**BSW Type**
NvMWriteVerification		EcucBooleanParamDef
BSW Description		
Defines if Write Verification is enabled. false: Write verification is disabled. true: Write Verification is enabled.		
Template Description		
Defines if Write Verification shall be enabled for this NVRAM Block.		
M2 Parameter		
CommonStructure::ServiceNeeds::NvBlockNeeds.writeVerification		
Mapping Rule		**Mapping Type**

1:1 mapping	full
Mapping Status	**Mapping ID**
valid	up_NvM_00010

C.3 WdgM

BSW Module	**BSW Context**	
WdgM	WdgM/WdgMConfigSet/WdgMMode/WdgMLocalStatusParams	
BSW Parameter		**BSW Type**
WdgMFailedAliveSupervisionRefCycleTol		EcucIntegerParamDef
BSW Description		
This parameter shall contain the acceptable amount of reference cycles with incorrect/failed alive supervisions for this Supervised Entity.		
Template Description		
Number of consecutive failed alive cycles for this SupervisedEntity which shall be tolerated until the supervision status of the SupervisedEntity is set to WDGM_ALIVE_EXPIRED (see SWS WdgM for more details). Note that this value has to be recalculated with respect to the WdgM's own cycle time for ECU configuration.		
M2 Parameter		
CommonStructure::ServiceNeeds::SupervisedEntityNeeds.toleratedFailedCycles		
Mapping Rule		**Mapping Type**
1:1		full
Mapping Status		**Mapping ID**
valid		up_WdgM_00001

BSW Module	**BSW Context**	
WdgM	WdgM/WdgMGeneral	
BSW Parameter		**BSW Type**
WdgMSupervisedEntity		EcucParamConfContainerDef
BSW Description		
This container collects all common (mode-independent) parameters of a Supervised Entity to be supervised by the Watchdog Manager.		
Template Description		
Specifies the abstract needs on the configuration of the Watchdog Manager for one specific Supervised Entity.		
M2 Parameter		
CommonStructure::ServiceNeeds::SupervisedEntityNeeds		
Mapping Rule		**Mapping Type**
In case the owner of the SupervisedEntityNeeds is a BSW module then the WdgMSupervisedEntity.shortName = {capitalizedMip}_{ServiceDependency.symbolicNameProps.symbol}.		full
Mapping Status		**Mapping ID**
valid		up_WdgM_00002

BSW Module	**BSW Context**	
WdgM	WdgM/WdgMGeneral/WdgMSupervisedEntity	
BSW Parameter		**BSW Type**
WdgMCheckpoint		EcucParamConfContainerDef

BSW Description	
This container collects all Checkpoints of this Supervised Entity. Each Supervised Entity has at least one Checkpoint.	
Template Description	
Specifies the abstract needs on the configuration of the Watchdog Manager to support a Checkpoint for a Supervised Entity.	
M2 Parameter	
CommonStructure::ServiceNeeds::SupervisedEntityCheckpointNeeds	
Mapping Rule	**Mapping Type**
In case the owner of the SupervisedEntityCheckpointNeeds is a BSW module then the WdgMCheckpoint.shortName = {capitalizedMip}_{ServiceDependency.symbolicNameProps.symbol}.	full
Mapping Status	**Mapping ID**
valid	up_WdgM_00003

C.4 Dcm

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData	
BSW Parameter		**BSW Type**
DcmDspDataFreezeCurrentStateFnc		EcucFunctionNameDef
BSW Description		
Function name to request to application to freeze the current state of an IOControl. (FreezeCurrentState-function). This parameter is related to the interface Xxx_FreezeCurrentState.		
Template Description		
DiagnosticServiceSwMapping.mappedBswServiceDependency: This is supposed to represent a reference to a BswServiceDependency. the latter is not derived from Referrable and therefore this detour needs to be implemented to still let BswServiceDependency become the target of a reference. **DiagnosticIoControlNeeds.freezeCurrentStateSupported**: This attribute determines, if the referenced port supports temporary freezing of I/O value.		
M2 Parameter		
DiagnosticExtract::ServiceMapping::DiagnosticServiceSwMapping.mappedBswServiceDependency, CommonStructure::ServiceNeeds::DiagnosticIoControlNeeds.freezeCurrentStateSupported		
Mapping Rule		**Mapping Type**
It could be possible to get the FNC name via BswServiceDependency		full
Mapping Status		**Mapping ID**
valid		up_Dcm_00004

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort	
BSW Parameter		**BSW Type**
USE_DATA_ASYNCH_FNC		EcucEnumerationLiteralDef
BSW Description		
The DCM will access the Data using the functions that are defined in parameters of type EcucFunctionNameDef (but without DcmDspDataReadDataLengthFnc) in the DcmDspData container. DCM_E_PENDING return is allowed. OpStatus is existing as IN parameter.		

