Document Title	Specification of BSW Module Description Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	089
Document Classification	Standard

Document Version	2.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Description**
01.11.2011	2.2.0	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
22.10.2010	2.1.0	AUTOSAR Administration	• Reworked description of Memory Section • Added chapter on Implementation Conformance Statement

13.11.2009	2.0.0	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
06.08.2008	1.1.0	AUTOSAR Administration	• Added OBD Features
15.02.2008	1.0.1	AUTOSAR Administration	• Layout adaptations
27.11.2007	1.0.0	AUTOSAR Administration	• Initial Release

Table of Contents

References

[1] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[2] Requirements on Basic Software Module Description Template
AUTOSAR_RS_BSWModuleDescriptionTemplate.pdf

[3] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[4] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[5] Methodology
AUTOSAR_TR_Methodology.pdf

[6] Glossary
AUTOSAR_TR_Glossary.pdf

[7] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[8] System Template
AUTOSAR_TPS_SystemTemplate.pdf

[9] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules.pdf

[10] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions.pdf

[11] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[12] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

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

[14] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[15] Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

[16] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[17] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[18] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[19] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping.pdf

[20] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate.pdf

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

[22] AUTOSAR BSW & RTE Conformance Test Specification Part 1: Background
AUTOSAR_PD_BSWCTSpecBackground.pdf

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 [3]
- Requirements on BSW Module Description Template [2]
- General Requirements on Basic Software Modules [4]
- AUTOSAR Methodology [5]
- AUTOSAR Glossary [6]
- Software Component Template [7]
- System Template [8]
- AUTOSAR Model Persistence Rules for XML [9]

1.3 Abbreviations

Abbreviation	*Meaning*
BSW	Basic Software
BSWMD	Basic Software Module Description
BSWMDT	Basic Software Module Description Template
ECU	Electronic Control Unit
ECUC	ECU Configuration
ICC1, ICC2, ICC3	AUTOSAR Implementation Conformance Class 1...3
ISR	Interrupt Service Routine
ICS	Implementation Conformance Statement
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

2 Requirements Traceability

The following table references the requirements specified in AUTOSAR BSWMD Requirements [2] and denotes how they are satisfied by the meta-model.

Requirement	Description	Satisfied by
[BSWMD0001]	Main source of information on BSW Module ECU Configuration activity and integration	Complete BSWMDT
[BSWMD0005]	Description of the memory needs of the BSW Module implementation	MM:ResourceConsumption. stackUsage, MM:ResourceConsumption. memorySection
[BSWMD0007]	Provide vendor-specific published information	MM:BswImplementation. preconfiguredConfiguration
[BSWMD0008]	BSW Module Description SHALL be tool processable	Generated XML schema
[BSWMD0009]	Description of peripheral register usage	MM:BswImplementation. requiredHW
[BSWMD0010]	Compiler version and settings	MM:Implementation. compiler
[BSWMD0011]	Guaranteed execution context of API calls	MM: BswModuleDependency. requiredEntry. executionContext
[BSWMD0013]	Describe configuration class of ECU Configuration Parameters	MM:BswImplementation. vendorSpecificModuleDef
[BSWMD0014]	Support of BSW Module clusters	Complete BSWMDT
[BSWMD0015]	Timing requirements	See document [10]
[BSWMD0016]	Timing guarantees	MM:ResourceConsumption. executionTime
[BSWMD0019]	Modeling of Dataflow dependencies between BSW Modules	N.a., requirement was rejected
[BSWMD0024]	Support description of module specific published information	MM:BswImplementation. vendorSpecificModuleDef
[BSWMD0025]	Support for shipment information	This is not specific for the shipment of BSWMD. It is handled in general by the root element of an AUTOSAR description MM:AUTOSAR. adminData
[BSWMD0026]	Description of supported hardware	MM:BswImplementation. requiredHW
[BSWMD0027]	Provide Vendor-Specific Module Definition	MM:BswModuleDescription. vendorSpecificModuleDef
[BSWMD0028]	Development according to the AUTOSAR Generic Structure Template document	Complete BSWMDT
[BSWMD0029]	Transformation of BSWMD modeling according to the AUTOSAR Model Persistence Rules for XML	Implicitly solved by having the BSWMDT in the same EAP file as all templates
[BSWMD0030]	Publish resource needs for the BSW Scheduler	MM:BswBehavior
[BSWMD0031]	Description of used memory section names	MM:ResourceConsumption. memorySection
[BSWMD0032]	Recommended ECU Configuration Values	MM:BswImplementation. recommendedConfiguration
[BSWMD0033]	Pre-configured ECU Configuration Values	MM:BswImplementation. preconfiguredConfiguration

Requirement	Description	Satisfied by
[BSWMD0034]	ECU Configuration Editor and Generation supported tool version information	MM:Implementation. requiredGeneratorTool
[BSWMD0035]	Provide Standardized Module Definition	MM:BswImplementation. vendorSpecificModuleDef. refinedConfiguration
[BSWMD0037]	Needed libraries	MM:Implementation. requiredArtifact
[BSWMD0038]	Required execution context of API calls	MM: BswModuleDescription. providedEntry. executionContext
[BSWMD0039]	Identification of implemented API and functions	MM:BswModuleDescription. providedEntry
[BSWMD0040]	Identification of required API and functions	MM:BswModuleDescription. bswModuleDependency. requiredEntry
[BSWMD0041]	Declaration of the provided API argument data types	MM:BswModuleDescription. providedEntry
[BSWMD0042]	Description of the required API argument data types	MM:BswModuleDescription. bswModuleDependency. requiredEntry
[BSWMD0043]	Support description of common published information	MM: Attributes of BswImplementation
[BSWMD0044]	Description of generated artifacts	MM:Implementation. generatedArtifact
[BSWMD0045]	Publish resources needed from AUTOSAR Services	MM: BswModuleDependency. serviceItem
[BSWMD0046]	Publish OS resource usage	MM:BswBehavior...
[BSWMD0047]	Modeling of call-chain dependencies between BSW Modules	MM:BswModuleEntity. calledEntry
[BSWMD0048]	Tagging of Vendor-Specific Module Definition	Solved in the ECU Parameter Definition Template, MM:ConfigParameter. origin
[BSWMD0049]	Describe constraints on optional and required elements	Chapter: BSW Variant Handling
[BSWMD0050]	Allow vendor-specific modification of Standardized Module Definition	MM:BswImplementation. vendorSpecificModuleDef
[BSWMD0051]	Description of libraries	Added use case and category; no impact on meta-model
[BSWMD0052]	Description of the generated RTE	This is possible by generating XML artifacts based on BswImplementation, Implementation and ResourceConsumption
[BSWMD0053]	Cyclic time based scheduling of BSW Main Functions	MM:BswTimingEvent
[BSWMD0054]	Mode Switches for BSW modules shall be supported	MM:BswModeSwitchEvent
[BSWMD0055]	Simultaneous Mode transitions	MM: SwcBswMapping
[BSWMD0056]	API for Mode switch notification of BSW modules	MM: BswEntity. managedMode
[BSWMD0057]	Triggering of BSW Main Functions by Triggered Events	MM: BswExternalTriggerOccurredEvent
[BSWMD0058]	Synchronized Triggering by Triggered Events	MM: BswSwcMapping

Requirement	Description	Satisfied by
[BSWMD0059]	API for Triggering BSW modules by Triggered Events	MM: BswEntity. issuedTrigger
[BSWMD0060]	Support exclusive areas in BSW Modules and Application Software Components	MM: InternalBehavior. exclusiveArea
[BSWMD0062]	Provide Measurement and Calibration Support	MM: McSupportData

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 (AUTOSAR R4.0) 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 [5], 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 [11] 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 [5]), 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.

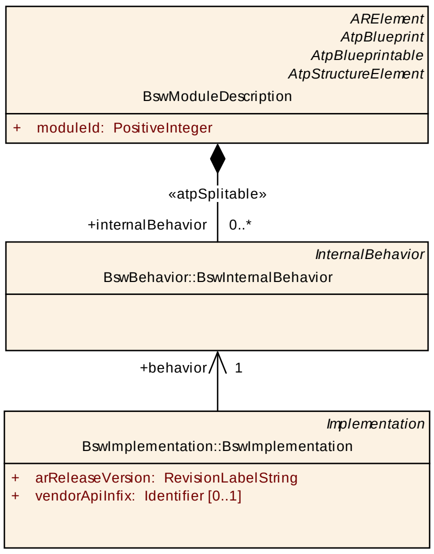
ARElement
AtpBlueprint
AtpBlueprintable
AtpStructureElement
BswModuleDescription
+ moduleId: PositiveInteger
«atpSplitable»
+internalBehavior
0..*
InternalBehavior
BswBehavior::BswInternalBehavior
+behavior
1
Implementation
BswImplementation::BswImplementation
+ arReleaseVersion: RevisionLabelString
+ vendorApiInfix: Identifier [0..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 BswBehavior, 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 BswBehavior 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 BswModuleBehavior.

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 BswBehavior would aggregate BswImplementation, a concrete XML artifact of a BswBehavior 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 BswBehavior 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 `BswModuleBehavior` describe a type of a BSW module or cluster, for which different implementations may exist which are represented by different `BswModuleImplementations` (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 `BswModuleBehavior`.

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 `BswModuleBehavior`, but there must be a mechanism to ensure, that the globally visible C symbols derived from the `BswModuleDescription` and `BswModuleBehavior` 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 [7] and [6]).

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 [7]) which all are derived from `AtomicSwComponentType`.

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

[TPS_BSWMDT_4000] 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. ⌋

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 [7].

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 6.7 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 in AUTOSAR Release 4.0 are both generated during RTE contract phase). For the standardized AUTOSAR Services defined up to now (AUTOSAR release 4.0) 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 (Release 4.0), it is required to specify a runnable entity 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.

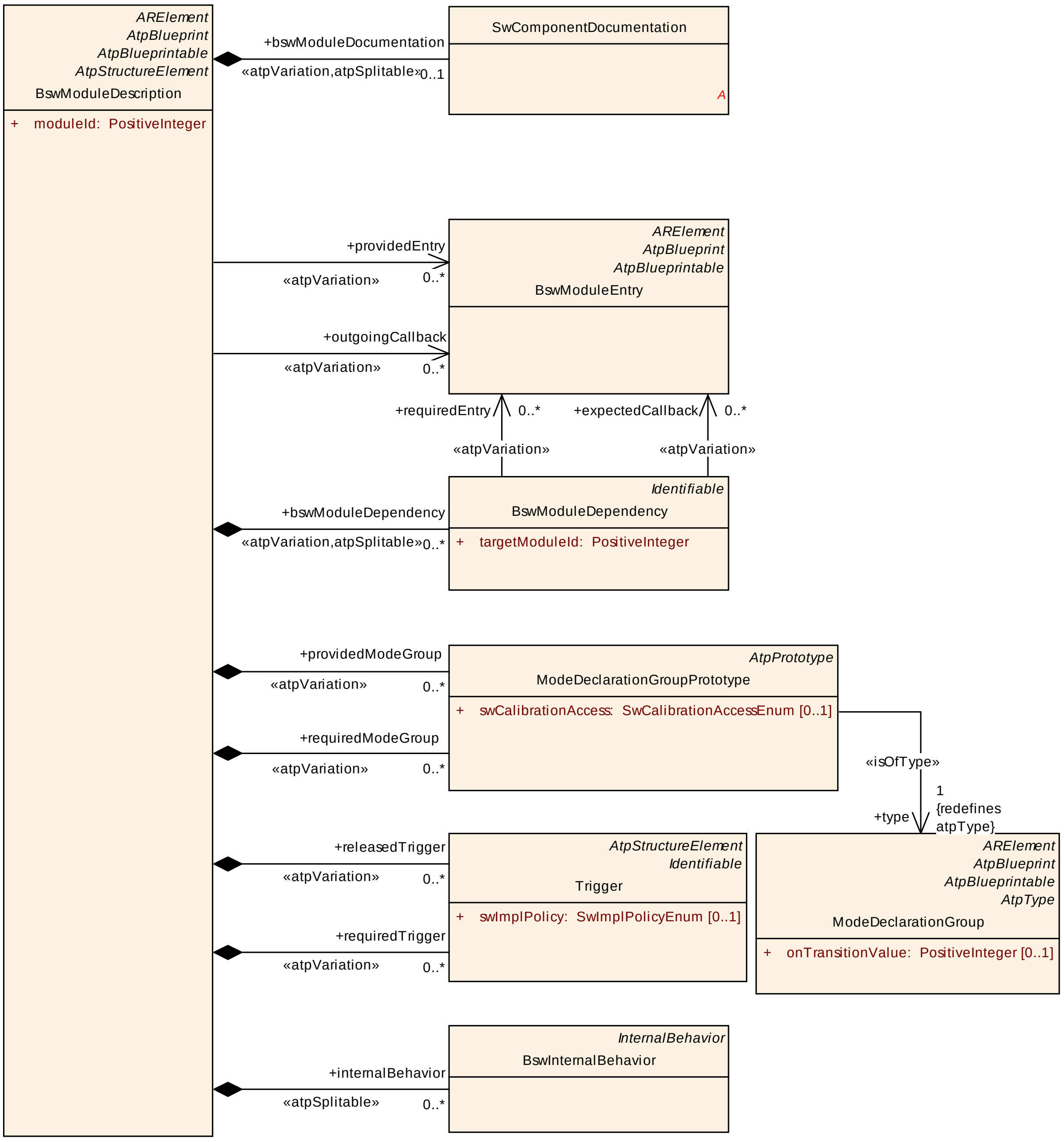

Figure 4.1: BSW Module Description Overview

First of all, 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 [12], if applicable. This identifier can also be used to distinguish modules which are not standardized (i.e. if they belong to the ECU Abstraction or are Complex Device Drivers), or to identify ICC2 clusters. In this case the identifier must be chosen differently from the ones given in [12]. ⌋

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.

[constr_4020] Categories of `BswModuleDescription` ⌈

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

Other values or an empty value are not allowed. ⌋

[TPS_BSWMDT_4001] 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 [7].⌋

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

[TPS_BSWMDT_4002] Usage of `BswModuleEntry` ⌈The interface exported by a `BswModuleDescription` is a set of `providedEntry`-s provided for the usage by other modules (including "main"-functions called by the BSW Scheduler) and of `outgoingCallback`s which this module declares and which it calls if another modules requires it.⌋

The distinction between between provided functions and callbacks must be unambiguous:
[constr_4036] Entries linked to `BswModuleDescription` ⌈

- `BswModuleDescription.providedEntry.callType` must not be `'callback'`.
- `BswModuleDescription.outgoingCallback.callType` must always be `'callback'`.

⌋
(for the definition of the attribute `BswModuleEntry.callType` see next section).

[TPS_BSWMDT_4003] `BswModuleDependency` ⌈ With the help of class `BswModuleDependency` it is possible to describe the requirements of a given BSW module onto another BSW module which among other things includes the interface imported from the other module, namely a set of `requiredEntries` and `expectedCallbacks`.⌋

[TPS_BSWMDT_4004] **`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`.⌋

[TPS_BSWMDT_4005] **`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`.⌋

[TPS_BSWMDT_4006] BswModuleDescription.internalBehavior ⌈By the aggregation of class `BswBehavior` in `BswModuleDescription` it is possible to add scheduling aspects to the description.⌋

The declaration of function calls, dependencies, triggers and modes make up the interface of a module or cluster, the details are described in chapter 5. For `BswBehavior` 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	***Datatype***	***Mul.***	***Kind***	***Note***
bswModuleDependency	BswModuleDependency	*	aggr	Describes the dependency to another BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel xml.sequenceOffset=20
bswModuleDocumentation	SwComponentDocumentation	0..1	aggr	This adds a documentation to the BSW module. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=bswModuleDocumentation, variationPoint.shortLabel xml.sequenceOffset=6
internalBehavior	BswInternalBehavior	*	aggr	The various BswInternalBehaviors associated with a BswModuleDescription can be distributed over several physical files. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName xml.sequenceOffset=45

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
moduleId	PositiveInteger	1	attr	Refers to the BSW Module Identifier defined by the AUTOSAR standard. For non-standardized modules, a proprietary identifier can be chosen. **Tags:** xml.sequenceOffset=5
outgoingCallback	BswModuleEntry	*	ref	Specifies a callback, which will be called from this module if required by another module. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=15
providedEntry	BswModuleEntry	*	ref	Specifies an entry provided by this module which can be called by other modules. This includes "main" functions and interrupt routines, but not callbacks (because the signature of a callback is defined by the caller). **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=10
providedModeGroup	ModeDeclarationGroupPrototype	*	aggr	A set of modes which is owned and provided by this module or cluster. It can be connected to the requiredModeGroups 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, EcuAbstractionSwComponentType or ComplexDeviceDriverSwComponentType. **Stereotypes:** atpVariation **Tags:** 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, EcuAbstractionSwComponentType or ComplexDeviceDriverSwComponentType. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=35

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
requiredModeGroup	ModeDeclarationGroupPrototype	*	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:** atpVariation **Tags:** 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:** atpVariation **Tags:** 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 `providedEntries`, declaration of mode groups, declaration of triggers and the dependencies from other modules.

5.1 BSW Module Entry

[TPS_BSWMDT_4007] **`BswModuleEntry`** ⌈The class `BswModuleEntry` is used to model the signature of a C-function call⌋, see figure 5.1.

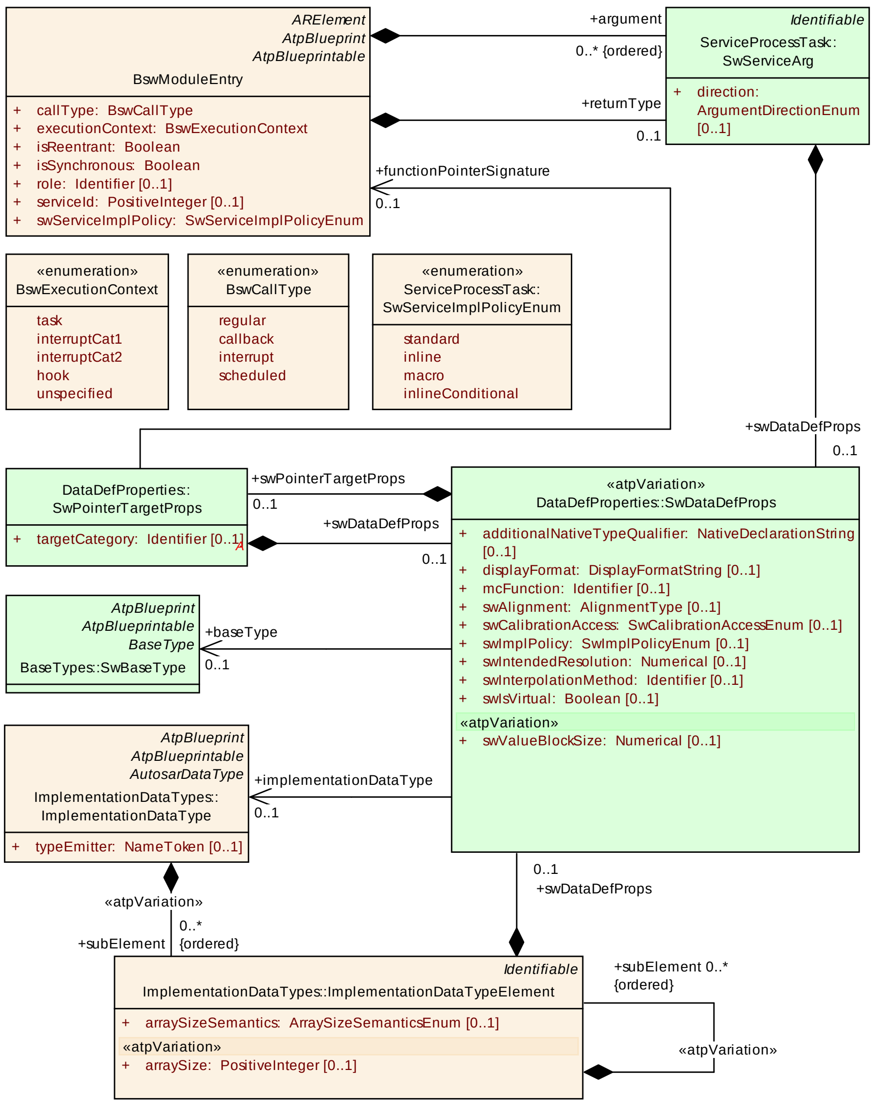
AREElement
AtpBlueprint
AtpBlueprintable
BswModuleEntry
+ callType: BswCallType
+ executionContext: BswExecutionContext
+ isReentrant: Boolean
+ isSynchronous: Boolean
+ role: Identifier [0..1]
+ serviceId: PositiveInteger [0..1]
+ swServiceImplPolicy: SwServiceImplPolicyEnum
+argument
0..* {ordered}
+returnType
0..1
Identifiable
ServiceProcessTask::SwServiceArg
+ direction: ArgumentDirectionEnum [0..1]
+functionPointerSignature
0..1
«enumeration» BswExecutionContext
task
interruptCat1
interruptCat2
hook
unspecified
«enumeration» BswCallType
regular
callback
interrupt
scheduled
«enumeration» ServiceProcessTask::SwServiceImplPolicyEnum
standard
inline
macro
inlineConditional
+swDataDefProps
0..1
DataDefProperties::SwPointerTargetProps
+ targetCategory: Identifier [0..1]
+swPointerTargetProps
0..1
+swDataDefProps
0..1
«atpVariation» DataDefProperties::SwDataDefProps
+ additionalNativeTypeQualifier: NativeDeclarationString [0..1]
+ displayFormat: DisplayFormatString [0..1]
+ mcFunction: Identifier [0..1]
+ swAlignment: AlignmentType [0..1]
+ swCalibrationAccess: SwCalibrationAccessEnum [0..1]
+ swImplPolicy: SwImplPolicyEnum [0..1]
+ swIntendedResolution: Numerical [0..1]
+ swInterpolationMethod: Identifier [0..1]
+ swIsVirtual: Boolean [0..1]
«atpVariation»
+ swValueBlockSize: Numerical [0..1]
AtpBlueprint
AtpBlueprintable
BaseType
BaseTypes::SwBaseType
+baseType
0..1
AtpBlueprint
AtpBlueprintable
AutosarDataType
ImplementationDataTypes::ImplementationDataType
+ typeEmitter: NameToken [0..1]
+implementationDataType
0..1
«atpVariation»
+subElement
0..* {ordered}
0..1
+swDataDefProps
Identifiable
ImplementationDataTypes::ImplementationDataTypeElement
+ arraySizeSemantics: ArraySizeSemanticsEnum [0..1]
«atpVariation»
+ arraySize: PositiveInteger [0..1]
+subElement 0..* {ordered}
«atpVariation»

Figure 5.1: Details of class BswModuleEntry

The attributes of 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. ⌋

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,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
argument (ordered)	SwServiceArg	*	aggr	An argument belonging to this BswModuleEntry. **Tags:** xml.sequenceOffset=45
callType	BswCallType	1	attr	The type of call associated with this service. **Tags:** xml.sequenceOffset=25
executionC ontext	BswExecutionC ontext	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
isReentran t	Boolean	1	attr	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
isSynchron ous	Boolean	1	attr	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

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
role	Identifier	0..1	ref	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
swServiceImplPolicy	SwServiceImplPolicyEnum	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	**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
interruptCat1	CAT1 interrupt context always
interruptCat2	CAT2 interrupt context always
task	Task context always
unspecified	The execution context is not specified by the API

Table 5.2: BswExecutionContext

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)
interrupt	Interrupt routine
regular	Regular API call
scheduled	Called by the scheduler

Table 5.3: BswCallType

Enumeration	**SwServiceImplPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceProcessTask
Note	This specifies the legal values for the implementation policies for services (in AUTOSAR: BswModuleEntry-s).
Literal	***Description***
inline	inline service definition.
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.
macro	macro service definition.
standard	Standard service and default value, if nothing is defined.

