Document Title	Software Component Template
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	062
Document Classification	Standard

Document Version	4.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Description
2011-10-26	4.2.0	AUTOSAR Administration	• Added `CompuMethod` categories SCALE_LINEAR_AND_TEXTTABLE and SCALE_RATIONAL_AND_TEXTTABLE (table 5.67) • Clarification concerning the usage of invalid values • Revised support for data filters • Support for partial networking • Support for the specification of local connections between software-components • Improved description of service needs • Change history of constraints and specification items • Miscellaneous improvements and clarifications • “Support for Standardization” moved to Standardization Template [1]

2010-10-14	4.1.0	AUTOSAR Administration	• Remove restriction on data type of inter-runnable variables • Rework end-to-end communication protection • Add more constraints on the usage of the meta-model • Various fixes and clarifications
2009-09-15	4.0.0	AUTOSAR Administration	• New requirements tracing table • Support for fixed data exchange • Implementation of meta-model cleanup • Fundamental revision of the data type concept • Support for variant handling • Support for end-to-end communication protection • Support for documentation • Support for stopping and restarting of software-components • Support for triggered events • Support for explicit mapping of interface elements • Revised concept of mode management • Support for integrity and scaling at ports • Support for standardization within AUTOSAR
2008-07-02	3.1.0	AUTOSAR Administration	• Improved support for on-board diagnostics • Small layout adaptations made

2007-11-13	3.0.0	AUTOSAR Administration	• Improved support for measurement and calibration • Improved semantics of delegation ports • Introduction of abstract memory classes • Document meta information extended • Small layout adaptations made
2007-01-31	2.1.0	AUTOSAR Administration	• Harmonization of the document with other specifications (e.g. RTE) • Introduction of a new concept to support calibration and measurement - harmonized with RTE • Description of needs of the Software Component Template toward AUTOSAR services and of the interaction of the Software Component Template and services (on XML level) • Legal disclaimer revised • Release notes added • "'Advice for users"' added • "'Revision information"' added
2006-05-18	2.0.0	AUTOSAR Administration	Second
2005-05-09	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

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

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

Table of Contents

References

[1] Standardization Template
AUTOSAR_TPS_StandardizationTemplate.pdf

[2] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[3] Virtual Functional Bus
AUTOSAR_EXP_VFB.pdf

[4] Methodology
AUTOSAR_TR_Methodology.pdf

[5] Specification of Interoperability of AUTOSAR Tools
AUTOSAR_TR_InteroperabilityOfAutosarTools.pdf

[6] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[7] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[8] Specification of Timing Extensions
AUTOSAR_TPS_TimingExtensions.pdf

[9] Requirements on Timing Extensions
AUTOSAR_RS_TimingExtensions.pdf

[10] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate.pdf

[11] System Template
AUTOSAR_TPS_SystemTemplate.pdf

[12] AUTOSAR Template Modeling Patterns
AUTOSAR_TemplateModelingPatterns.pdf

[13] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[14] Requirements on Software Component Template
AUTOSAR_RS_SoftwareComponentTemplate.pdf

[15] Specification of I/O Hardware Abstraction
AUTOSAR_SWS_IOHardwareAbstraction.pdf

[16] Communication
http://portal.osek-vdx.org/files/pdf/specs/
osekcom303.pdf

[17] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_SWS_E2ELibrary.pdf

[18] Specification of Communication Manager
AUTOSAR_SWS_COMManager.pdf

[19] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager.pdf

[20] Specification of Communication
AUTOSAR_SWS_COM.pdf

[21] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[22] ASAM MCD 2 Harmonized Data Objects Version 1.1
harmonized-data-objects-V1.1.pdf

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

[24] ASAM AE Calibration Data Format V2.0.0
http://www.asam.net
ASAM-AE-CDF-V2_0_0-Users-Guide.pdf

[25] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[26] Specification of ECU Configuration Parameters
AUTOSAR_MOD_ECUConfigationParameters.pdf

[27] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.pdf

[28] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager.pdf

[29] Specification of ECU State Manager with fixed state machine
AUTOSAR_SWS_ECUStateManagerFixed.pdf

[30] Specification of Crypto Service Manager
AUTOSAR_SWS_CryptoServiceManager.pdf

[31] Specification of Function Inhibition Manager
AUTOSAR_SWS_FunctionInhibitionManager.pdf

[32] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[33] Specification of Diagnostic Communication Manager
AUTOSAR_SWS_DiagnosticCommunicationManager.pdf

[34] Specification of Diagnostic Log and Trace
AUTOSAR_SWS_DiagnosticLogAndTrace.pdf

[35] Specification of Synchronized Time-Base Manager

AUTOSAR_SWS_SynchronizedTimeBaseManager.pdf

[36] Glossary
AUTOSAR_TR_Glossary.pdf

[37] ASAM AE Functional Specification Exchange Format V1.0.0
http://www.asam.net
AE-FSX_V1.0.0.pdf

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

1 Introduction

1.1 Overview

This document contains the specification of the AUTOSAR `Software-Component Template`. Actually, it has been created as a supplement to the formal definition of the `Software-Component Template` by means of the AUTOSAR meta-model. In other words, this document in addition to the formal specification provides introductory description and rationale for the part of the AUTOSAR meta-model relevant for the definition of software-components.

In this context, the term software-component refers to a formally described piece of software existing that needs the AUTOSAR RTE [2] for execution.

Please note that the general ideas behind the semantics of application software-components have been described in the specification of the `Virtual Functional Bus` [3]. The latter, however, represents conceptual work that strongly influences but does not totally govern the formal definition of software-components.

Note further that this document does not provide any "best practice" recommendations of software-component modeling nor does it require or enforce a certain methodology. Note however, that the methodology aspect is covered by the specification of the AUTOSAR methodology [4].

Although it is beyond any doubt reasonable to use a suitable AUTOSAR Authoring Tool for dealing with AUTOSAR software-components, this specification does not make any assumptions nor does it give recommendations regarding the tooling. Please refer to [5] for more details about AUTOSAR Authoring Tools are supposed to work and interact.

1.2 Scope

As already mentioned in chapter 1.1, the Scope of this document is the description of AUTOSAR software-components. This work covers the following three aspects:

- A general description of `SwComponentType`s using `PortPrototype`s and `PortInterface`s, i.e. this document defines the `SwComponentType` as an entity which can be described through `PortPrototype`s which provide or require `PortInterface`s.
- A description of `CompositionSwComponentType`s which are sub-systems consisting out of connected instances of software-components, i.e. software-components may be defined in the form of hierarchical subsystems which in turn consist of software-components again. The description of such hierarchical structures is in scope of this document.

- A description of AtomicSwComponentType which is implemented as a piece of software that can be mapped to an AUTOSAR ECU.
  An AtomicSwComponentType therefore shows up in the ECU Software Architecture depicted in Figure 1.1. In this figure, the green (vertically striped) and blue (diagonally striped) borders show the aspects that are described by the Software-Component Template.

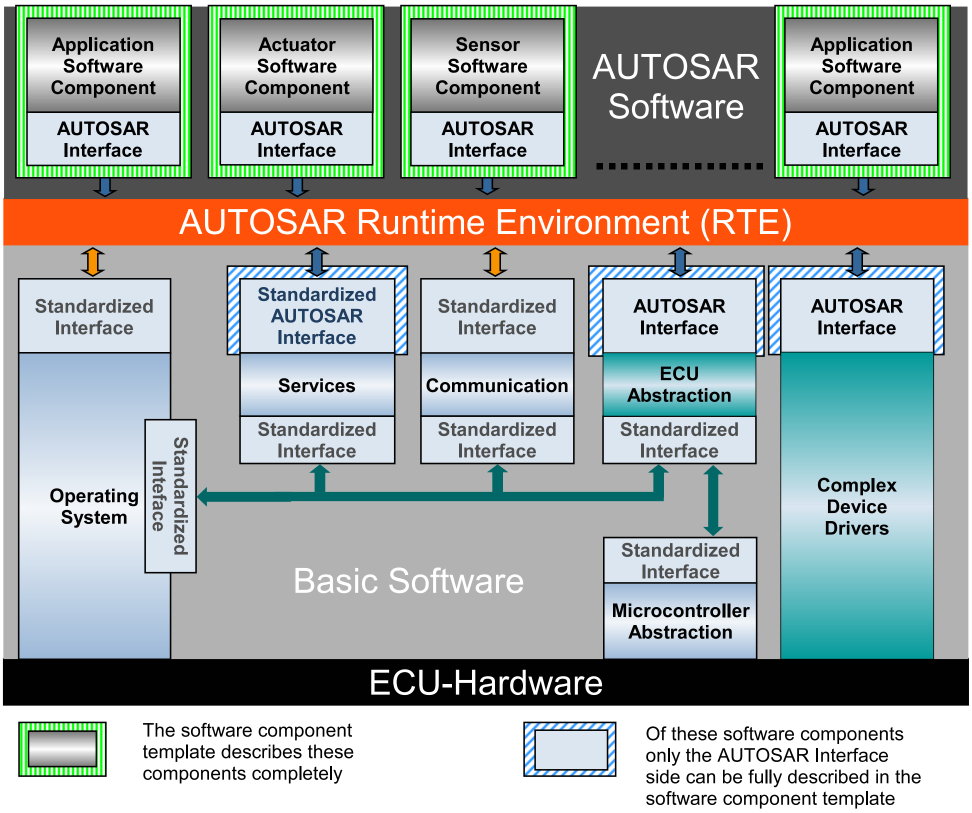

Figure 1.1: Scope of this document in the ECU SW Architecture [6]

Aspects of AUTOSAR Basic Software not relevant for the RTE are out of scope; these are covered by the Basic Software Module Description Template [7].

Also, the document does not cover aspects of timing analysis with respect to the execution of AUTOSAR software-components. This issue is explained in the Specification of Timing Extensions [8] as well as the corresponding requirements specification [9].

1.3 Organization of the Meta-Model

Figure 1.2 sketches the overall structure of the meta-model which formally defines the vocabulary required to describe AUTOSAR software-components. As the diagram points out, other template specifications (e.g. ECU Resource Template [10] and System Template [11]) also use the same modeling approach in order to define an overall consistent model of AUTOSAR software description.

The dashed arrows in the diagram describe dependencies in terms of import-relationships between the packages within the meta-model. For example, the package `SWComponentTemplate` imports meta-classes defined in the packages `GenericStructure` [12] and `ECUResourceTemplate` [10].

Please note that this specification document will (with some well-defined exceptions) mostly discuss meta-model elements defined in the package `SWComponentTemplate`.

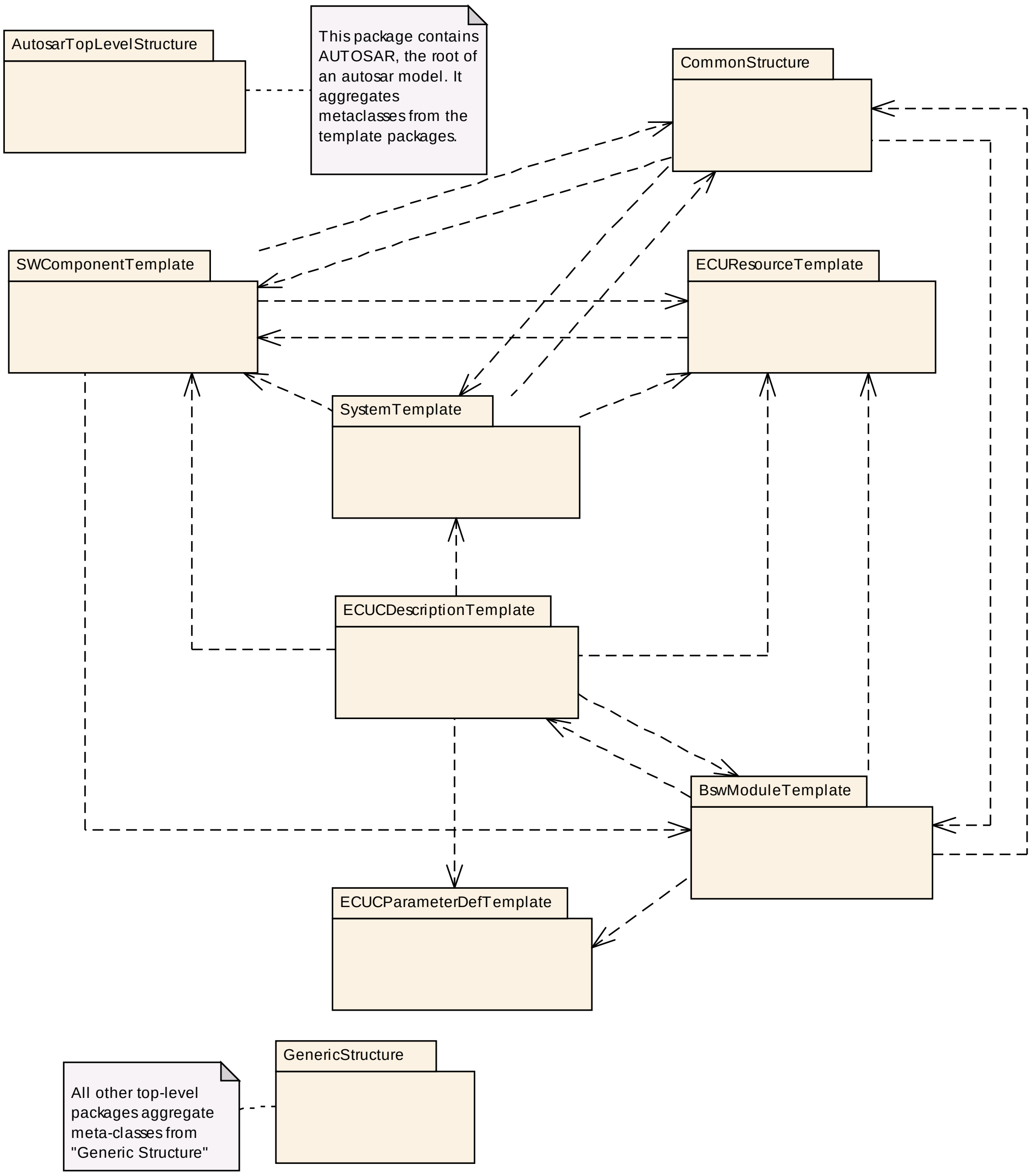

Figure 1.2: Structure of the meta-model

For clarification, please note that the package `GenericStructure` contains some fundamental infrastructure meta-classes and common patterns that are described in [13]. As these are used by all other template specification the dependency associations are not depicted in the diagram for the sake of clarity.

Generic Structure provides Details about

- AUTOSAR Top level structure,
- Commonly used meta-classes and primitives
- Variant Handling
- Documentation

1.4 Structure of the Template

AUTOSAR software components are described on three distinctive levels, as shown in Figure 1.3.

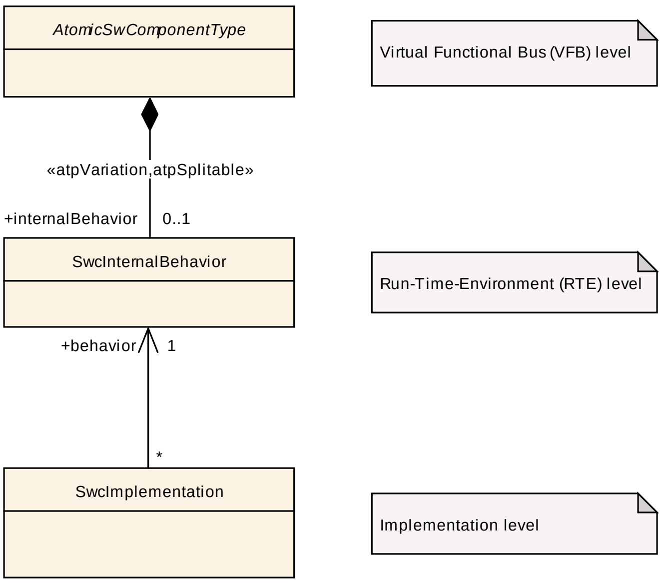

Figure 1.3: The description of a software component is done on three levels

1.4.1 Description of Software Components on VFB Level

The highest (most abstract) description level is the `Virtual Functional Bus` [3]. In this document `SwComponentType`s are described with the means of `DataTypes`, `PortInterface`s, `PortPrototype`s, and connections between them. At this level, the fundamental communication properties of components and their communication relationships among each other are expressed.

In the diagram depicted in Figure 1.3, this aspect is expressed by means of the description of `AtomicSwComponentType`[1].

[1]To avoid clutter and require additional up-front information about the meta model, `CompositionSwComponentType`s have not been added to the diagram.

1.4.2 Description of Software Components on RTE Level

The middle level allows for behavior description of a given AtomicSwComponentType. This so-called SwcInternalBehavior is expressed according to AUTOSAR RTE concepts, e.g. RTEEvents and in terms of schedulable units, so-called RunnableEntitys.

For instance, for a ClientServerOperation defined in the scope of a particular ClientServerInterface on the VFB, the behavior specifies which RunnableEntity is activated as a consequence of the invocation of the specific ClientServerOperation.

As sketched by Figure 1.3, there may be zero or one SwcInternalBehaviors aggregated by a given AtomicSwComponentType. In response to the existence of the stereotype ≪atpSplitable≫ at the aggregation it is possible to distribute the aggregation over several physical files.

1.4.3 Descriptions of Software Components on Implementation Level

The lowest level of description specifies the implementation (i.e. in terms of the AUTOSAR meta-model: the SwcImplementation) of a given SwcInternalBehavior description. More precisely, the RunnableEntitys of such a behavior are mapped to code (source code or object code).

There may be different SwcImplementations that reference a specific SwcInternalBehavior description, e.g. in different programming languages, or with differently optimized code.

Please note that Implementation has been described in previous versions of this document. In response to the evolution of the AUTOSAR concept the description of the Implementation aspect has been moved to the "GenericStructure" (see Figure 1.2) because it is also used for creating the Basic Software Module Description Template [7].

However, the SwcImplementation still remains in the scope of this document as it exclusively covers aspects of software-components rather than basic software modules.

1.5 Document Conventions

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

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

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

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

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

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:** xml.globalElement=true			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data of an Autosar file. **Tags:** xml.sequenceOffset=10
arPackage	ARPackage	*	aggr	This is the top level package in an AUTOSAR model. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivation Time atp.Splitkey=shortName xml.sequenceOffset=30
introduction	Documentation Block	0..1	aggr	This represents an introduction on the Autosar file. It is intended for example to rpresent disclaimers and legal notes. **Tags:** xml.sequenceOffset=20

Table 1.1: AUTOSAR

The first rows in the table have the following meaning:

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

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

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

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

The headers in the table have the following meaning:

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

Datatype: The datatype of an attribute of the class.

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

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

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

1.6 Requirements Tracing

The following table references the requirements specified in [14] as well as [2] and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document. In this case chapter 1.7 provides more background about why the requirement is not fulfilled.