Template Description	
The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request.	
M2 Parameter	
CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleAsynchronous	
Mapping Rule	**Mapping Type**
DiagnosticServiceSwMapping is having a BswServiceDependency and ServiceNeeds::DiagnosticProcessingStyleEnum is equal to processingStyleAsynchronous	full
Mapping Status	**Mapping ID**
valid	up_Dcm_00250

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort	
BSW Parameter		**BSW Type**
USE_DATA_ASYNCH_FNC_ERROR		EcucEnumerationLiteralDef
BSW Description		
The DCM will access the Data using the functions that are defined in parameters of type EcucFunctionNameDef (but without DcmDspDataReadDataLengthFnc) in the DcmDspData container. DCM_E_PENDING return is allowed. OpStatus is existing as IN parameter. The parameter ErrorCode can be returned to allow the application to trigger a negative response during the operation.		
Template Description		
The software-component processes the request in background but still the Dcm has to issue the call again to eventually obtain the result of the request or handle error code.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleAsynchronousWithError		
Mapping Rule		**Mapping Type**
DiagnosticServiceSwMapping is having a BswServiceDependency and ServiceNeeds::DiagnosticProcessingStyleEnum is equal to processingStyleAsynchronousWithError		full
Mapping Status		**Mapping ID**
valid		up_Dcm_00086

BSW Module	BSW Context	
Dcm	Dcm/DcmConfigSet/DcmDsp/DcmDspData/DcmDspDataUsePort	
BSW Parameter		**BSW Type**
USE_DATA_SYNCH_FNC		EcucEnumerationLiteralDef
BSW Description		
The DCM will access the Data using the functions that are defined in parameters of type EcucFunctionNameDef (but without DcmDspDataReadDataLengthFnc) in the DcmDspData container. DCM_E_PENDING return value is not allowed and OpStatus parameter is not existing in the prototype.		
Template Description		
The software-component is supposed to react synchronously on the request.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticProcessingStyleEnum.processingStyleSynchronous		
Mapping Rule		**Mapping Type**
DiagnosticServiceSwMapping is having a BswServiceDependency and ServiceNeeds::DiagnosticProcessingStyleEnum is equal to processingStyleSynchronous		full
Mapping Status		**Mapping ID**

valid	up_Dcm_00089

C.5 Dem

BSW Module	BSW Context	
Dem	Dem/DemConfigSet	
BSW Parameter		**BSW Type**
DemEventParameter		EcucParamConfContainerDef
BSW Description		
This container contains the configuration (parameters) for events.		
Template Description		
Specifies the abstract needs on the configuration of the Diagnostic Event Manager for one diagnostic event. Its shortName can be regarded as a symbol identifying the diagnostic event from the viewpoint of the component or module which owns this element. In case the diagnostic event specifies a production error, the shortName shall be the name of the production error.		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticEventNeeds		
Mapping Rule		**Mapping Type**
In case the owner of the DiagnosticEventNeeds is a BSW module then the DemEventParameter.shortName = {capitalizedMip}_{ServiceDependency.symbolicNameProps.symbol}.		full
Mapping Status		**Mapping ID**
valid		up_Dem_00002

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter	
BSW Parameter		**BSW Type**
DemReportBehavior		EcucEnumerationParamDef
BSW Description		
Indicates the reporting behavior of the BSW Module (DemEventKind == DEM_EVENT_KIND_BSW) in order to determine the size of the reporting queue. If the parameter is not defined it means REPORT_BEFORE_INIT.		
Template Description		
This switch indicates whether or not the BSW module is allowed to report the related Events before Dem_Init().		
M2 Parameter		
CommonStructure::ServiceNeeds::DiagnosticEventNeeds.reportBehavior		
Mapping Rule		**Mapping Type**
1:1 mapping		full
Mapping Status		**Mapping ID**
valid		up_Dem_00003