Table 5.4: 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_4008] 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.⌋ For details see 7.

As a result, also the code of a module requiring a `SwService` 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_4009] Usage of `SwServiceArg` ⌈Class `SwServiceArg` [1] is used to declare the properties of the function arguments as well as of the return type. ⌋

Class	**SwServiceArg**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceProcessTask			
Note	Specifies the properties of a data object exchanged during the call of an SwService, e.g. an argument or a return value.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***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 5.5: SwServiceArg

[TPS_BSWMDT_4010] `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. ⌋

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

[TPS_BSWMDT_4011] `SwServiceArg.swDataDefProps.swPointerTargetProps` ⌈ together with it category (see [7]) is used to declare an argument or return type as a pointer to either another data object or to a function: ⌋

[1] `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 [7] and [13] for more explanation.

Class	**SwPointerTargetProps**			
Package	M2::AUTOSARTemplates::CommonStructure::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	***Datatype***	***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	ref	This specifies the category of the target: • In case of a data pointer, it must specify the category of the referenced data. • In case of a function pointer, it could be used to denote the category of the referenced BswModuleEntry. Since currently no categories for BswModuleEntry are defined, it will be empty. **Tags:** xml.sequenceOffset=5

Table 5.6: 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 [7]. 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.

Function signature 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.

[TPS_BSWMDT_4012] `SwServiceArg.direction` ⌈ allows to declare the direction of data flow ⌋ (the attribute was introduced in R4.0.3 and is optional for backwards compatibility reasons):

Enumeration	**ArgumentDirectionEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types
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.
inout	The argument value is passed to the callee but also passed back from the callee to the caller.
out	The argument value is passed from the callee to the caller.

Table 5.7: 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. ⌋

5.2 BSW Mode Declaration

[TPS_BSWMDT_4013] 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 mod-

ules which require to be controlled by the mode group, must declare a compatible `ModeDeclarationGroupPrototype` as attribute `requiredModeGroup`. ⌋

For the compatibility of `ModeDeclarationGroupPrototype`s see [7]. 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 [7].

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.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	This allows for specifying whether or not the enclosing ModeDeclarationGroupPrototype can be measured at run-time.
type	ModeDeclarationGroup	1	tref	The "collection of ModeDeclarations" (= ModeDeclarationGroup) supported by a component **Stereotypes:** isOfType

Table 5.8: 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 [14].

Class	**ModeDeclarationGroup**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:** atp.recommendedPackage=ModeDeclarationGroups			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initialMode	ModeDeclaration	1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
modeDeclaration	ModeDeclaration	1..*	aggr	The ModeDeclarations collected in this ModeDeclarationGroup.
onTransitionValue	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 Statuus.

Table 5.9: ModeDeclarationGroup

Class	**ModeDeclaration**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Declaration of one Mode. The name and semantics of a special mode is not defined in the meta-model.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*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 ModeDeclaration.

Table 5.10: ModeDeclaration

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 ModeDeclarationGroups (via requiredModeGroup and/or providededModeGroup) having the same shortName but different elements. ⌋

The attributes ModeDeclaration.values and ModeDeclarationGroup.transitionValue and the category of ModeDeclarationGroup allow to determine the generation of source code from the formal definition. For constraints on these attributes refer to [14].

[TPS_BSWMDT_4014] ModeRequestTypeMap in BSW ⌈Furthermore, it is required to define a ModeRequestTypeMap in order to explicitly specify by which data type a ModeDeclarationGroup is implemented: ⌋

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	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
implementationDataType	ImplementationDataType	1	ref	This is the corresponding ImplementationDataType. It shall be modeled along the idea of an "unsigned integer-like" data type.
modeGroup	ModeDeclarationGroup	1	ref	This is the corresponding ModeDeclarationGroup.

Table 5.11: 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 [14] 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 `ImplementationDataType`s on the level of BSWMDs only.

5.3 BSW Trigger Declaration

[TPS_BSWMDT_4015] 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 `Trigger`s refer to [7]). These declarations together with the associated event model (see chapter 6.4) allow for the appropriate API generation and coordination by the BSW Scheduler. ⌋

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	*Datatype*	*Mul.*	*Kind*	*Note*
swImplPolicy	SwImplPolicyEnum	0..1	attr	This attribute, when set to value queued, allows for a queued processing of Triggers.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table 5.12: Trigger

A `Trigger` declaration can optionally set an attribute to define its queuing behavior. This is in more detail explained in [7]. The usage of the enumeration type `Trigger.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 `Trigger`s positively uses a queue). ⌋

5.4 BSW Module Dependency

5.4.1 General

Figure 5.2 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 the dependencies are not expressed by associations between instances of `BswModuleDescription`. This allows to maintain each BSWMD separately.

A module cannot state a dependency to itself:
[constr_4038] `bswModuleDependency` must refer to a different module ⌈ `BswModuleDescription.bswModuleDependency.targetModuleId` must have a different value than `BswModuleDescription.moduleId`. ⌋

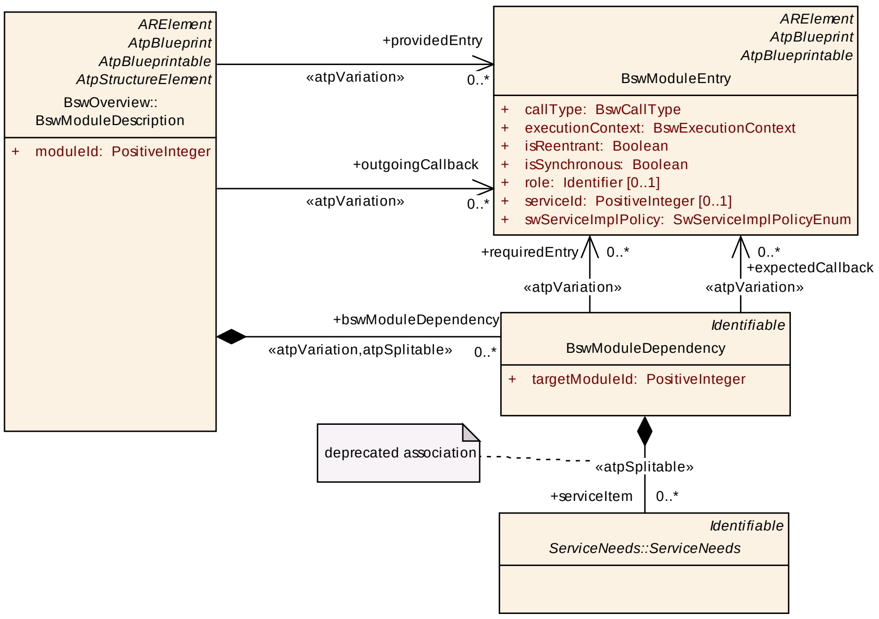

Figure 5.2: Details of class BswModuleDependency

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	**Datatype**	**Mul.**	**Kind**	**Note**
expectedCallback	BswModuleEntry	*	ref	Indicates a callback expected to be called from another module and implemented by this module. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=15
requiredEntry	BswModuleEntry	*	ref	Indicates an entry into another modules which is required by this module. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=10

Attribute	*Datatype*	*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=obsolete xml.sequenceOffset=20
targetModuleId	PositiveInteger	1	attr	AUTOSAR identifier of the target module of which the dependencies are defined. **Tags:** xml.sequenceOffset=5

Table 5.13: BswModuleDependency

The set of `requiredEntry`-s and `expectedCallbacks` 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 `ARElement`s 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 `ARPackage`s.

Rules for the package location of standardized M1 model elements are given in [3], chapter *Identifying M1 elements in packages*. As a consequence we can state:

[TPS_BSWMDT_4016] Location of standardized `BswModuleEntrys` ⌈ Instances of standardized `BswModuleEntry`s defined for an AUTOSAR module <module>[2] shall be located under a package

`AUTOSAR_<module>/BswModuleEntrys/`

⌋

for example

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

`AUTOSAR_Can/BswModuleEntrys/Can_SetControllerMode`

[TPS_BSWMDT_4017] Reference to standardized `BswModuleEntrys` ⌈ If a BSWMD refers to a standardized `BswModuleEntry` via its `BswModuleDependency.requiredEntry` or `BswModuleDependency.expectedCallBack` it shall also use the path

`AUTOSAR_<module>/BswModuleEntrys/`

thus indicating that it relies on the AUTOSAR compliant implementation of the referred API functions. ⌋

It is highly recommended to follow an analog pattern (but not starting with AUTOSAR) for the package names of non-standardized `ARElement`s too.[3] 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 this example, we would instead of *Can* use a non-standardized module name if the referred module is a complex device driver. In this case, the module name would be equal to the `BswModuleDescription.shortName` of the BSWMD of that complex device driver.

5.4.3 Dependency: Examples and Constraints

Note that `requiredEntries` and `expectedCallbacks` 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
- `bswModuleDependency.expectedCallback` = signature+name of "‘EndJobNotification"
- `bswModuleDependency.expectedCallback.callType = ’callback’`

[3]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.

- `bswModuleDependency.expectedCallback.executionContext = 'interrupt'` (i.e. the required context)

The distinction between between required (i.e. called) functions and expected (i.e. implemented) callbacks must be unambiguous and is related to the attribute `callType`:

[constr_4037] Entries linked to ARTechTermBswModuleDependency ⌈

- `BswModuleDependency.requiredEntry.callType` must always be `'regular'`.
- `BswModuleDependency.expectedCalback.callType` must always be `'callback'`.

⌋

Figure 5.3 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"[4].

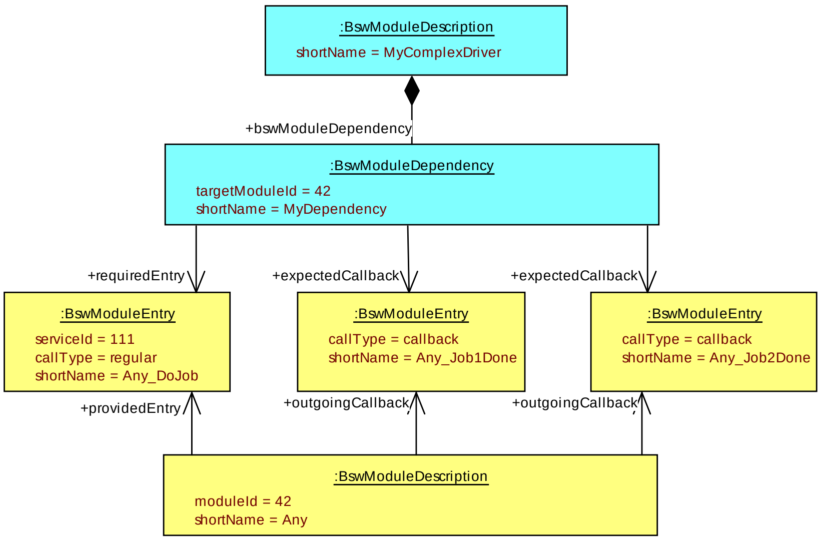

Figure 5.3: Example for an M1 model of a dependency between two modules

Note that the model of the outgoing callbacks can (in general) only be completed at configuration time, because the number and names of the `BswModuleEntry`s 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 a `Blueprint` 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 [15].

[4]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.

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 6.8.

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.

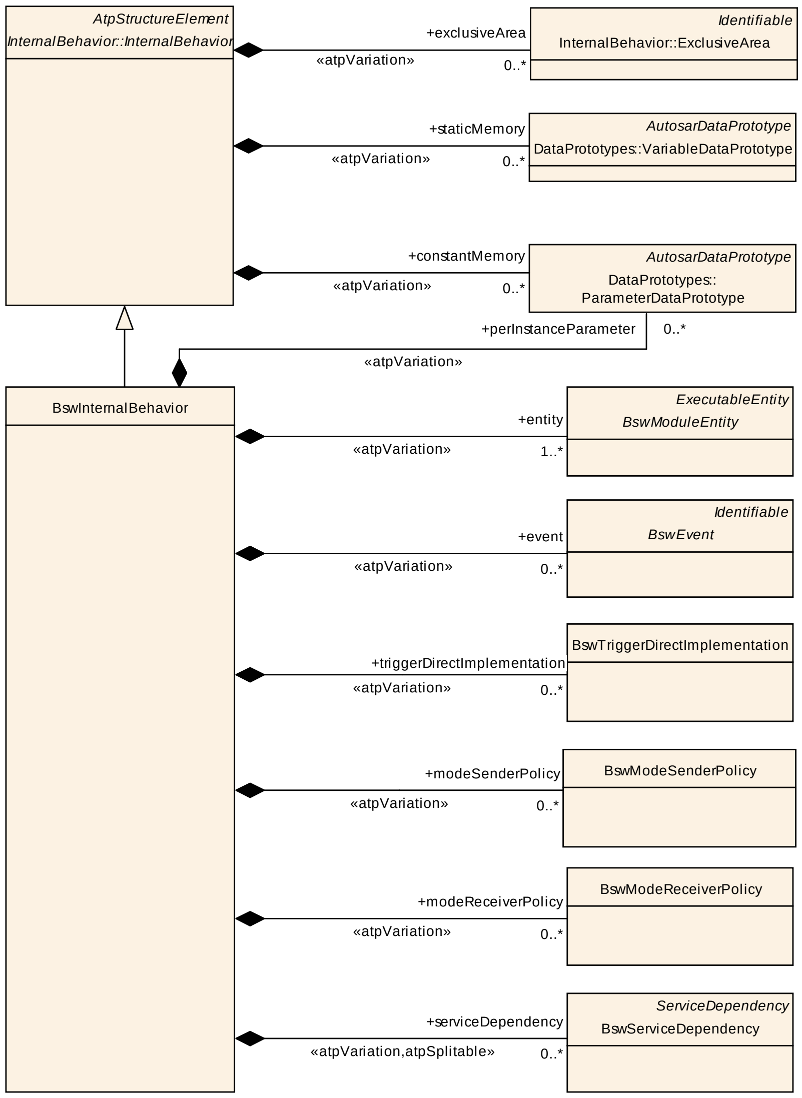

Figure 6.1: Overview of class BswModuleBehavior

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,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
constantMemory	ParameterDataPrototype	*	aggr	Describes a read only memory object containing characteristic value(s) implemented by this InternalBehavior. The shortName of ParameterElementPrototype has to be equal to the ”C’ identifier of the described constant. The characteristic value(s) might be shared between SwComponentPrototypes of the same SwComponentType. 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:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular InternalBehavior
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular InternalBehavior
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 BswModuleEntities. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
staticMemory	VariableDataPrototype	*	aggr	Describes a read and writeable static memory object representing measurment variables implemented by this software component. 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. The shortName of DataElementPrototype 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 components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** 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,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
entity	BswModuleEntity	1..*	aggr	A code entity for which the behavior is described **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=5
event	BswEvent	*	aggr	An event required by this module behavior. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=10
internalTriggeringPoint	BswInternalTriggeringPoint	*	aggr	An internal triggering point. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=30
modeReceiverPolicy	BswModeReceiverPolicy	*	aggr	Implementation policy for the reception of mode switches. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=25

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
modeSenderPolicy	BswModeSenderPolicy	*	aggr	Implementation policy for providing a mode group. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20
perInstanceParameter	ParameterDataPrototype	*	aggr	Describes a read only memory object containing characteristic value(s) needed by this BswInternalBehavior. The role name perInstanceParameter is chosen in analogy to the similar role in the context of SwcInternalBehavior. In contrast to constantMemory, this object is not allocated locally be 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:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=45
schedulerNamePrefix	BswSchedulerNamePrefix	*	aggr	Optional definition of one or more prefixes to be used for the BswScheduler. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=50
serviceDependency	BswServiceDependency	*	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:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel xml.sequenceOffset=40
triggerDirectImplementation	BswTriggerDirectImplementation	*	aggr	Specifies a trigger to be directly implemented via OS calls. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=15

Table 6.2: BswInternalBehavior

6.2 BSW Module Entity

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. Figure 6.1 show how `BswModuleEntity` is related to other parts of the BSWMDT.

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.

[TPS_BSWMDT_4018] `BswModuleEntity.calledEntry` ⌈ This attribute allows to declare which entry of another module (or the same module) is called by this code entity. ⌋

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.

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

[constr_4015] `calledEntry` constraints ⌈

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

⌋

[TPS_BSWMDT_4019] `BswModuleEntity` attributes ⌈ 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. ⌋

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 `providedEntry` or as `bswModuleDependency.expectedCallback` of the enclosing `BswModuleDescription`

- BswModuleEntity.calledEntry must refer to an element declared as outgoingCallback, providedEntry or as bswModuleDependency.requiredEntry of the enclosing BswModuleDescription
- 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.accessedModeGroupmust refer to an element declared as requiredModeGroup of the enclosing BswModuleDescription

⌋

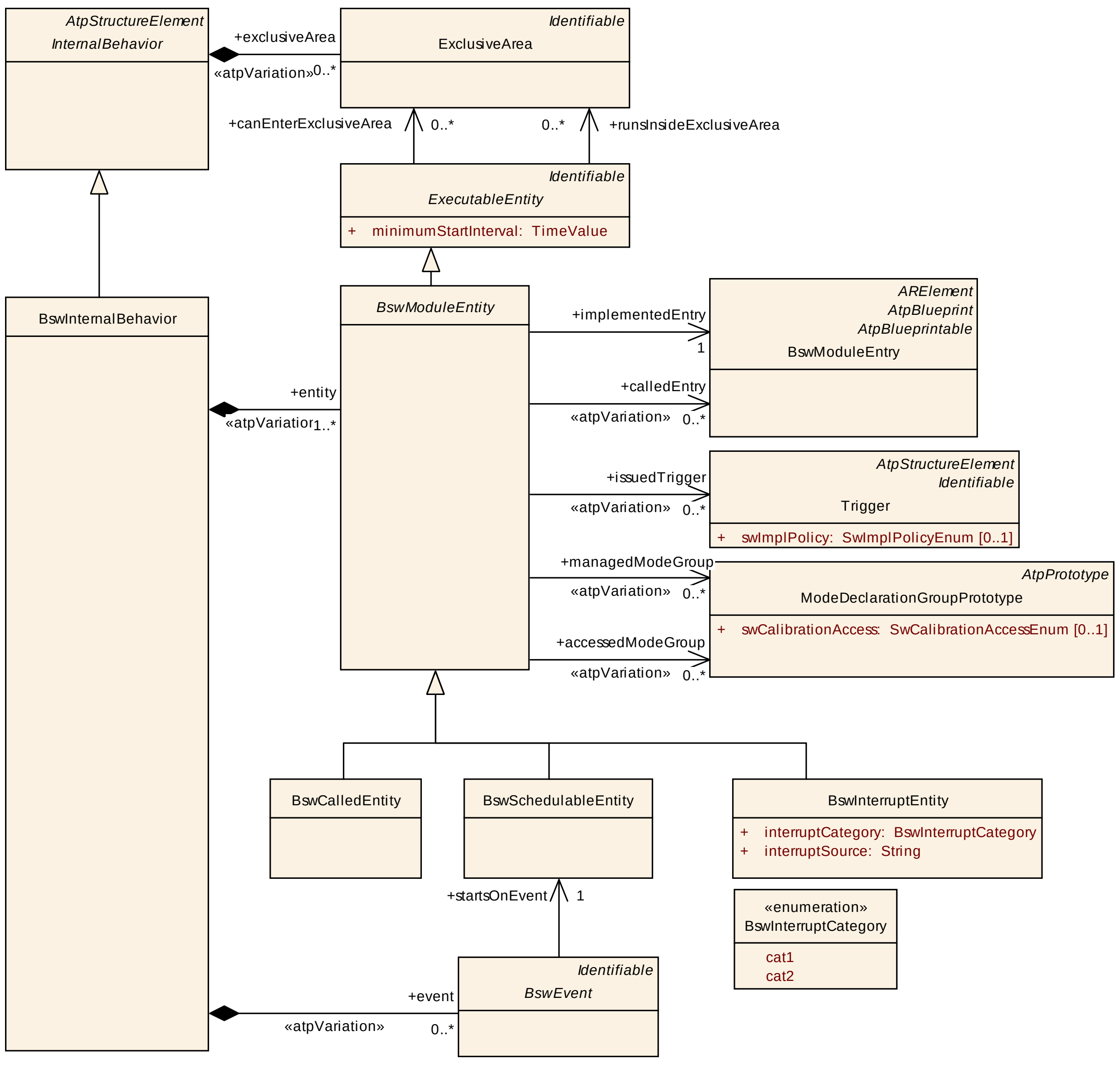

Figure 6.2: Details of class BswModuleEntity

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			
Attribute	***Datatype***	***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
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. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
implementedEntry	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
managedModeGroup	ModeDeclarationGroupPrototype	*	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

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
scheduler NamePrefix	BswSchedulerNamePrefix	0..1	ref	A prefix to be used in generated names for the BswModuleScheduler 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 6.3: BswModuleEntity

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,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.4: BswCalledEntity

This 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 not restricted

⌋

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

Table 6.5: BswSchedulableEntity

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

[constr_4017] `BswScheduledEntity` constraints ⌈

- `BswScheduledEntity.implementedEntry.callType` must be `'scheduled'`
- `BswScheduledEntity.implementedEntry.executionContext` must be `'task'`

⌋

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,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
interruptCategory	BswInterruptCategory	1	attr	Category of the interrupt
interruptSource	String	1	attr	Allows a textual documentation of the intended interrupt source.

Table 6.6: 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).
cat2	Cat2 interrupt routines are controlled by the OS and they are allowed to make OS calls. The BswInterruptEntity is implemented by the interrupt handler, which is called from the OS.

Table 6.7: BswInterruptCategory

This 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'`

⌋

The classes ExclusiveArea and ExecutableEntity are not specific for the Basic Software, they are imported from the CommonStructure package of the meta-model:

Class	**ExecutableEntity (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::InternalBehavior			
Note	Abstraction of executable code.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
canEnterExclusiveArea	ExclusiveArea	*	ref	This means that the executable entity can enter/leave the referenced exclusive area through explicit API calls.
minimumStartInterval	TimeValue	1	attr	Specifies the time in seconds by which two consecutive starts of an ExecutableEntity are guaranteed to be separated.
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 6.8: ExecutableEntity

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	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.9: ExclusiveArea

6.3 BSW Scheduler Name Prefix

[TPS_BSWMDT_4020] 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. ⌋

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 6.3 and the following class table show how the meta-class `BswSchedulerNamePrefix` is placed in the meta-model. Refer to [16] for the details how this information is used by the RTE generator.

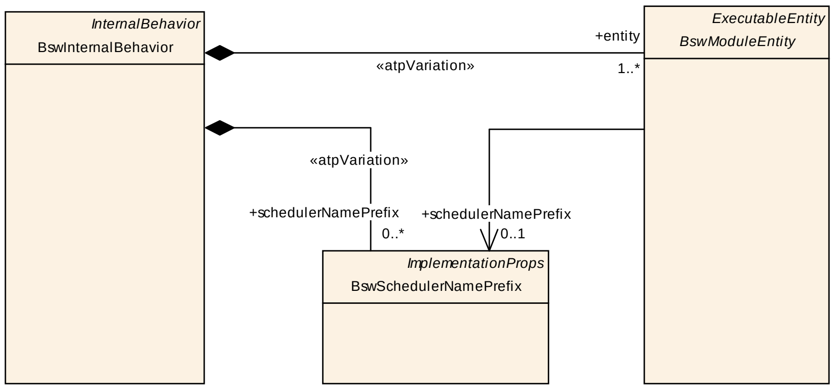

Figure 6.3: Name Prefix for BSW Scheduler artifacts

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	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.10: BswSchedulerNamePrefix

Class	**ImplementationProps (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as prefix when generating code artifacts.			
Base	ARObject,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
symbol	CIdentifier	1	ref	The symbol to be used as prefix.