Requirement	Description	Satisfied by
[Dlt495]	No description	[TPS_SWCT_2506]
[Dlt496]	No description	[TPS_SWCT_2506]
[Dlt497]	No description	[TPS_SWCT_2506]
[Dlt498]	No description	[TPS_SWCT_2506]
[FIM090]	No description	[TPS_SWCT_2505]
[NVM734]	No description	[TPS_SWCT_2501] [TPS_SWCT_2502] [TPS_SWCT_2503] [TPS_SWCT_2504]
[NVM735]	No description	[TPS_SWCT_2501] [TPS_SWCT_2502] [TPS_SWCT_2503] [TPS_SWCT_2504]
[NVM736]	No description	[TPS_SWCT_2501] [TPS_SWCT_2502] [TPS_SWCT_2503] [TPS_SWCT_2504]
[NVM737]	No description	[TPS_SWCT_2501] [TPS_SWCT_2502] [TPS_SWCT_2503] [TPS_SWCT_2504]
[NVM738]	No description	[TPS_SWCT_2504]
[RS_SWCT_0010]	AUTOSAR shall support inter- and intra-ECU-communication mechanisms with high reliability	[TPS_SWCT_1025] [TPS_SWCT_1026] [TPS_SWCT_1027] [TPS_SWCT_1069] [TPS_SWCT_1070] [TPS_SWCT_1111]
[RS_SWCT_0020]	AUTOSAR shall provide open and standardized software interfaces for intra-ECU and inter-ECU communication	[TPS_SWCT_1002]

Requirement	Description	Satisfied by
[RS_SWCT_0030]	AUTOSAR shall provide complete interfaces to application software and basic software modules	[TPS_SWCT_1002]
[RS_SWCT_0040]	AUTOSAR shall ease the re-usability of software and its concepts and implementations	
[RS_SWCT_0050]	AUTOSAR shall provide a software architecture that is applicable across different functional domains	
[RS_SWCT_0070]	AUTOSAR shall provide an abstraction of the application software from hardware	[TPS_SWCT_1030] [TPS_SWCT_1097] [TPS_SWCT_1098]
[RS_SWCT_0080]	AUTOSAR shall provide an independency of application software from in-vehicle communication technologies	[TPS_SWCT_1025] [TPS_SWCT_1026] [TPS_SWCT_1027] [TPS_SWCT_1069] [TPS_SWCT_1070]
[RS_SWCT_0090]	AUTOSAR should provide an independency of application software from operating systems	[TPS_SWCT_1030] [TPS_SWCT_1097] [TPS_SWCT_1098]
[RS_SWCT_0110]	AUTOSAR shall provide a functional interface view of the entire system	[TPS_SWCT_1025] [TPS_SWCT_1026] [TPS_SWCT_1027] [TPS_SWCT_1069] [TPS_SWCT_1070]
[RS_SWCT_0120]	AUTOSAR shall provide protection/unlock mechanisms for software through appropriate services in the infrastructure	[TPS_SWCT_1031] [TPS_SWCT_1049] [TPS_SWCT_1050] [TPS_SWCT_1051] [TPS_SWCT_1052] [TPS_SWCT_1053] [TPS_SWCT_1054] [TPS_SWCT_1055] [TPS_SWCT_1321]
[RS_SWCT_0130]	AUTOSAR shall provide interoperability with legacy software	
[RS_SWCT_0150]	AUTOSAR shall provide means to protect SW-Components from malicious SW-components	[TPS_SWCT_1002]
[RS_SWCT_0160]	AUTOSAR shall provide means to achieve compositionality	[TPS_SWCT_1002]
[RS_SWCT_0170]	AUTOSAR shall provide diagnostics means during runtime, for production and services purposes	[TPS_SWCT_1028] [TPS_SWCT_1029] [TPS_SWCT_1131] [TPS_SWCT_1132] [TPS_SWCT_1133] [TPS_SWCT_1134] [TPS_SWCT_1135] [TPS_SWCT_1136] [TPS_SWCT_1137] [TPS_SWCT_1138] [TPS_SWCT_1139] [TPS_SWCT_1140] [TPS_SWCT_1425] [TPS_SWCT_1426] [TPS_SWCT_1427] [TPS_SWCT_1428] [TPS_SWCT_2002] [TPS_SWCT_2003] [TPS_SWCT_2004] [TPS_SWCT_2005] [TPS_SWCT_2006] [TPS_SWCT_2007] [TPS_SWCT_2008] [TPS_SWCT_2009] [TPS_SWCT_2010] [TPS_SWCT_2011] [TPS_SWCT_2012] [TPS_SWCT_2013] [TPS_SWCT_2014] [TPS_SWCT_2015] [TPS_SWCT_2016] [TPS_SWCT_2017]

Requirement	Description	Satisfied by
[RS_SWCT_0190]	AUTOSAR shall support hierarchical design methods	[TPS_SWCT_1032] [TPS_SWCT_1033] [TPS_SWCT_1034] [TPS_SWCT_1035] [TPS_SWCT_1036] [TPS_SWCT_1037]
[RS_SWCT_0200]	Definitions of relations between SW components are exhaustive and formal	[TPS_SWCT_1002] [TPS_SWCT_1322] [TPS_SWCT_1323] [TPS_SWCT_1325] [TPS_SWCT_1326] [TPS_SWCT_1327] [TPS_SWCT_1328] [TPS_SWCT_1329] [TPS_SWCT_1330] [TPS_SWCT_1331] [TPS_SWCT_1333] [TPS_SWCT_1334] [TPS_SWCT_1335] [TPS_SWCT_1336] [TPS_SWCT_1337] [TPS_SWCT_1338] [TPS_SWCT_1339] [TPS_SWCT_1340] [TPS_SWCT_1341] [TPS_SWCT_1342] [TPS_SWCT_1343] [TPS_SWCT_1344] [TPS_SWCT_1345] [TPS_SWCT_1346] [TPS_SWCT_1347] [TPS_SWCT_1348] [TPS_SWCT_1349] [TPS_SWCT_1350] [TPS_SWCT_1351] [TPS_SWCT_1352] [TPS_SWCT_1353]
[RS_SWCT_0210]	SW components are protected from illegal access	[TPS_SWCT_1002]
[RS_SWCT_0220]	Management of vehicle diversity is supported by AUTOSAR	[TPS_SWCT_1038] [TPS_SWCT_1039] [TPS_SWCT_1040] [TPS_SWCT_1041] [TPS_SWCT_1042]
[RS_SWCT_2000]	Top-down hierarchical design	[TPS_SWCT_1032] [TPS_SWCT_1033] [TPS_SWCT_1034] [TPS_SWCT_1035] [TPS_SWCT_1036] [TPS_SWCT_1037]
[RS_SWCT_2010]	Interfaces of atomic software-components	[TPS_SWCT_1002]
[RS_SWCT_2020]	Bottom-up design of CompositionTypes	[TPS_SWCT_1032] [TPS_SWCT_1033] [TPS_SWCT_1034] [TPS_SWCT_1035] [TPS_SWCT_1036] [TPS_SWCT_1037]
[RS_SWCT_2030]	Specification of Communications	[TPS_SWCT_1002] [TPS_SWCT_1025] [TPS_SWCT_1026] [TPS_SWCT_1027]
[RS_SWCT_2050]	Specification of timing resources for software-component description	
[RS_SWCT_2060]	Consider interaction with basic software	[TPS_SWCT_1043] [TPS_SWCT_1044] [TPS_SWCT_1045] [TPS_SWCT_1046]
[RS_SWCT_2080]	Designing a Sensor Actuator Component	[TPS_SWCT_1047] [TPS_SWCT_1048]
[RS_SWCT_2090]	Data-consistency for communication among RunnableEntities	[TPS_SWCT_1031] [TPS_SWCT_1049] [TPS_SWCT_1050] [TPS_SWCT_1051] [TPS_SWCT_1052] [TPS_SWCT_1053] [TPS_SWCT_1054] [TPS_SWCT_1055]
[RS_SWCT_2100]	Definition of physical units	[TPS_SWCT_1056] [TPS_SWCT_1057] [TPS_SWCT_1058] [TPS_SWCT_1059] [TPS_SWCT_1060] [TPS_SWCT_1061] [TPS_SWCT_1068]
[RS_SWCT_2110]	Definition of comments	[TPS_SWCT_1062]
[RS_SWCT_3000]	Compositions	[TPS_SWCT_1032] [TPS_SWCT_1033] [TPS_SWCT_1034] [TPS_SWCT_1035] [TPS_SWCT_1036] [TPS_SWCT_1037]

Requirement	Description	Satisfied by
[RS_SWCT_3010]	Interfaces	[TPS_SWCT_1025] [TPS_SWCT_1026] [TPS_SWCT_1069] [TPS_SWCT_1070]
[RS_SWCT_3020]	Libraries	
[RS_SWCT_3030]	Integration on object code level	
[RS_SWCT_3040]	Behavior	[TPS_SWCT_1075] [TPS_SWCT_1108]
[RS_SWCT_3050]	Schedulability	[TPS_SWCT_1030] [TPS_SWCT_1097] [TPS_SWCT_1098]
[RS_SWCT_3060]	Sequence of execution of runnable entities	
[RS_SWCT_3070]	Needed resources for SW Components	
[RS_SWCT_3080]	Timing-requirements of SW-Components	
[RS_SWCT_3090]	Needed and usable sensors and actuators	[TPS_SWCT_1047] [TPS_SWCT_1048]
[RS_SWCT_3100]	Support of variant handling	[TPS_SWCT_1038] [TPS_SWCT_1040] [TPS_SWCT_1041] [TPS_SWCT_1042] [TPS_SWCT_1370] [TPS_SWCT_1371] [TPS_SWCT_1372] [TPS_SWCT_1373]
[RS_SWCT_3110]	Support of modes	[TPS_SWCT_1071] [TPS_SWCT_1190] [TPS_SWCT_1376] [TPS_SWCT_1377] [TPS_SWCT_1378] [TPS_SWCT_1379] [TPS_SWCT_1380] [TPS_SWCT_1381] [TPS_SWCT_1382] [TPS_SWCT_1383] [TPS_SWCT_1384] [TPS_SWCT_1385] [TPS_SWCT_1386] [TPS_SWCT_1387] [TPS_SWCT_1388]
[RS_SWCT_3120]	Dependency on modes	[TPS_SWCT_1077]
[RS_SWCT_3128]	No description	[TPS_SWCT_1128]
[RS_SWCT_3130]	Connections between PortInterfaces	[TPS_SWCT_1079] [TPS_SWCT_1080] [TPS_SWCT_1081] [TPS_SWCT_1082] [TPS_SWCT_1083] [TPS_SWCT_1084] [TPS_SWCT_1113]
[RS_SWCT_3135]	Record Type Subsetting	[TPS_SWCT_1023] [TPS_SWCT_1024]
[RS_SWCT_3136]	Record Type Subsetting with Primitive Types	[TPS_SWCT_1195]
[RS_SWCT_3140]	Conditional existence of PortPrototypes	[TPS_SWCT_1038]
[RS_SWCT_3141]	Conditional existence of data element prototypes, operation prototypes, parameter prototypes in an interface.	[TPS_SWCT_1106]
[RS_SWCT_3142]	Conditional existence of ComponentPrototypes	[TPS_SWCT_1038]
[RS_SWCT_3143]	Conditional existence of ConnectorPrototypes	[TPS_SWCT_1040]
[RS_SWCT_3144]	Configurable size of Arrays	[TPS_SWCT_1076] [TPS_SWCT_1078]
[RS_SWCT_3145]	Describe supported combinations of System Constant Value of an Software Component Type	

Requirement	Description	Satisfied by
[RS_SWCT_3146]	Describe supported combinations of System Constant Value of an InternalBehavior	
[RS_SWCT_3147]	Describe supported combinations of System Constant Value of an Implementation	
[RS_SWCT_3148]	Attributes swMinAxisPoints and swMaxAxisPoints shall be adjustable by an System Constant Definition	[TPS_SWCT_1107] [TPS_SWCT_1181]
[RS_SWCT_3149]	Conditional existence of RunnableEntitys	[TPS_SWCT_1085]
[RS_SWCT_3150]	Conditional existence of RTEEvents	[TPS_SWCT_1085]
[RS_SWCT_3151]	Conditional existence of InterRunnableVariables	[TPS_SWCT_1085]
[RS_SWCT_3152]	Conditional accessibility for measurement	[TPS_SWCT_1130]
[RS_SWCT_3153]	Conditional existence of parameter prototypes	[TPS_SWCT_1085]
[RS_SWCT_3154]	Support of conditional ports for SW-C	[TPS_SWCT_1038]
[RS_SWCT_3155]	Support of Interfaces with different resolutions	[TPS_SWCT_1099] [TPS_SWCT_1100] [TPS_SWCT_1101] [TPS_SWCT_1102] [TPS_SWCT_1103] [TPS_SWCT_1104] [TPS_SWCT_1105]
[RS_SWCT_3170]	Fixed data exchange	[TPS_SWCT_1102] [TPS_SWCT_1103] [TPS_SWCT_1104]
[RS_SWCT_3175]	M2 support for definition of calibration datasets	[TPS_SWCT_1177] [TPS_SWCT_1178] [TPS_SWCT_1188]
[RS_SWCT_3180]	Support of SAE J1939 Protocol Features	[TPS_SWCT_1076]
[RS_SWCT_3181]	Need data type and access support for arrays of variable number of elements within the maximum size	[TPS_SWCT_1076]
[RS_SWCT_3182]	Need data type and access support for byte arrays of variable number of elements	[TPS_SWCT_1127]
[RS_SWCT_3190]	Ability to publish/specify the diagnostic capabilities and its resources of an SWC	[TPS_SWCT_1129]
[RS_SWCT_3200]	Add support for Vehicle and Application Mode Management Concept	[TPS_SWCT_1008] [TPS_SWCT_1009] [TPS_SWCT_1010] [TPS_SWCT_1011] [TPS_SWCT_1063] [TPS_SWCT_1064] [TPS_SWCT_1065] [TPS_SWCT_1066] [TPS_SWCT_1067] [TPS_SWCT_1071]
[RS_SWCT_3201]	Add support for Portgroups	[TPS_SWCT_1096] [TPS_SWCT_1126]
[RS_SWCT_3202]	Enable SWCs to request dedicated Modes	[TPS_SWCT_1086] [TPS_SWCT_1201]

Requirement	Description	Satisfied by
[RS_SWCT_3203]	Propagation of Mode Information	[TPS_SWCT_1086] [TPS_SWCT_1087] [TPS_SWCT_1200] [TPS_SWCT_1201] [TPS_SWCT_1202]
[RS_SWCT_3210]	Integrity and Scaling at Ports	[TPS_SWCT_1023] [TPS_SWCT_1024] [TPS_SWCT_1099] [TPS_SWCT_1100] [TPS_SWCT_1101] [TPS_SWCT_1102] [TPS_SWCT_1103] [TPS_SWCT_1104] [TPS_SWCT_1105] [TPS_SWCT_1158] [TPS_SWCT_1159] [TPS_SWCT_1160] [TPS_SWCT_1161] [TPS_SWCT_1162] [TPS_SWCT_1163] [TPS_SWCT_1164] [TPS_SWCT_1165] [TPS_SWCT_1166] [TPS_SWCT_1167] [TPS_SWCT_1168]
[RS_SWCT_3215]	Need to add application data type on top of implementation data type	[TPS_SWCT_1072] [TPS_SWCT_1073] [TPS_SWCT_1074] [TPS_SWCT_1189] [TPS_SWCT_1229] [TPS_SWCT_1231] [TPS_SWCT_1235] [TPS_SWCT_1236]
[RS_SWCT_3216]	Application data type	[TPS_SWCT_1072] [TPS_SWCT_1073] [TPS_SWCT_1179] [TPS_SWCT_1180] [TPS_SWCT_1181] [TPS_SWCT_1183] [TPS_SWCT_1184] [TPS_SWCT_1185] [TPS_SWCT_1189] [TPS_SWCT_1191] [TPS_SWCT_1229] [TPS_SWCT_1230] [TPS_SWCT_1231] [TPS_SWCT_1235] [TPS_SWCT_1236] [TPS_SWCT_1237] [TPS_SWCT_1240] [TPS_SWCT_1241] [TPS_SWCT_1242] [TPS_SWCT_1243] [TPS_SWCT_1249] [TPS_SWCT_1256]
[RS_SWCT_3217]	Implementation data type	[TPS_SWCT_1072] [TPS_SWCT_1074] [TPS_SWCT_1183] [TPS_SWCT_1184] [TPS_SWCT_1189] [TPS_SWCT_1191] [TPS_SWCT_1229] [TPS_SWCT_1231] [TPS_SWCT_1232] [TPS_SWCT_1233] [TPS_SWCT_1235] [TPS_SWCT_1236] [TPS_SWCT_1237] [TPS_SWCT_1248] [TPS_SWCT_1250] [TPS_SWCT_1251] [TPS_SWCT_1252] [TPS_SWCT_1253] [TPS_SWCT_1254] [TPS_SWCT_1255] [TPS_SWCT_1257] [TPS_SWCT_1258] [TPS_SWCT_1259]
[RS_SWCT_3220]	Allow Communication Attributes on Compositions	[TPS_SWCT_1088]
[RS_SWCT_3225]	Enhancing the Non-Volatile (NV) memory interface	[TPS_SWCT_1141] [TPS_SWCT_1142] [TPS_SWCT_1143] [TPS_SWCT_1227] [TPS_SWCT_1228]
[RS_SWCT_3230]	Documentation of M1 artifacts	[TPS_SWCT_1062]
[RS_SWCT_3240]	Support end-to-end communication protection	[TPS_SWCT_1089] [TPS_SWCT_1090] [TPS_SWCT_1091] [TPS_SWCT_1092] [TPS_SWCT_1093] [TPS_SWCT_1094] [TPS_SWCT_1095]
[RS_SWCT_3241]	Support for partial networking	[TPS_SWCT_1169] [TPS_SWCT_1170] [TPS_SWCT_1171] [TPS_SWCT_1172] [TPS_SWCT_1173] [TPS_SWCT_1174] [TPS_SWCT_1175]