BSW Module	BSW Context	
Dem	Dem/DemConfigSet/DemEventParameter/DemReportBehavior	
BSW Parameter		**BSW Type**
REPORT_AFTER_INIT		EcucEnumerationLiteralDef
BSW Description		

Indicates that the Event will not be reported before Dem_Init().	
Template Description	
This allows reporting related events after initialization	
M2 Parameter	
CommonStructure::ServiceNeeds::ReportBehaviorEnum.reportAfterInit	
Mapping Rule	**Mapping Type**
	full
Mapping Status	**Mapping ID**
valid	up_Dem_00005

BSW Module	**BSW Context**
Dem	Dem/DemConfigSet/DemEventParameter/DemReportBehavior
BSW Parameter	**BSW Type**
REPORT_BEFORE_INIT	EcucEnumerationLiteralDef
BSW Description	
Indicates that the Event may be reported before Dem_Init().	
Template Description	
This allows reporting related events before initialization	
M2 Parameter	
CommonStructure::ServiceNeeds::ReportBehaviorEnum.reportBeforeInit	
Mapping Rule	**Mapping Type**
	full
Mapping Status	**Mapping ID**
valid	up_Dem_00004

BSW Module	**BSW Context**
Dem	Dem/DemGeneral
BSW Parameter	**BSW Type**
DemRatio	EcucParamConfContainerDef
BSW Description	
This container contains the OBD-specific in-use-monitor performance ratio configuration. It is related to a specific event, a FID, and an IUMPR group.	
Template Description	
ObdRatioServiceNeeds: Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular "ratio monitoring" which is supported by this component or module. **DiagnosticIumprGroup**: This meta-class represents the ability to model a IUMPR groups.	
M2 Parameter	
CommonStructure::ServiceNeeds::ObdRatioServiceNeeds, DiagnosticExtract::Dem::DiagnosticEvent::DiagnosticIumprGroup	
Mapping Rule	**Mapping Type**
In case the owner of the ObdRatioServiceNeeds is a BSW module then the DemRatio.shortName = {capitalizedMip}_{ServiceDependency.symbolicName Props.symbol}. For the DiagnosticIumprGroup the mapping rule is 1:1	full
Mapping Status	**Mapping ID**
valid	up_Dem_00001

C.6 FiM

BSW Module	BSW Context		
FiM	FiM/FiMConfigSet		
BSW Parameter		**BSW Type**	
FiMFID		EcucParamConfContainerDef	
BSW Description			
This container includes symbolic names of all FIDs.			
Template Description			
FunctionInhibitionNeeds: Specifies the abstract needs on the configuration of the Function Inhibition Manager for one Function Identifier (FID). This class currently contains no attributes. Its name can be regarded as a symbol identifying the FID from the viewpoint of the component or module which owns this class. **DiagnosticFunctionIdentifier**: This meta-class represents a diagnostic function identifier (a.k.a. FID).			
M2 Parameter			
CommonStructure::ServiceNeeds::FunctionInhibitionNeeds DiagnosticExtract::Fim::DiagnosticFunctionIdentifier			
Mapping Rule			**Mapping Type**
In case the owner of the FunctionInhibitionNeeds is a BSW module then the FiMFID.shortName= {capitalizedMip}_{ServiceDependency.symbolicNameProps.symbol}.			full
Mapping Status			**Mapping ID**
valid			up_FiM_00001

C.7 ComM

BSW Module	BSW Context		
ComM	ComM/ComMConfigSet		
BSW Parameter		**BSW Type**	
ComMUser		EcucParamConfContainerDef	
BSW Description			
This container contains a list of identifiers that are needed to refer to a user in the system which is designated to request Communication modes.			
Template Description			
Specifies the abstract needs on the configuration of the Communication Manager for one "user".			
M2 Parameter			
CommonStructure::ServiceNeeds::ComMgrUserNeeds			
Mapping Rule			**Mapping Type**
In case the owner of the ComMgrUserNeeds is a BSW module then the ComMUser.shortName = {capitalizedMip}_{ServiceDependency.symbolicNameProps.symbol}.			full
Mapping Status			**Mapping ID**
valid			up_ComM_00003