Table 6.11: ImplementationProps

6.4 BSW Event

[TPS_BSWMDT_4021] Usage of `BswEvent` ⌈ The abstract class `BswEvent` is used as base class for all kinds of events which can start a `BswSchedulableEntity` (which means it does not include direct function calls). ⌋ Figure 6.4 gives an overview on these events and their association to `BswSchedulableEntity`.

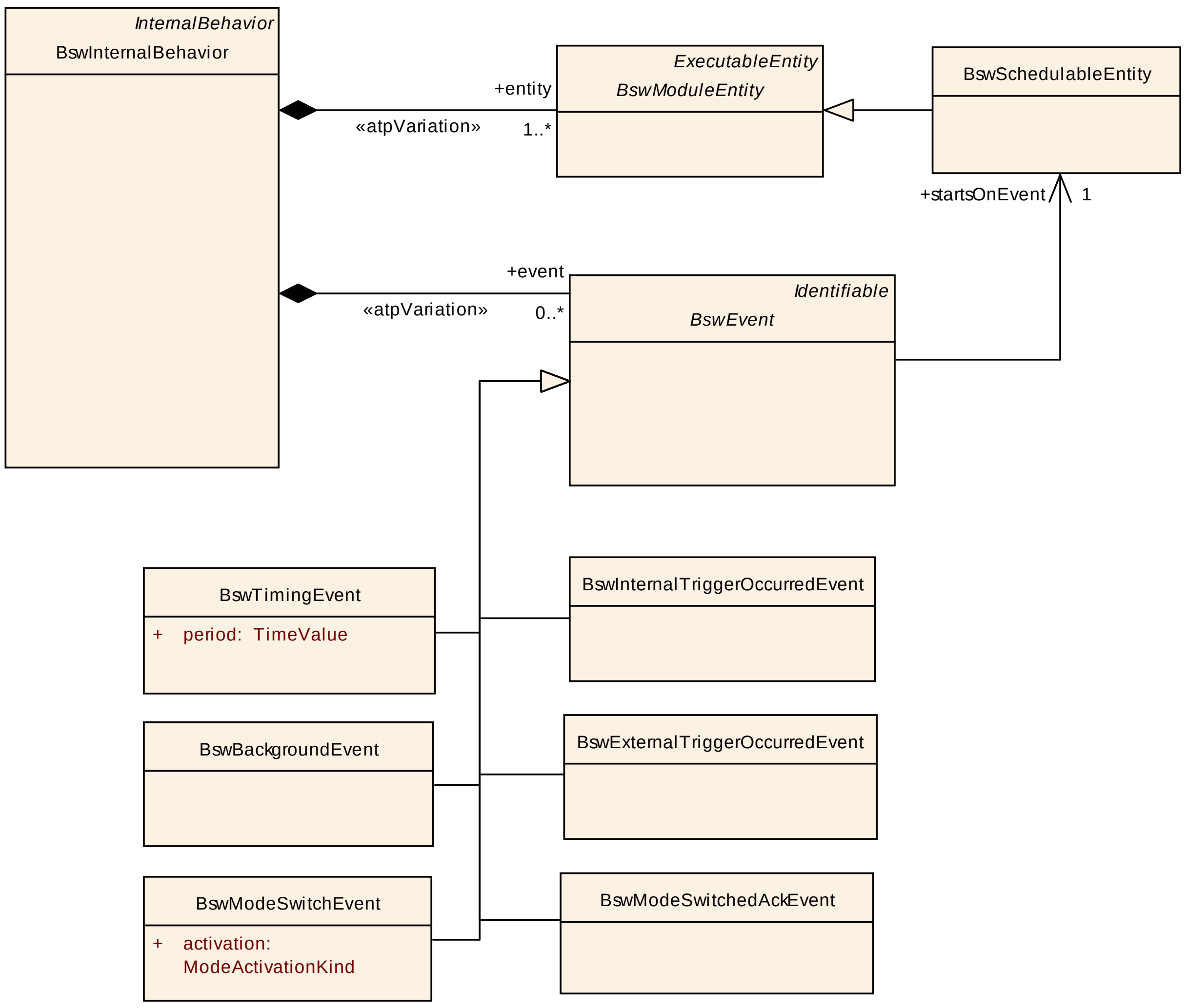

Figure 6.4: Overview on `BswEvents`

Class	**BswEvent (abstract)**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Defines an event which is used to trigger a schedulable entity of this BSW module or cluster. The event is local to the BSW module or cluster. The short name of the class instance is intended as an input to configure the required API of the BSW Scheduler.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
disabledIn Mode	ModeDeclaratio n	*	iref	The modes, in which this event is disabled.
startsOnEv ent	BswSchedulabl eEntity	1	ref	This entity which is started by the event.

Table 6.12: BswEvent

[TPS_BSWMDT_4022] Timing and background events for BSW ⌈ A `BswTimingEvent` and `BswBackgroundEvent` are directly driven by the Scheduler resp. OS without external sources. ⌋

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

Table 6.13: 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,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.14: BswBackgroundEvent

Figure 6.5 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_4023] 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. ⌋ Further information can be found in [16].

[TPS_BSWMDT_4024] External trigger event for BSW ⌈ The `BswExternalTriggerOccurredEvents`, 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. ⌋

[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. ⌋

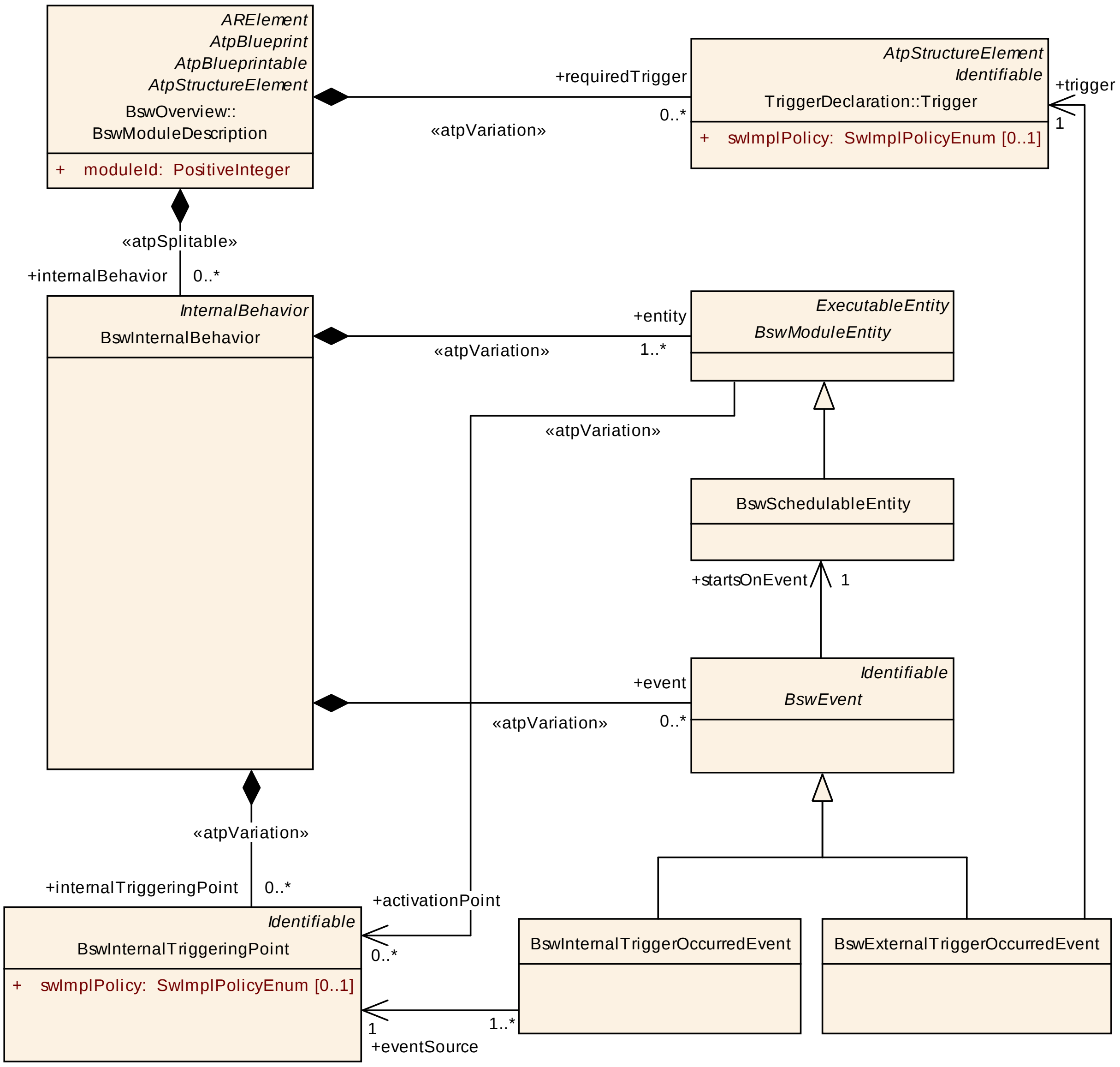

Figure 6.5: Details on BSW Trigger Events

Class	**BswInternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	Represents the activation point for one or more BswInternalTriggerOccurredEvents.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***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 6.15: 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 `BswInternalTriggering-Point.swImplPolicy` ⌈ The **only** allowed values for the attribute `Trigger.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,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
eventSource	BswInternalTriggeringPoint	1	ref	The activation point is the source of this event.

Table 6.16: BswInternalTriggerOccurredEvent

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

Table 6.17: 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.

Figure 6.6 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_4025] 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 `BswEntity` at the entry or exit of a specific mode.
- At the sender side of a mode switch (i.e. in the module providing the mode group), a `BswModeSwitchedAckEvent` can be used to start a `BswEntity` after a mode switch has been acknowledged by the BswScheduler.

⌋

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_4025] Modes used by BSW mode switch event ⌈ The ModeDeclaration used by BswModeSwitchEvent must belong to the ModeDeclarationGroupPrototype referred as BswModuleEntry.accessedModeGroup of the enclosing BswModuleBehavior. ⌋

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

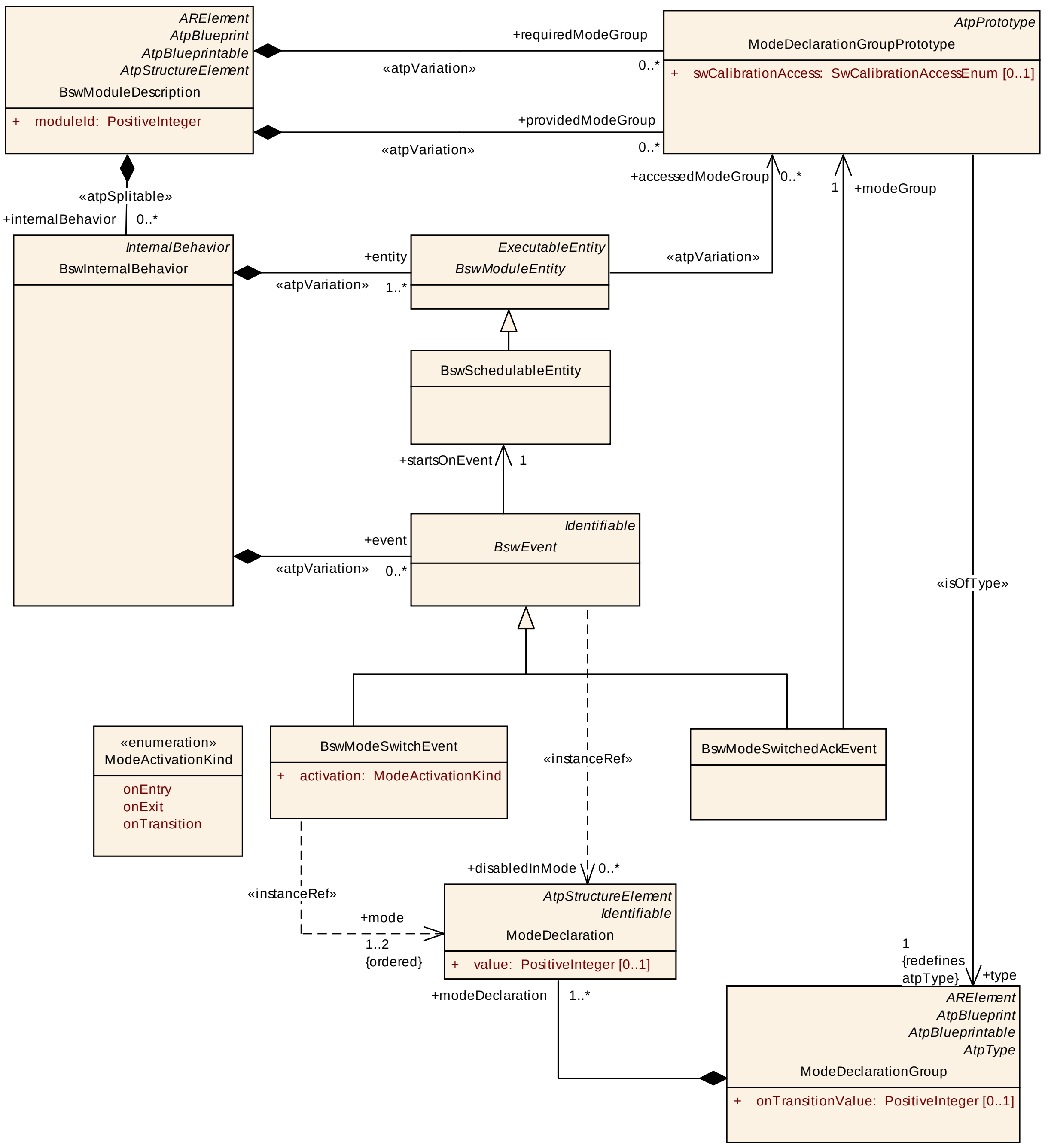

Figure 6.6: Details on BSW Events related to Mode Switches

Class	**BswModeSwitchEvent**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswBehavior			
Note	A BswEvent resulting from a mode switch.			
Base	ARObject,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
activation	ModeActivation Kind	1	attr	Kind of activation w.r.t. to the referred mode.
mode (ordered)	ModeDeclaration	1..2	iref	Reference to one or two Modes that initiate the Mode Switch Event.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Table 6.18: 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,BswEvent,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	1	ref	A mode group provided by this module. The acknowlede of a switch of this group raises this event.

Table 6.19: BswModeSwitchedAckEvent

Class	**ModeDeclaration**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Declaration of one Mode. The name and semantics of a special mode is not defined in the meta-model.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***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 ModeDeclaration.

Table 6.20: ModeDeclaration

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.
onExit	On exiting the referred mode.
onTransition	On transition of the 1st referred mode to the 2nd referred mode.

Table 6.21: ModeActivationKind

6.5 Mode and Trigger Implementation Policy

The implementation of triggers and mode switches can follow various policies which have to be known by the RTE resp. the BswScheduler in order to generate the correct "glue" code. The required attributes are shown in Figure 6.7 and are explained in the class tables below.

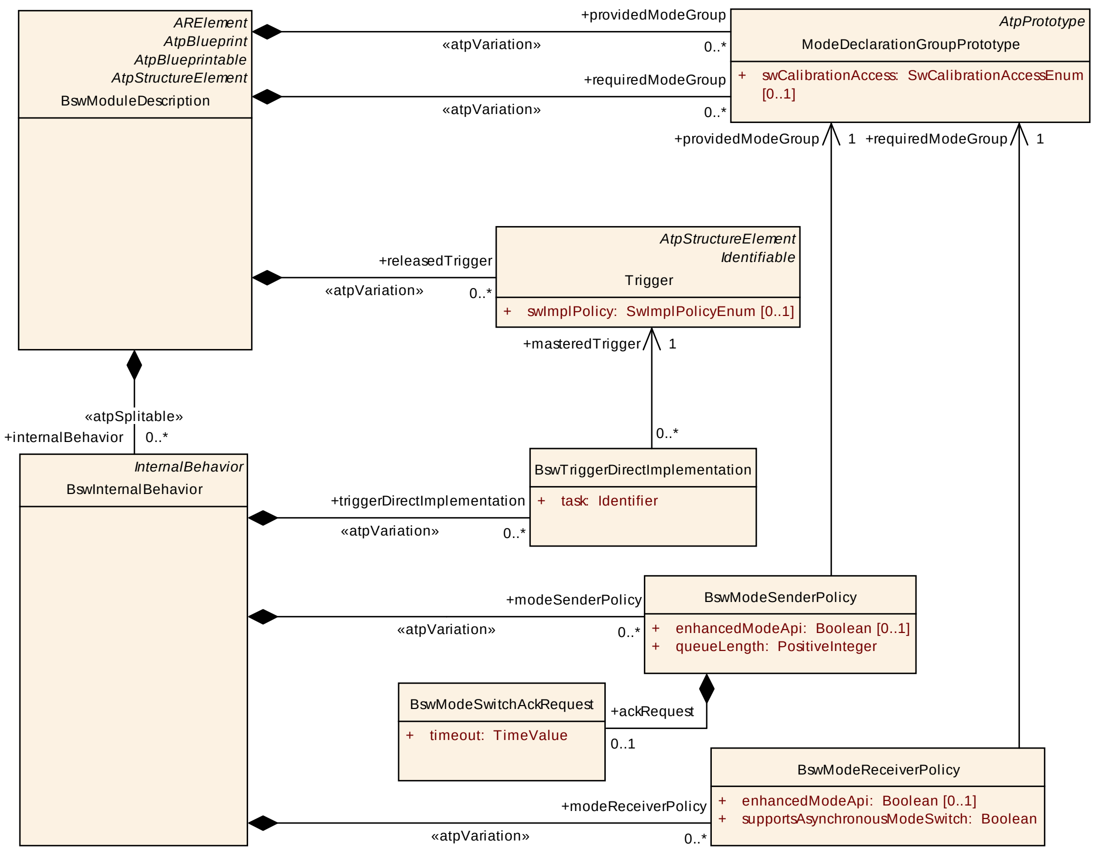

Figure 6.7: Special Implementation Policy for Modes and Triggers

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	***Datatype***	***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	ref	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.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Table 6.22: 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	***Datatype***	***Mul.***	***Kind***	***Note***
ackRequest	BswModeSwitchAckRequest	0..1	aggr	Request for acknowledgement
enhancedModeApi	Boolean	0..1	attr	
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 6.23: BswModeSenderPolicy

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

Table 6.24: 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	***Datatype***	***Mul.***	***Kind***	***Note***
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.
requiredModeGroup	ModeDeclarationGroupPrototype	1	ref	The required mode group for which the policy is specified.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
supportsAsynchronousModeSwitch	Boolean	1	attr	Specifies whether the module can handle the reception of an asynchronous mode switch (true) or not (false).

Table 6.25: BswModeReceiverPolicy

6.6 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 6.8)

[TPS_BSWMDT_4026] 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

⌋

[TPS_BSWMDT_4027] 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.perInstanceMemory`

⌋

For compatibility reasons with the SWCT these various data are declared on the behavior level using the abstract class `InternalBehavior` as shown in figure 6.8. The class table for `InternalBehavior` has already been listed in chapter 6.1.

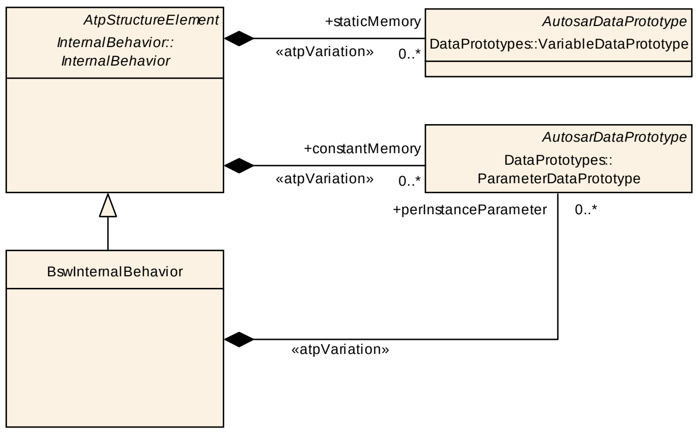

Figure 6.8: BSW Local Data

These data use the type system of `AutosarDataProtoype`s which is explained in more detail in [14]:

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,Data Prototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the ParameterDataPrototype

Table 6.26: 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,Data Prototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table 6.27: VariableDataPrototype

6.7 Synchronization with a Corresponding SWC

BSW modules which implement a ServiceSwComponentType, EcuAbstractionSwComponentType or ComplexDriverSwComponentType 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: A BSW modules 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 [16]). 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.

Another use case is the specification of a RunnableEntity in order to allow calls to or from the RTE via ports. In this case, an BswExecutableEntity 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 an RunnableEntity runs in interrupt context.

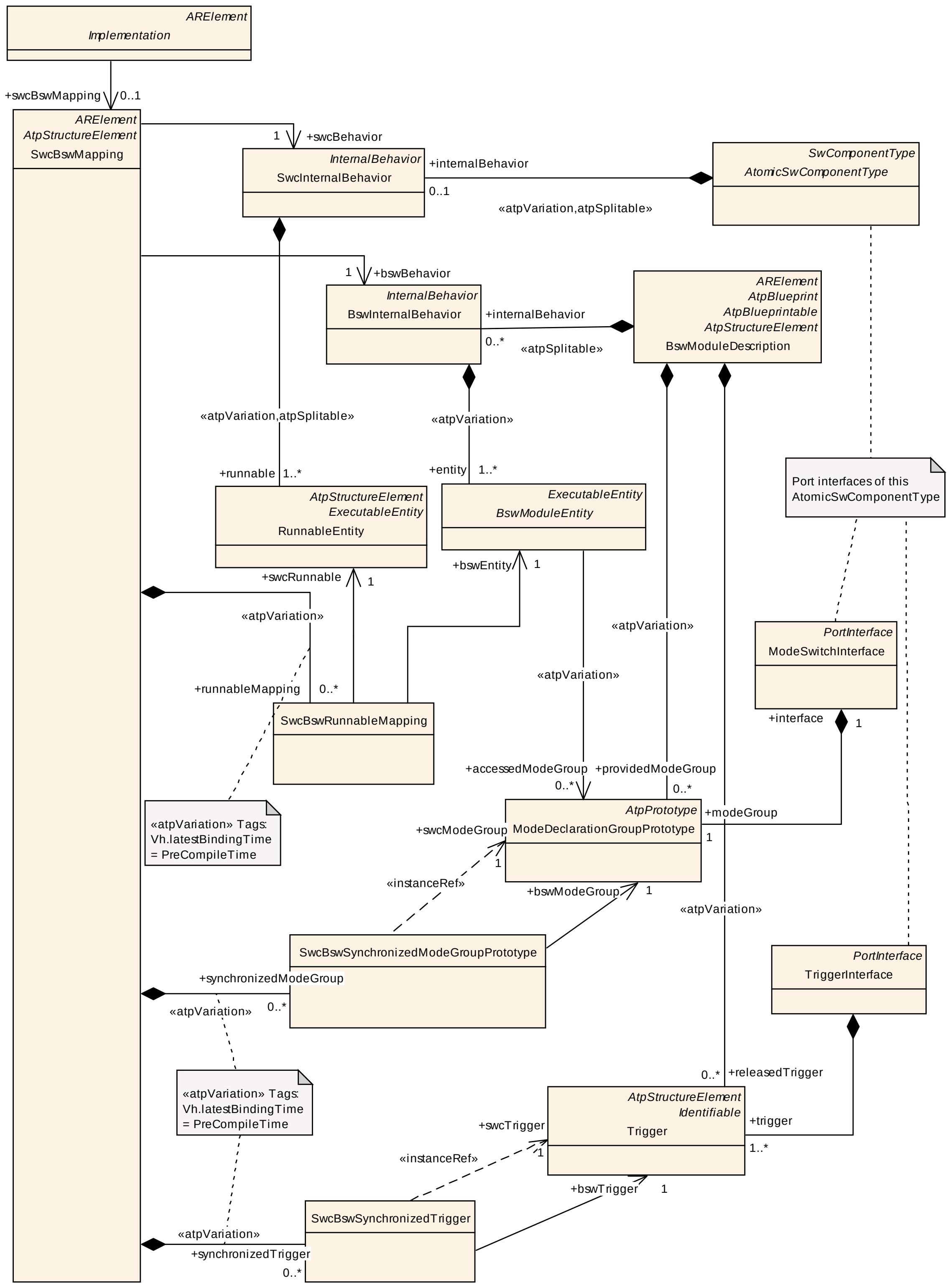

Figure 6.9: Mapping between an SWC and a BSW module.