Requirement	Description	Satisfied by
[SWS_CSM_0775]	No description	[TPS_SWCT_2020]
[SWS_CSM_0776]	No description	[TPS_SWCT_2021]
[SWS_CSM_0777]	No description	[TPS_SWCT_2022]
[SWS_CSM_0778]	No description	[TPS_SWCT_2023]
[SWS_CSM_0779]	No description	[TPS_SWCT_2024]
[SWS_CSM_0780]	No description	[TPS_SWCT_2025]
[SWS_CSM_0781]	No description	[TPS_SWCT_2026]
[SWS_CSM_0782]	No description	[TPS_SWCT_2027]
[SWS_CSM_0783]	No description	[TPS_SWCT_2028]
[SWS_CSM_0784]	No description	[TPS_SWCT_2029]
[SWS_CSM_0785]	No description	[TPS_SWCT_2030]
[SWS_CSM_0786]	No description	[TPS_SWCT_2031]
[SWS_CSM_0787]	No description	[TPS_SWCT_2032]
[SWS_CSM_0788]	No description	[TPS_SWCT_2033]
[SWS_CSM_0789]	No description	[TPS_SWCT_2034]
[SWS_CSM_0790]	No description	[TPS_SWCT_2035]
[SWS_CSM_0791]	No description	[TPS_SWCT_2036]
[SWS_CSM_0792]	No description	[TPS_SWCT_2037]
[SWS_CSM_0793]	No description	[TPS_SWCT_2038]
[SWS_CSM_0794]	No description	[TPS_SWCT_2039]
[SWS_CSM_0795]	No description	[TPS_SWCT_2040]
[SWS_CSM_0796]	No description	[TPS_SWCT_2041]
[SWS_CSM_0797]	No description	[TPS_SWCT_2042]
[SWS_CSM_0798]	No description	[TPS_SWCT_2043]
[SWS_CSM_0799]	No description	[TPS_SWCT_2044]
[SWS_CSM_0800]	No description	[TPS_SWCT_2045]
[SWS_CSM_0801]	No description	[TPS_SWCT_2020] [TPS_SWCT_2021] [TPS_SWCT_2022] [TPS_SWCT_2023] [TPS_SWCT_2024] [TPS_SWCT_2025] [TPS_SWCT_2026] [TPS_SWCT_2027] [TPS_SWCT_2028] [TPS_SWCT_2029] [TPS_SWCT_2030] [TPS_SWCT_2031] [TPS_SWCT_2032] [TPS_SWCT_2033] [TPS_SWCT_2034] [TPS_SWCT_2035] [TPS_SWCT_2036] [TPS_SWCT_2037] [TPS_SWCT_2038] [TPS_SWCT_2039] [TPS_SWCT_2040] [TPS_SWCT_2041] [TPS_SWCT_2042] [TPS_SWCT_2043] [TPS_SWCT_2044] [TPS_SWCT_2045]
[SWS_ComM_0741]	No description	[TPS_SWCT_1021]
[SWS_ComM_0847]	No description	[TPS_SWCT_1019]
[SWS_ComM_0848]	No description	[TPS_SWCT_1020]
[SWS_DCM_0685]	No description	[TPS_SWCT_2015]
[SWS_DCM_0686]	No description	[TPS_SWCT_2002] [TPS_SWCT_2005] [TPS_SWCT_2008]
[SWS_DCM_0687]	No description	[TPS_SWCT_2003] [TPS_SWCT_2006] [TPS_SWCT_2009]
[SWS_DCM_0688]	No description	[TPS_SWCT_2010]
[SWS_DCM_0689]	No description	[TPS_SWCT_2011]
[SWS_DCM_0690]	No description	[TPS_SWCT_2004]
[SWS_DCM_0691]	No description	[TPS_SWCT_2012]
[SWS_DCM_0692]	No description	[TPS_SWCT_2016]
[SWS_DCM_0694]	No description	[TPS_SWCT_2017]

Requirement	Description	Satisfied by
[SWS_DCM_0695]	No description	[TPS_SWCT_2014]
[SWS_DCM_0698]	No description	[TPS_SWCT_2013]
[SWS_DCM_0699]	No description	[TPS_SWCT_2014]
[SWS_DEM_0600]	No description	[TPS_SWCT_1426]
[SWS_DEM_0601]	No description	[TPS_SWCT_1132]
[SWS_DEM_0602]	No description	[TPS_SWCT_1133]
[SWS_DEM_0603]	No description	[TPS_SWCT_1133]
[SWS_DEM_0604]	No description	[TPS_SWCT_1134]
[SWS_DEM_0605]	No description	[TPS_SWCT_1135]
[SWS_DEM_0606]	No description	[TPS_SWCT_1136]
[SWS_DEM_0607]	No description	[TPS_SWCT_1137]
[SWS_DEM_0608]	No description	[TPS_SWCT_1138]
[SWS_DEM_0609]	No description	[TPS_SWCT_1428]
[SWS_DEM_0610]	No description	[TPS_SWCT_2007]
[SWS_DEM_0611]	No description	[TPS_SWCT_2007]
[SWS_DEM_0612]	No description	[TPS_SWCT_1139]
[SWS_DEM_0614]	No description	[TPS_SWCT_1131]
[SWS_DEM_0616]	No description	[TPS_SWCT_1426]
[SWS_DEM_0617]	No description	[TPS_SWCT_1140]
[SWS_DEM_0618]	No description	[TPS_SWCT_1425]
[SWS_DEM_0619]	No description	[TPS_SWCT_1426]
[SWS_DEM_0621]	No description	[TPS_SWCT_1427]
[SWS_EcuM_2749]	No description	[TPS_SWCT_1012]
[SWS_EcuM_2762]	No description	[TPS_SWCT_1012] [TPS_SWCT_1013]
[SWS_RTE_2568]	Definition for mode declaration	[TPS_SWCT_1010]
[SWS_WDGM_0333]	No description	[TPS_SWCT_2018]
[SWS_WDGM_0335]	No description	[TPS_SWCT_2018]
[SWS_WDGM_0336]	No description	[TPS_SWCT_2019]

1.7 Comments regarding non-fulfilled requirements

This section contains a list of requirements that are not fulfilled by this document along with more background information.

Requirement	Description	Comment
RS_SWCT_0040 AUTOSAR shall ease the reusability of software and its concepts and implementations	In general, this requirement is sort of the reason or motivation why the software-component template exists, i.e. AUTOSAR software-components can be deployed and redeployed to different ECUs.	This requirement is therefore fulfilled by all chapters of this document.

Requirement	Description	Comment
RS_SWCT_0050 AUTOSAR shall provide a software architecture that is applicable across different functional domains	The original main requirement explains that the term *functional domain* boils down to • Body/Comfort • Power train • Chassis • Safety • Multimedia • Human-machine-interface	This document does not contain any parts that ensure or else contradict the specific requirement. In other words, the document does not specifically address applicability in different functional domains.
RS_SWCT_0130 AUTOSAR shall provide interoperability with legacy software	This document does not make any assumption about how software-components inter-operate with legacy software. Please note that the `Implementation` meta-class provides support for `requiredArtifact` but still this is out of scope.	This requirement is not fulfilled by this document. More information can be found in [7].
RS_SWCT_2050 Specification of timing resources for software-component description	Although there is certainly a solid conceptual connection to the formal definition of software-components this requirement is addressed in [8].	This requirement is not fulfilled. For more information about how this aspect is considered in AUTOSAR please refer to [8].
RS_SWCT_3020 Sequence of execution of runnable entities		
RS_SWCT_3030 Sequence of execution of runnable entities		
RS_SWCT_3060 Sequence of execution of runnable entities	The fulfillment of this requirement would mean to describe relations among `RunnableEntities` that control the order of execution with respect to the enclosing `AtomicSwComponentType`.	This requirement is out of the scope this document. More information can be found in [8].
RS_SWCT_3070 Needed resource for SW Components	The meta-model in general covers this requirement by the existence of `Implementation` but the latter is not in the scope of this document.	The requirement is not fulfilled by this document. More information can be found in [7].
RS_SWCT_3080 Timing-requirements of SW-Components	The meta-model in general covers this requirement by the existence of `TimingExtension` but the latter is not in the scope of this document.	The requirement is not in the scope of this document. More information can be found in [8].
RS_SWCT_3145 Describe supported combinations of System Constant Value of an Software Component Type	This requirement is addressed by the variant handling concept.	This requirement is out of the scope of this document. Please find more information in [13].

Requirement	Description	Comment
RS_SWCT_3146 Describe supported combinations of System Constant Values of an InternalBehavior	This requirement is addressed by the variant handling concept.	This requirement is out of the scope of this document. Please find more information in [13].
RS_SWCT_3147 Describe supported combinations of System Constant Value of an Implementation	This requirement is addressed by the variant handling concept.	This requirement is out of the scope of this document. Please find more information in [13].

2 Conceptual Aspects

2.1 Introduction

For the sake of a compact description of relevant meta-model elements the discussion and explanation of conceptual aspects has been concentrated in this chapter.

2.2 Measurement and Calibration

2.2.1 Basic Approach of Measurement and Calibration

While performing the calibration process using a MCD tool (Measurement, Calibration, and Diagnostic) the calibration engineer needs to have a specific insight to the data within the CPU at runtime.

This insight is provided by access to ECU internal variables (also called measurements) as well as calibration parameters (sometimes also called characteristic value).

[TPS_SWCT_1417] Define calibration parameters common to all `SwComponentPrototypes` of the same `SwComponentType` ⌈ Similar to `DataPrototype`s, calibration parameters can be defined for a `SwcInternalBehavior` common for all `SwComponentPrototype`s of the same `SwComponentType`, individually for a `SwComponentPrototype` (similar to `PerInstanceMemory`) as well as for several `SwComponentPrototype`s (using the port-/interface-concept with `ParameterInterface`s). ⌋

Therefore, the description of measurement variables and calibration parameters is basically the same. In AUTOSAR both appear finally as `DataPrototype`s.

2.2.2 Calibration Parameters Overview

A Calibration Parameter is a parameter which characterizes the dynamics of a control algorithm. From a software implementation point of view, it is a variable with only read-access during the normal operation of an ECU. Characteristics are specialized `DataPrototype` entities in terms of its associated type but are used in a similar way.

[TPS_SWCT_1418] Ways to define a calibration parameter ⌈ This means that Calibration Parameters can be defined

- individually for a `SwComponentPrototype` in the `SwcInternalBehavior` of a `SwComponentType` via an aggregation of an `ParameterDataPrototype` in the role of `perInstanceParameter` (similar to `PerInstanceMemory`) (see chapter 2.2.3.3)
- sharing between all `SwComponentPrototype`s of the same `SwComponentType` in its `SwcInternalBehavior` via an aggregation of an `ParameterDat-`

aPrototype in the role of sharedParameter or constantMemory (see chapter 2.2.3.2)

- for several SwComponentPrototypes (using the port-/interface-concept with ParameterInterfaces) (see chapter 2.2.3.1).

⌋

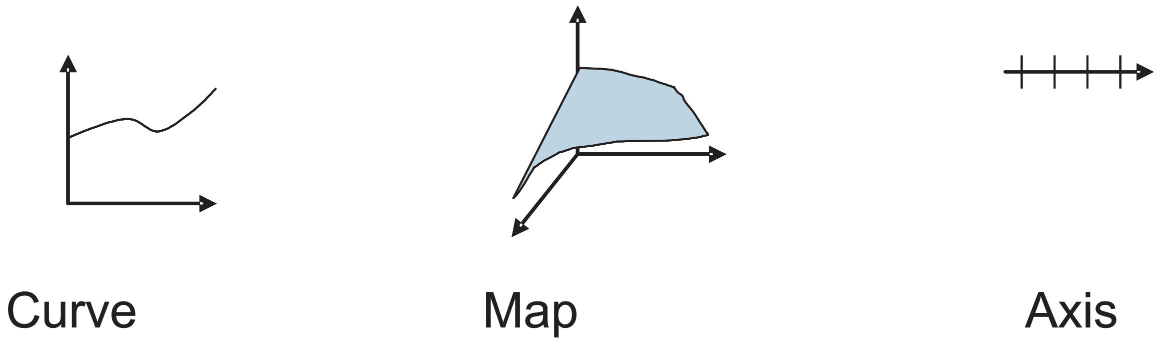

Figure 2.1: Some Categories of Calprms

2.2.3 Using Calibration Parameters

As mentioned above, a ParameterDataPrototype can be used in the context of SwcInternalBehavior as well as in the context of PortPrototypes.

2.2.3.1 Sharing Calibration Parameters within Compositions

To provide calibration parameters for being visible in other SwComponentTypes, a dedicated ParameterSwComponentType (see Figure 3.2) that inherits from SwComponentType has to be used as a SwComponentPrototype within a CompositionSwComponentType.

[TPS_SWCT_1419] ParameterSwComponentType shall never aggregate a SwcInternalBehavior ⌈ Please note that a ParameterSwComponentType shall never aggregate a SwcInternalBehavior and also owns exclusively PPortPrototypes of type ParameterInterface. This aspect is covered by [constr_1092].
⌋

Class	**ParameterSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ParameterSwComponentType defines parameters and characteristic values accessible via provided Ports. The provided values are the same for all connected SwComponentPrototypes **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
constantMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular ParameterSwComponentType
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular ParameterSwComponentType
instantiationDataDefProps	InstantiationDataDefProps	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 2.1: ParameterSwComponentType

[TPS_SWCT_1420] `SwComponentType` requiring access to shared calibration parameters needs `RPortPrototype` typed by a `ParameterInterface` ⌈ Every software `SwComponentType` requiring access to shared calibration parameters will have an `RPortPrototype` typed by a `ParameterInterface`. The definition of this shared calibration access in the context of a `CompositionSwComponentType` will be defined by creating a `SwConnector` between the relevant `SwComponentPrototypes`. ⌋

Class	**ParameterInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A parameter interface declares a number of parameter and characteristic values to be exchanged between parameter components and software components. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
parameter	ParameterDataPrototype	1..*	aggr	The ParameterDataPrototype of this ParameterInterface.

Table 2.2: ParameterInterface

[TPS_SWCT_1421] `ParameterInterface` is not restricted to parameters which can actually can be calibrated ⌈ Note that a `ParameterInterface` is not restricted to parameters which can actually can be calibrated. It can be used whenever there shall be no write access to the data during normal operation of the software, i.e. only constant date are visible over the interface. ⌋

The compatibility rules for `ParameterInterface`s are described in chapter 6.4; the compatibility rules for `ParameterDataPrototype`s are described in chapter 6.4.3.

[TPS_SWCT_1422] Delegation of `PortPrototypes` typed bay a`ParameterInterface` ⌈ Access to shared calibration parameters can be provided and required even over `CompositionSwComponentType`s using `DelegationSwConnector`s and `AssemblySwConnector`s.

This means that each access to calibration parameters between `SwComponentPrototype`s is explicitly visible. If a `SwConnector` spans after the mapping of software `SwComponentPrototype`s over two different ECUs the system generation process has to ensure the proper allocation of the `ParameterDataPrototype` (see Figure 2.2) while the calibration system has to cope with setting the parameter synchronously on the affected ECUs. ⌋

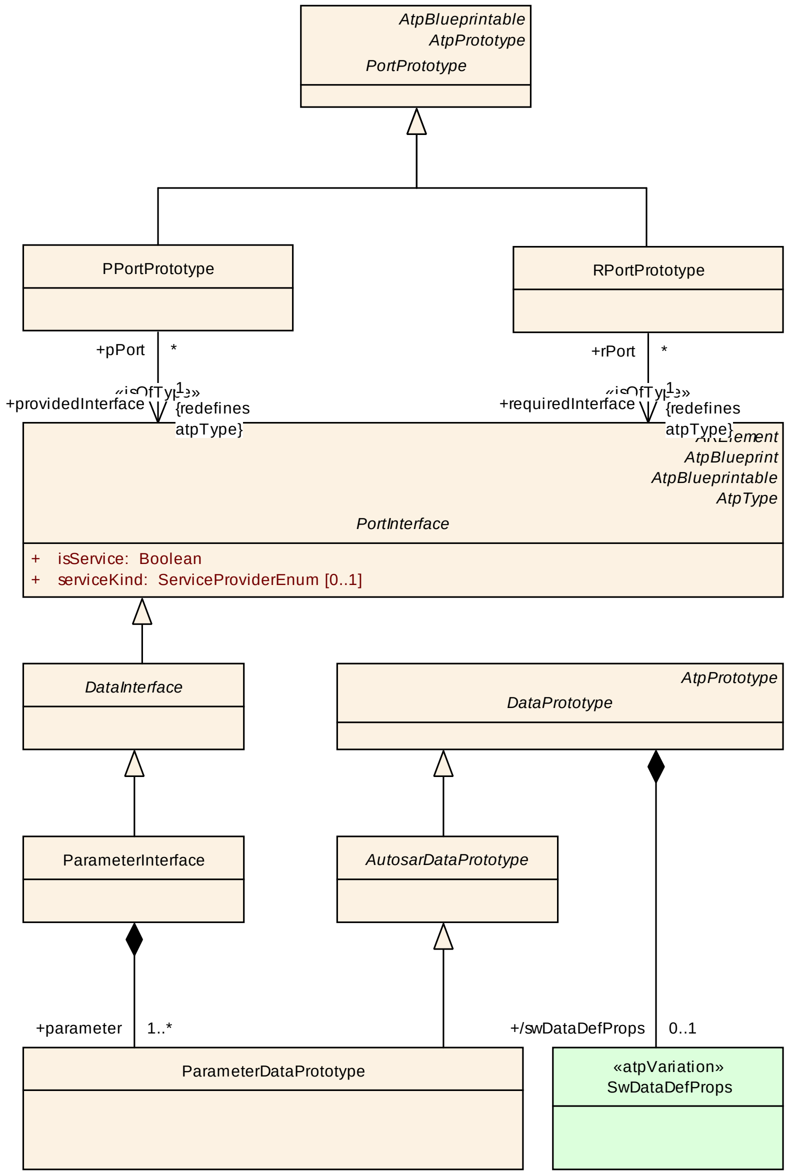

Figure 2.2: ParameterInterface

2.2.3.2 Sharing Calibration Parameters between SwComponentPrototypes of the Same SwComponentType

To share calibration parameters between several `SwComponentPrototype`s of the same `SwComponentType`, a `ParameterDataPrototype` is attached to an `SwcInternalBehavior` in `sharedParameter` role (see [TPS_SWCT_1417]).

When the `SwcInternalBehavior` is aggregated by an `AtomicSwComponentType` the actual calibration parameters of the `ParameterDataPrototype` is the same for all `SwComponentPrototype`s.

[TPS_SWCT_1423] `ParameterDataPrototype` aggregated in the role `constantMemory` ⌈ Additionally, it is possible to describe the implementation of shared characteristic values a via a ParameterDataPrototype which attached to an SwcInternalBehavior in constantMemory role.

In contrast to the ParameterDataPrototype in sharedParameter role this kind of memory is not instantiated by the RTE. This supports more efficient implementations (especially for software components provided as object code) by avoidance of the additional indirection caused by the RTE's component data structure. ⌋

Further on this kind of memory reduces the dependencies of the software-component's implementation to generated RTE code which is appreciated for safety related functionalities.

Nevertheless the information about these characteristic values has to be taken into account for the A2L file generation.

A typical example for this kind of sharing code between instances is dealing with two lambda sensors in multiple cylinder-bank engines, where (at least) two SwComponentPrototypes for each lambda sensor will use the very same Calibration Parameters.

2.2.3.3 Providing Instance Individual Characteristic Data

[TPS_SWCT_1424] ParameterDataPrototype aggregated in the role perInstanceParameter ⌈ To provide instance individual calibration parameters a ParameterDataPrototype is owned by a SwcInternalBehavior in perInstanceParameter role.