C.8 StbM

BSW Module	BSW Context	
StbM	StbM	
BSW Parameter		**BSW Type**

StbMSynchronizedTimeBase		EcucParamConfContainerDef
BSW Description		
Synchronized time.base collects the information about a specific time-base provider within the system.		
Template Description		
This represents the ability to define a global time domain.		
M2 Parameter		
SystemTemplate::GlobalTime::GlobalTimeDomain		
Mapping Rule	**Mapping Type**	
For each GlobalTimeDomain where - the configured Ecu is connected to as slave or - the configured Ecu is connected to as master if the Ecu is not in the role of a GlobalTimeGateway for this GlobalTimeDomain an instance of StbMSynchronizedTimeBase shall be created.	full	
Mapping Status	**Mapping ID**	
valid	up_StbM_00001	

D Splitable Elements in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpSplitable≫ in the scope of this document.

Each entry in Table D.1 consists of the identification of the specific model element itself and the applicable value of the tagged value atp.Splitkey.

For more information about the concept of splitable model elements and how these shall be treated please refer to [1].

Name of splitable element	Splitkey
ARPackage.arPackage	shortName, variationPoint.shortLabel
ARPackage.element	shortName, variationPoint.shortLabel
ARPackage.referenceBase	shortLabel
BswEvent.disabledInMode	disabledInMode
BswInternalBehavior.arTypedPerInstanceMemory	shortName, variationPoint.shortLabel
BswInternalBehavior.bswPerInstanceMemoryPolicy	shortName, variationPoint.shortLabel
BswInternalBehavior.clientPolicy	clientPolicy, variationPoint.shortLabel
BswInternalBehavior.distinguishedPartition	shortName, variationPoint.ShortLabel
BswInternalBehavior.entity	shortName, variationPoint.shortLabel
BswInternalBehavior.event	shortName, variationPoint.shortLabel
BswInternalBehavior.exclusiveAreaPolicy	exclusiveAreaPolicy, variationPoint.shortLabel
BswInternalBehavior.includedDataTypeSet	includedDataTypeSet
BswInternalBehavior.internalTriggeringPoint	shortName, variationPoint.shortLabel
BswInternalBehavior.internalTriggeringPointPolicy	internalTriggeringPointPolicy, variationPoint.shortPoint
BswInternalBehavior.modeReceiverPolicy	modeReceiverPolicy, variationPoint.shortLabel
BswInternalBehavior.modeSenderPolicy	modeSenderPolicy, variationPoint.shortLabel
BswInternalBehavior.parameterPolicy	parameterPolicy, variatioPoint.shortLabel
BswInternalBehavior.perInstanceParameter	atp.Splitkey shortName, variationPoint.shortLabel
BswInternalBehavior.receptionPolicy	receptionPolicy, variationPoint.shortLabel
BswInternalBehavior.releasedTriggerPolicy	releasedTriggerPolicy, variationPoint.shortLabel
BswInternalBehavior.schedulerNamePrefix	schedulerNamePrefix, variationPoint.ShortLabel
BswInternalBehavior.sendPolicy	sendPolicy, variationPoint.shortLabel
BswInternalBehavior.serviceDependency	serviceDependency, variationPoint.shortLabel
BswInternalBehavior.triggerDirectImplementation	triggerDirectImplementation, variationPoint.shortLabel
BswInternalBehavior.variationPointProxy	shortName
BswModuleDependency.serviceItem	shortName
BswModuleDescription.bswModuleDependency	shortName, variationPoint.shortLabel
BswModuleDescription.bswModuleDocumentation	bswModuleDocumentation, variationPoint.shortLabel
BswModuleDescription.expectedEntry	expectedEntry, variationPoint.shortLabel
BswModuleDescription.implementedEntry	implementedEntry, variationPoint.shortLabel