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 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,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
bswBehavi or	BswInternalBeh avior	1	ref	The mapped BswInternalBehavior
runnableM apping	SwcBswRunnab leMapping	*	aggr	A mapping between a pair of SWC and BSW runnables. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
swcBehavi or	SwcInternalBeh avior	1	ref	The mapped SwcInternalBehavior.
synchroniz edModeGr oup	SwcBswSynchr onizedModeGro upPrototype	*	aggr	A pair of SWC and BSW mode group prototypes to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
synchroniz edTrigger	SwcBswSynchr onizedTrigger	*	aggr	A pair of SWC and BSW Triggers to be synchronized by the scheduler. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 6.28: 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 Device 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	***Datatype***	***Mul.***	***Kind***	***Note***
bswEntity	BswModuleEntit y	1	ref	The mapped BswModuleEntity
swcRunna ble	RunnableEntity	1	ref	The mapped SWC runnable.

Table 6.29: 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	***Datatype***	***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 6.30: 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	***Datatype***	***Mul.***	***Kind***	***Note***
bswTrigger	Trigger	1	ref	The BSW Trigger.
swcTrigger	Trigger	1	iref	The SWC Trigger provided by a particular port.

Table 6.31: SwcBswSynchronizedTrigger

[TPS_BSWMDT_4028] 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.`⌋

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

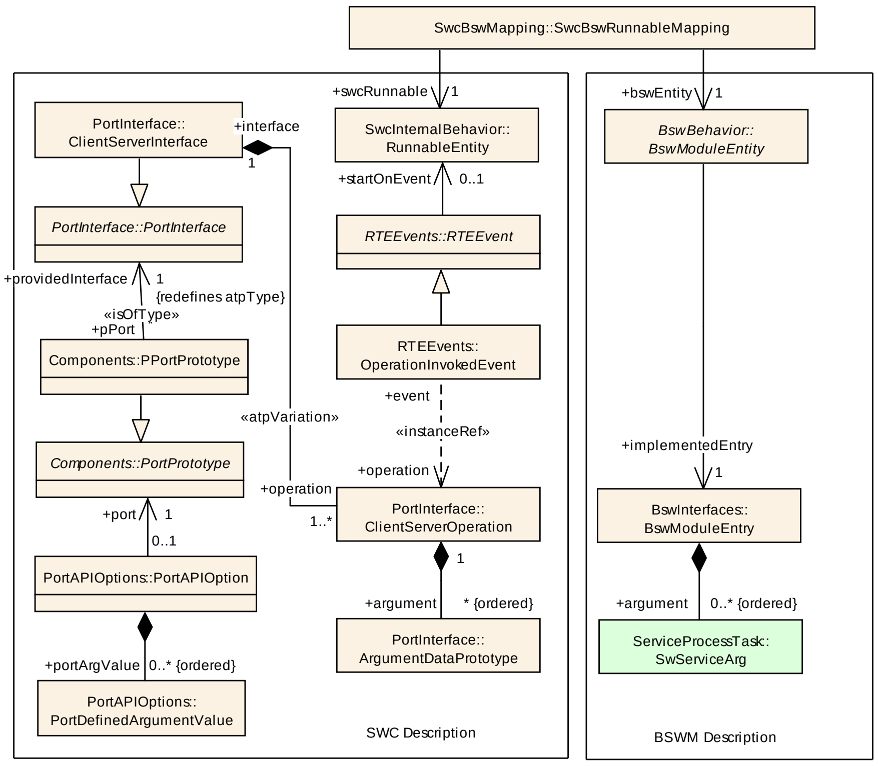

Figure 6.10: Mapping of function arguments between an SWC and a BSW module.

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`.

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`. ⌋

Further constraints are:

[constr_4040] Synchronized mode groups must have same type ⌈ `SwcBswSynchronizedModeGroupPrototype` can only refer to equally typed `ModeDeclarationGroupPrototype`s, i.e. which have identical `ModeDeclarationGroup`s. ⌋

[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 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 `ModeDeclarationGroup`s having the same `shortName` but different elements. This holds especially if these mode groups are not synchronized but used independently. ⌋

6.8 BSW Service Needs

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 6.11 shows the various meta-classes which can be used on the behavior level of BSW modules and SWCs in order to express these dependencies.

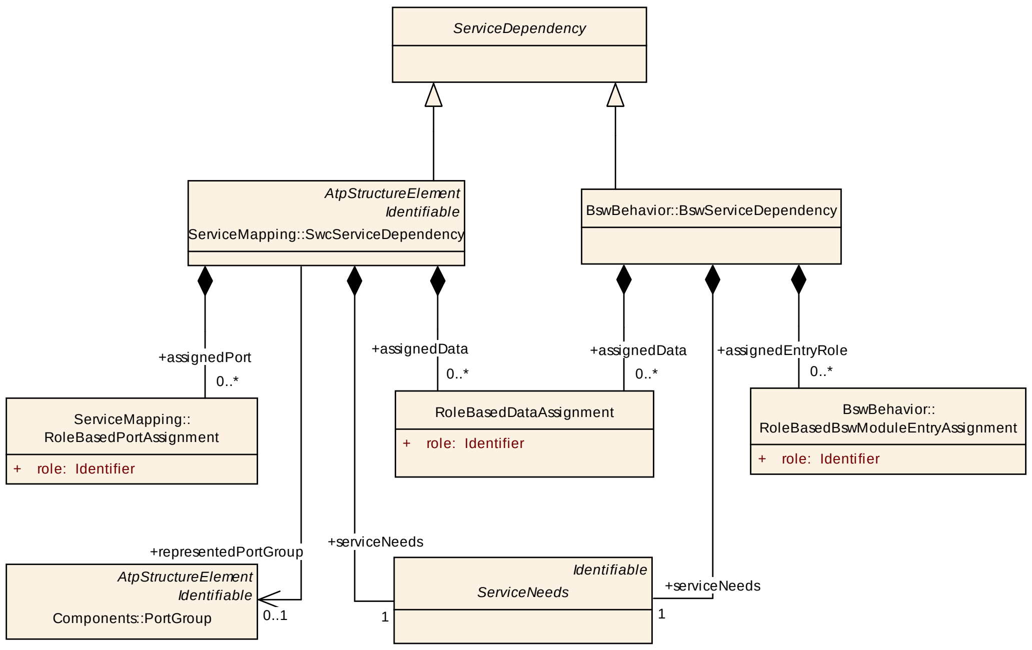

Figure 6.11: Concept of `ServiceDependency` for BSW and SWC

[TPS_BSWMDT_4029] Usage of `BswServiceDependency` ⌈ In figure 6.12 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). ⌋

For further explanation refer to the class tables below.

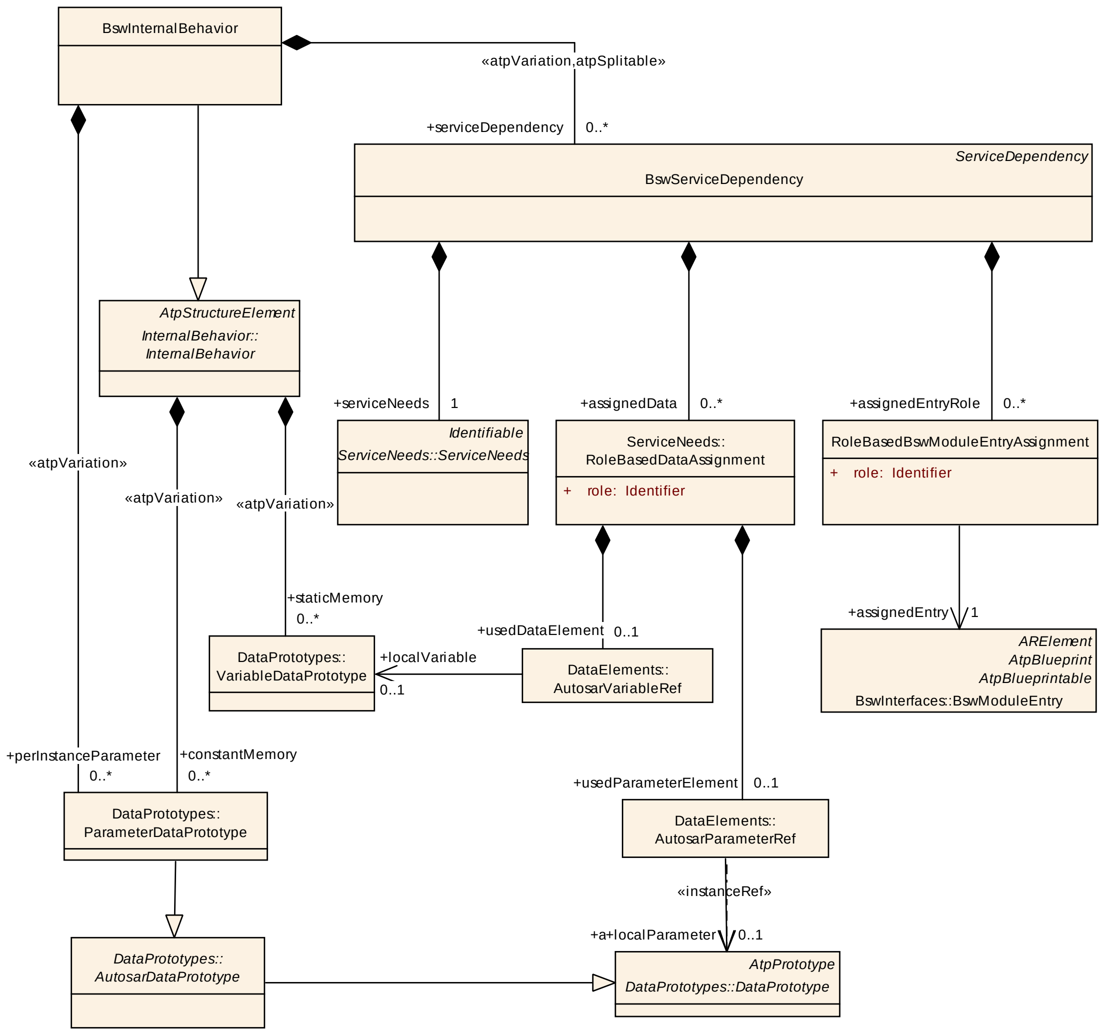

Figure 6.12: `BswServiceDependency` attached to a `BswModuleBehavior`

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 Nv block, a diagnostic event etc.). It defines the quality of service (ServiceNeeds) 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			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.32: 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	*Datatype*	*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.
assignedEntryRole	RoleBasedBswModuleEntryAssignment	*	aggr	Defines the role of an associated BswModuleEntry in the context of the ServiceNeeds element.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table 6.33: 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.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
assignedEntry	BswModuleEntry	1	ref	The assigned entry. It should be a providedEntry or expectedCallback of the module or cluster that requires the ServiceNeeds.
role	Identifier	1	ref	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 6.34: 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	**Datatype**	**Mul.**	**Kind**	**Note**
role	Identifier	1	ref	This is the role of the assigned data in the given context, for example for an Nv 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: • **ramMirror** indicates data to be used as a mirror for an Nv block. • **defaultData** indicates constant data to be used as default in the context of this ServiceNeeds, e.g. for an Nv block. • **signalBasedDiagnostics** indicates the RoleBasedDataAssignment shall be used for signal based diagnostics.
usedDataElement	AutosarVariableRef	0..1	aggr	The VariableDataPrototype used in this role, e.g. • RAM mirror for an Nv block which must belong to the same SwcInternalBehavior or BswInternalBehavior. • In the role signalBasedDiagnostics it has to refer to a VariableDataPrototype in a SenderReceiverInterface or a NvDataInterface.
usedParameterElement	AutosarParameterRef	0..1	aggr	The ParameterDataPrototype used in this role, e.g. • ROM default for an Nv block. It must belong to the same SwcInternalBehavior or BswInternalbehavior. • In the role signalBasedDiagnostics it has to refer to a ParameterDataPrototype in a ParameterInterface.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
usedPim	PerInstanceMe mory	0..1	ref	The (untyped) PerInstanceMemory used in this role (e.g. as a RAM mirror for an Nv block).

Table 6.35: RoleBasedDataAssignment

Note that several kinds of data assignments are restricted to be used within an SWC because they need RTE support:

[constr_4051] `RoleBasedDataAssignment` in BSW ⌈ When used in the context of `BswServiceDependency`, the following restriction hold for date references described by `RoleBasedDataAssignment`:

- Within `RoleBasedDataAssignment.usedDataEelement`, 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.

⌋

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			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.36: ServiceNeeds

The class `ServiceNeeds` and its derivatives is defined in the `CommonStructure` package of the meta-model. These classes are shown in two figures (6.13 and 6.14).

The subsequent tables show those specialized `ServiceNeeds` which are of interest for the basic software. Not all classes for diagnostic capabilities are shown here, because they are mainly of interest for application software. For a detailed description of the diagnostic capabilities refer to [14].

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 topic "Derived Parameter Definition" in [17]. 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.

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.

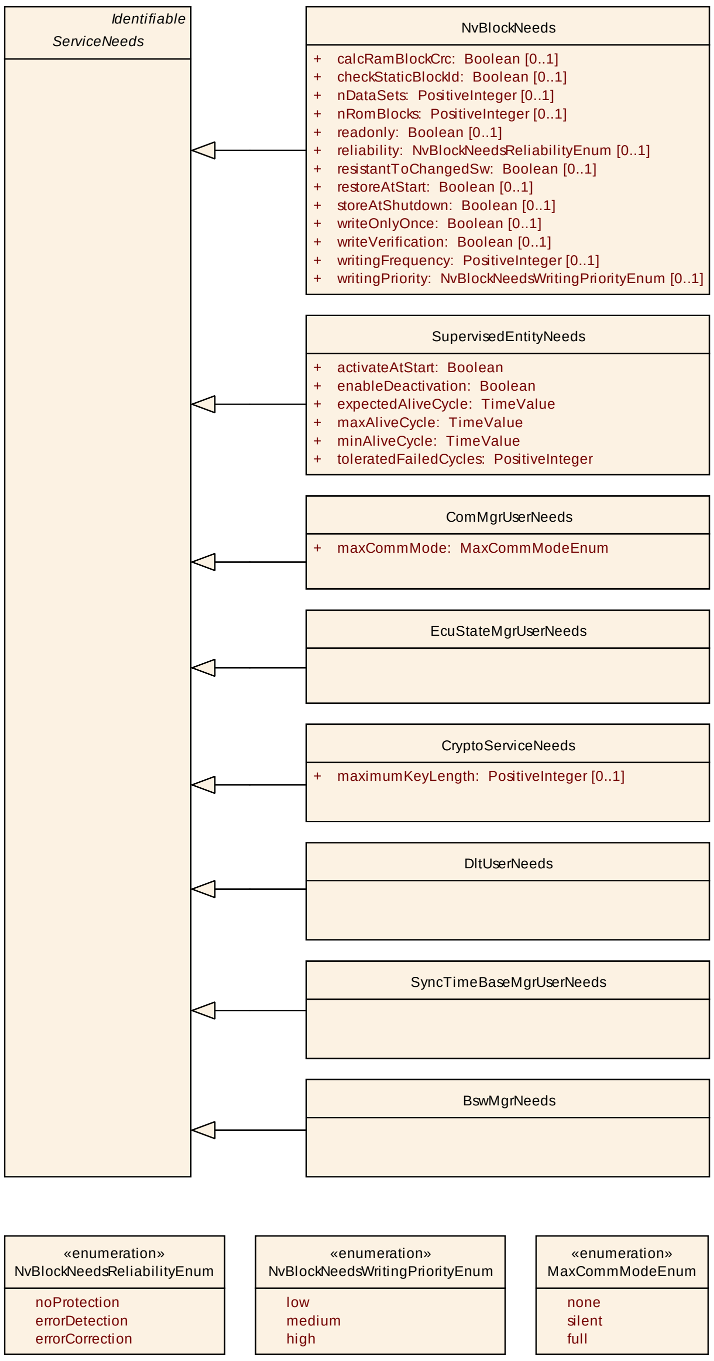
Identifiable
ServiceNeeds
NvBlockNeeds
+ calcRamBlockCrc: Boolean [0..1]
+ checkStaticBlockId: Boolean [0..1]
+ nDataSets: PositiveInteger [0..1]
+ nRomBlocks: PositiveInteger [0..1]
+ readonly: Boolean [0..1]
+ reliability: NvBlockNeedsReliabilityEnum [0..1]
+ resistantToChangedSw: Boolean [0..1]
+ restoreAtStart: Boolean [0..1]
+ storeAtShutdown: Boolean [0..1]
+ writeOnlyOnce: Boolean [0..1]
+ writeVerification: Boolean [0..1]
+ writingFrequency: PositiveInteger [0..1]
+ writingPriority: NvBlockNeedsWritingPriorityEnum [0..1]
SupervisedEntityNeeds
+ activateAtStart: Boolean
+ enableDeactivation: Boolean
+ expectedAliveCycle: TimeValue
+ maxAliveCycle: TimeValue
+ minAliveCycle: TimeValue
+ toleratedFailedCycles: PositiveInteger
ComMgrUserNeeds
+ maxCommMode: MaxCommModeEnum
EcuStateMgrUserNeeds
CryptoServiceNeeds
+ maximumKeyLength: PositiveInteger [0..1]
DltUserNeeds
SyncTimeBaseMgrUserNeeds
BswMgrNeeds
«enumeration»
NvBlockNeedsReliabilityEnum
noProtection
errorDetection
errorCorrection
«enumeration»
NvBlockNeedsWritingPriorityEnum
low
medium
high
«enumeration»
MaxCommModeEnum
none
silent
full

Figure 6.13: Class `ServiceNeeds` from `CommonStructure` and some derived classes

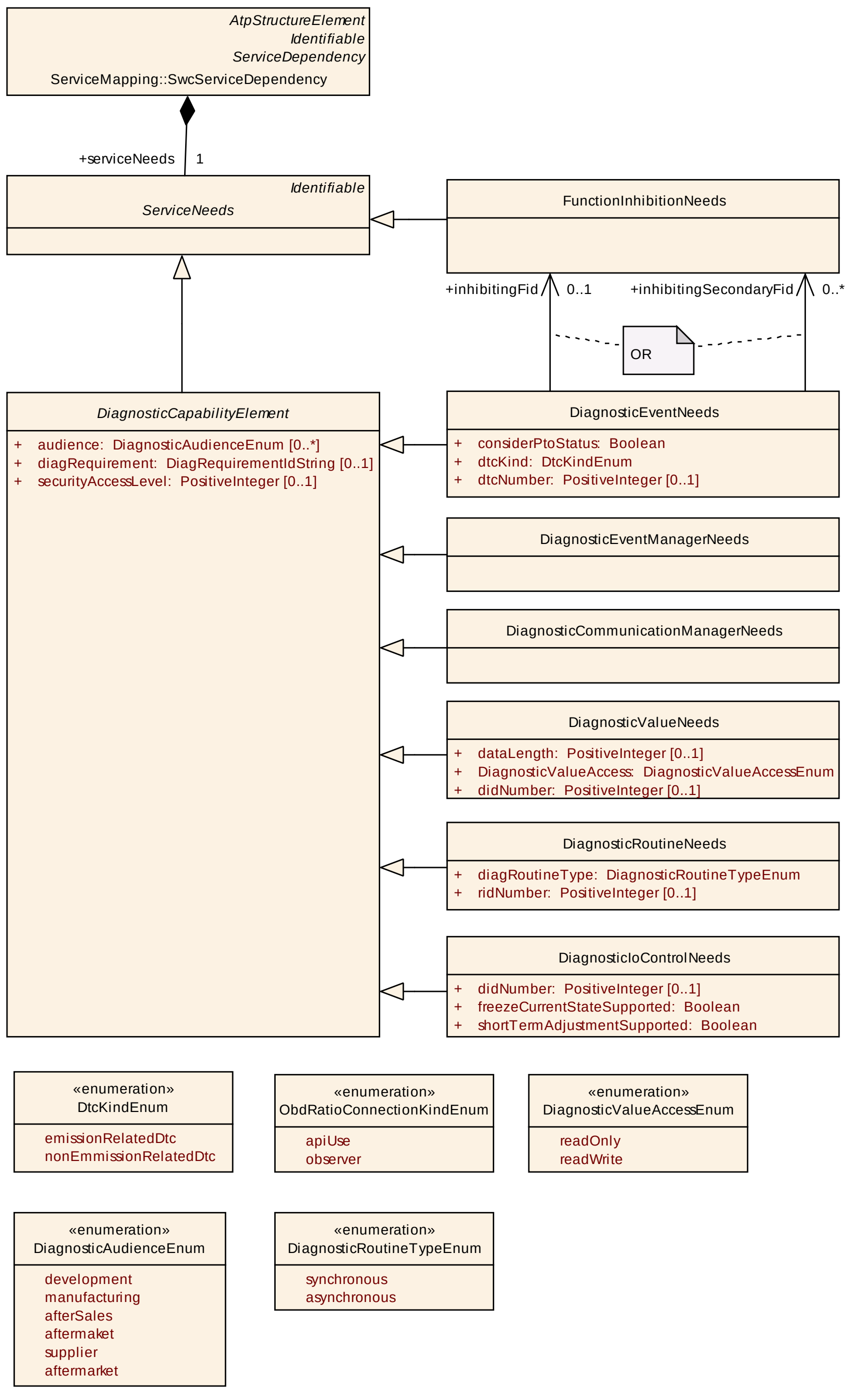
AtpStructureElement
Identifiable
ServiceDependency
ServiceMapping::SwcServiceDependency
+serviceNeeds
1
Identifiable
ServiceNeeds
FunctionInhibitionNeeds
+inhibitingFid
0..1
+inhibitingSecondaryFid
0..*
OR
DiagnosticCapabilityElement
+ audience: DiagnosticAudienceEnum [0..*]
+ diagRequirement: DiagRequirementIdString [0..1]
+ securityAccessLevel: PositiveInteger [0..1]
DiagnosticEventNeeds
+ considerPtoStatus: Boolean
+ dtcKind: DtcKindEnum
+ dtcNumber: PositiveInteger [0..1]
DiagnosticEventManagerNeeds
DiagnosticCommunicationManagerNeeds
DiagnosticValueNeeds
+ dataLength: PositiveInteger [0..1]
+ DiagnosticValueAccess: DiagnosticValueAccessEnum
+ didNumber: PositiveInteger [0..1]
DiagnosticRoutineNeeds
+ diagRoutineType: DiagnosticRoutineTypeEnum
+ ridNumber: PositiveInteger [0..1]
DiagnosticIoControlNeeds
+ didNumber: PositiveInteger [0..1]
+ freezeCurrentStateSupported: Boolean
+ shortTermAdjustmentSupported: Boolean
«enumeration»
DtcKindEnum
emissionRelatedDtc
nonEmmissionRelatedDtc
«enumeration»
ObdRatioConnectionKindEnum
apiUse
observer
«enumeration»
DiagnosticValueAccessEnum
readOnly
readWrite
«enumeration»
DiagnosticAudienceEnum
development
manufacturing
afterSales
aftermaket
supplier
aftermarket
«enumeration»
DiagnosticRoutineTypeEnum
synchronous
asynchronous

Figure 6.14: 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 Nv block.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***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.
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 NV Blocks.
nRomBlocks	PositiveInteger	0..1	attr	Number of ROM blocks to be provided by the NVRAM manager for this block. Please not that these multiple ROM Blocks are given in a contiguous area.
readonly	Boolean	0..1	attr	True: data of this 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 Nv block shall be treated resistant to configuration changes (true) or not (false). For details how to handle initialization in the latter case, refer to the NVRAM specification.
restoreAtStart	Boolean	0..1	attr	Defines whether the associated RAM mirror block shall be implicitly restored during startup by the basic SW or not. Only relevant if a RAM mirror block is associated with this port (for Software Components the latter is modeled via SwcServiceDependency).
storeAtShutdown	Boolean	0..1	attr	Defines whether or not the associated RAM mirror block shall be implicitly stored during shutdown by the basic SW. This is only relevant if a RAM mirror block is associated with this port (for software-components the latter is modeled by means of a SwcServiceDependency).
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 SWC. false: No such restriction.
writeVerification	Boolean	0..1	attr	Defines if Write Verification shall be enabled for this Nv Block.
writingFrequency	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	NvBlockNeedsWritingPriorityEnum	0..1	attr	Requires the priority of writing this block in case of concurrent requests to write other blocks.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Table 6.37: NvBlockNeeds

Class	**SupervisedEntityNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs on the configuration of the Watchdog Manager for one specific Supervised Entity (SE).			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
activateAtStart	Boolean	1	attr	True/false: supervision activation status of SE shall be enabled/disabled at start.
enableDeactivation	Boolean	1	attr	True: software-component shall be allowed to deactivate supervision of this SE false: not
expectedAliveCycle	TimeValue	1	attr	Expected cycle time of alive trigger of this SE (in seconds).
maxAliveCycle	TimeValue	1	attr	Maximum cycle time of alive trigger of this SE (in seconds).
minAliveCycle	TimeValue	1	attr	Minimum cycle time of alive trigger of this SE (in seconds).
toleratedFailedCycles	PositiveInteger	1	attr	Number of consecutive failed alive cycles for this SE which shall be tolerated until the supervision status of the SE is set to EXPIRED (see WdgM documentation for details). Note that this has to be recalculated w.r.t. the WdgMs own cycle time for ECU configuration.