When the SwcInternalBehavior is attached to an AtomicSwComponentType, the actual calibration values are specific for each SwComponentPrototype. ⌋

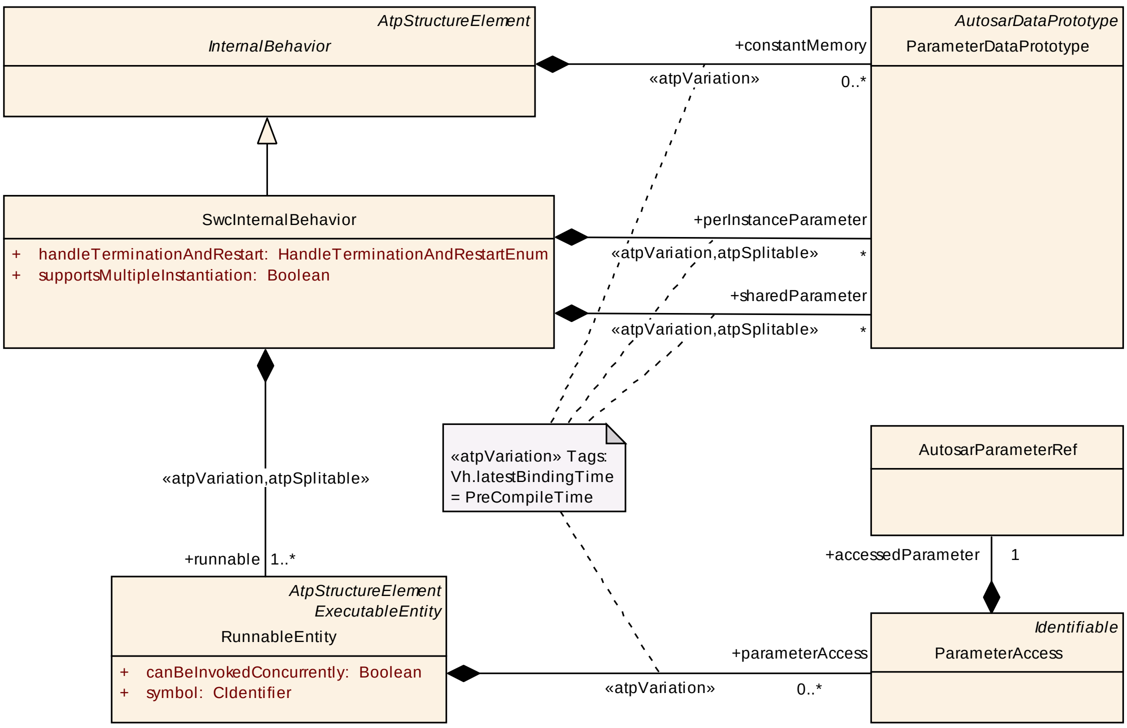

Figure 2.3: ParameterDataPrototypes in internal behavior

2.3 Runtime and Data Consistency Aspects

2.3.1 Background: the Issues

This section gives some background information and lists possible strategies concerning the implementation of the RunnableEntitys and the RTE with respect to efficient communication between the RunnableEntitys.

The communication among RunnableEntitys can very efficiently be implemented by means of "sharing memory"[1].

This is technically feasible because it is always guaranteed that the RunnableEntitys within an AtomicSwComponentType are always gathered at a specific processing unit (in other words: distribution is not an option).

Note that the purpose of communication among the RunnableEntitys is to establish a data flow scheme. The latter is a very popular pattern in the application of control theory to automotive embedded systems. So if "global variables" are used for establishing internal communication among RunnableEntitys they acquire the semantics of so called state-messages.

Nevertheless, directly sharing memory between RunnableEntitys requires a serious problem to be solved: the guarantee of data consistency among communicating RunnableEntitys. The RunnableEntitys will indeed be mapped to tasks so that one RunnableEntity of an AtomicSwComponentType may be preempted by a different RunnableEntity of the same AtomicSwComponentType.

Please note that a purist approach to achieving data consistency not only applies to single accesses of concurrently accessed variables. Rather, it would not be permitted that the value of a concurrently accessed variable (with state-message semantics) is unintentionally changed during the run-time of a RunnableEntity.

The following paragraphs describe some common strategies that can be used to ensure the required data-consistency. We do not attempt to describe the pros or cons of these approaches.

2.3.1.1 Mutual Exclusion with Semaphores

Multi-threaded operating systems provide mutexes (mutual exclusion semaphores) that protect access to an exclusive resource that is used from within several tasks.

[1] Please note that the term "sharing memory" can be interpreted on different levels. It is e.g. in the C language possible to use variables with external linkage (a.k.a. "global variables", although this term is not officially defined by the C language) for the purpose of inter-Runnable communication.

The RTE could use these OS-provided mutexes to make sure that the RunnableEntitys sharing a memory-space would never run concurrently. The RTE would make sure the task running the RunnableEntity has taken an appropriate mutex before accessing the memory shared between the RunnableEntitys.

2.3.1.2 Interrupt Disabling

Another alternative would be the disabling of interrupts during the run-time of RunnableEntitys or at least for a period in time identical to the interval from the first to the last usage of a concurrently accessed variable in a RunnableEntity. This approach could lead to seriously non-deterministic execution timing.

2.3.1.3 Priority Ceiling

Priority ceiling allows for a non-blocking protection of shared resources. Provided that the priority scheme is static, the AUTOSAR OS is capable of temporarily raising the priority of a task that attempts to access a shared resource to the highest priority of all tasks that would ever attempt to access the resource.

By this means is technically impossible that a task in temporary possession of a resource is ever preempted by a task that attempts to access the resource as well.

2.3.1.4 Implicit Communication by Means of Variable Copies

Another alternative is the usage of copies of concurrently accessed variables with state message semantics. Note that this approach directly corresponds to the semantics of "implicit" sender-receiver communication (see 7.5.1.2).

This means in particular that for a concurrently used variable a copy is created on which a RunnableEntity entity can work without any danger of data inconsistency.

This concept requires additional code to write the value of the concurrently accessed variable to the copy before the RunnableEntity that accesses the variable is executed. The value of the copy shall be written back to the concurrently accessed variable after the RunnableEntity has been terminated.

This concept is sketched in Figure 2.4. Since it would be too expensive and error-prone to manually care about the copy routines it would be a good idea to leave the creation of the additional code to a suitable code generator.

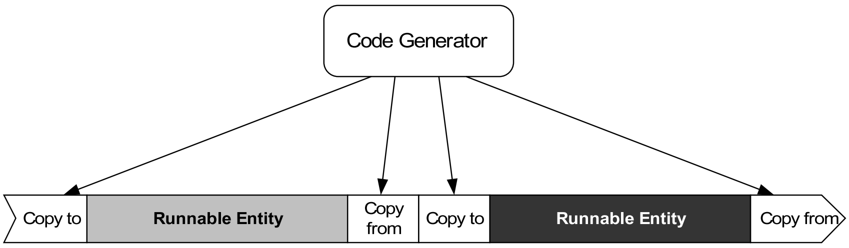

Figure 2.4: Generation of copy routines around `RunnableEntitys`

The additional copy routines as sketched in Figure 2.4 already protect the particular `RunnableEntitys` from unintended changes of concurrently accessed variables. It would, however, be possible to further optimize the process by reducing the additional code at the beginning and end of each task (see Figure 2.5).

2.3.2 Data Consistency at Runtime

In addition, copy routines will only be inserted where appropriate, e.g. a copy routine for writing the value of a copy back to the concurrently accessed variable will only be inserted if the `RunnableEntity` has write access to the concurrently used variable.

Please note that the copy routines have to temporarily make sure that the copy process is not interrupted in order to be capable of consistently copying the values from and to the concurrently accessed variable.

These periods, however, are supposed to be very short compared with the overall run-time consumption of the `RunnableEntity` and thus would not have a significant impact on the runtime behavior.

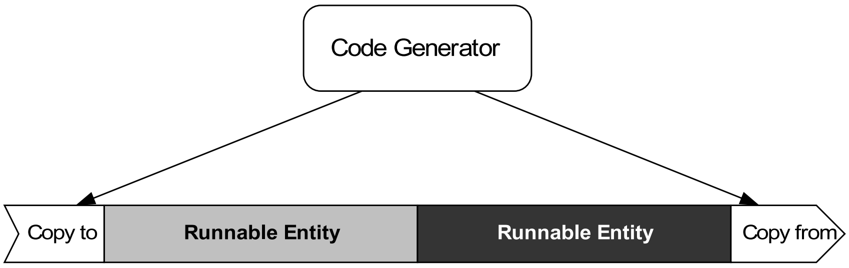

Figure 2.5: Optimized insertion of copy routines

Further optimization criteria can be applied, for example: it would be perfectly safe to avoid the creation of copies for `RunnableEntitys` that are scheduled in the task with the highest priority of all tasks that (via contained `RunnableEntitys`) access a certain concurrently accessed variable.

In order to keep the application code free of any dependencies from the code generation, access to concurrently accessed variables will be guarded by macros that are later resolved by the code generator.

The presence of the guard macros directly supports the reuse on the level of source code. The reuse on the level of object code is only possible if the scheduling scenario (in terms of the assignment of RunnableEntitys to priority levels) does not change.

This concept can only be implemented properly with the aid of a code generator if the variables in question can be identified. In other words: the description of an AtomicSwComponentType has to expose all concurrently accessed variables to the outside world.

2.3.3 Modeling Aspects of Data Consistency

The intrinsic meaning of the terms "explicit communication" and "implicit communication" is explained in section 7.5.1.1. It would be fair to say that the distinction between implicit and explicit communication establishes a usage pattern in the application domain, i.e. in the world of the developer of AUTOSAR software-components and their implementation.

There is another facet to this subject, however, namely the question how this pattern is implemented in the meta-model. With respect to the application of the pattern for port-based communication the details can be found in section 7.5.1.2, more specifically in section 7.5.1.3. The consideration of the internal communication based on so-called "inter-runnable variables" is described in section 7.4.2.

By reading the respective text sections it becomes apparent that the two applications of the pattern are modeled differently. The port-based communication uses the VariableAccess to formalize different roles of accessing communication elements. Some of the roles used for this purpose imply explicit communication (e.g. dataSendPoint) and some represent implicit communication (e.g. dataWriteAccess).

The important thing about using the VariableAccess, however, is that the modeling of communication roles is abstracted from the actual communication elements and represents a uniform (meaning: it can refer to the target directly or by a so-called InstanceRef) modeling approach that is applied for all use cases[2].

Admittedly, this is handled in a different way for the internal communication. Here, the additional layer of abstraction is not used (although it would have been technically feasible to do so) with respect to the clear separation of "inter-runnable variables with implicit behavior" and "inter-runnable variables with explicit behavior" in the RTE. The implementation of different communication roles (i.e. implicit vs. explicit) is done by directly aggregating VariableDataPrototype in the roles explicitInterRunnableVariable and implicitInterRunnableVariable.

On the other hand, access to internal communication **never** requires the usage of an InstanceRef and therefore the abstraction might be considered unnecessary overhead that blows up the M1 model.

[2]On a related note, even for non-communication related data access the same pattern applies implemented by ParameterAccess

2.4 Variant Handling in the Software Component Template

The Software Component Template supports the creation of *Variant*s in a subset of its model elements.

[TPS_SWCT_1038] Support for Variant Handling in the in Software Component Template ⌈ The Variant Handling support in the in Software Component Template is mainly driven by the purpose to describe a variable system on Virtual Functional Bus[3] level by varying

- the existence of SwComponentPrototypes
- the existence of SwConnectors
- the existence of Chapters of SwComponentDocumentation
- the existence of PortPrototypes

⌋*(RS_SWCT_0220, RS_SWCT_3100, RS_SWCT_3140, RS_SWCT_3142, RS_SWCT_3154)*

[TPS_SWCT_1039] Purpose of variant handling ⌈ This supports adjusting the number and kind of software-component instances as well as their interconnection in a particular system variant. ⌋*(RS_SWCT_0220)*

The first three cases are supporting *PostBuild* binding. For the existence of PortPrototypes only *PreCompileTime* is supported as latest Binding Time.

[TPS_SWCT_1040] SwConnector exists depending on a *PostBuild* condition ⌈ A SwConnector which exists depending on a *PostBuild* condition has an impact on the behavior of API function calls that apply on a PortPrototype to which the SwConnector is attached. If the SwConnector does not exist the behavior of the RTE API functions need to take this into account. ⌋*(RS_SWCT_0220, RS_SWCT_3100, RS_SWCT_3143)*

This means that the RTE implementation of this PortPrototype resembles the behavior of an unconnected PortPrototype. Please find more details in the specification of the RTE [2].

[TPS_SWCT_1041] API functions of not existing SwConnector are still part of the software-component's implementation ⌈ If SwConnectors do not exist the corresponding API functions are still part of the software-component's implementation. It is not possible to remove the API functions in a *PostBuild* step. Therefore the latest reasonable Binding Time for the conditional existence of a PortPrototype is *PreCompileTime*. ⌋*(RS_SWCT_0220, RS_SWCT_3100)*

[TPS_SWCT_1085] Variation on the behavior level ⌈ In addition to variation of the VFB-related model elements, the description of variant software-component implementations is supported. Please note that this requires a broad support of variability in the *Internal Behavior*.

The identified main use case are

- the varying existence of RunnableEntitys
- the varying existence of memory objects used inside the software-component implementation
- varying data structures
- the existence of RunnableEntitys
- the existence of RTEEvents
- the existence of VariableDataPrototypes in the roles implicitInterRunnableVariable and explicitInterRunnableVariable
- the existence of ParameterDataPrototypes in the roles perInstanceParameter, sharedParameter, and constantMemory

⌋*(RS_SWCT_3149, RS_SWCT_3150, RS_SWCT_3151, RS_SWCT_3153)*

For the same reason that applies on the existence of PortPrototype the latest Binding Time of these kinds of variability is *PreCompileTime*.

In the meta-model, all locations that may exhibit variability are marked with the stereotype ≪atpVariation≫. This allows the definition of possible variation points. Tagged Values are used to specify additional information, for example the latest binding time.

[TPS_SWCT_1042] Four types of locations in the meta-model which may exhibit variability ⌈ There are four types of locations in the meta-model which may exhibit variability:

- Aggregations
- Associations
- Attribute Values
- Classes providing property sets

⌋*(RS_SWCT_0220, RS_SWCT_3100)*

The reasons for the attachment of the stereotype ≪atpVariation≫ to certain model elements and the consequences for other model elements are explained in class tables in the following chapters. More details about the AUTOSAR Variant Handling Concept can be found in the AUTOSAR Generic Structure Template [13].

2.5 Communication Specification of Composition Component Types

[TPS_SWCT_1088] ComSpecs defined by CompositionSwComponents ⌈ It shall be possible to attach ComSpecs to PortPrototypes owned by CompositionSwComponents. ⌋*(RS_SWCT_3220)*

2.5.1 Rationale

ComSpecs attached to a PortPrototype owned by an AtomicSwComponentType have a direct impact on the generation of the RTE. The RTE Generator, on the other hand, does not consider the existence of CompositionSwComponentTypes.

Nevertheless, there are some cases where the definition of a ComSpec attached to a PortPrototype owned by a CompositionSwComponentType does make sense.

That is, in case an OEM wants to submit the definition of a CompositionSwComponentType to a supplier for adding more details and implementing the behavior the OEM might want to point out that from the OEM's point of view initValues apply for the elements of PortInterfaces used to type the delegation PortPrototypes.

The idea is that the supplier takes over the initValues attached to the delegation PortPrototypes and *copies* them to the PortPrototypes owned by SwComponentPrototypes of the CompositionSwComponentType.

The RTE Generator would still *only* take the initial values of the PortPrototypes of AtomicSwComponentTypes and ignore the initValues at the delegation PortPrototypes.

Therefore, the initValues of the delegation PortPrototype would be taken as *mere templates* for the detailing of PortPrototypes connected to the delegation PortPrototypes.

It is not required that the initValues of delegated PortPrototype and a PortPrototype connected by means of a DelegationSwConnector match.

Although this would certainly make sense in many cases it is eventually still left to the supplier to decide on the specific initValues applicable inside the CompositionSwComponentType.

On the other hand, a requirement that the initValues defined on the surface of CompositionSwComponentType and the inside of the CompositionSwComponentType shall be consistent in any case might effectively prevent the reuse of existing AtomicSwComponentTypes.

Please note that the ability to define a ComSpec in the context of a CompositionSwComponentType implies that it shall be possible to define mappings of ApplicationDataTypes used in a PortInterface to their corresponding ImplementationDataTypes.

For this purpose the CompositionSwComponentType owns a DataTypeMappingSet in the role dataTypeMapping and a ConstantSpecificationMappingSet in the role constantValueMapping.

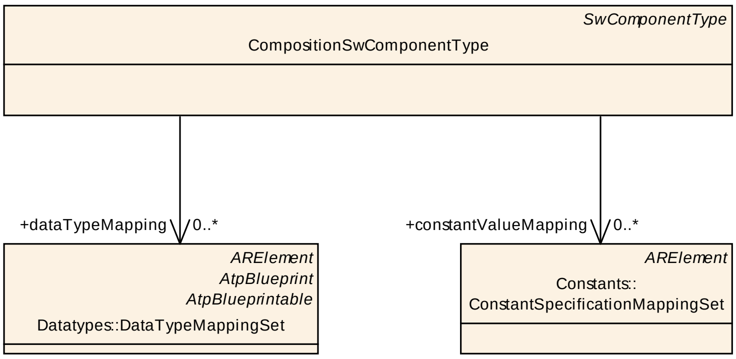

Figure 2.6: Specification of data type mapping for `CompositionSwComponentType`

3 Overview: Software Components, Ports, and Interfaces

3.1 Introduction

The detailed introduction of all aspects of the `Software Component Template` in one move is considered too complex. This chapter therefore provides an overview of the main conceptual aspects of software components, ports and interfaces. The overview will then be broken down into further details in chapter 4.

One of the goals of the AUTOSAR concept is the support of re-usability on the level of application software. In other words: it should be possible to re-use existing artifacts to create further model elements instead of being forced to create every single modeling detail from scratch. One of the consequences of this approach is the application of the so-called type-prototype pattern [13].

Among other things, this concept allows for creating hierarchical structures of software-components with arbitrary complexity. However, the creation of hierarchical structures itself does not have an impact on the run-time behavior of the overall system. The actual behavior is completely defined within the individual software-components.

This conclusion is backed by the understanding that software-components are developed against the so-called *Virtual Functional Bus* (VFB), an abstract communication channel without direct dependency on ECUs and communication buses. The VFB does not provide any means for expressing a hierarchy of software-components.

Of course, the usage of the VFB has further consequences on the design of software-components which shall not directly call the operating system or the communication hardware. As a result, software-components can be deployed to actual ECUs at a rather late stage in the development process.