BswModuleDescription.internalBehavior	shortName
BswModuleDescription.providedClientServerEntry	shortName, variationPoint.shortLabel
BswModuleDescription.providedData	shortName, variationPoint.shortLabel
BswModuleDescription.providedModeGroup	shortName, variationPoint.shortLabel
BswModuleDescription.releasedTrigger	shortName, variationPoint.shortLabel
BswModuleDescription.requiredClientServerEntry	shortName, variationPoint.shortLabel
BswModuleDescription.requiredData	shortName, variationPoint.shortLabel
BswModuleDescription.requiredModeGroup	shortName, variationPoint.shortLabel
BswModuleDescription.requiredTrigger	shortName, variationPoint.shortLabel
BswServiceDependency.assignedEntryRole	assignedEntryRole, variationPoint.shortLabel
Implementation.mcSupport	mcSupport
Implementation.resourceConsumption	shortName
ImplementationDataType.symbolProps	shortName
InternalBehavior.constantMemory	shortName, variationPoint.shortLabel
InternalBehavior.constantValueMapping	constantValueMapping
InternalBehavior.dataTypeMapping	dataTypeMapping
InternalBehavior.exclusiveArea	shortName, variationPoint.shortLabel
InternalBehavior.exclusiveAreaNestingOrder	shortName, variationPoint.shortLabel
InternalBehavior.staticMemory	shortName, variationPoint.shortLabel
McDataInstance.symbol	symbol
McFunction.defCalprmSet	variationPoint.shortLabel
McFunction.inMeasurementSet	variationPoint.shortLabel
McFunction.locMeasurementSet	variationPoint.shortLabel
McFunction.outMeasurementSet	variationPoint.shortLabel
McFunction.refCalprmSet	variationPoint.shortLabel
McFunction.subFunction	subFunction
McFunctionDataRefSet.flatMapEntry	flatMapEntry
McFunctionDataRefSet.mcDataInstance	mcDataInstance
McGroup.mcFunction	mcFunction
McGroup.refCalprmSet	variationPoint.shortLabel
McGroup.refMeasurementSet	variationPoint.shortLabel
McGroup.subGroup	subGroup
McGroupDataRefSet.flatMapEntry	flatMapEntry
McGroupDataRefSet.mcDataInstance	mcDataInstance
McSupportData.mcParameterInstance	shortName, variationPoint.shortLabel
McSupportData.mcVariableInstance	shortName, variationPoint.shortLabel
McSupportData.rptSupportData	rptSupportData
ResourceConsumption.accessCountSet	shortName, variationPoint.shortLabel
ResourceConsumption.executionTime	shortName, variationPoint.shortLabel
ResourceConsumption.heapUsage	shortName, variationPoint.shortLabel
ResourceConsumption.memorySection	shortName, variationPoint.shortLabel
ResourceConsumption.stackUsage	shortName, variationPoint.shortLabel

Table D.1: Usage of splitable elements

E Variation Points in the Scope of this Document

This chapter contains a table of all model elements stereotyped ≪atpVariation≫ in the scope of this document.

Each entry in Table E.1 consists of the identification of the model element itself and the applicable value of the tagged value vh.latestBindingTime.

For more information about the concept of variation points and how model elements that contain variation points shall be treated please refer to [1].

Variation Point	Latest Binding Time
ARPackage.arPackage	blueprintDerivationTime
ARPackage.element	systemDesignTime
ArrayValueSpecification.element	preCompileTime
BswInternalBehavior.arTypedPerInstanceMemory	preCompileTime
BswInternalBehavior.bswPerInstanceMemoryPolicy	preCompileTime
BswInternalBehavior.clientPolicy	preCompileTime
BswInternalBehavior.distinguishedPartition	preCompileTime
BswInternalBehavior.entity	preCompileTime
BswInternalBehavior.event	preCompileTime
BswInternalBehavior.exclusiveAreaPolicy	preCompileTime
BswInternalBehavior.internalTriggeringPoint	preCompileTime
BswInternalBehavior.internalTriggeringPointPolicy	preCompileTime
BswInternalBehavior.modeReceiverPolicy	preCompileTime
BswInternalBehavior.modeSenderPolicy	preCompileTime
BswInternalBehavior.parameterPolicy	preCompileTime
BswInternalBehavior.perInstanceParameter	preCompileTime
BswInternalBehavior.receptionPolicy	preCompileTime
BswInternalBehavior.releasedTriggerPolicy	preCompileTime
BswInternalBehavior.schedulerNamePrefix	preCompileTime
BswInternalBehavior.sendPolicy	preCompileTime
BswInternalBehavior.serviceDependency	preCompileTime
BswInternalBehavior.triggerDirectImplementation	preCompileTime
BswModuleDependency.targetModuleRef	preCompileTime
BswModuleDescription.bswModuleDependency	preCompileTime
BswModuleDescription.bswModuleDocumentation	preCompileTime
BswModuleDescription.expectedEntry	preCompileTime
BswModuleDescription.implementedEntry	preCompileTime
BswModuleDescription.providedClientServerEntry	preCompileTime
BswModuleDescription.providedData	preCompileTime
BswModuleDescription.providedModeGroup	preCompileTime
BswModuleDescription.releasedTrigger	preCompileTime
BswModuleDescription.requiredClientServerEntry	preCompileTime
BswModuleDescription.requiredData	preCompileTime
BswModuleDescription.requiredModeGroup	preCompileTime
BswModuleDescription.requiredTrigger	preCompileTime
BswModuleEntity.accessedModeGroup	preCompileTime
BswModuleEntity.activationPoint	preCompileTime
BswModuleEntity.calledEntry	preCompileTime
BswModuleEntity.callPoint	preCompileTime
BswModuleEntity.dataReceivePoint	preCompileTime
BswModuleEntity.dataSendPoint	preCompileTime