Table 6.38: 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	***Datatype***	***Mul.***	***Kind***	***Note***
maxCommMode	MaxCommModeEnum	1	attr	Maximum communication mode requested by this ComM user.

Table 6.39: 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	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 6.40: 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 an SWC belong to this ConfigID.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
maximumKeyLength	PositiveInteger	0..1	attr	The maximum length of a cryptographic key, that is used by the SWC or module for this configuration.

Table 6.41: CryptoServiceNeeds

Class	**DltUserNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	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 ports of an SWC belong to this SessionId in order to group the request and response ports of the same SessionId. The actual SessionId value is stored in the PortDefinedArgumentValue of the respective port specification.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.42: 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	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.43: SyncTimeBaseMgrUserNeeds

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 name can be regarded as a symbol identifying the diagnostic event from the viewpoint of the component or module which owns this class.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
considerPtoStatus	Boolean	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.
diagEventDebounceAlgorithm	DiagEventDebounceAlgorithm	1	aggr	Specifies the abstract need on the Debounce Algorithm applied by the Diagnostic Event Manager.
dtcKind	DtcKindEnum	1	attr	This attribute indicates the kind of the diagnostic monitor according to the SWS Diagnostic Event Manger.
dtcNumber	PositiveInteger	0..1	attr	This represents a reasonable Diagnostic Trouble Code. This allows to predefine the Diagnostic Trouble Code if the a function developer has received a particular requirement from the OEM or from a standardization body.
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.
inhibitingSecondaryFid	FunctionInhibitionNeeds	*	ref	This represents the secondary 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. The "primary" and all "secondary" FID inhibitions are combined by "OR".

Table 6.44: DiagnosticEventNeeds

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	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 6.45: FunctionInhibitionNeeds

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).

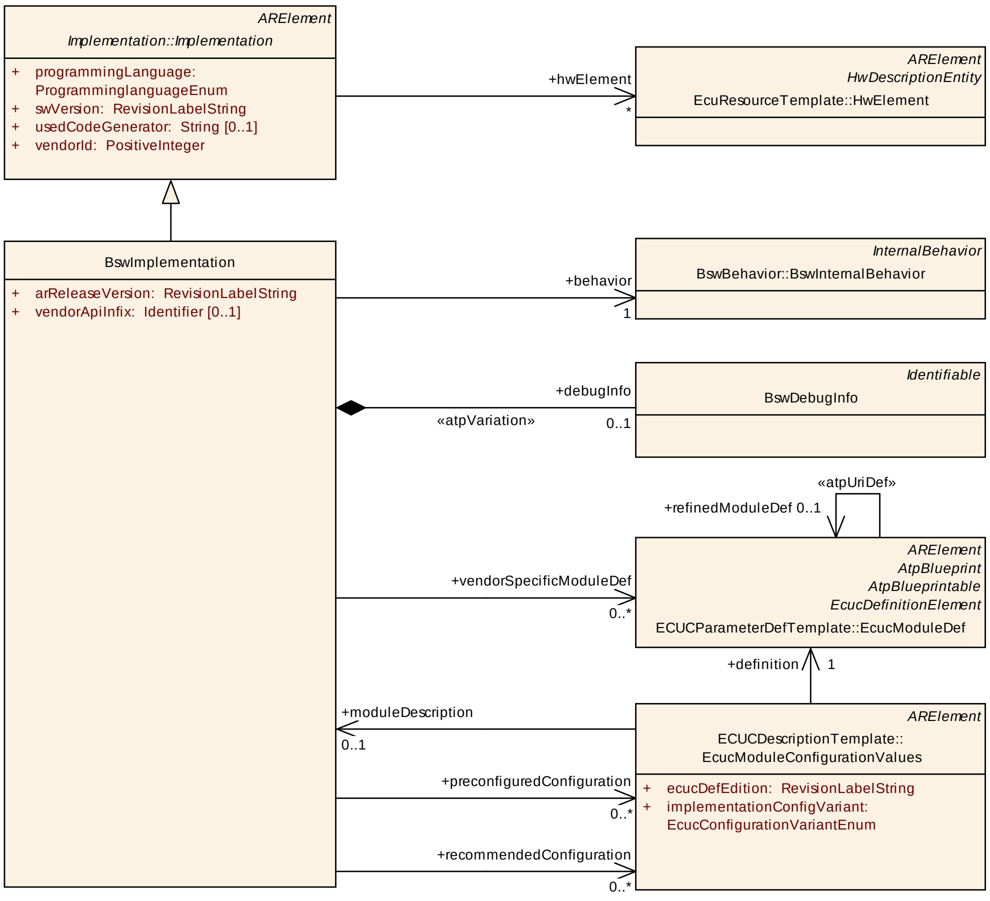
ARElement
Implementation::Implementation
+ programmingLanguage: ProgramminglanguageEnum
+ swVersion: RevisionLabelString
+ usedCodeGenerator: String [0..1]
+ vendorId: PositiveInteger
+hwElement
*
ARElement
HwDescriptionEntity
EcuResourceTemplate::HwElement
BswImplementation
+ arReleaseVersion: RevisionLabelString
+ vendorApiInfix: Identifier [0..1]
+behavior
1
InternalBehavior
BswBehavior::BswInternalBehavior
+debugInfo
«atpVariation»
0..1
Identifiable
BswDebugInfo
«atpUriDef»
+refinedModuleDef 0..1
+vendorSpecificModuleDef
0..*
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionElement
ECUCParameterDefTemplate::EcucModuleDef
+definition
1
+moduleDescription
0..1
ARElement
ECUCDescriptionTemplate::EcucModuleConfigurationValues
+ ecucDefEdition: RevisionLabelString
+ implementationConfigVariant: EcucConfigurationVariantEnum
+preconfiguredConfiguration
0..*
+recommendedConfiguration
0..*

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 (BswModuleDescription 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,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arRelease Version	RevisionLabelSt ring	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	BswInternalBeh avior	1	ref	The behavior of this implementation.
debugInfo	BswDebugInfo	0..1	aggr	Collects the debug info for this implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
preconfigur edConfigur ation	EcucModuleCo nfigurationValue s	*	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
recommen dedConfig uration	EcucModuleCo nfigurationValue s	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiI nfix	Identifier	0..1	ref	In driver modules which can be instantiated several times on a single ECU, BSW00347 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.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
vendorSpecificModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this BswImplementation if it represents a cluster of modules • one or no EcucModuleDefs used in this BswImplementation if it represents a library **Tags:** xml.roleWrapperElement=true

Table 7.1: BswImplementation

[TPS_BSWMDT_4030] **`BswImplementation.arReleaseVersion`** ⌈The inclusion of the AUTOSAR version information `arReleaseVersion` is specific for AUTOSAR BSW and specified per instance of `BswImplementation`. ⌋

[TPS_BSWMDT_4031] Instances of `BswImplementation` ⌈ Note that in case a BSW module is 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`. ⌋

With attribute `debugInfo` it is possible to specify information for the AUTOSAR BSW Debug Module. This is further explained in chapter 7.3.

[TPS_BSWMDT_4032] **`BswImplementation.requiredHW`** ⌈ The attribute `requiredHW` allows to document special hardware dependencies of a BSW module or cluster in addition to what can be expressed by the generic attributes `Implementation.processor` and `Implementation.resourceConsumption` ⌋ (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_4033] Reference to vendor specific configuration parameters ⌈ Vendor specific configuration parameters are expressed by an association from `BswImplementation` to `EcucModuleDef`. ⌋

[TPS_BSWMDT_4034] Reference to predefined or recommended configuration values ⌈ Predefined or recommended configuration parameter values are expressed by associations from `BswImplementation` to `EcucModuleConfigurationValues`. ⌋

The meta-classes `EcucModuleDef` and `EcucModuleConfigurationValues` are specified in the ECU Configuration Specification document [18].

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 `EcuModuleConfigurationValues` if the underlying `BswModuleDescription` has the `category` BSW_CLUSTER. In all other cases, it shall refer to at most one instance of `EcuModuleConfigurationValues` (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 [18] 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 `PreconfiguredConfiguration` shall only be referenced in the role `preconfiguredConfiguration` and no other value for `implementation-ConfigVariant` is allowed in this role. ⌋

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

[TPS_BSWMDT_4035] 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 `preconfigured-Configuration`. ⌋

[TPS_BSWMDT_4036] 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. ⌋

7.3 BSW Debug Information

A BSW Module can declare local data for being accessible be the AUTOSAR BSW Debug Module. Note that this is a limited kind of debugging available for the integrator and has nothing to do with more powerful debugging tools the developer might use.

[TPS_BSWMDT_4037] `BswDebugInfo` ⌈As shown in Figure 7.2 the container class `BswDebugInfo` is used to aggregate all data declarations exported from one module for debugging. These can be local data, which otherwise would be not visible in the description, or data that are already declared on the behavior level for measurement or calibration. ⌋

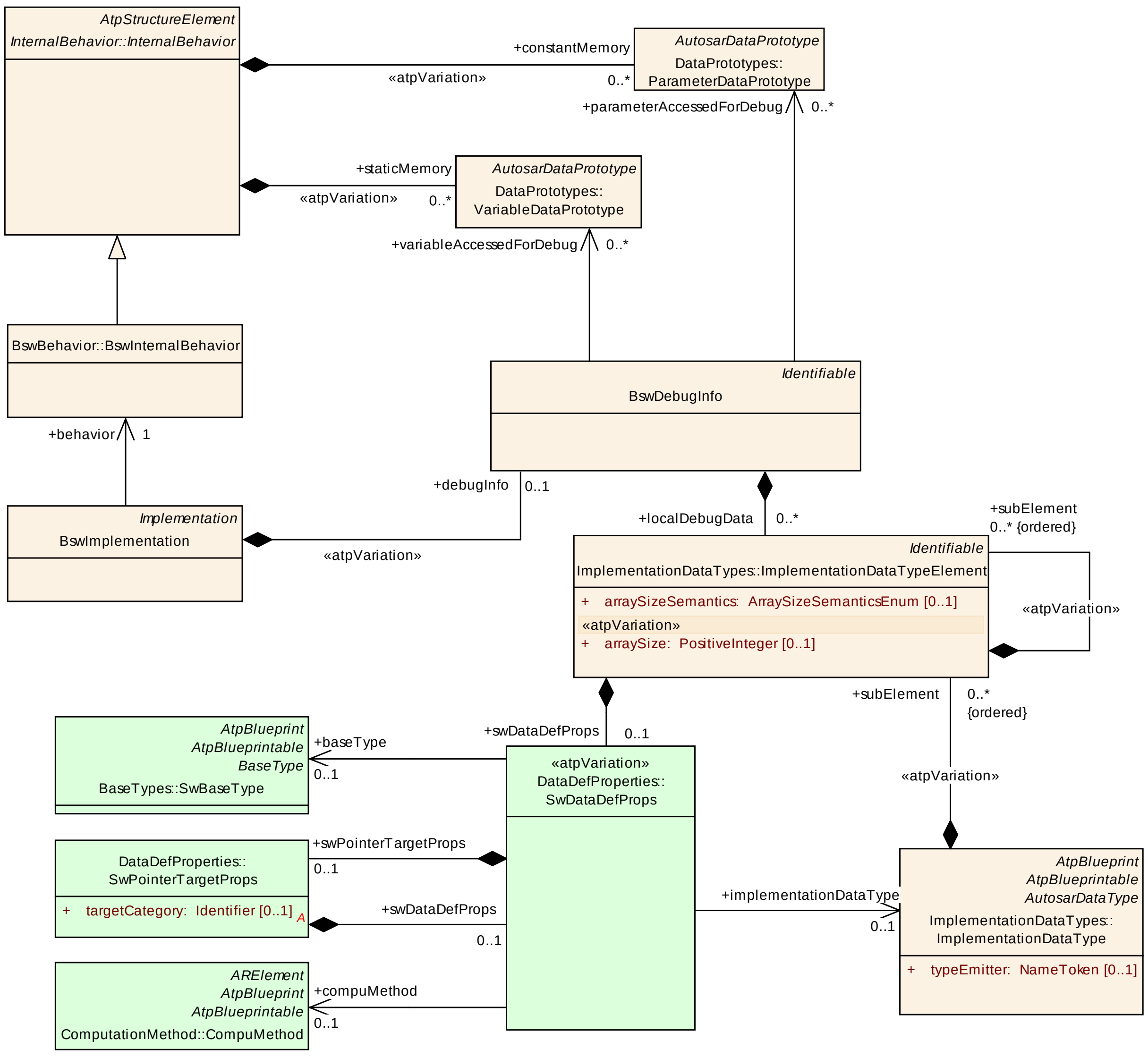

Figure 7.2: Aggregation of BswDebugInfo

Class	**BswDebugInfo**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Collects the information on the data provided to the AUTOSAR debug module.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
localDebug Data	Implementation DataTypeEleme nt	*	aggr	A data element declared locally to this module, cluster or library. It shall be used (within AUTOSAR) only for debugging purposes.
parameter AccessedF orDebug	ParameterData Prototype	*	ref	Indicates a parameter as to be debugged.
variableAc cessedFor Debug	VariableDataPr ototype	*	ref	Indicates a variable as to be debugged.

Table 7.2: BswDebugInfo

[TPS_BSWMDT_4038] Data types for debug data ⌈For the further detailing of `BswDebugInfo.localDebugData`, the system of `ImplementationDataTypes` is used which is defined in the CommonStructure part of the meta-model. ⌋

The usage of these data types is similar to the the declaration of `SwServiceArg` as explained in chapter chapter 5.1. For more details refer to [7].

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)

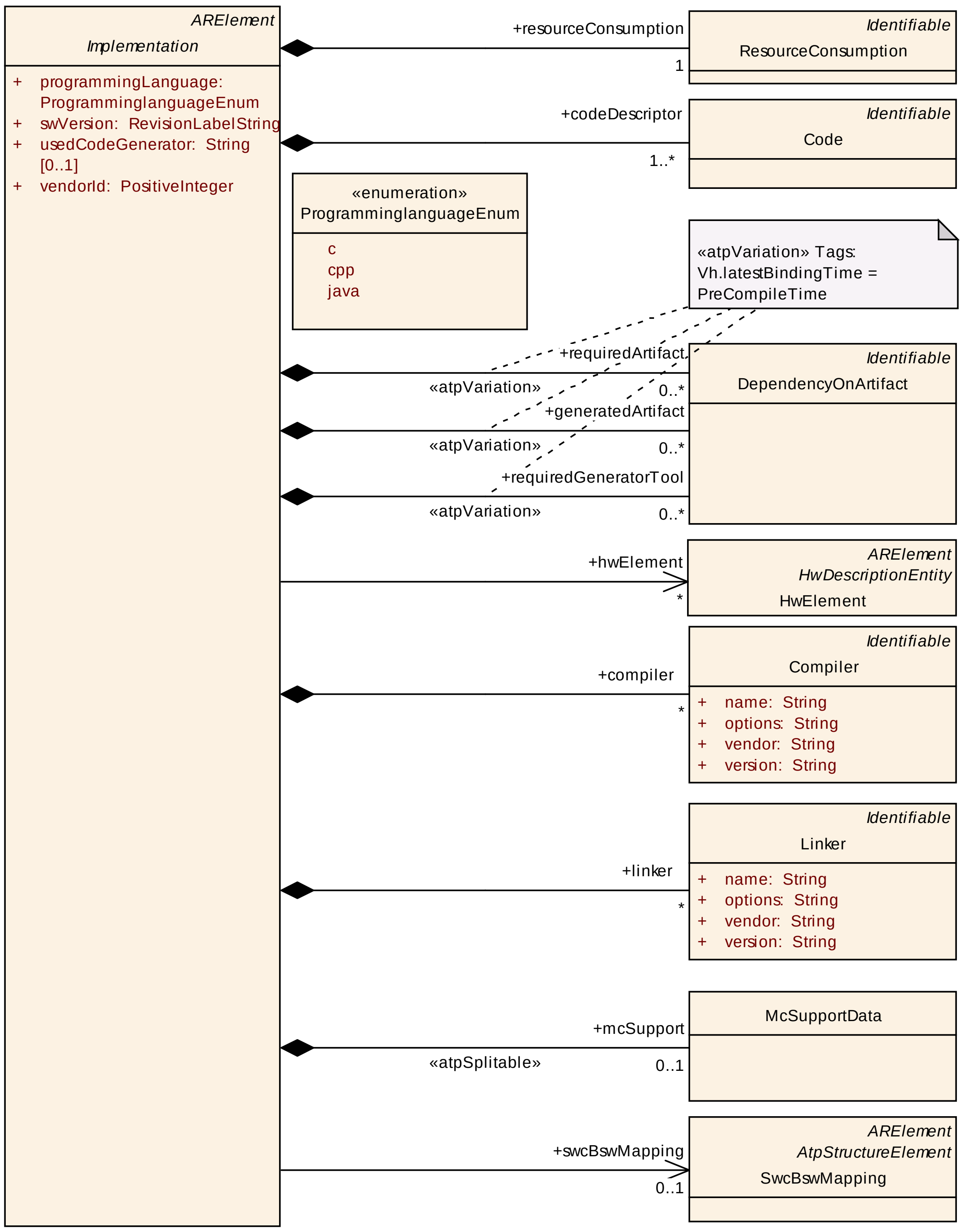

Figure 8.1: Overview of implementation description

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,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
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

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
resourceConsumption	ResourceConsumption	1	aggr	All static and dynamic resources for each implementation are described within the ResourceConsumption class.
swVersion	RevisionLabelString	1	attr	Software version of this implementation. The numbering contains three levels (like major, minor, patch), its values are vendor specific.
swcBswMapping	SwcBswMapping	0..1	ref	This allows a mapping between an SWC and a BSW behavior to be attached to an implementation description (for 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 BswImplementtion or for both.
usedCodeGenerator	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_4039] 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 `BswBehavior`. 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`. ⌋

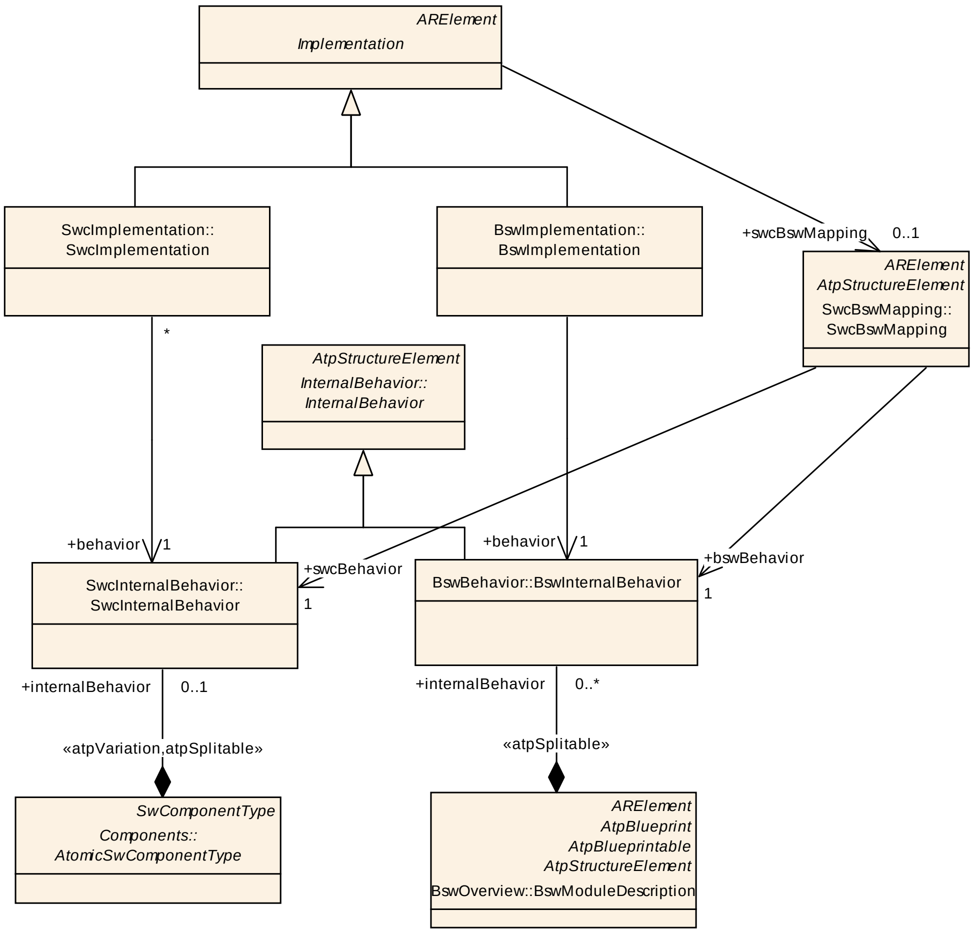

Figure 8.2: An implementation is associated with a single software component or module

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_4040] **`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. ⌋

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

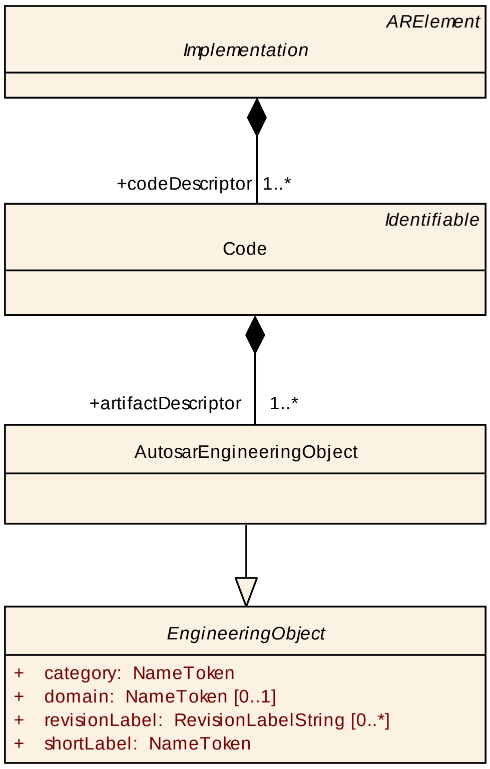

Figure 8.3: An `Implementation` references the code artifacts through the `Code` class

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	***Datatype***	***Mul.***	***Kind***	***Note***
artifactDescriptor	AutosarEngineeringObject	1..*	aggr	Refers to the artifact belonging to this code descriptor.