In order to make the description more precise, the following text preferably uses accurate meta-model terms instead of the rather vague terminology of "composition" and "software-component".

3.2 Software Component

Application software within AUTOSAR is organized in self-contained units called `AtomicSwComponentType`s. Such `AtomicSwComponentType`s encapsulate the implementation of their functionality and behavior and merely expose well-defined connection points, called `PortPrototype`s, to the outside world.

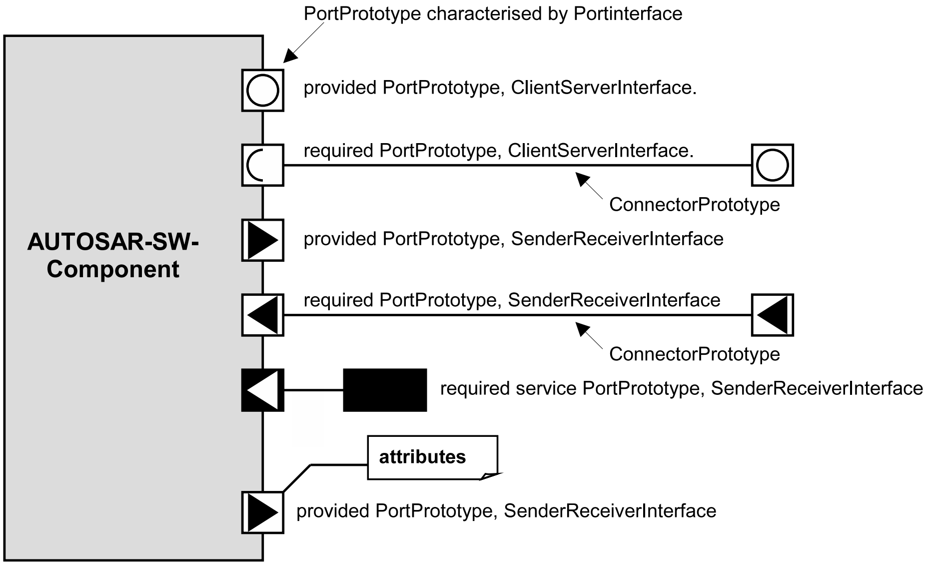

Figure 3.1: Graphical representation of software-components in AUTOSAR

The graphical appearance of AUTOSAR software-components according to [3] is depicted in Figure 3.1.

Class	**SwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for AUTOSAR software components.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
port	PortPrototype	*	aggr	The ports through which this component can communicate. The aggregation of PortPrototype is subject to variability with the purpose to support the conditional existence of PortPrototypes. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel
portGroup	PortGroup	*	aggr	A port group being part of this component. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
swComponentDocumentation	SwComponentDocumentation	0..1	aggr	This adds a documentation to the SwComponentType. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=swComponentDocumentation, variationPoint.shortLabel xml.sequenceOffset=-10

Table 3.1: SwComponentType

[TPS_SWCT_1002] SwComponentTypes may only interact by means of their PortPrototypes ⌈ AtomicSwComponentTypes (and also the more general SwComponentTypes may only interact by means of their PortPrototypes). Hidden communication dependencies that are *not* expressed by means of PortPrototypes are strictly forbidden. ⌋*(RS_SWCT_0020, RS_SWCT_0030, RS_SWCT_0150, RS_SWCT_0160, RS_SWCT_0200, RS_SWCT_0210, RS_SWCT_2010, RS_SWCT_2030)*

Therefore, software-components are in theory exchangeable as long as they implement the same functionality and provide the same public communication interface to the remaining system.

[TPS_SWCT_1096] PortGroup ⌈ PortPrototypes can be logically grouped into PortGroups. This mechanism is used for implementing mode management features and further explained in chapter 4.6. ⌋*(RS_SWCT_3201)*

[TPS_SWCT_1108] Added value of an AtomicSoftwareComponentType ⌈ As mentioned before, the term AtomicSwComponentType is a specific form of the general concept of the SwComponentType. The added value of an AtomicSwComponentType is that it can aggregate an InternalBehavior (see chapter 7). ⌋*(RS_SWCT_3040)*

[TPS_SWCT_1109] Adding the SwcInternalBehavior in a later process step ⌈ The aggregation of SwcInternalBehavior is stereotyped ≪atpSplitable≫ to allow for adding the SwcInternalBehavior in a later process step. In other words, it is possible to completely develop the VFB view of a software-component and later add more details like InternalBehavior. ⌋

Class	**AtomicSwComponentType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	An atomic software component is atomic in the sense that it cannot be further decomposed and distributed across multiple ECUs.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
internalBehavior	SwcInternalBehavior	0..1	aggr	The SwcInternalBehaviors owned by an AtomicSwComponentType can be located in a different physical file. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=internalBehavior, variationPoint.shortLabel
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the AtomicSwComponentType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName

Table 3.2: AtomicSwComponentType

There are several specialized `SwComponentType`s to describe specific software-components used in the different parts of the AUTOSAR Layered Architecture [6]. Further details are mentioned in chapter 10 and 11.

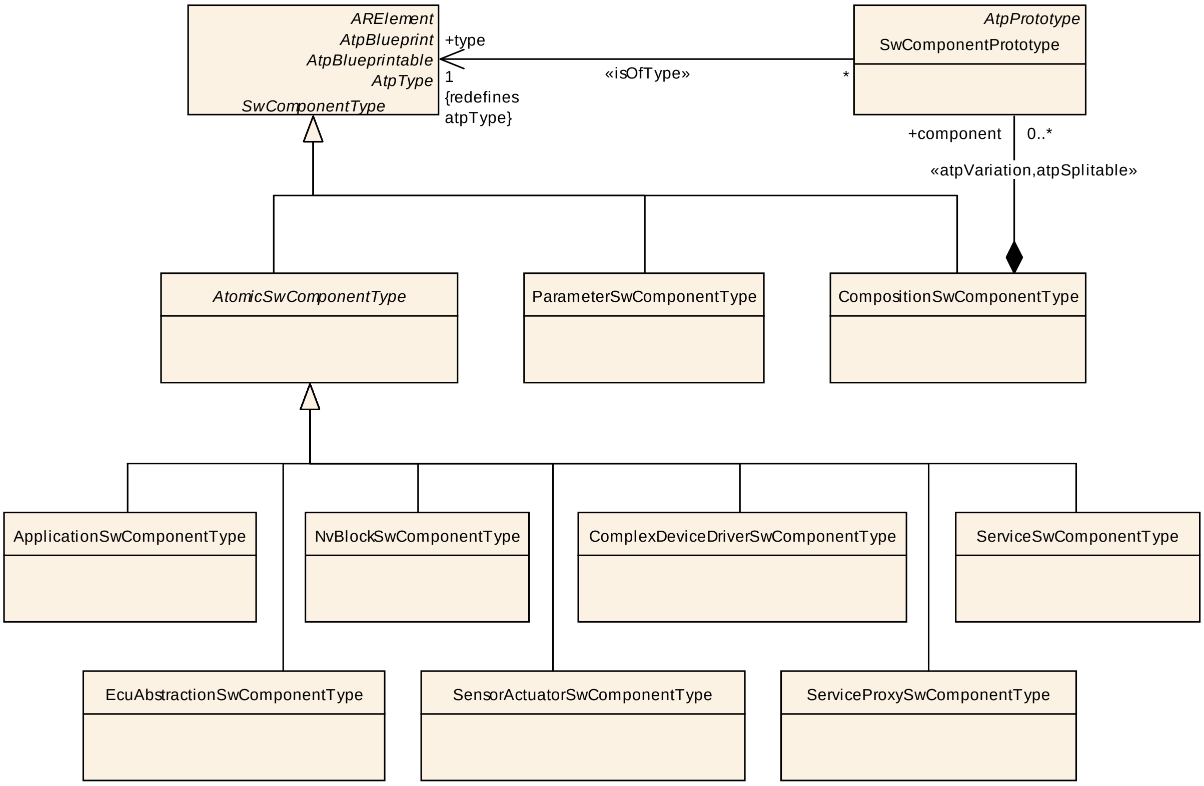

Figure 3.2: Overview of Component Types

The `ApplicationSwComponentType` is a specialization of `AtomicSwComponentType` for representing hardware-independent application software. The `ParameterSwComponentType` is a specialization of `SwComponentType` that can - in contrast to `AtomicSwComponentType` - not aggregate `SwcInternalBehavior`.

The purpose of the `NvBlockSwComponentType` is described in detail in section 11.5.2. The `ServiceSwComponentType` is described in section 11.3. Further on, the `EcuAbstractionSwComponentType` and the `ComplexDeviceDriverSwComponentType` are discussed in detail in section 10.

A description of the `ServiceProxySwComponentType` can be found in section 11.4 while the `SensorActuatorSwComponentType` is described in section 10.4.

[constr_1092] `ParameterSwComponentType` ⌈ A `ParameterSwComponentType` shall never aggregate a `SwcInternalBehavior` and also owns exclusively `PPortPrototype`s of type `ParameterInterface`. ⌋

However, a `ParameterSwComponentType` shall have the ability to aggregate `InstantiationDataDefProps`. By this means it is possible to define role-specific data properties of elements of composite data types used for the definition of calibration parameters in the scope of a `ParameterSwComponentType`.

For more information about this aspect please refer to section 7.5.4.

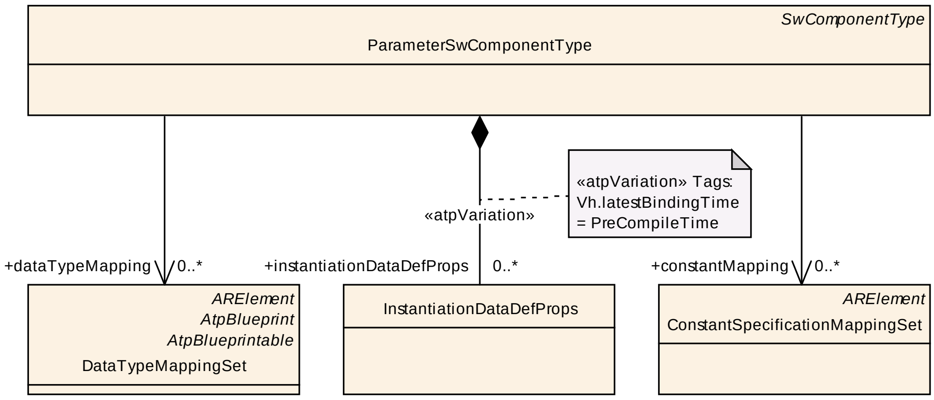

Figure 3.3: Details of `ParameterSwComponentType`

Class	**ApplicationSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ApplicationSwComponentType is used to represent the application software. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,Atp Classifier,AtpType,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 3.3: ApplicationSwComponentType

Please note that an `AtomicSwComponentType` manifests itself in the source code of an RTE into which an instance of the `AtomicSwComponentType` is deployed. This implies potential naming conflicts if instances of `AtomicSwComponentType` that have identical `shortName`s are deployed into a specific RTE.

[TPS_SWCT_1110] Symbolic name of a software-component ⌈ To mitigate this potential hazard it is possible to provide the `AtomicSwComponentType` along with an accompanying symbolic name that can be used for resolving the name clash. The symbolic name is provided by means of the attribute `symbol` of the meta-class `SymbolProps` owned by `AtomicSwComponentType` in the role `symbolProps` (for more information, please refer to Figure 3.4). ⌋

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Attribute	Datatype	Mul.	Kind	Note

Table 3.4: SymbolProps

[TPS_SWCT_1000] Usage of attribute `symbol` of the `symbolProps` ⌈ In particular, the RTE generator shall take over the value of the attribute `symbol` of the `swcSymbolProps` owned by a given `AtomicSwComponentType`. If and only if `swcSymbolProps` is not defined the RTE generator shall take the `shortName` of the `AtomicSwComponentType`. ⌋

[TPS_SWCT_1001] Prefix symbols generated for the `RunnableEntity` ⌈ The value of the attribute `symbol` of an `swcSymbolProps` owned by an `AtomicSwComponentType` shall also be taken for prefixing the symbols generated for the `RunnableEntity`s owned by the `AtomicSwComponentType`. ⌋

This is a further measure to mitigate the risk of potential name clashes in the RTE code.

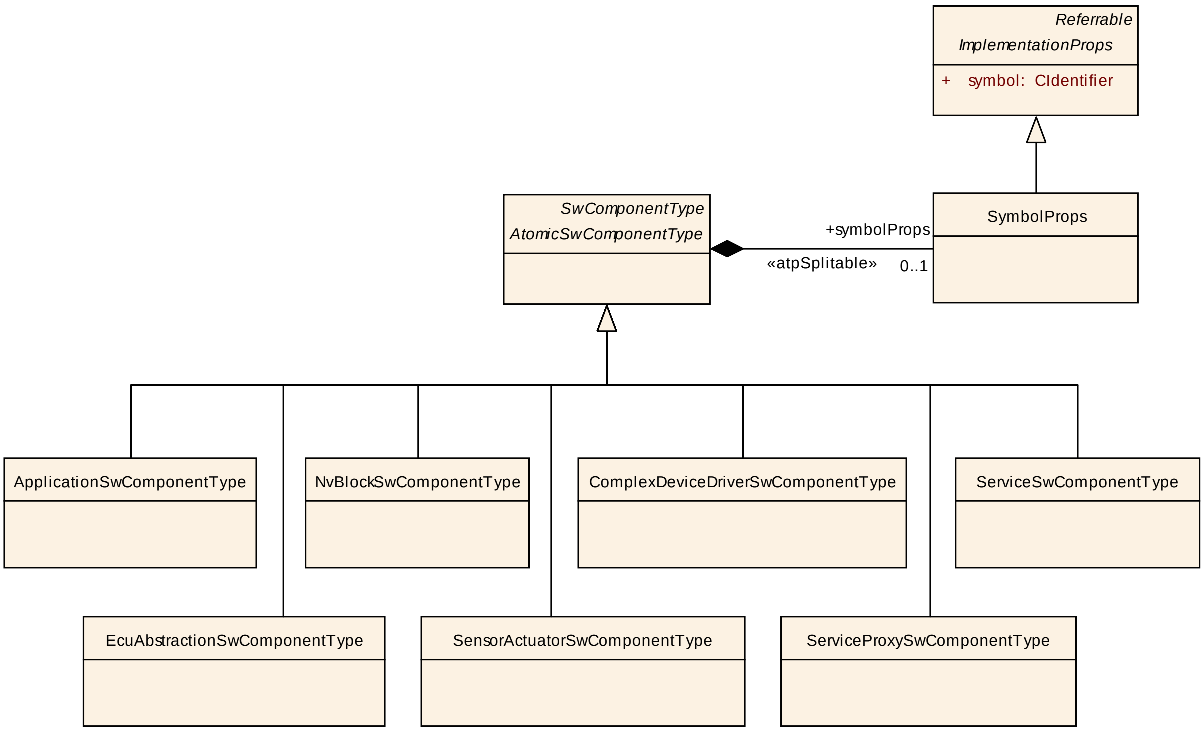

Figure 3.4: Overview of `AtomicSwComponentType`

Please note that `PortPrototype`s of a `SwComponentType` are supposed to be used for attaching `SwConnector`s that establish an actual connection between `SwComponentPrototype`s (see chapter 3.3).

Class	**PortPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Base class for the ports of an AUTOSAR software component. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports.			
Base	ARObject,AtpBlueprintable,AtpFeature,AtpPrototype,Identifiable,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
clientServerAnnotation	ClientServerAnnotation	*	aggr	Annotation of this PortPrototype with respect to client/server communication.
delegatedPortAnnotation	DelegatedPortAnnotation	0..1	aggr	Annotations on this delegated port.
ioHwAbstractionServerAnnotation	IoHwAbstractionServerAnnotation	*	aggr	Annotations on this IO Hardware Abstraction port.
modePortAnnotation	ModePortAnnotation	*	aggr	Annotations on this mode port.
nvDataPortAnnotation	NvDataPortAnnotation	*	aggr	Annotations on this non voilatile data port.
parameterPortAnnotation	ParameterPortAnnotation	*	aggr	Annotations on this parameter port.
senderReceiverAnnotation	SenderReceiverAnnotation	*	aggr	Collection of annotations of this ports sender/receiver communication.
triggerPortAnnotation	TriggerPortAnnotation	*	aggr	Annotations on this trigger port.

Table 3.5: PortPrototype

[TPS_SWCT_1111] `PortPrototypes` need an additional model artifact, the `PortInterface` ⌈ Please note that `PortPrototype`s actually need an additional model artifact, the `PortInterface`, for fully describing the details of the `PortPrototype`. The concept of the `PortInterface` as another means for establishing a high degree of re-usability is described in chapter 3.4. ⌋*(RS_SWCT_0010)*

[TPS_SWCT_1112] `PortPrototype`s are either *require*- or *provide*-ports. ⌈ As depicted in Figure 3.5, ports are either *require*- or *provide*-ports. A require-port (in technical terms: `RPortPrototype`) requires certain services or data, while a provide-port (or `PPortPrototype`) on the other hand provides those services or data. ⌋

[TPS_SWCT_1113] Connecting two `PortPrototypes` ⌈ Two `SwComponentPrototype`s are eventually connected by hooking up a `PPortPrototype` of one `SwComponentPrototype` to a compatible `RPortPrototype` of the other `SwComponentPrototype`s. Please find more information concerning the definition of "compatibility" in section 6. ⌋*(RS_SWCT_3130)*

Class	**RPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port requiring a certain port interface.			
Base	ARObject,AtpBlueprintable,AtpFeature,AtpPrototype,Identifiable,Multilanguage Referrable,PortPrototype,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
requiredComSpec	RPortComSpec	*	aggr	Required communication attributes, one for each interface element.
requiredInterface	PortInterface	1	tref	The interface that this port requires, i.e. the port depends on another port providing the specified interface. **Stereotypes:** isOfType

Table 3.6: RPortPrototype

Class	**PPortPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Component port providing a certain port interface.			
Base	ARObject,AtpBlueprintable,AtpFeature,AtpPrototype,Identifiable,Multilanguage Referrable,PortPrototype,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
providedComSpec	PPortComSpec	*	aggr	Provided communication attributes per interface element (data element or operation).
providedInterface	PortInterface	1	tref	The interface that this port provides. **Stereotypes:** isOfType

Table 3.7: PPortPrototype

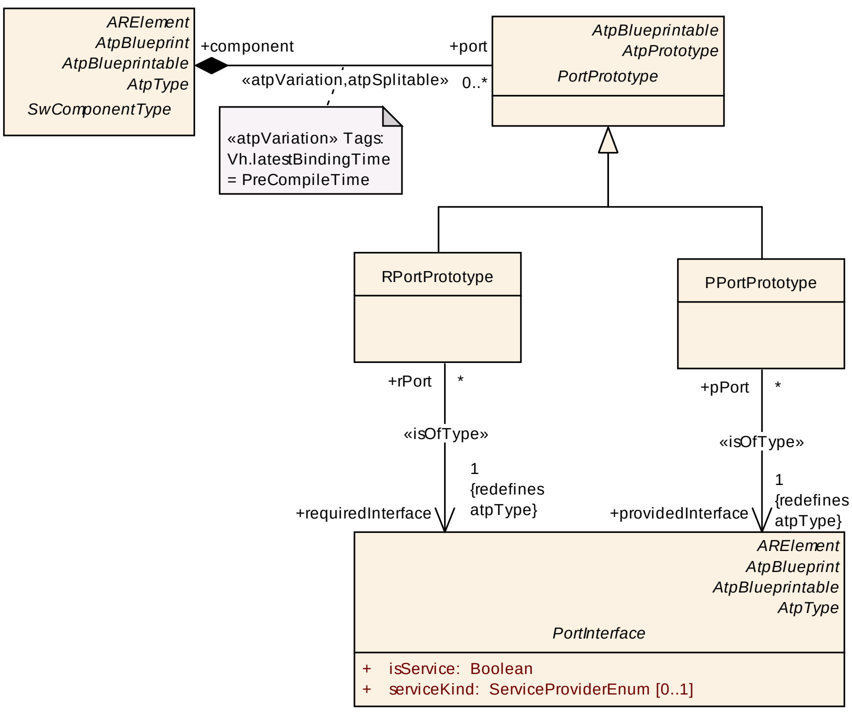

Figure 3.5: Components and Ports

3.3 Composition

[TPS_SWCT_1032] **`CompositionSwComponentType`** ⌈ The purpose of an AUTOSAR `CompositionSwComponentType` is to allow the encapsulation of specific functionality by aggregating existing software-components. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

[TPS_SWCT_1033] **Nested definition of `CompositionSwComponentTypes`** ⌈ Since a `CompositionSwComponentType` is also a `SwComponentType`, it again may be aggregated in further `CompositionSwComponentTypes`. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

This recursive relation is formally expressed in Figure 3.6.