BswModuleEntity.issuedTrigger	preCompileTime
BswModuleEntity.managedModeGroup	preCompileTime
BswModuleEntry.argument	blueprintDerivationTime
BswServiceDependency.assignedData	preCompileTime
BswServiceDependency.assignedEntryRole	preCompileTime
ErrorTracerNeeds.tracedFailure	preCompileTime
Implementation.buildActionManifest	codeGenerationTime
Implementation.generatedArtifact	preCompileTime
Implementation.requiredArtifact	preCompileTime
Implementation.requiredGeneratorTool	preCompileTime
ImplementationDataType.subElement	preCompileTime
ImplementationDataTypeElement.arraySize	preCompileTime
ImplementationDataTypeElement.subElement	preCompileTime
InternalBehavior.constantMemory	preCompileTime
InternalBehavior.exclusiveArea	preCompileTime
InternalBehavior.exclusiveAreaNestingOrder	preCompileTime
InternalBehavior.staticMemory	preCompileTime
McDataInstance.subElement	preCompileTime
McFunctionDataRefSet	preCompileTime
McGroupDataRefSet	preCompileTime
McSupportData.emulationSupport	preCompileTime
McSupportData.mcParameterInstance	postBuild
McSupportData.mcVariableInstance	postBuild
ModeDeclarationGroup.modeDeclaration	blueprintDerivationTime
NumericalOrText.vf	preCompileTime
NumericalValueSpecification.value	preCompileTime
RecordValueSpecification.field	preCompileTime
ResourceConsumption.accessCountSet	preCompileTime
ResourceConsumption.executionTime	preCompileTime
ResourceConsumption.heapUsage	preCompileTime
ResourceConsumption.memorySection	preCompileTime
ResourceConsumption.sectionNamePrefix	preCompileTime
ResourceConsumption.stackUsage	preCompileTime
RptComponent.rptExecutableEntity	preCompileTime
RptExecutableEntity.rptExecutableEntityEvent	preCompileTime
RptExecutableEntity.rptRead	preCompileTime
RptExecutableEntity.rptWrite	preCompileTime
RptSupportData.rptComponent	preCompileTime
RptSupportData.rptServicePoint	preCompileTime
RuleArguments.vf	preCompileTime
RuleArguments.vtf	preCompileTime
RuleBasedValueSpecification.arguments	preCompileTime
ServiceDependency.assignedDataType	preCompileTime
SupervisedEntityNeeds.checkpoints	preCompileTime
SwcBswMapping.runnableMapping	preCompileTime
SwcBswMapping.synchronizedModeGroup	preCompileTime
SwcBswMapping.synchronizedTrigger	preCompileTime
SwDataDefProps	codeGenerationTime
SwDataDefProps.swValueBlockSize	preCompileTime
SwDataDefProps.swValueBlockSizeMult	preCompileTime
SwTextProps.swMaxTextSize	preCompileTime
ValueList.vf	preCompileTime

Table E.1: Usage of variation points