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_4041] `DependencyOnArtifact` ⌈This is described by the class `DependencyOnArtifact` which relates to meta-information via the class `AutosarEngineeringObject` as shown in Figure 8.4. ⌋

[TPS_BSWMDT_4042] 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. ⌋This mechanism is described in more detail in the AUTOSAR Methodology, see [5].

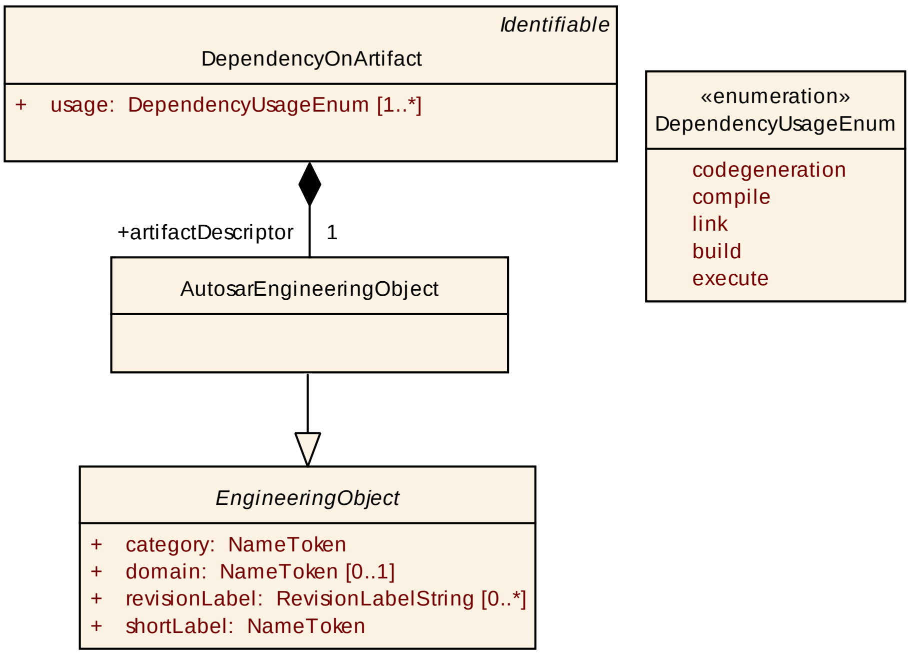

Figure 8.4: Dependencies of an `Implementation`

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	***Datatype***	***Mul.***	***Kind***	***Note***
artifactDescriptor	AutosarEngineeringObject	1	aggr	The specified artifact needs to exist.
usage	DependencyUsageEnum	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.
codegeneration	The object referred by the dependency is required during code generation
compile	The object referred by the dependency is required during compilation.
execute	The object referred by the dependency is required at execution time.
link	The object referred by the dependency is required during linking.

Table 8.4: DependencyUsageEnum

Class	**AutosarEngineeringObject**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Engineering Object			
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	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 8.5: AutosarEngineeringObject

Class	**EngineeringObject (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Engineering Object			
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			
Attribute	***Datatype***	***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

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
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_4043] **`Compiler`** ⌈For the specification of the used (or to be used) compiler the `Compiler` element shall be used: ⌋

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	*Datatype*	*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_4044] **`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	*Datatype*	*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

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* [8] 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_4045] **`Implementation.resourceConsumption`** ⌈The `ResourceConsumption` is attached to an `Implementation`. For each `Implementation`, there is one `ResourceConsumption` description. ⌋

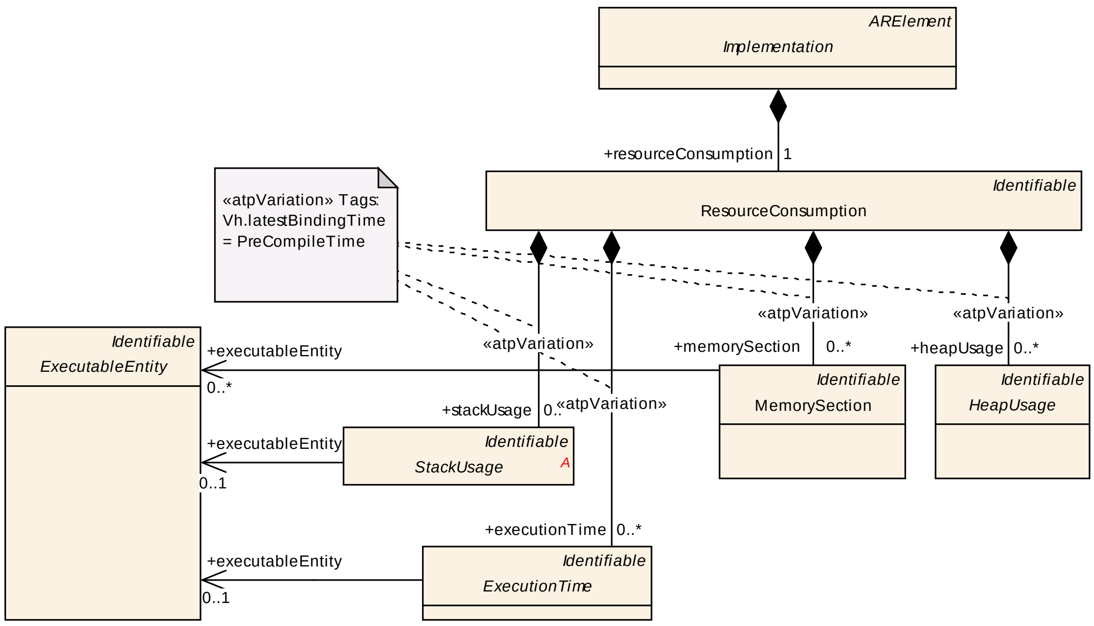

Figure 9.1: Resource consumption overview

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 `BswBehavior` 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	***Datatype***	***Mul.***	***Kind***	***Note***
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:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
heapUsage	HeapUsage	*	aggr	Collection of the heap memory allocated by this implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
memorySection	MemorySection	*	aggr	An abstract memory section required by this Implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
sectionNamePrefix	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:** atpVariation **Tags:** 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

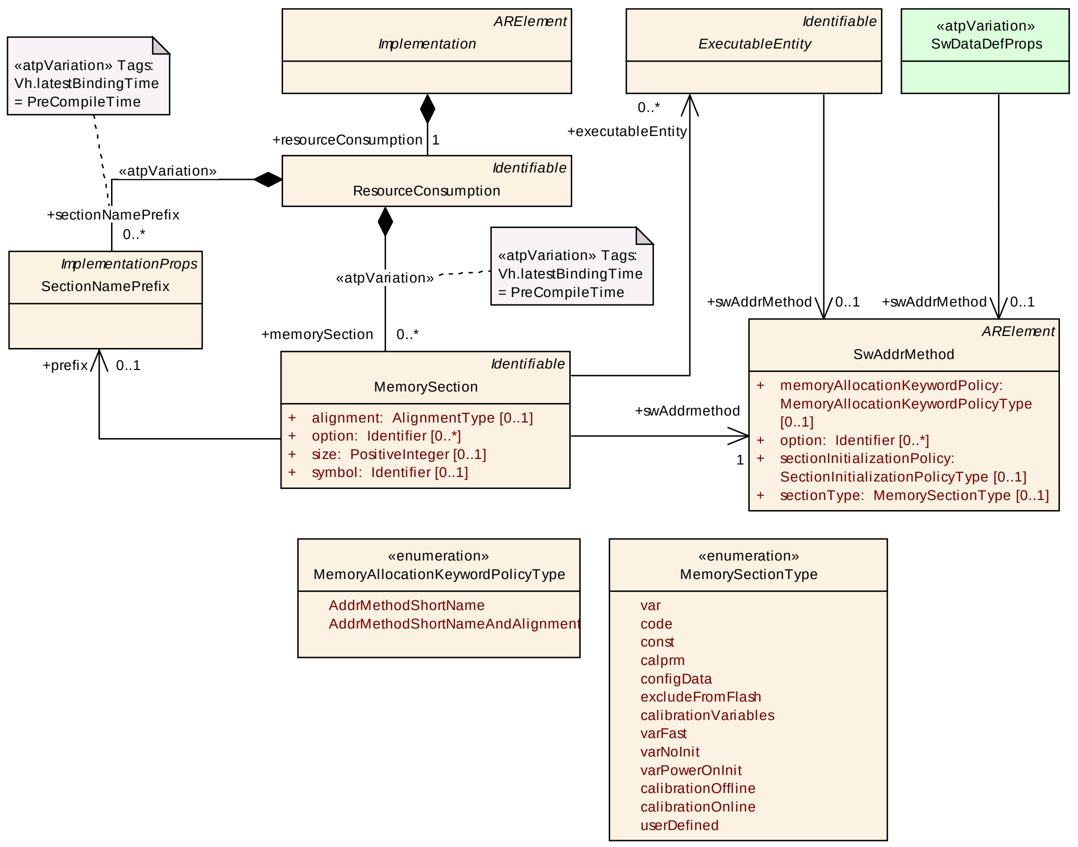

Figure 9.2: Meta-model related to the `MemorySection`

[TPS_BSWMDT_4046] 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. ⌋

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_4047] Memory section prefix ⌈It is possible to supersede these prefixes by more fine granular values using the meta-class `SectionNamePrefix`. The details are explained in the diagrams, tables and constraints below. ⌋

The mapping of the allocation keywords to the compiler specific code is done via header files (MemMap.h for modules or <SWC>_MemMap.h for components). It is possible (since AUTOSAR R4.0 rev. 0002) 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.

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

[TPS_BSWMDT_4048] 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. ⌋

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].

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 must 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 must 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> where • is the shortName of the referenced SwAddrMethod • 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. • 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 SwcComponentType. It can be superseded by the prefix attribute.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***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 ExecutableEntitities in different sections even if the associated SwAddrmethod is the same. This is applicable to code sections only.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
option	Identifier	*	ref	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 BswModuleDescription'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 assocation indicates that this module specific (abstract) memory section is part of an overal SwAddrMethod, referred by the upsream declarations (e.g. calibration parameters, data element prototypes, code entities) which share a common addressing strategy. This can be evaulated for the ECU configuration of the build support. This association must 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.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
symbol	Identifier	0..1	ref	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 sectionNamePrefixes.

Table 9.2: MemorySection

Primitive	**AlignmentType**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types
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 or BOOLEAN. Typical values for numbers are 8, 16, 32. **Tags:** xml.xsd.customType=ALIGNMENT-TYPE; xml.xsd.pattern=[1-9][0-9]*|0x[0-9a-f]*|0[0-7]*|0b[0-1]*|UNSPECIFIED|UNKNOWN|B OOLEAN; xml.xsd.type=string

Table 9.3: AlignmentType

Class	**SwAddrMethod**			
Package	M2::AUTOSARTemplates::CommonStructure::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,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
memoryAllocationKeywordPolicy	MemoryAllocationKeywordPolicyType	0..1	attr	Enumeration to specify the name pattern of the Memory Allocation Keyword.
option	Identifier	*	ref	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 MemMapAddressingModeSet.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
sectionInitializationPolicy	SectionInitializationPolicyType	0..1	attr	Specifies the expected initialization of the variables (inclusive those which are implementing VariableDataPrototypes). 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 AutosarDataPrototypes 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::AUTOSARTemplates::CommonStructure::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Literal	***Description***
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.
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 the alignment attribute of the MemorySection. This requests a separation of objects in memory dependent from the alignment and is not applicable for SwAddrMethods referred by RunnableEntitys and BswSchedulableEntitys.

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 must 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::AUTOSARTemplates::CommonStructure::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***
calibration Offline	Program data which can only be used for offline calibration. **Note**: This value is deprecated and shall be substituted by calPrm. **Tags:** atp.Status=obsolete
calibration Online	Program data which can be used for online calibration. **Note**: This value is deprecated and shall be substituted by calPrm. **Tags:** atp.Status=obsolete
calibration Variables	Values which are available in the ECU but do not exist in the Hex-file. No upload is required to obtain access to the ECU data. The ECU will never be touched by the instrumentation tool with the exception of upload. These are calculated values which are not represented in the CPU memory (no address is associated).
calprm	To be used for calibratable constants of ECU-functions.
code	To be used for mapping code to application block, boot block, external flash etc.
configData	Constants with attributes that show that they reside in one segment for module configuration.
const	To be used for global or static constants.

excludeFrom Flash	Values existing in the ECU but not dropped down in the binary file. No upload should be needed to obtain access to the ECU data. The ECU will never be touched by the instrumentation tool, with the exception of upload. These are memory areas which are not overwritten by downloading the executable.
userDefined	No specific categorization of sectionType possible. **Note**: This value is deprecated and shall be substituted by var, code, const, calprm, configData, excludeFromFlash and the appropriate values of the orthogonal attributes sectionInitializationPolicy, memoryAllocationKeywordPolicy and option. **Tags:** atp.Status=obsolete
var	To be used for global or static variables. The expected initialization is specified with the attribute sectionInitializationPolicy.
varFast	To be used for all global or static variables that have at least one of the following properties: - accessed bit-wise - frequently used - high number of accesses in source code Some platforms allow the use of bit instructions for variables located in this specific RAM area as well as shorter addressing instructions. This saves code and runtime. **Note**: This value is deprecated and shall be substituted by var and the appropriate values of the orthogonal attributes sectionInitializationPolicy, memoryAllocationKeywordPolicy and option. **Tags:** atp.Status=obsolete
varNoInit	To be used for all global or static variables that are never initialized. **Note**: This value is deprecated and shall be substituted by var and the appropriate values of the orthogonal attributes sectionInitializationPolicy, memoryAllocationKeywordPolicy and option. **Tags:** atp.Status=obsolete
varPowerOn Init	To be used for all global or static variables that are initialized only after power on reset. **Note**: This value is deprecated and shall be substituted by var and the appropriate values of the orthogonal attributes sectionInitializationPolicy, memoryAllocationKeywordPolicy and option. **Tags:** atp.Status=obsolete

Table 9.7: MemorySectionType

Class	**SectionNamePrefix**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::Memory SectionUsage			
Note	A prefix to be used for generated code artifacts defining a memory section name in the source code of the using module.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 9.8: SectionNamePrefix

Class	**ImplementationProps (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Implementation			
Note	Defines a symbol to be used as prefix when generating code artifacts.			
Base	ARObject,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
symbol	CIdentifier	1	ref	The symbol to be used as prefix.

Table 9.9: ImplementationProps

[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 the `userDefined` `sectionType` is used 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 `option`. This allows a more formal support for selecting the memory segment during integration. (see [19]).

[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_4049] 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. ⌋

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.

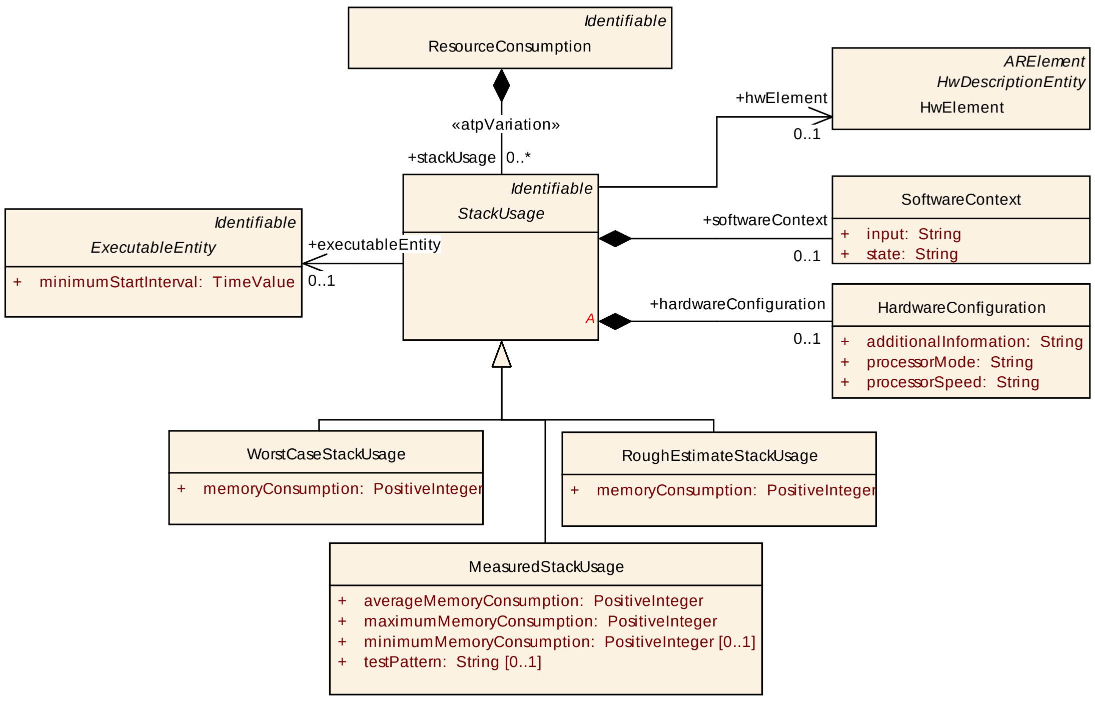

Figure 9.3: Stack Memory Consumption

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			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
executable Entity	ExecutableEntit y	0..1	ref	The executable entity for which this stack usage is described.
hardwareC onfiguratio n	HardwareConfig uration	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.
softwareC ontext	SoftwareContex t	0..1	aggr	Contains details about the software context this stack usage is provided for.

Table 9.10: StackUsage

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

Table 9.11: WorstCaseStackUsage

Class	**MeasuredStackUsage**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::StackUsage			
Note	The stack usage has been measured.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,StackUsage			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
averageMe moryCons umption	PositiveInteger	1	attr	The average stack usage measured.
maximum MemoryCo nsumption	PositiveInteger	1	attr	The maximum stack usage measured.
minimumM emoryCon sumption	PositiveInteger	0..1	attr	The minimum stack usage measured.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
testPattern	String	0..1	attr	Description of the test pattern used to acquire the measured values.

Table 9.12: 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	***Datatype***	***Mul.***	***Kind***	***Note***
memoryConsumption	PositiveInteger	1	attr	Rough estimate of the stack usage.

Table 9.13: 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.

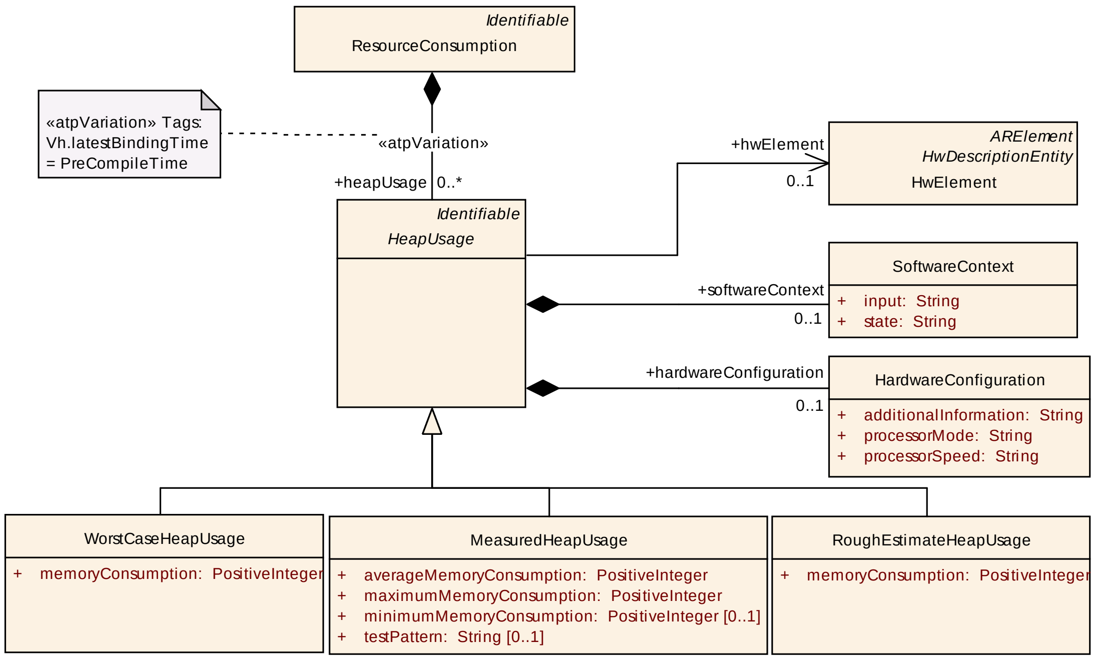
Identifiable
ResourceConsumption
«atpVariation» Tags: Vh.latestBindingTime = PreCompileTime
«atpVariation»
+heapUsage 0..*
Identifiable
HeapUsage
+hwElement 0..1
ARElement
HwDescriptionEntity
HwElement
SoftwareContext
+ input: String
+ state: String
+softwareContext 0..1
+hardwareConfiguration 0..1
HardwareConfiguration
+ additionalInformation: String
+ processorMode: String
+ processorSpeed: String
WorstCaseHeapUsage
+ memoryConsumption: PositiveInteger
MeasuredHeapUsage
+ averageMemoryConsumption: PositiveInteger
+ maximumMemoryConsumption: PositiveInteger
+ minimumMemoryConsumption: PositiveInteger [0..1]
+ testPattern: String [0..1]
RoughEstimateHeapUsage
+ memoryConsumption: PositiveInteger

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			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
hardwareConfiguration	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.
softwareContext	SoftwareContext	0..1	aggr	Contains details about the software context this heap usage is provided for.

Table 9.14: HeapUsage

Class	**WorstCaseHeapUsage**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption::HeapUsage			
Note	Provides a formal worst case heap usage.			
Base	ARObject,HeapUsage,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
memoryConsumption	PositiveInteger	1	attr	Worst case heap consumption.

Table 9.15: WorstCaseHeapUsage

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

Table 9.16: 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	***Datatype***	***Mul.***	***Kind***	***Note***
memoryConsumption	PositiveInteger	1	attr	Rough estimate of the heap usage.