It is important to understand that while compositions allow for (sub-) system abstraction, they are solely an *architectural element for the implementation of model scalability*. They simply group existing software-components and thereby take away complexity when viewing or designing logical software architecture.

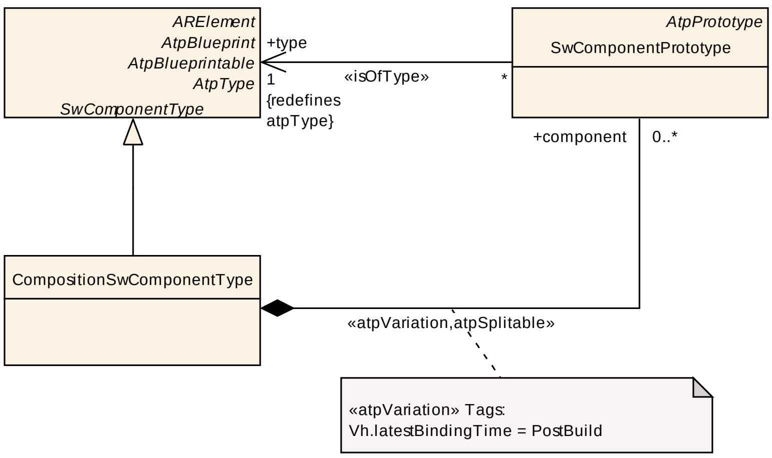

Figure 3.6: The recursive relation of software-components and compositions

Therefore, the definition of `CompositionSwComponentTypes` has no effect on how software-components interact with the Virtual Functional Bus (VFB). `CompositionSwComponentTypes` do not add any new functionality to what is already provided by the software-components they aggregate.

[TPS_SWCT_1034] **`CompositionSwComponentTypes` do not have any binary footprint** ⌈ As the main consequence, `CompositionSwComponentTypes` do not have any binary footprint in the ECU software. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

[TPS_SWCT_1035] **`CompositionSwComponentType` aggregates `SwComponentPrototypes`** ⌈ In terms of the AUTOSAR meta-model, a composition of software-components realized by the meta-class `CompositionSwComponentType` aggregates `SwComponentPrototype`s which in turn are typed by a `SwComponentType`. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

Please note that a `CompositionSwComponentType` is also a `SwComponentType`.

Class	**CompositionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A CompositionSwComponentType aggregates SwComponentPrototypes (that in turn are typed by SwComponentTypes) as well as SwConnectors for primarily connecting SwComponentPrototypes among each others and towards the surface of the CompositionSwComponentType. By this means hierarchical structures of software-components can be created. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
component	SwComponentP rototype	*	aggr	The instantiated components that are part of this composition. The aggregation of SwComponentPrototype is subject to variability with the purpose to support the conditional existence of a SwComponentPrototype. Please be aware; if the conditional existence of SwComponentPrototypes is resolved postbuild the deselected SwComponentPrototypes are still contained in the ECUs build but the instances are inactive in in that they are not scheduled by the RTE. The aggregation is marked as atpSplitable in order to allow the addition of service components to the ECU extract during the ECU integration. The use case for having 0 components owned by the CompositionSwComponentType could be to deliver an empty CompositionSwComponentType to e.g. a supplier for filling the internal structure. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PostBuild atp.Splitkey=shortName, variationPoint.shortLabel

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
connector	SwConnector	*	aggr	SwConnectors have the principal ability to establish a connection among PortPrototypes. They can have many roles in the context of a CompositionSwComponentType. Details are refined by subclasses. The aggregation of SwConnectors is subject to variability with the purpose to support variant data flow. The aggregation is marked as atpSplitable in order to allow the extension of the ECU extract with AssemblySwConnectors between ApplicationSwComponentTypes and ServiceSwComponentTypes during the ECU integration. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PostBuild atp.Splitkey=shortName, variationPoint.shortLabel
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortComSpec. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementers mainly because the RTE generator is not concerned about the CompositionSwComponentTypes. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility.

Attribute	Datatype	Mul.	Kind	Note
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the used ApplicationDataTypes in PortInterfaces. Background: when developing subsystems it may happen that ApplicationDataTypes are used on the surface of CompositionSwComponentTypes. In this case it would be reasonable to be able to also provide the intended mapping to the ImplementationDataTypes. However, this mapping shall be informal and not technically binding for the implementers mainly because the RTE generator is not concerned about the CompositionSwComponentTypes. Rationale: if the mapping of ApplicationDataTypes on the delegated and inner PortPrototype matches then the mapping to ImplementationDataTypes is not impacting compatibility.

Table 3.8: CompositionSwComponentType

Class	**SwComponentPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	Role of a software component within a composition.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
type	SwComponentType	1	tref	Type of the instance. **Stereotypes:** isOfType

Table 3.9: SwComponentPrototype

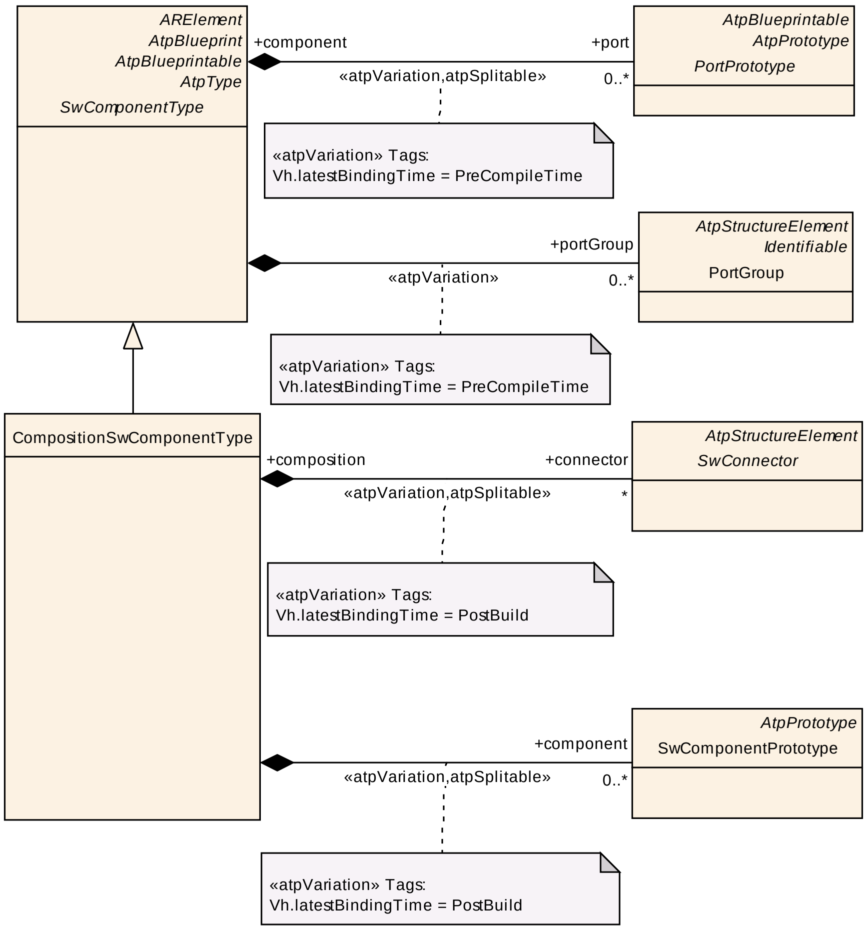

Figure 3.7: Composition and the meta-classes aggregated

[TPS_SWCT_1036] `SwComponentPrototype` implements a specific role ⌈ Therefore, a `SwComponentPrototype` implements the usage of a `SwComponentType` in a specific *role*. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

[TPS_SWCT_1037] arbitrary numbers of `SwComponentPrototypes` can be created ⌈ In general, arbitrary numbers of `SwComponentPrototype`s that refer to specific `SwComponentType`s can be created. ⌋*(RS_SWCT_0190, RS_SWCT_2000, RS_SWCT_2020, RS_SWCT_3000)*

[TPS_SWCT_1079] `SwConnector` ⌈ Note that `CompositionSwComponentType` also aggregates the abstract meta-class `SwConnector` for connection the `SwComponentPrototype`s contained among each others (see Figure 3.7). ⌋*(RS_SWCT_3130)*

Example: a `SwComponentPrototype` "LeftDoorControl" fulfills the role of implementing the `SwComponentType` "DoorControl" for the left door of a vehicle while the `SwComponentPrototype` "RightDoorControl" fulfills the role of the `SwComponentType` "DoorControl" for the right door.

[TPS_SWCT_1080] Delegation ports ⌈ Note that being a `SwComponentType`, a `CompositionSwComponentType` also exposes `PortPrototype`s to the outside world. However, the PortPrototypes are only delegated and do not play the same role as PortPrototypes attached to AtomicSwComponentTypes. ⌋*(RS_SWCT_3130)*

[TPS_SWCT_1081] Implications of being a delegation port ⌈ Being a PortPrototype attached to a CompositionSwComponentType has the following implications:

- The delegation has to follow the rules defined in chapter 6.
- By creating PortPrototypes on the surface of a specific CompositionSwComponentType it is explicitly decided whether or not the contents of an "inner" port contained in the CompositionSwComponentType is exposed to the outside world.

⌋*(RS_SWCT_3130)*

Please note that the semantics of the delegation of PortPrototypes are similar to encapsulation mechanisms like public and private members in object-oriented programming languages.

CompositionSwComponentTypes contain two kinds of SwConnectors:

- **[TPS_SWCT_1082] AssemblySwConnector** ⌈ AssemblySwConnectors interconnect PortPrototypes of SwComponentPrototypes that are part of the CompositionSwComponentType. ⌋*(RS_SWCT_3130)*
- **[TPS_SWCT_1083] DelegationSwConnector** ⌈ DelegationSwConnectors connect from "inner" PortPrototypes to delegated "outer" PortPrototypes. ⌋*(RS_SWCT_3130)*

  [constr_1032] DelegationSwConnector can only connect PortPrototypes of the same kind ⌈ A DelegationSwConnector can only connect PortPrototypes of the same kind, i.e. PPortPrototype to PPortPrototype and RPortPrototype to RPortPrototype. ⌋

  [TPS_SWCT_1084] Outer PortPrototype is referenced by multiple DelegationSwConnectors ⌈ In the case that an outer PortPrototype is referenced by multiple DelegationSwConnectors the semantic is the multiplication of the AssemblySwConnectors referencing the outer PortPrototypes.⌋*(RS_SWCT_3130)*

[constr_1086] SwConnector between two specific PortPrototypes ⌈ Each pair of PortPrototypes can only be connected by one and only one SwConnector ⌋

In other words, it is not supported to create two different SwConnectors that connect the same pair of PortPrototypes.

[constr_1087] AssemblySwConnector inside CompositionSwComponentType ⌈ An AssemblySwConnector can only connect PortPrototypes of SwComponentPrototypes that are owned by the same CompositionSwComponentType ⌋

[constr_1088] `DelegationSwConnector` inside `CompositionSwComponentType` ⌈ A `DelegationSwConnector` can only connect a `PortPrototype` of a `SwComponentPrototype` that is owned by the same `CompositionSwComponentType` that also owns the connected delegation `PortPrototype`. ⌋

[constr_1100] Unconnected `RPortPrototype` typed by a `DataInterface` ⌈ For any element in an unconnected `RPortPrototype` typed by a `DataInterface` there shall be a `requiredComSpec` that defines an `initValue`. ⌋

Class	**SwConnector (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	The base class for connectors between ports. Connectors have to be identifiable to allow references from the system constraint template.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
mapping	PortInterfaceMapping	0..1	ref	Reference to a PortInterfaceMapping specifying the mapping of unequal named PortInterface elements of the two different PortInterfaces typing the two PortPrototypes which are referenced by the ConnectorPrototype.

Table 3.10: SwConnector

Class	**AssemblySwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	AssemblySwConnectors are exclusively used to connect SwComponentPrototypes in the context of a CompositionSwComponentType.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable,SwConnector			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
provider	PPortPrototype	1	iref	Instance of providing port.
requester	RPortPrototype	1	iref	Instance of requiring port.

Table 3.11: AssemblySwConnector

Class	≪`atpStructureElement`≫ **DelegationSwConnector**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Composition			
Note	A delegation connector delegates one inner PortPrototype (a port of a component that is used inside the composition) to a outer PortPrototype of compatible type that belongs directly to the composition (a port that is owned by the composition).			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable,SwConnector			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
innerPort	PortPrototype	1	iref	The port that belongs to the ComponentPrototype in the composition
outerPort	PortPrototype	1	ref	The port that is located on the outside of the CompositionType

Table 3.12: DelegationSwConnector

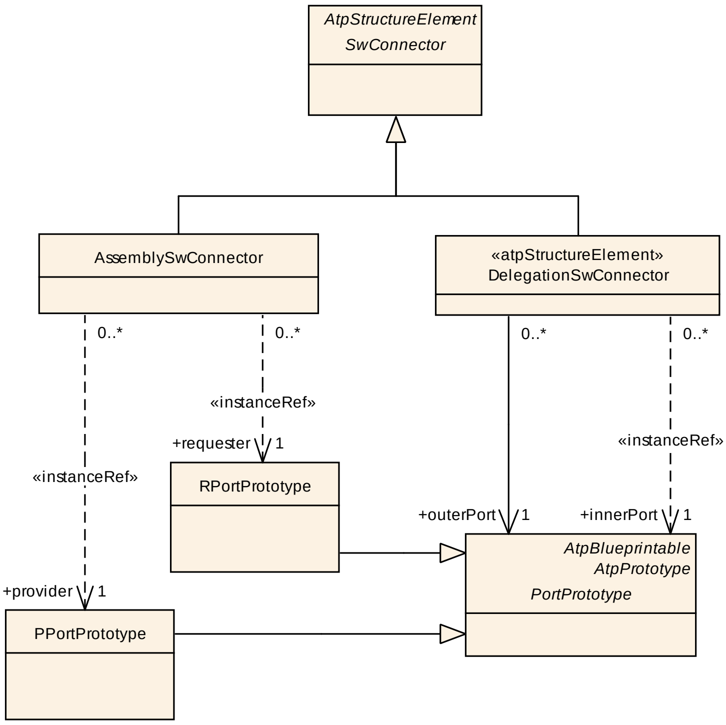

Figure 3.8: Connectors

One implication of the concept of CompositionSwComponentType is that the application software of an entire vehicle eventually is represented by one CompositionSwComponentType. This so-called top-level composition has a special role in the context of the AUTOSAR System Template [11].

However, please note note that a top-level composition might have (unconnected) PortPrototypes in order to allow for reuse as part of another system.

[constr_1035] Recursive definition of CompositionSwComponentType ⌈ The recursive definition of a CompositionSwComponentType that eventually contains a SwComponentPrototype typed by the same CompositionSwComponentType shall not be feasible. ⌋

3.4 Port Interface

[TPS_SWCT_1025] The role of PortPrototypes in the AUTOSAR architecture ⌈ A PortPrototype mainly contributes the functionality of being a connection point to the AUTOSAR concept. The details, i.e. what kind of information is actually transported between two PortPrototypes is defined by the PortInterface. ⌋*(RS_SWCT_0010, RS_SWCT_0080, RS_SWCT_0110, RS_SWCT_2030, RS_SWCT_3010)*

[TPS_SWCT_1026] The role of PortInterfaces in the AUTOSAR architecture ⌈ PortInterfaces (see Figure 3.10) are used to support a design-by-contract work flow, i.e. they provide means to formally verify structural and dynamic compatibility between software-components. ⌋*(RS_SWCT_0010, RS_SWCT_0080, RS_SWCT_0110, RS_SWCT_2030, RS_SWCT_3010)*

In other words: `PortInterface`s represent a pivotal point in the AUTOSAR concept.

Please note that a `PortInterface` creates a name space for the information contained. This allows for defining the details of a specific `PortInterface` without having to care for possible side-effects on other `PortInterface`s. Again, this property of the AUTOSAR concept directly supports re-usability.

[TPS_SWCT_1027] Different flavors of `PortInterfaces` ⌈ Within the AUTOSAR concept, different flavors of `PortInterface`s are defined:

- `SenderReceiverInterface`,
- `NvDataInterface`,
- `ParameterInterface`.
- `ModeSwitchInterface`,
- `ClientServerInterface`, and the
- `TriggerInterface`,

⌋*(RS_SWCT_0010, RS_SWCT_0080, RS_SWCT_0110, RS_SWCT_2030)*

[TPS_SWCT_1069] `DataInterface` is defined as abstract base class ⌈ Please note that the conceptual relationship of `SenderReceiverInterface`, `NvDataInterface`, and `ParameterInterface` is expressed by the abstract base class `DataInterface`. ⌋*(RS_SWCT_0010, RS_SWCT_0080, RS_SWCT_0110, RS_SWCT_3010)*

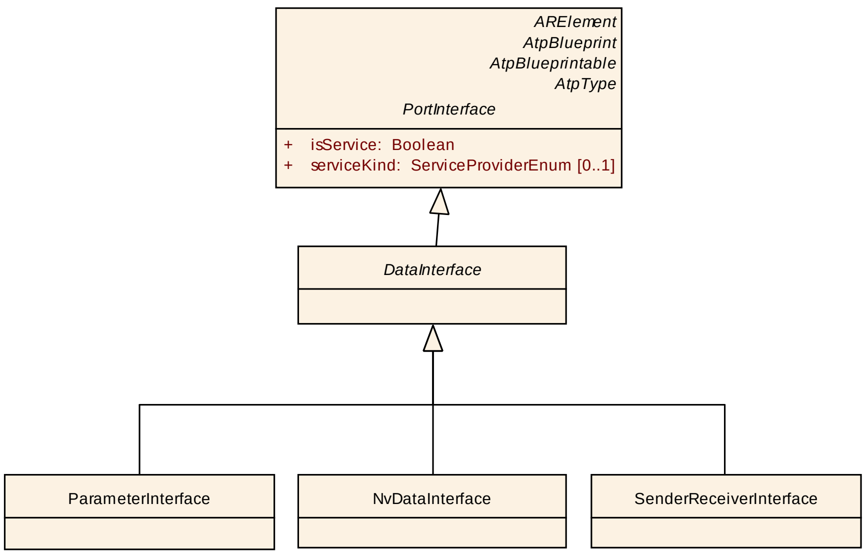

Figure 3.9: `DataInterface` as an abstract base class

Please find more details about the specialization of the `PortInterface` concept in chapter 4.2.3 and 4.2.2.

Class	**PortInterface (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
isService	Boolean	1	attr	This flag is set if the PortInterface is to be used for communication between an • ApplicationSwComponentType or • ServiceProxySwComponentType or • SensorActuatorSwComponentType or • ComplexDeviceDriverSwComponentType or • EcuAbstractionSwComponentType and a ServiceSwComponentType (namely an AUTOSAR Service) located on the same ECU. Otherwise the flag is not set.
serviceKin d	ServiceProvider Enum	0..1	attr	This attribute provides further details about the nature of the applied service.

Table 3.13: PortInterface

Class	**DataInterface (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	The purpose of this meta-class is to act as an abstract base class for subclasses that share the semantics of being concerned about data (as opposed to e.g. operations).			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,PortInterface,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 3.14: DataInterface

[TPS_SWCT_1070] `PortInterface` acts as a *type* for a `PortPrototype` ⌈ From an abstract point of view, a `PortInterface` acts as a *type* for a `PortPrototype`. This means in particular that several `PortPrototype`s can be typed by the same `PortInterface`. ⌋*(RS_SWCT_0010, RS_SWCT_0080, RS_SWCT_0110, RS_SWCT_3010)*

Of course, this aspect facilitates the creation of valid connections between software-components dramatically. By using a specific `PortInterface` for typing particular `PortPrototype`s the latter are eligible for being connected to each other by definition.

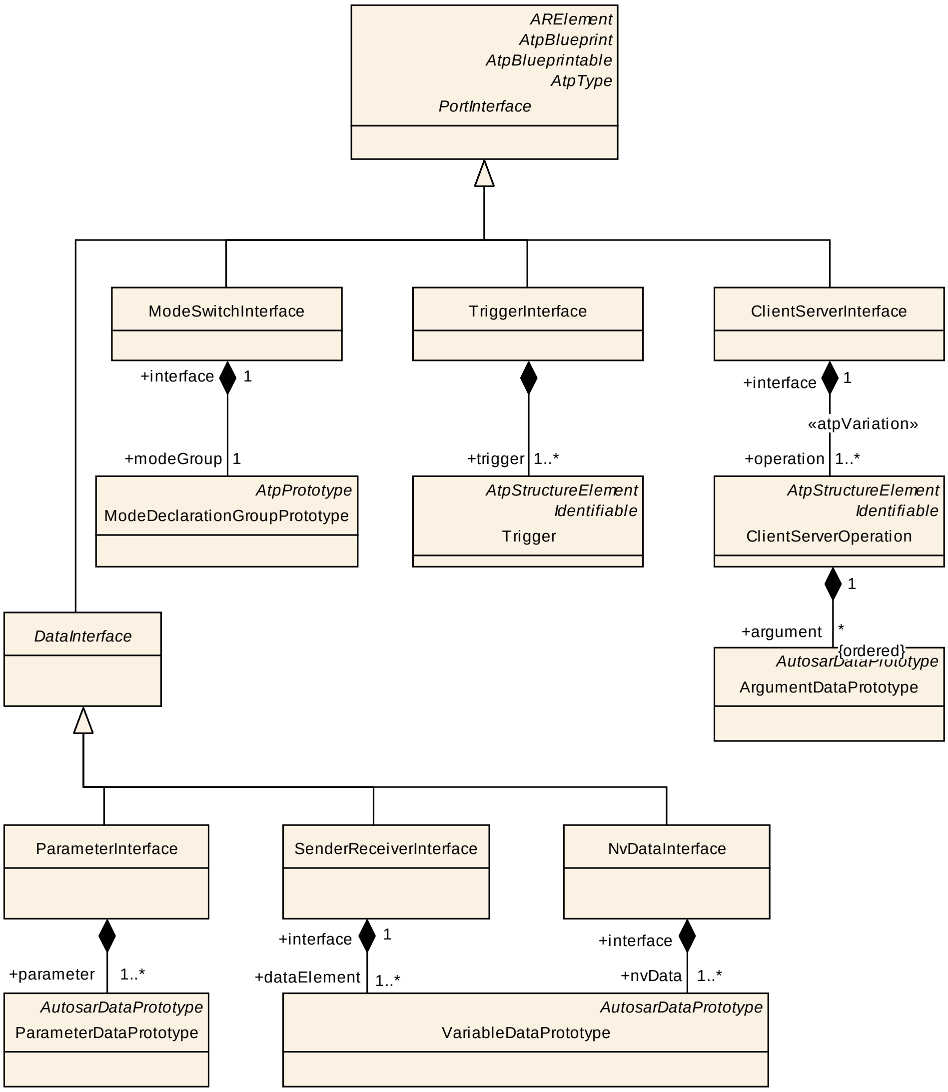

Figure 3.10: `PortInterface`s in the AUTOSAR meta-model

However, the creation of a valid connection does not need to be based on the usage of identical `PortInterface`s. It is also possible to use different, but *compatible* `PortInterface`s. The details about compatibility of `PortInterface`s are described in chapter 6.

[constr_1036] Connect kinds of `PortInterfaces` ⌈ It shall not be possible to connect `PortPrototype`s typed by `PortInterface`s of different kinds. Subclasses of `DataInterface` make an exception from this rule and can be used for creating connections to each others. ⌋

For clarification, a connection between a `PortPrototype` typed by a `SenderReceiverInterface` and a `PortPrototype` typed by a `ClientServerInterface` shall not be possible. However, the creation of a connection between a `PortPrototype` typed by a `SenderReceiverInterface` and a `PortPrototype` typed by a `ParameterInterface` is supported.

[constr_1137] Applicability of `ParameterInterface` ⌈ A `PPortPrototype` typed by a `ParameterInterface` can **only** be owned by a `ParameterSwComponentType`. ⌋

Please note that `PortInterface`s also play an important role in the context of defining so-called AUTOSAR services. In particular, by means of the attribute `isService` a `PortInterface` can define whether or not it is supposed to be used in the context of an AUTOSAR service and in addition to this it may define (by means of the attribute `serviceKind`) what kind of service is intended.

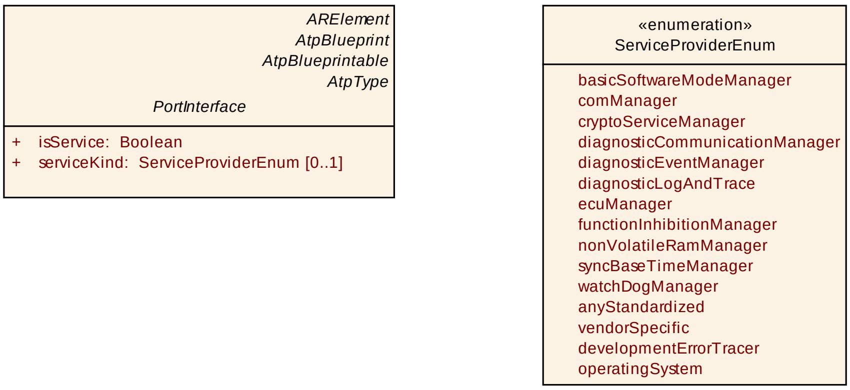

Figure 3.11: `PortInterface`s and AUTOSAR services

The information contained in `serviceKind` can be used in various ways. The primary intent is to distinguish between the usage of standardized AUTOSAR services from the usage of a vendor-specific service. this information may have an impact on the development- and build process of software-components that use the `PortInterface`.

In addition, it is also possible to use the information contained in `serviceKind` for filtering the presentation of an AUTOSAR model in an AUTOSAR authoring tool and e.g. display the nature of the service `PortPrototype`s independently of the content of the corresponding `PortInterface`.

[TPS_SWCT_1003] Inconsistencies regarding the value of `serviceKind` and the actual implementation of the `PortInterface` ⌈ In case of inconsistencies between the value of `serviceKind` and the actual implementation of the `PortInterface` the implementation of the `PortInterface` wins over the attribute's value (which for the intended purpose shall be considered as an annotation rather than a semantically binding information). ⌋

[TPS_SWCT_1004] Default value if `serviceKind` is not defined ⌈ if the attribute `serviceKind` is not defined in the context of a specific `PortInterface` the default value `anyStandardized` shall be assumed. ⌋

[constr_1174] `PortInterface`s used in the context of `CompositionSwComponentType`s cannot refer to AUTOSAR services ⌈ `CompositionSwComponentType`s shall not own `PortPrototype`s typed by `PortInterface`s where the attribute `isService` is set to `TRUE`. ⌋

Enumeration	**ServiceProviderEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This represents a list of possible service providers
Literal	***Description***
anyStandardized	This value means that the specific nature is either unknown or it is not important for the given purpose. This is also the default value for any attribute of type ServiceProviderEnum
basicSoftwareMode Manager	The service relates to the Basic Software Mode Manager (BswM)
comManager	The service relates to the COM Manager (ComM).
cryptoService Manager	The service relates to the Crypto Service Manager (CsM).
development ErrorTracer	The service relates to the Development Error Tracer (DET).
diagnostic CommunicationManager	The service relates to the Diagnostic Communication Manager (DCM).
diagnostic EventManager	The service relates to the Diagnostic Event Manager (DEM).
diagnostic LogAndTrace	The service relates to the Diagnostic Log and Trace (DLT).
ecuManager	The service relates to the ECU Manager (EcuM).
function Inhibition Manager	The service relates to the Function Inhibition Manager (FIM).
nonVolatile RamManager	The service relates to the Non-Volatile RAM Manager (NvM).
operating System	The service relates to the Operating System (OS).
syncBase TimeManager	The service relates to the Sync Time Base Manager (StbM).
vendorSpecific	This value denotes a vendor-specific service.
watchDog Manager	The service relates to the Watchdog Manager (WdgM).

Table 3.15: ServiceProviderEnum

[TPS_SWCT_1005] Usage of `SwcServiceDependency`s for vendor-specific services ⌈ `SwcServiceDependency`s can also be used for vendor-specific services. In this case the `SwcServiceDependency` shall not contain any of the standardized `ServiceNeeds`. ⌋

Please find more details about the relation of `PortInterface`s to AUTOSAR services in chapter 11.

4 Details: Software Components, Ports, and Interfaces

4.1 Introduction

The specification of the Virtual Functional Bus (VFB) [3] explains the main communication paradigms for communication among software-components: *client/server* for operation-based communication, and *sender/receiver* for data-based communication.

The nature of the two communication paradigms is quite different, and so is the modeling of `SenderReceiverInterface`s and `ClientServerInterface`s and their related meta-classes.

`PortInterface`s are limited to the description of the static structure of the exchanged information; the dynamic attributes (please refer to chapter 4.5) relevant for communication are attached to `PortPrototype`s.

4.2 Port Interface Details

4.2.1 Introduction

The usage of value encodings (for more information please refer to section 5.2.6) is limited within the context of `PortInterface`s.

[constr_1045] Supported value encodings for `SwBaseType` in the context of `PortInterfaces` ⌈ The supported value encodings for the usage within a `PortInterface` are:

- `2C`: Two's complement
- `IEEE754`: floating point numbers
- `ISO-8859-1`: ASCII-Strings
- `ISO-8859-2`: ASCII-Strings
- `WINDOWS-1252`: ASCII-Strings
- `UTF-8`: UCS Transformation Format 8
- `UCS-2`: Universal Character Set 2
- `NONE`: Unsigned Integer
- `BOOLEAN`: This represents an integer to be interpreted as boolean.

⌋

[constr_1046] Applicability of [constr_1045] ⌈ [constr_1045] applies **only** if the value of the attribute `isService` is set to `FALSE`. ⌋

4.2.2 Sender Receiver Communication

[TPS_SWCT_1114] **SenderReceiverInterface** ⌈ SenderReceiverInterfaces allow for the specification of the typically asynchronous communication pattern where a sender provides data that is required by one or more receivers. While the actual communication takes place via the respective PortPrototypes, a SenderReceiverInterface allows for formally describing what kind of information is sent and received. ⌋

Class	**SenderReceiverInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A sender/receiver interface declares a number of data elements to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataElement	VariableDataPrototype	1..*	aggr	The data elements of this SenderReceiverInterface.
invalidationPolicy	InvalidationPolicy	*	aggr	InvalidationPolicy for a particular dataElement

Table 4.1: SenderReceiverInterface

Class	**InvalidationPolicy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies whether the component can actively invalidate a particular dataElement. If no invalidationPolicy points to a dataElement this is considered to yield the identical result as if the handleInvalid attribute was set to dontInvalidate.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataElement	VariableDataPrototype	1	ref	Reference to the dataElement for which the InvalidationPolicy applies.
handleInvalid	HandleInvalidEnum	0..1	attr	This attribute defines the action performed upon a reception timeout violation.

Table 4.2: InvalidationPolicy

Enumeration	**HandleInvalidEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Strategies of handling the reception of invalidValue.
Literal	*Description*
dontInvalidate	Invalidation is switched off.
keep	The application software is supposed to handle signal invalidation on RTE API level either by DataReceiveErrorEvent or check of error code on read access.

replace	Replace a received invalidValue. The replacement value is specified by the initValue.

Table 4.3: HandleInvalidEnum

A `SenderReceiverInterface` focuses on the description of information items represented by `VariableDataPrototypes` (see section 5.3).

A `VariableDataPrototype` aggregated in the role of `dataElement` represents an atomic[1] piece of information transmitted among `PortPrototypes` typed by a `SenderReceiverInterface`.

[TPS_SWCT_1115] `InvalidationPolicy` ⌈ A `InvalidationPolicy` specifies whether the sending component can actively invalidate a particular `dataElement` and which strategy of handling the reception of `invalidValue` on the receiver side shall be implemented. ⌋

Further information about the `invalidValue` is provided in chapter 5.4.2

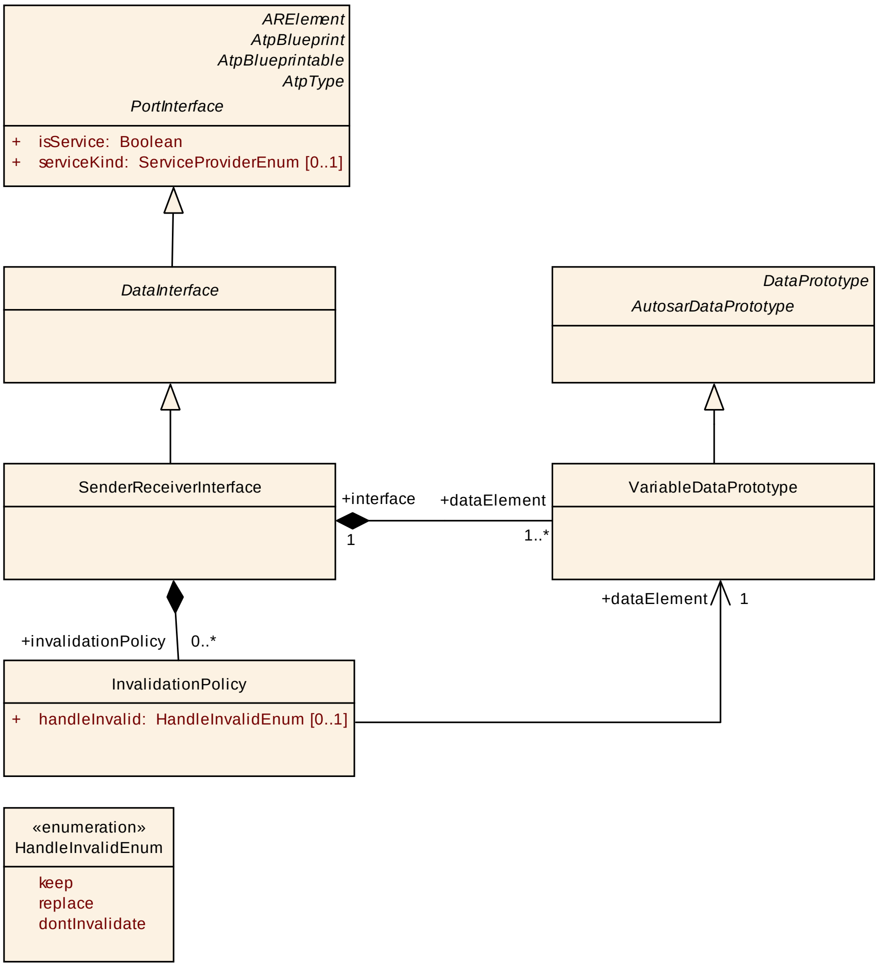

Figure 4.1: `dataElements` of a `SenderReceiverInterface`

[1]Note that the term "atomic" does not have any implication on the implementation on a concrete computing platform

Note that a `SenderReceiverInterface` provides a name space for the definition of `VariableDataPrototype`s. In terms of the AUTOSAR meta-model this aspect is indicated by the inheritance relation to `DataPrototype` (which in turn inherits from `Identifiable`). Please find more information on the creation of name spaces in [13].

[TPS_SWCT_1116] `swImplPolicy` ⌈The `swImplPolicy` (see section 5.4) indicates the way how a `VariableDataPrototype` shall be processed at the receiver's side. If set to `queued` the semantics is that the corresponding `VariableDataPrototype` needs to be added to a *queue* (or in other words: a FIFO data structure) from which it is later consumed by the actual receiver software-component. ⌋

[TPS_SWCT_1176] last-is-best semantics for sender-receiver communication ⌈ If `swImplPolicy` is set to any other valid value of `SwImplPolicyEnum` then *last is best* semantics applies. ⌋

Please note that the definition of `VariableDataPrototype` may possibly come very close to the reader's idea of a *signal*. However, different kinds of signals have a specific meaning in the AUTOSAR concept, especially in the context of the AUTOSAR System Template [11].