Table 9.17: RoughEstimateHeapUsage

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 [6] 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_4050] `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`. ⌋

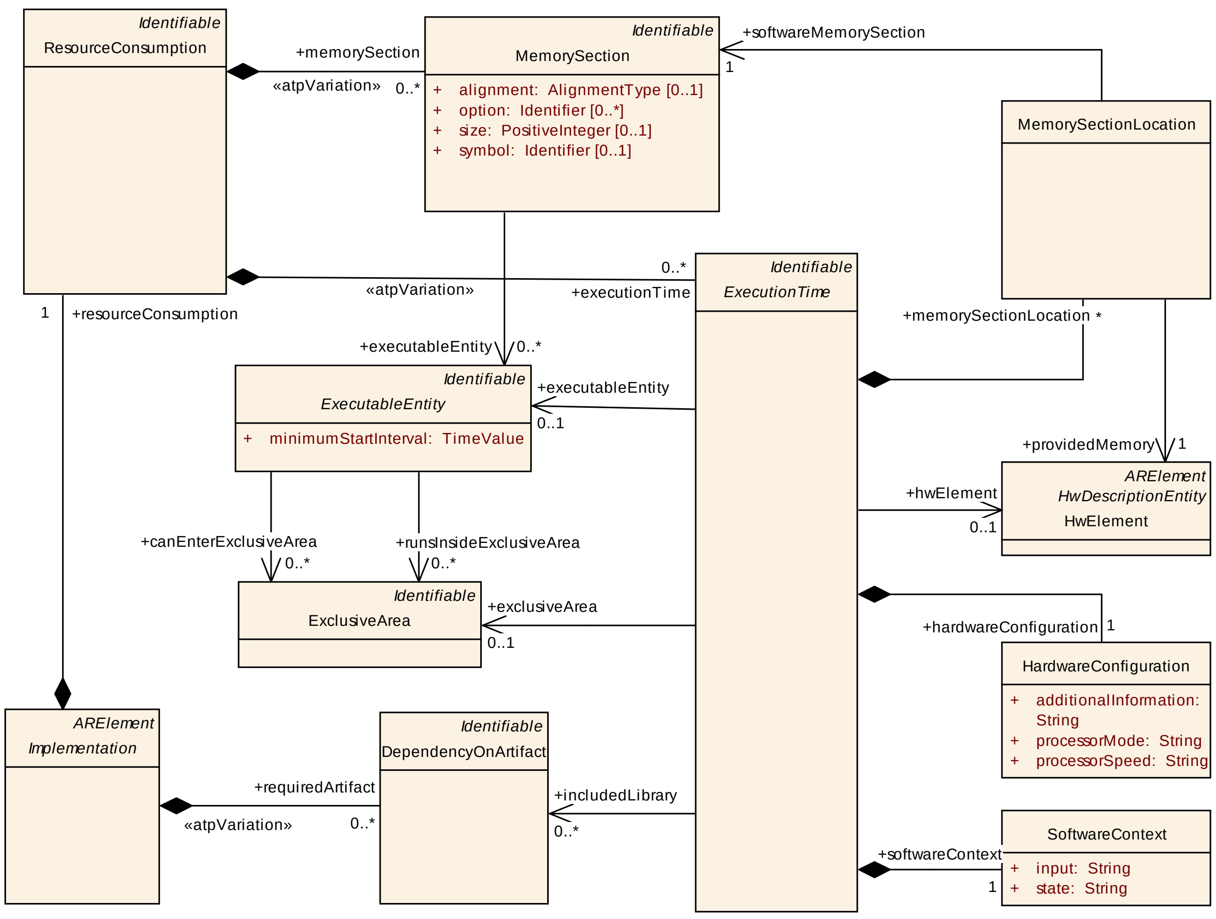

Figure 9.5: Detailed relations of an `ExecutionTime` description

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 ([10]).

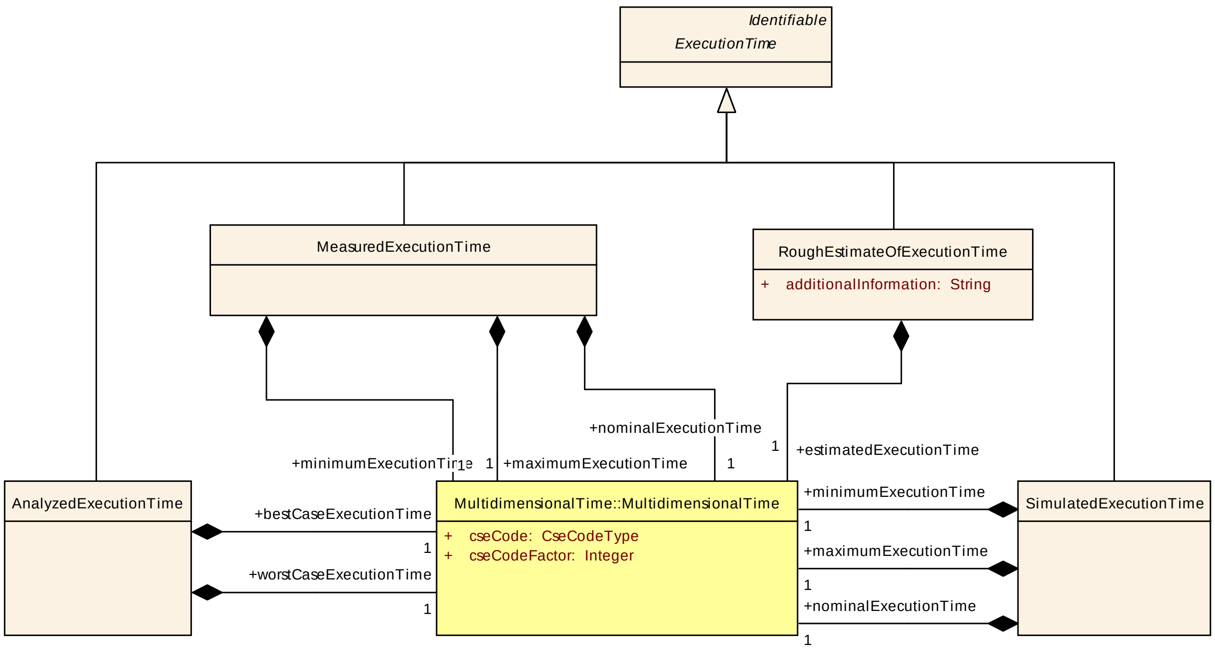

Figure 9.6: Sub-classes of `ExecutionTime` and their usage of `TimeValue`

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			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
exclusiveArea	ExclusiveArea	0..1	ref	Reference to the ExclusiveArea this execution time is provided for.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
executable Entity	ExecutableEntit y	0..1	ref	The executable entity for which this execution time is described.
hardwareC onfiguratio n	HardwareConfig uration	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.
includedLi brary	DependencyOn Artifact	*	ref	If this dependency is specified, the execution time of the library code is included in the execution time data for the runnable.
memorySe ctionLocati on	MemorySection Location	*	aggr	Provides information on the MemorySectionLocation which is involved in the ExecutionTime description.
softwareC ontext	SoftwareContex t	1	aggr	Provides information on the detailed SoftwareContext used to provide the ExecutionTime description.

Table 9.18: ExecutionTime

9.5.5.2 ExecutionTime References an "ECU"

[TPS_BSWMDT_4051] `ExecutionTime` references an ECU ⌈The `ExecutionTime` references an ECU (the concept ECU is defined by the ECU-Resource-Template [20]) via the attribute `hwElement`. This reference uniquely describes the hardware for which the `ExecutionTime` is provided. ⌋ 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_4052] `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 HW-Configuration 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. ⌋

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

Table 9.19: HardwareConfiguration

9.5.5.4 ExecutionTime Includes a MemorySectionLocation

[TPS_BSWMDT_4053] **`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 `memorySection` on the software are described in the `softwareMemorySection` 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. ⌋

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

Table 9.20: MemorySectionLocation

9.5.5.5 ExecutionTime Includes a SoftwareContext

[TPS_BSWMDT_4054] **`ExecutionTime.softwareContext`** ⌈The `SoftwareContext` 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. ⌋

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 `ExecutionTime`.

Class	**SoftwareContext**			
Package	M2::AUTOSARTemplates::CommonStructure::ResourceConsumption			
Note	Specifies the context of the software for this resource consumption.			
Base	ARObject			
Attribute	***Datatype***	***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 ExecutionTime is provided.

Table 9.21: SoftwareContext

9.5.5.6 Dependency on External Libraries

[TPS_BSWMDT_4055] **`ExecutionTime.includedLibrary`** ⌈The `ExecutionTime` 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`.
⌋

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 `worstCaseExecutionTime` 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	***Datatype***	***Mul.***	***Kind***	***Note***
bestCaseExecutionTime	MultidimensionalTime	1	aggr	The best case execution time (BCET) defines the minimum amount of time the related executable entity requires for its execution.
worstCaseExecutionTime	MultidimensionalTime	1	aggr	The worst case execution time (WCET) defines the maximum amount of time the related executable entity requires for its execution.

Table 9.22: AnalyzedExecutionTime

[constr_4031] Analyzed execution time ⌈ The attribute values of `AnalyzedExecutionTime` must fulfill:
`bestCaseExecutionTime` <= `worstCaseExecutionTime` ⌋

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	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
maximumExecutionTime	MultidimensionalTime	1	aggr	The maximum measured execution time.
minimumExecutionTime	MultidimensionalTime	1	aggr	The minimum measured execution time.
nominalExecutionTime	MultidimensionalTime	1	aggr	The nominal measured execution time.

Table 9.23: 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	***Datatype***	***Mul.***	***Kind***	***Note***
maximumExecutionTime	MultidimensionalTime	1	aggr	The maximum simulated execution time.
minimumExecutionTime	MultidimensionalTime	1	aggr	The minimum simulated execution time.
nominalExecutionTime	MultidimensionalTime	1	aggr	The nominal simulated execution time.

Table 9.24: SimulatedExecutionTime

[constr_4033] Simulated execution time ⌈ The attribute values of `SimuletedExecutionTime` must fulfill:

`minimumExecutionTime` <= `nominalExecutionTime` <= `maximumExecution-Time`⌋

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::Execution Time			
Note	Provides a description of a rough estimate on the ExecutionTime.			
Base	ARObject,ExecutionTime,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
additionalI nformation	String	1	attr	Provides description on the rough estimate of the ExecutionTime.
estimatedE xecutionTi me	Multidimensiona lTime	1	aggr	The estimated execution time.

Table 9.25: 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 6.6 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 [21]).

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 [5] for the method and [8] 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 [7]). 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.

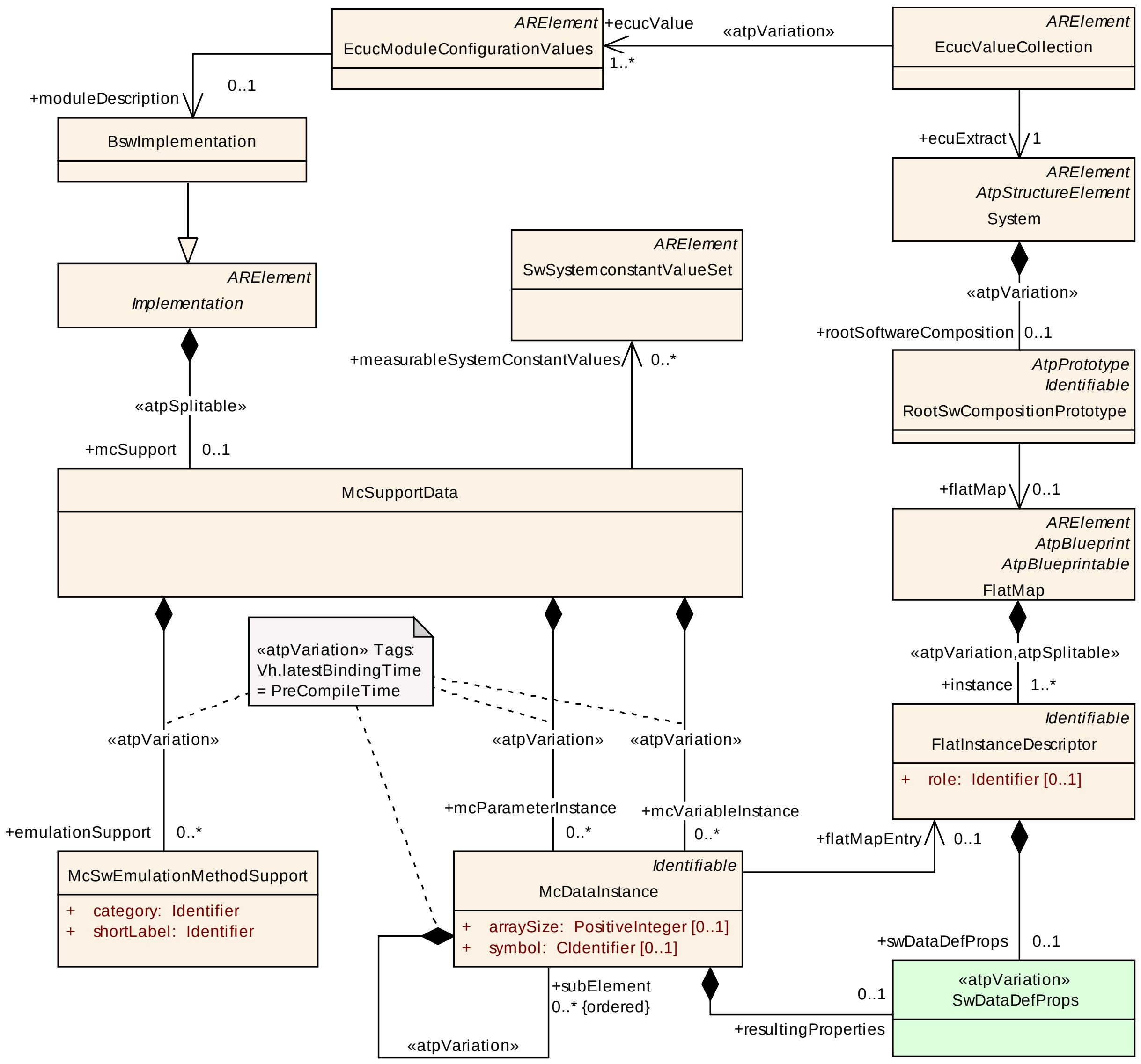

Figure 10.1: Calibration Support Data attached to Implementation

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_4056] Multiplicity of `McSupportData` ⌈Thus in an ECU there will be at most one (generated) instance of `McSupportData` for each `Implementation` instance: ⌋

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	***Datatype***	***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:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
mcVariableInstance	McDataInstance	*	aggr	A data instance to be used for measurement. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
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.

Table 10.1: McSupportData

[TPS_BSWMDT_4057] 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 three 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 `McSupport` artifact will depend on the system constants needed to resolve these variants.

⌋

[TPS_BSWMDT_4058] **`McSupportData.measuredSystemConstantValues`** ⌈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.measuredSystemConstantValues`. Note that the values of system constants are also possibly subject to compile-time variation (not visible in the figure). ⌋

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 [5]) :

- 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 [8]) 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 `ARPackage`s (this kind of name conflict cannot be resolved by a `FlatMap`).
- Administrative data for the A2L-File which are nor delivered by AUTOSAR.

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.

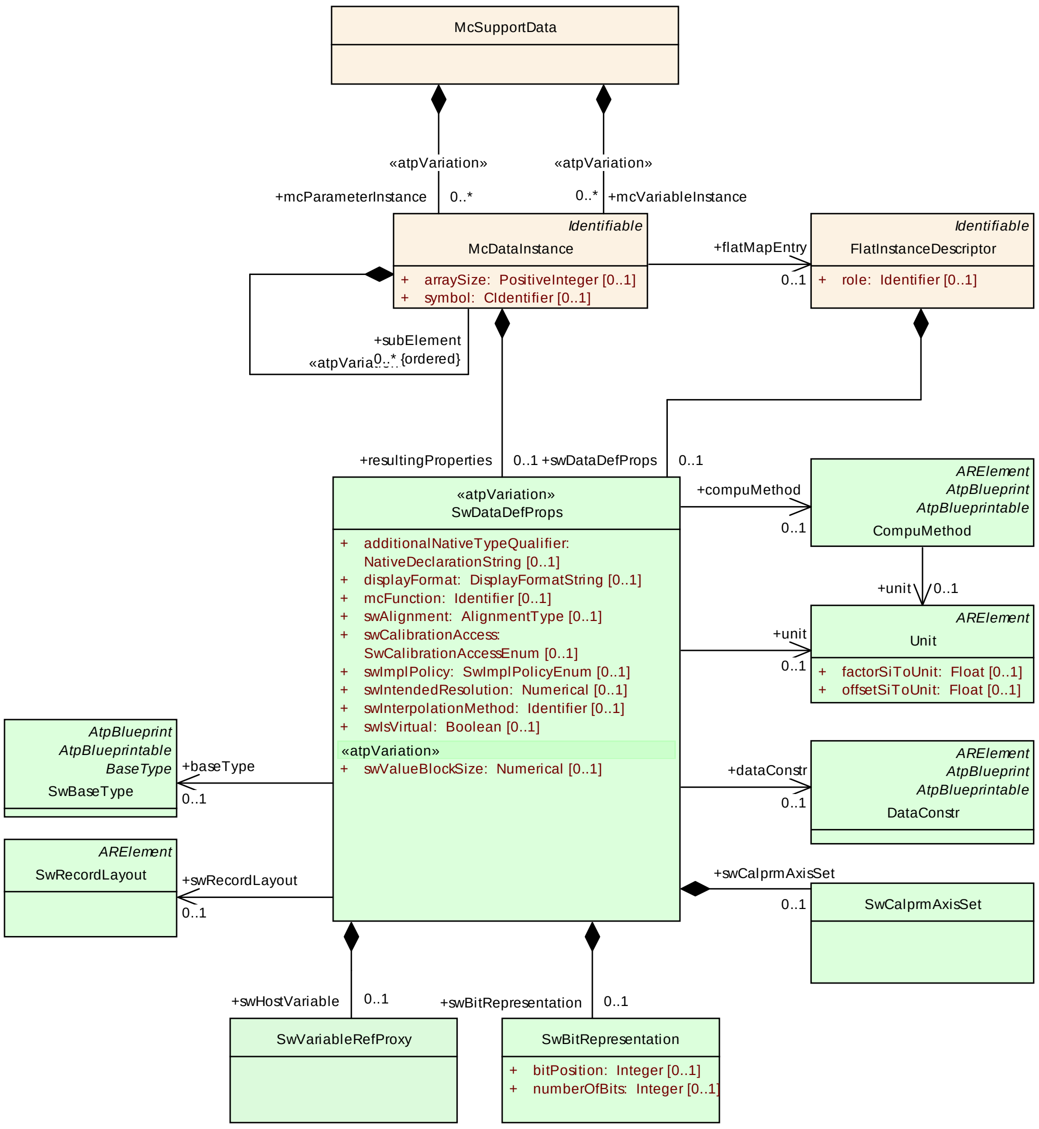
McSupportData
«atpVariation»
+mcParameterInstance
0..*
«atpVariation»
0..*
+mcVariableInstance
Identifiable
McDataInstance
+ arraySize: PositiveInteger [0..1]
+ symbol: CIdentifier [0..1]
+subElement
«atpVariation» 0..* {ordered}
+flatMapEntry
0..1
Identifiable
FlatInstanceDescriptor
+ role: Identifier [0..1]
+resultingProperties
0..1
+swDataDefProps
0..1
«atpVariation»
SwDataDefProps
+ additionalNativeTypeQualifier: NativeDeclarationString [0..1]
+ displayFormat: DisplayFormatString [0..1]
+ mcFunction: Identifier [0..1]
+ swAlignment: AlignmentType [0..1]
+ swCalibrationAccess: SwCalibrationAccessEnum [0..1]
+ swImplPolicy: SwImplPolicyEnum [0..1]
+ swIntendedResolution: Numerical [0..1]
+ swInterpolationMethod: Identifier [0..1]
+ swIsVirtual: Boolean [0..1]
«atpVariation»
+ swValueBlockSize: Numerical [0..1]
+compuMethod
0..1
ARElement
AtpBlueprint
AtpBlueprintable
CompuMethod
+unit
0..1
+unit
0..1
ARElement
Unit
+ factorSiToUnit: Float [0..1]
+ offsetSiToUnit: Float [0..1]
+dataConstr
0..1
ARElement
AtpBlueprint
AtpBlueprintable
DataConstr
+swCalprmAxisSet
0..1
SwCalprmAxisSet
AtpBlueprint
AtpBlueprintable
BaseType
SwBaseType
+baseType
0..1
ARElement
SwRecordLayout
+swRecordLayout
0..1
+swHostVariable
0..1
+swBitRepresentation
0..1
SwVariableRefProxy
SwBitRepresentation
+ bitPosition: Integer [0..1]
+ numberOfBits: Integer [0..1]

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			
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 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) 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	***Datatype***	***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.
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.
instanceInMemory	ImplementationElementInParameterInstanceRef	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.
resultingProperties	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.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
subElement	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	CIdentifier	0..1	ref	This symbolic name is used to determine the memory address during final generation of the MC configuration data (e.g. "A2L" file). It must be the name of the variable used by the linker. This name 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.

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_4059] Granularity of `McDataInstance.subElements` ⌈Note that is 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. ⌋

[TPS_BSWMDT_4060] `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 [14]) 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. ⌋

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 [11] 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 device 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 device 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 `InternalBehavior` 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 6.8.)

The generated data descriptions of the RTE are an M1 model of `DataPrototype`s based on `ImplementationDataType`s 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 [11] 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.

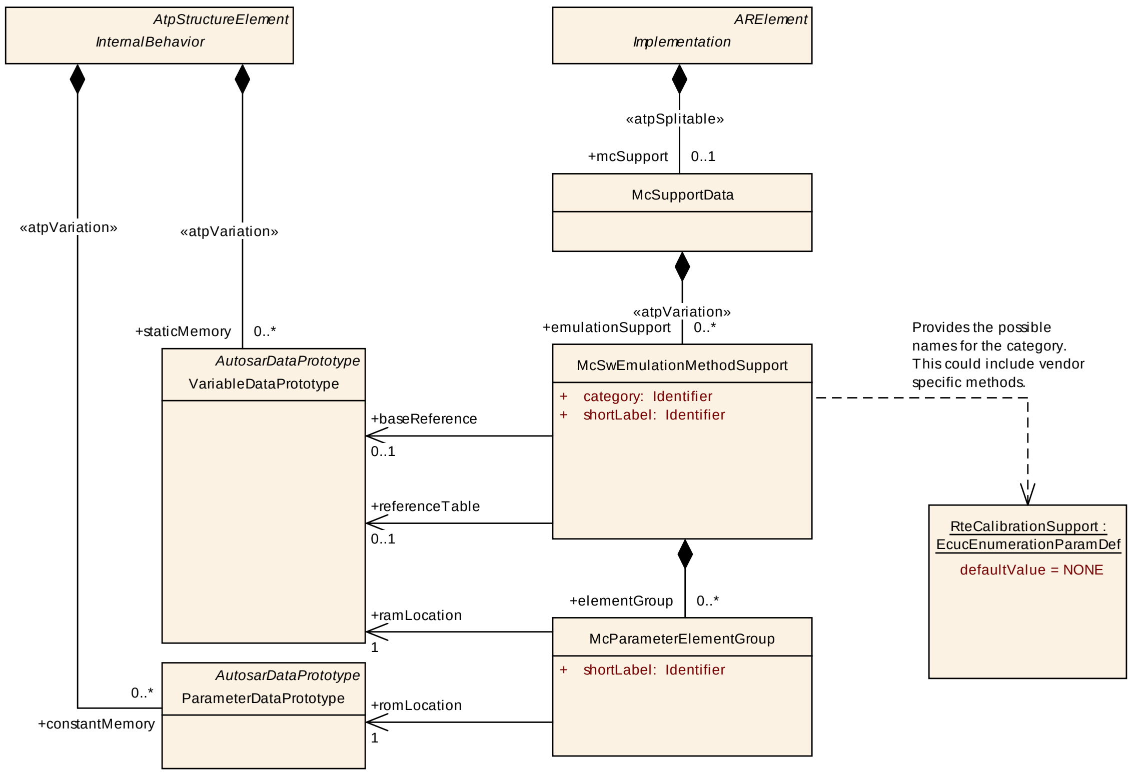

Figure 10.3: Describing the Software Emulation Method for Calibration Data

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 device 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	***Datatype***	***Mul.***	***Kind***	***Note***
category	Identifier	1	ref	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.
shortLabel	Identifier	1	ref	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	***Datatype***	***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	ref	Assigns a name to this element. **Tags:** xml.sequenceOffset=-100

Table 10.4: McParameterElementGroup

[TPS_BSWMDT_4061] **`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 [11]), or it can be chosen differently in order to denote a vendor specific method. ⌋

[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 `elementGroup`s must exist.

⌋

[TPS_BSWMDT_4062] 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 `ImplementationDataType`s 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. ⌋

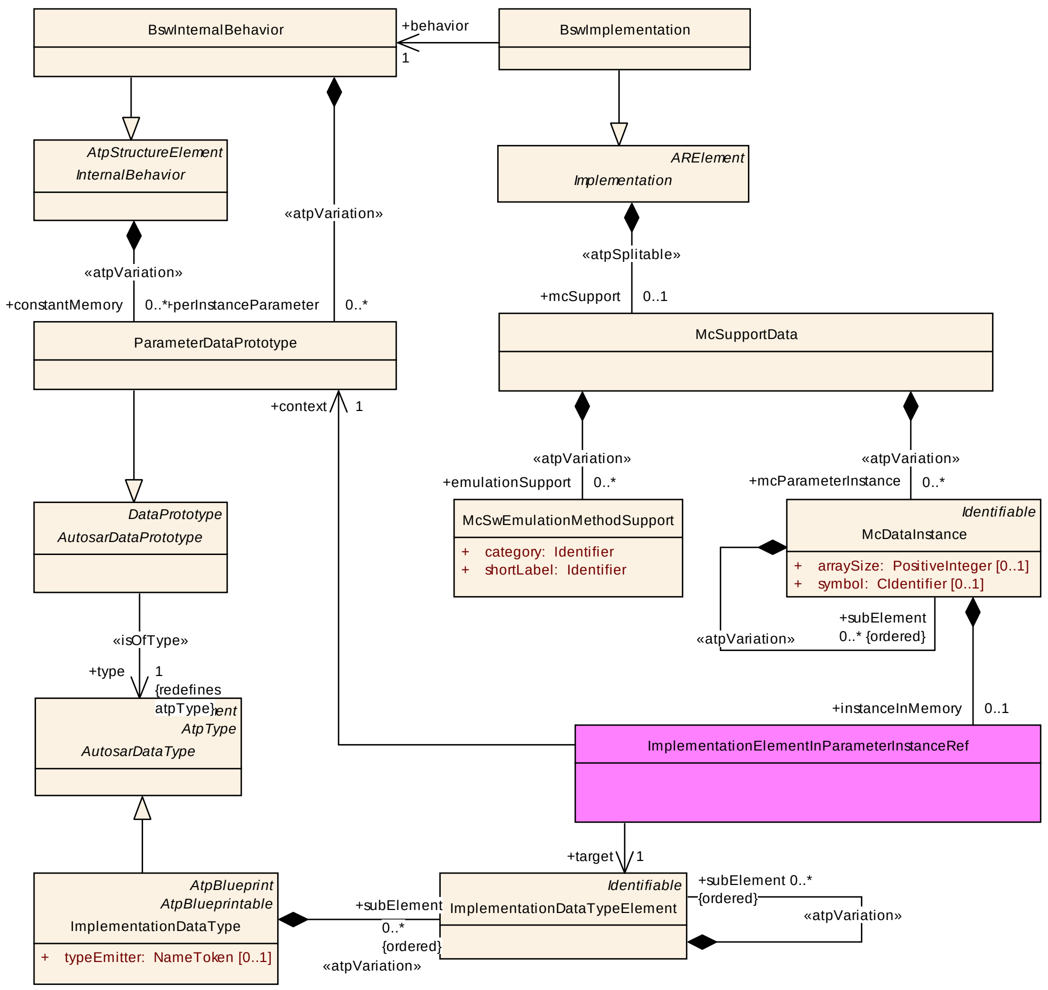

Figure 10.4: Reference to the Implemented Data needed for Emulation

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 ParameterDataPrototype 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 ImplementationDataTypeElement isn't derived from AtpPrototype.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
context	ParameterData Prototype	1	ref	The context for the referred element. **Tags:** xml.sequenceOffset=20

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
target	Implementation DataTypeEleme nt	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 subElementof 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.

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_4063] BSW Interface Variation Points ⌈The variation points in the scope of `BswModuleDescription` allow to declare variable sets of optional documentation, C-interfaces, dependencies, triggers and mode groups as part of one BSW module description, see figure 11.1. Further variation points within this class hierarchy are not allowed in order to keep the meta-model and the resulting M1 models maintainable. ⌋ 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, this is possible by variation of the associations to `BswModuleEntry`, but is is not possible to declare a single `BswModuleEntry` with two variants just for a single argument.

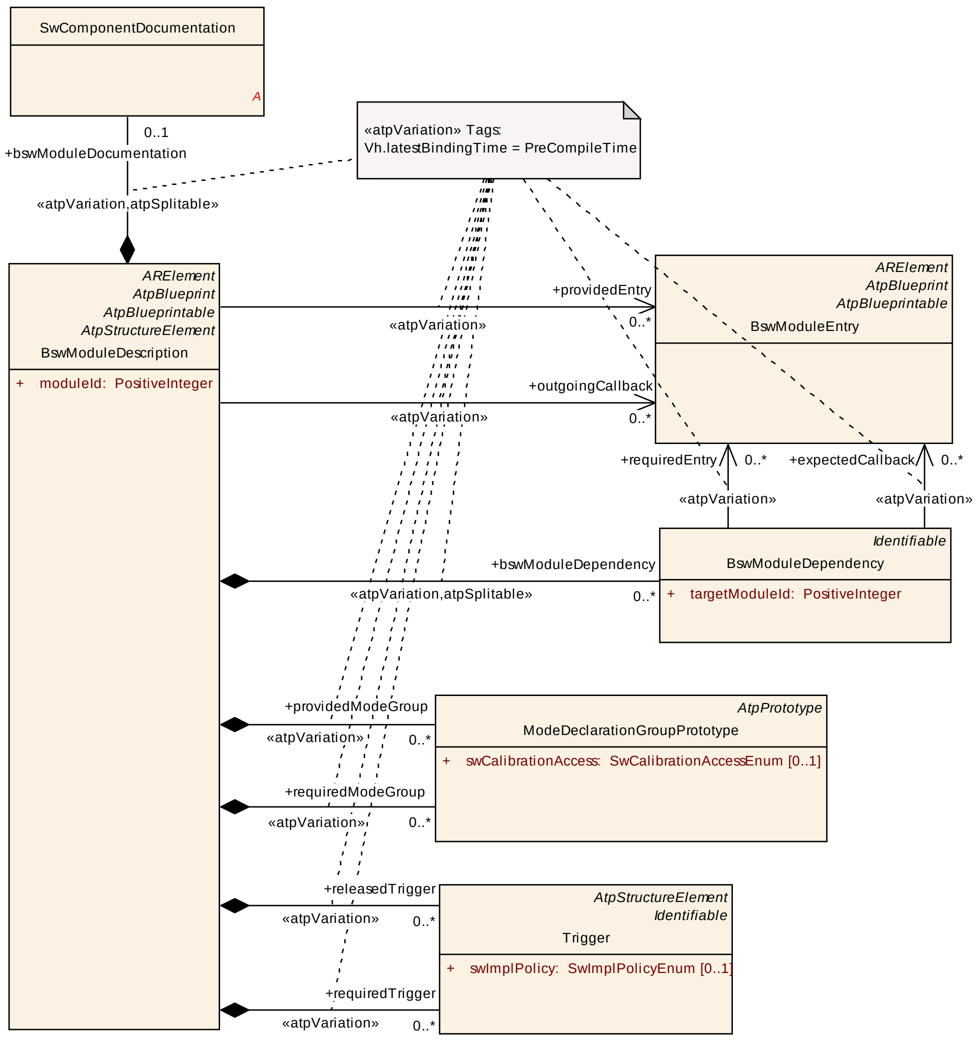

Figure 11.1: Variation points under `BswModuleDescription`

One use case is to maintain a specification which includes optional or alternative interfaces/dependencies for a module at design time. For example, 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 all 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 all the 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 `outgoingCallback-`

/expectedCallback). 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_4064] BSW Behavior Variation Points ⌈In a similar way, variation points underneath BswInternalBehavior allow to declare variants in the aggregation of BswModuleEntity-s, BswEvents and further elements. Likewise, the roles calledEntry, accessedModeGroup, managedModeGroup and issuedTrigger are variation points, see figure 11.2.

This figure also shows the variation point in the aggregation of local data for calibration and measurement and of BswExclusiveArea by the base class InternalBehavior.⌋

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.

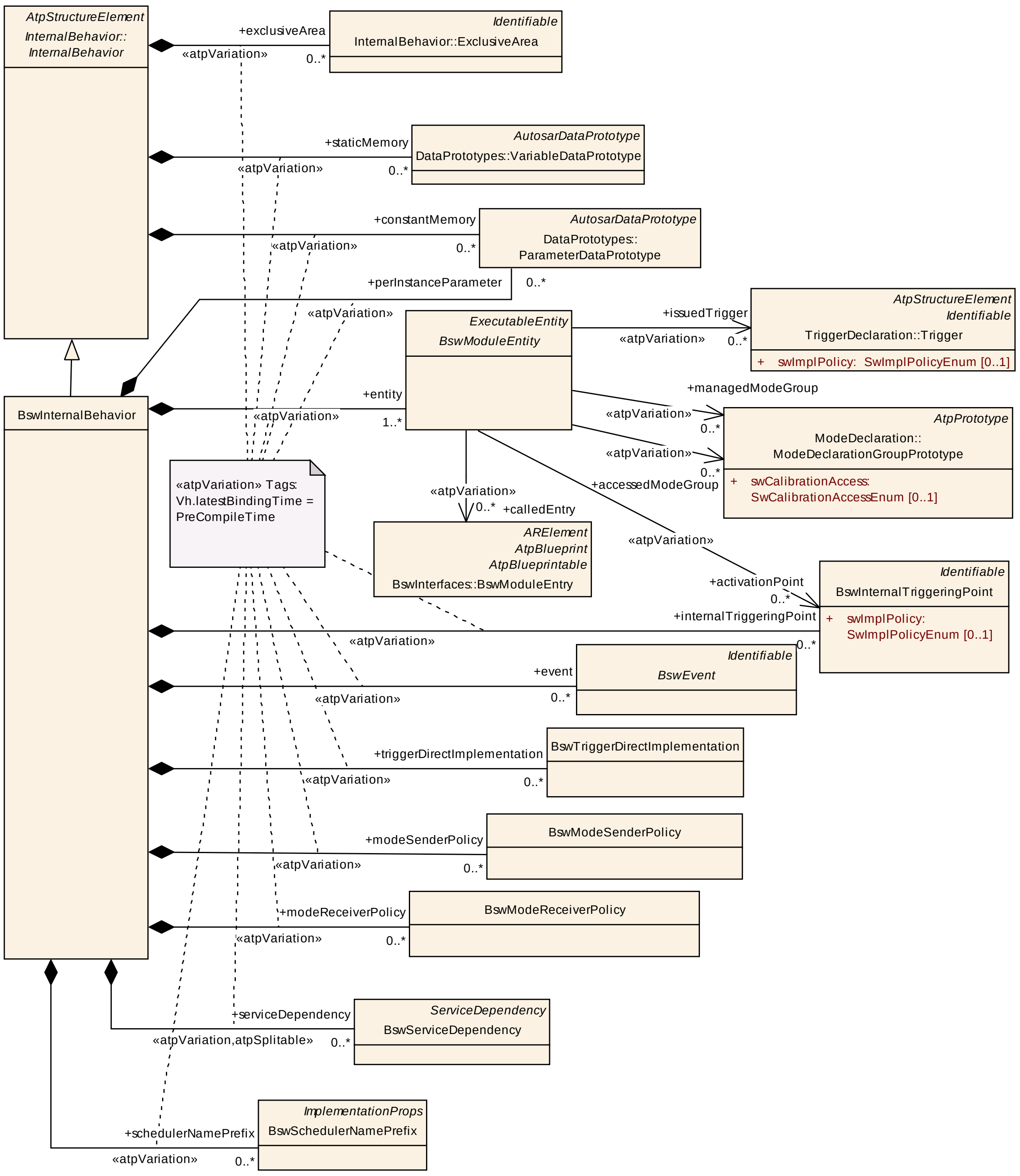

Figure 11.2: Variation points under `BswBehavior`

11.3 BSW Implementation Variation Points

[TPS_BSWMDT_4065] BSW Implementation Variation Points ⌈Figure 11.3 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. ⌋

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.

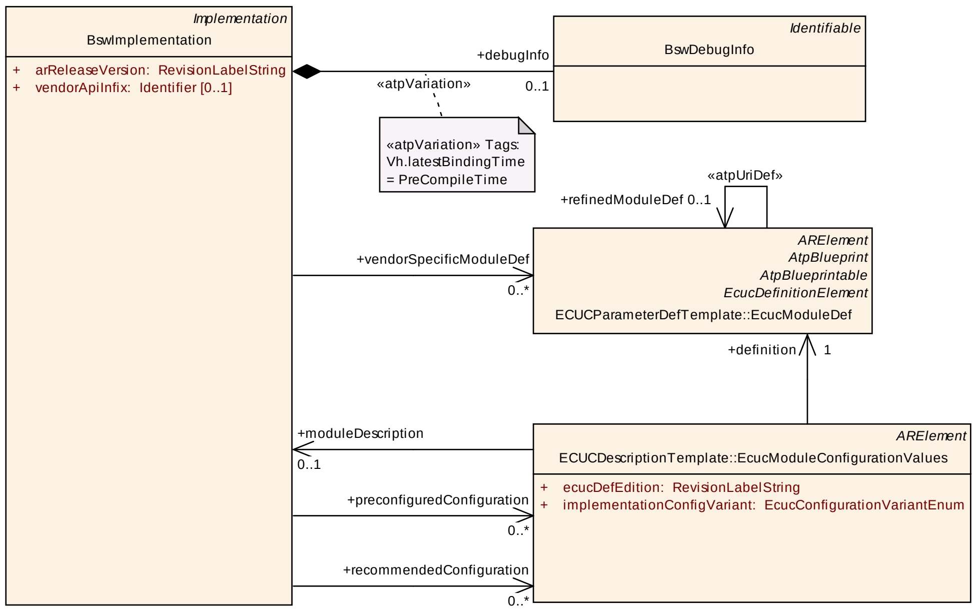
Implementation
BswImplementation
+ arReleaseVersion: RevisionLabelString
+ vendorApiInfix: Identifier [0..1]
+debugInfo
«atpVariation»
0..1
Identifiable
BswDebugInfo
«atpVariation» Tags: Vh.latestBindingTime = PreCompileTime
«atpUriDef»
+refinedModuleDef 0..1
+vendorSpecificModuleDef
0..*
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionElement
ECUCParameterDefTemplate::EcucModuleDef
+definition
1
+moduleDescription
0..1
ARElement
ECUCDescriptionTemplate::EcucModuleConfigurationValues
+ ecucDefEdition: RevisionLabelString
+ implementationConfigVariant: EcucConfigurationVariantEnum
+preconfiguredConfiguration
0..*
+recommendedConfiguration
0..*

Figure 11.3: 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 [22].

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 [6] 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 [3] 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_4066] 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.providedEntry`
  `BswModuleDescription.outgoingCallback`

  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`
  `BswModuleDescription.bswModuleDependency.requiredEntry`
  `BswModuleDescription.bswModuleDependency.expectedCallback`

  These elements are required as far as they describe the dependency on standardized elements of other standardized ICC3 modules (identified by the `targetModuleId`). There is one exception: The following element is deprecated in R4.0.3 since it has been moved to the Internal Behavior Level. It is not considered to be relevant for the conformance test (see also chapter 12.3):
  `BswModuleDescription.bswModuleDependency.serviceItem`

  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. `providedEntry` - 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 `argument`s and the `returnType`. Please refer to chapter “Implementation Data Type” of [7] for example how to describe C data types in this way.

⌋

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_4067] 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 ⌋ 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 6.8).

12.4 Implementation Level

[TPS_BSWMDT_4068] 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.

⌋

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_4069] 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 `preconfiguredConfiguration`s (and likewise several `BswImplementation`s and object files) but only one `vendorSpecificModuleDef` (the one belonging to the release of this module).

⌋

The following is obviously not relevant for the conformance test, because the tester cannot change the configuration:

- `BswImplementation.recommendedConfiguration`

[TPS_BSWMDT_4070] 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⌋.

13 Constraint and Specification History

13.1 Constraint History of this Document according to AUTOSAR R4.0.1

13.1.1 Changed Constraints in R4.0.1

N/A

13.1.2 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]	BswScheduledEntity 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 13.1: Added Constraints in R4.0.1

[1] this constraint was by mistake named **Bsw service identifier** in R4.0.1 and R4.0.2

13.1.3 Deleted Constraints

N/A

13.2 Constraint History of this Document according to AUTOSAR R4.0.2

13.2.1 Changed Constraints in R4.0.2

N/A

13.2.2 Added Constraints in R4.0.2

Number	Heading
[constr_4047]	Multiplicity of vendor specific configuration parameters
[constr_4048]	Multiplicity of preconfigured values

Table 13.2: Added Constraints in R4.0.2

13.2.3 Deleted Constraints in R4.0.2

N/A

13.3 Constraint and Specification History of this Document according to AUTOSAR R4.0.3

13.3.1 Changed Constraints in R4.0.3

N/A

13.3.2 Added Specification Items in R4.0.3

Number	Heading
[TPS_BSWMDT_4000]	BSW modules with AUTOSAR Interfaces
[TPS_BSWMDT_4001]	Attaching `SwComponentDocumentation` to a BSWMD
[TPS_BSWMDT_4002]	Usage of `BswModuleEntry`
[TPS_BSWMDT_4003]	`BswModuleDependency`
[TPS_BSWMDT_4004]	`BswModuleDescription.providedModeGroup`
[TPS_BSWMDT_4005]	`BswModuleDescription.releasedTrigger`
[TPS_BSWMDT_4006]	`BswModuleDescription.internalBehavior`
[TPS_BSWMDT_4007]	`BswModuleEntry`

[TPS_BSWMDT_4008]	C-symbol of BswModuleEntry
[TPS_BSWMDT_4009]	Usage of SwServiceArg
[TPS_BSWMDT_4010]	SwServiceArg.swDataDefProps.implementationDataType
[TPS_BSWMDT_4011]	SwServiceArg.swDataDefProps.swPointerTargetProps
[TPS_BSWMDT_4012]	SwServiceArg.direction
[TPS_BSWMDT_4013]	Usage of BswModuleDescription.providedModeGroup
[TPS_BSWMDT_4014]	ModeRequestTypeMap in BSW
[TPS_BSWMDT_4015]	Usage of Trigger in BSW
[TPS_BSWMDT_4016]	Location of standardized BswModuleEntrys
[TPS_BSWMDT_4017]	Reference to standardized BswModuleEntrys
[TPS_BSWMDT_4018]	BswModuleEntity.calledEntry
[TPS_BSWMDT_4019]	BswModuleEntity attributes
[TPS_BSWMDT_4020]	Usage of BswSchedulerNamePrefix
[TPS_BSWMDT_4021]	Usage of BswEvent
[TPS_BSWMDT_4022]	Timing and background events for BSW
[TPS_BSWMDT_4023]	Internal trigger and timing events for BSW
[TPS_BSWMDT_4024]	External trigger event for BSW
[TPS_BSWMDT_4025]	Mode switches and events in BSW
[TPS_BSWMDT_4026]	Local BSW data without RTE support
[TPS_BSWMDT_4027]	Local BSW data with RTE support
[TPS_BSWMDT_4028]	Determination of argument names for BSW functions called via ports
[TPS_BSWMDT_4029]	Usage of BswServiceDependency
[TPS_BSWMDT_4030]	BswImplementation.arReleaseVersion
[TPS_BSWMDT_4031]	Instances of BswImplementation
[TPS_BSWMDT_4032]	BswImplementation.requiredHW
[TPS_BSWMDT_4033]	Reference to vendor specific configuration parameters
[TPS_BSWMDT_4034]	Reference to predefined or recommended configuration values
[TPS_BSWMDT_4035]	Published parameter values
[TPS_BSWMDT_4036]	Back-reference from EcucModuleConfigurationValues
[TPS_BSWMDT_4037]	BswDebugInfo
[TPS_BSWMDT_4038]	Data types for debug data
[TPS_BSWMDT_4039]	Association of an Implementation with a component or module
[TPS_BSWMDT_4040]	Implementation.codeDescriptor
[TPS_BSWMDT_4041]	DependencyOnArtifact
[TPS_BSWMDT_4042]	Usage of DependencyOnArtifact
[TPS_BSWMDT_4043]	Compiler
[TPS_BSWMDT_4044]	Linker
[TPS_BSWMDT_4045]	Implementation.resourceConsumption
[TPS_BSWMDT_4046]	Memory section name
[TPS_BSWMDT_4047]	Memory section prefix
[TPS_BSWMDT_4048]	Scope of declared memory sections
[TPS_BSWMDT_4049]	Usage of MemorySection.executableEntity
[TPS_BSWMDT_4050]	ExecutionTime
[TPS_BSWMDT_4051]	ExecutionTime references an ECU
[TPS_BSWMDT_4052]	ExecutionTime.hardwareConfiguration
[TPS_BSWMDT_4053]	ExecutionTime.memorySectionLocation
[TPS_BSWMDT_4054]	ExecutionTime.softwareContext
[TPS_BSWMDT_4055]	ExecutionTime.includedLibrary
[TPS_BSWMDT_4056]	Multiplicity of McSupportData
[TPS_BSWMDT_4057]	Self-contained MC support artifact
[TPS_BSWMDT_4058]	McSupportData.measuredSystemConstantValues
[TPS_BSWMDT_4059]	Granularity of McDataInstance.subElements
[TPS_BSWMDT_4060]	McDataInstance.resultingProperties

[TPS_BSWMDT_4061]	McSwEmulationMethodSupport.category
[TPS_BSWMDT_4062]	Upstream reference for emulation support
[TPS_BSWMDT_4063]	BSW Interface Variation Points
[TPS_BSWMDT_4064]	BSW Behavior Variation Points
[TPS_BSWMDT_4065]	BSW Implementation Variation Points
[TPS_BSWMDT_4066]	Relevant elements for ICS on Interface level
[TPS_BSWMDT_4067]	No relevant elements for ICS on Internal Behavior level
[TPS_BSWMDT_4068]	Relevant elements for ICS on Implementation level
[TPS_BSWMDT_4069]	Configuration in ICS
[TPS_BSWMDT_4070]	Variants in ICS

Table 13.3: Added Specification Items in 4.0.3

13.3.3 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 13.4: Added Constraints in R4.0.3

13.3.4 Deleted Constraints in R4.0.2

N/A