[TPS_SWCT_1117] Communication patterns for sender-receiver communication ⌈`PortPrototype`s typed by a `SenderReceiverInterface` may be connected to establish a 1:n (i.e. one sender, multiple receivers) communication relationship. It is also possible to establish a n:1 (i.e. many senders, one receiver) communication pattern. ⌋

[constr_1033] Communication scenarios for sender/receiver communication ⌈ For sender/receiver communication, it is not allowed to create a communication scenario where n sender are connected to m receivers where m and n are **both** greater than 1. ⌋

4.2.3 Client Server Communication

The underlying semantics of a client/server communication is that a client may initiate the execution of an operation by a server that supports the operation. The server executes the operation and immediately provides the client with the result (synchronous operation call) or else the client checks for the completion of the operation by itself (asynchronous operation call).

[constr_1037] Client may not connect to multiple servers ⌈A client may not connect to multiple servers such that an operation call would be handled by more than one server. ⌋

4.2.3.1 Client Server Interface

A ClientServerInterface therefore to some extent is a counterpart to the SenderReceiverInterface[2].

Instead of defining pieces of information to be transferred among software-components, a ClientServerInterface defines a collection of ClientServerOperations.

Class	**ClientServerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A client/server interface declares a number of operations that can be invoked on a server by a client. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
operation	ClientServerOperation	1..*	aggr	ClientServerOperation(s) of this ClientServerInterface. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivationTime
possibleError	ApplicationError	*	aggr	Application errors that are defined as part of this interface.

Table 4.4: ClientServerInterface

[2]However, different connection patterns apply, see [constr_1037]

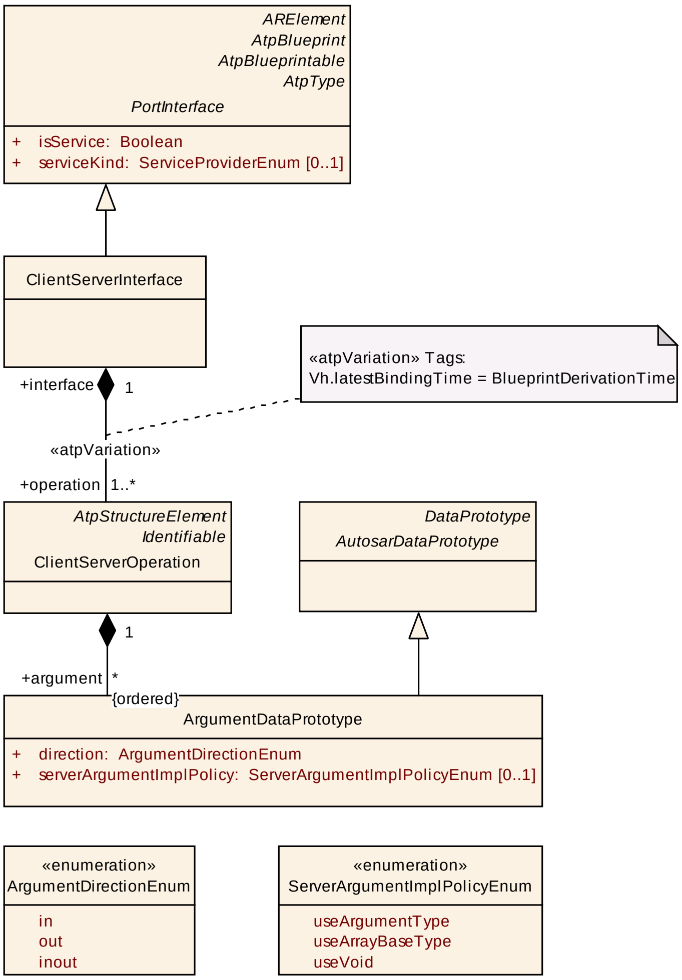

Figure 4.2: `Operations` of a `ClientServerInterface`

[TPS_SWCT_1118] **`ClientServerInterface`** ⌈ As depicted in Figure 4.2, a `ClientServerInterface` is composed of `ClientServerOperation`s, i.e. a `ClientServerOperation` cannot be reused in the context of a different `ClientServerInterface`⌋

[TPS_SWCT_1106] **`ClientServerOperation`** ⌈ A `ClientServerOperation` consists of 0..* `ArgumentDataPrototype`s. The latter may be

- passed to the operation (i.e. the direction is “in”)
- passed to and returned from the operation (i.e. the direction is “inout”)
- returned from the operation (i.e. the direction is “out”)

The aggregation represents a variation point. ⌋*(RS_SWCT_3141)*

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
argument (ordered)	ArgumentDataPrototype	*	aggr	The argument of this operation.
possibleError	ApplicationError	*	ref	Possible errors that may by raised by referring operation.

Table 4.5: ClientServerOperation

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is associated with a particular operation.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
direction	ArgumentDirectionEnum	1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantic as if the attribute is set to useArgumentType

Table 4.6: ArgumentDataPrototype

[TPS_SWCT_1119] Direction of `ArgumentDataPrototypes` ⌈ To cover these cases, `ArgumentDataPrototype` defines an attribute `direction`, possible values are `in` (pass to operation), `out` (return from operation), and `inout` (pass to and return from operation). ⌋

In many common programming languages (like *C*), an operation is yet another data type. This makes it for example possible to pass a reference to an operation as an argument to another operation.

This is *not* allowed in the AUTOSAR concept: it is not possible to pass a reference to a `ClientServerOperation` as an `ArgumentDataPrototype` in another `ClientServerOperation`.

Essentially all `ArgumentDataPrototype`s in a `ClientServerOperation` can be passed (conceptually) by value (from the client to the server and/or from the server to the client depending on the `direction` of the `ArgumentDataPrototype`). Extending the model to allow this causes a huge additional level of complication within the RTE (as the RTE now would need to deal with references to remote objects).

[TPS_SWCT_1120] Client needs to provide ArgumentDataPrototypes ⌈ When the client invokes an operation, it needs to provide a value for each ArgumentDataPrototype that is of direction in or inout. ⌋

[TPS_SWCT_1121] Pass correct data type ⌈ The value passed to an ArgumentDataPrototype of direction in or inout needs to be of the corresponding Datatype. ⌋

[TPS_SWCT_1122] Synchronous call of ClientServerOperation ⌈ In the case of synchronous operation call, the client expects to receive a response to the invocation of the operation.

As part of the response, it receives a value (of the correct Datatype) for each ArgumentDataPrototype that is of direction out or inout. ⌋

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 4.7: ArgumentDirectionEnum

Each ClientServerOperation provides a name space for its ArgumentDataPrototypes and therefore has a unique identifier which identifies the operation within the corresponding ClientServerInterface.

The ClientServerOperations have no ordering within a ClientServerInterface (there is no such thing as the "first" operation)[3].

[TPS_SWCT_1123] No default values for ArgumentDataPrototypes ⌈ It is not possible to define default values for ArgumentDataPrototypes defined in the context of a ClientServerOperation. Default values might lead to complicated mappings to programming languages. ⌋

[3]In different parts of the definition of a ClientServerInterface, a "calling-order" of the ClientServerOperations might be prescribed: the client might be required to use the ClientServerOperations in a certain logical ordering.

However, this ordering has nothing to do with the order in which the ClientServerOperations are listed in the definition of a ClientServerInterface

[TPS_SWCT_1124] Definition of `ArgumentDataPrototype`s within the context of a `ClientServerOperation` is ordered ⌈ In contrast to the unordered relationship of `ClientServerInterface` to `ClientServerOperation`, the definition of `ArgumentDataPrototype`s within the context of a `ClientServerOperation` is ordered, i.e. an `ClientServerOperation` may have a *first* argument[4]. ⌋

Please note that `ArgumentDataPrototype` inherits from `AutosarDataPrototype` and therefore has a reference to a concrete `AutosarDataType`.

The RTE Generator uses the referred `AutosarDataType`s to determine the data types of the arguments dependent from the attribute `serverArgumentImplPolicy`.

Enumeration	**ServerArgumentImplPolicyEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	This defines how the argument type of the servers RunnableEntity is implemented.
Literal	***Description***
useArgument Type	The argument type of the RunnableEntity is derived from the AutosarDataType of the ArgumentPrototype.
useArray BaseType	The argument type of the RunnableEntity is derived from the AutosarDataType of the elements of the array that corresponds to the ArgumentPrototype. This represents the base type of the array in C.
useVoid	The argument type of the RunnableEntity is void.

Table 4.8: ServerArgumentImplPolicyEnum

[TPS_SWCT_1125] `serverArgumentImplPolicy` ⌈ The option `useArrayBaseType` is intended to implement "'Server Runnables"' which are able to handle array typed arguments of different length. In this case the software component does have several Server Ports. At least one argument of the `Operation`s are typed by `AutosarDataType`s of category `ARRAY` but the length of the arrays defined by `maxNumberOfElements` respectively `arraySize` might be different for the individual Server Ports.

All `ClientServerOperation`s in the `PortPrototype`s are triggering the same `RunnableEntity` with `OperationInvokedEvent`s.

If the `serverArgumentImplPolicy` is set to `useArrayBaseType` the RTE Generator does not require the compatibility of `Operation`s for such `ArgumentDataPrototype`s and uses the base type of the array as the arguments data type instead the array data type with a particular length.

[4] Giving the `ArgumentDataPrototype`s of an `ClientServerOperation` both an ordering and a unique identifier might seem redundant.

For example, in the operation "foo(a, b, c)", we can refer to the "second argument" or to "the argument named b". In many common programming languages (like C or Java), only the *ordering* is actually used by the client during the invocation of the server (the client invokes the operation as "foo(1,2,3)" not as "foo(a=1,c=3,b=2)".

In addition, the names of the arguments represent an arbitrary choice made when implementing of the invocation. In *C*, only the data types and ordering of the arguments constitute the signature, *not* the names of the arguments.

The option useVoid is available to implement Server RunnableEntitys which are able to handle arbitrary typed arguments of different length - typically of category STRUCTURE. The design of the software component implementing the server is similar as explained for useArrayBaseType.

If the serverArgumentImplPolicy is set to useVoid the RTE Generator does not require the compatibility of Operations for such ArgumentDataPrototypes and uses void as the arguments data type.

Please note that the server RunnableEntity needs information about the currently used array length respectively structure size by usage of additionally arguments passed by the Client or via PortDefinedArgumentValue. ⌋

Note further that a ClientServerInterface does not define any timing information (how quickly the client expects a response of the server). It does not define how the threading works (if the client for example blocks until the response comes back from the server).

It also does not define explicitly how information is passed between an implementation of the client and the server and the underlying RTE (for example: through "pointers" or "by value").

4.2.3.2 Error Handling in Client/Server Communication

This section describes the handling of errors occurring either within an application software-component or during the communication across the VFB [3]. Errors that are created and consumed by basic software modules are not in scope.

Therefore, errors in the scope of this document are divided into two simple classes:

- infrastructure errors and
- application errors.

A software-component implementation uses RTE API methods to communicate with other software-components. During this communication certain errors can occur as a result of infrastructure faults, like a bus is not working, or an expected data value was not arriving in time.

These errors are listed in the RTE specification [2], as they are an inherent feature of the infrastructure provided by the VFB. Software-components will therefore typically not raise infrastructure errors on their own. Instead, the basic software and the RTE will determine infrastructure faults and communicate the corresponding error codes to the relevant software-components.

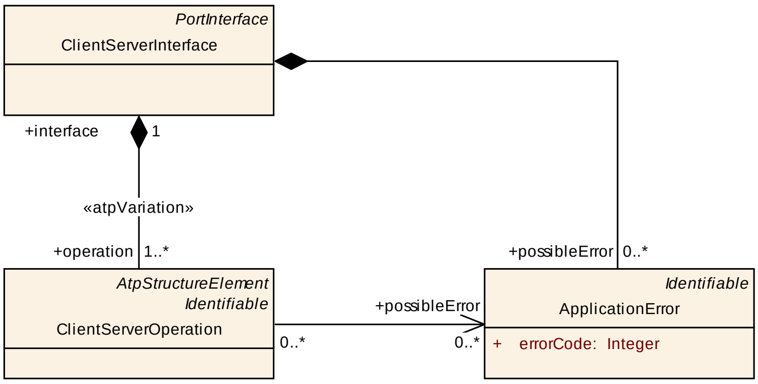

Figure 4.3: Application error meta-model

As the fixed set of infrastructure errors is defined as an implicit part of the VFB, a developer of an AUTOSAR system does not need to explicitly describe these. It is assumed that these might occur at run-time and application developers should take measures to handle them.

Application errors on the other hand are specific to the functionality or information that is described in form of a `PortInterface`. It is not possible to define such errors up front, instead they are defined at design time of a certain `PortInterface`.

In principle, such `ApplicationError`s could be part of all kinds of `PortInterface`s, but as of now, AUTOSAR supports (as depicted by Figure 4.3) `ApplicationError`s only for `ClientServerInterface`s.

[constr_1102] `ApplicationError` in the scope of one `SwComponentType` ⌈ A `SwComponentType` that has `PortPrototype`s typed by different `PortInterface`s with equal `shortName` but conflicting `ApplicationError`s. That is, `ApplicationError`s are considered conflicting if `ApplicationError`s with the same `shortName` do have different `errorCode`s. ⌋

[constr_1108] Value of `ApplicationError.errorCode` ⌈ The value of `ApplicationError.errorCode` shall not exceed the closed interval 1 .. 63. ⌋

By [constr_1108] it is possible to ensure that only the six least significant bits of a return value shall be used for indicating an application error.

Class	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
errorCode	Integer	1	attr	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Table 4.9: ApplicationError

Consequently, ClientServerOperations may be associated with a number of ApplicationErrors they possibly raise. These errors are defined as part of the ClientServerInterface.

[constr_1038] Reference to ApplicationError ⌈ A possibleError referenced by a ClientServerOperation shall be owned by the ClientServerInterface that also owns the ClientServerOperation. ⌋

4.2.4 External Trigger Event Communication

[TPS_SWCT_1196] Semantics of an external trigger event communication ⌈ The underlying semantics of an external trigger event communication is that a trigger source may initiate the execution of RunnableEntitys in the connected trigger sinks. Typically but not necessarily these RunnableEntitys are executed in a sequential order. ⌋

[TPS_SWCT_1197] TriggerInterface ⌈ The TriggerInterface defines a set of Trigger to be communicated between software-components. The Trigger represents a special kind of events at which occurrence the trigger sinks shall react in a particular manner. ⌋

Class	**TriggerInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A trigger interface declares a number of triggers that can be sent by an trigger source. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
trigger	Trigger	1..*	aggr	The Trigger of this trigger interface.

Table 4.10: TriggerInterface

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.
triggerPeriod	MultidimensionalTime	0..1	aggr	Optional definition of a period in case of a periodically (time or angle) driven external trigger.

Table 4.11: Trigger

Class	**MultidimensionalTime**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::MultidimensionalTime			
Note	This is used to specify a multidimensional time value based on ASAM CSE codes. It is specified by a code which defined the basis of the time and a scaling factor which finally determines the time value. If for example the the cseCode is 100 and the cseCodeFactor is 360, it represents 360 angular degrees. If the cseCode is 2 and the cseCodeFactor is 50 it represents 50 microseconds			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
cseCode	CseCodeType	1	attr	Specifies the time base by means of CSE codes.
cseCodeFactor	Integer	1	attr	The scaling factor for the time value based on the specified CSE code.

Table 4.12: MultidimensionalTime

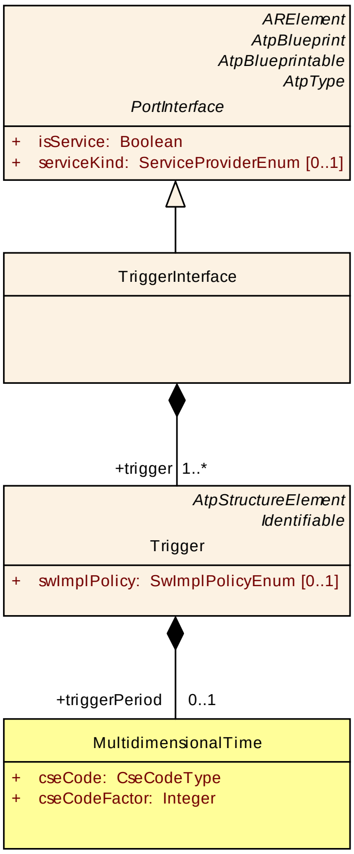

Figure 4.4: `Trigger` of a `TriggerInterface`

As illustrated in Figure 4.4, a `TriggerInterface` is composed of `Trigger`.

[TPS_SWCT_1198] Period for periodic triggering ⌈ A `Trigger` can optionally define a period for periodic triggering. It is expressed via the meta-class `MultidimensionalTime` in terms of time or angle. Note that the main use case for this is to specify the properties if the trigger is coming from the Basic Software e.g. from a Complex Driver, it is not used as an input for the RTE generator. ⌋

Apart from this, a TriggerInterface does not define any timing information (e.g. how quickly the source expects a reaction of the sinks). This is property of the timing information in the templates.

[constr_1104] Trigger sink and trigger source ⌈ An RPortPrototype typed by a TriggerInterface shall not be referenced by more than one SwConnectors that are in turn referencing PPortPrototypes typed by TriggerInterfaces that contain Triggers with the same shortName. ⌋

[constr_1104] boils down to the requirement that trigger communication shall not be implemented in a n:1 scenario.

[TPS_SWCT_1199] Queued processing of Triggers ⌈ It may happen that at least tentatively a Trigger source fires Triggers faster than they can be processed on the side of the Trigger sink. To support this use case it is possible to process trigger event communication in a queued manner.

In this case the Triggers are added to a queue from where the foremost trigger is dequeued and processed when the processing of the current Trigger is done. Please note that the queue size is **not** subject to definition in the scope of this document. The actual queue size is defined during the process of RTE configuration.

The specification of whether or not a Trigger is subject to queued processing is controlled by the attribute Trigger.swImplPolicy. ⌋

[constr_1169] Allowed values for Trigger.swImplPolicy ⌈ The **only** allowed values for the attribute Trigger.swImplPolicy are either STANDARD (in which case the Trigger processing does not use a queue) or QUEUED (in which case the processing of Triggers positively uses a queue). ⌋

4.2.5 Communication of Modes

There are two distinctive use cases for the communication of modes via ports:

1. An actual mode transition can be communicated from a mode manager component to its client components to enforce a mode switch.
2. A request for a mode transition can be communicated from any component to a mode manager.

[TPS_SWCT_1087] Propagation of mode information ⌈ For communicating a mode switch (i.e. the first use case), the Software-Component Template describes the concept of the communication of ModeDeclarationGroupPrototypes similar to the communication of VariableDataPrototypes but is uses a special type of PortInterface: the collections of ModeDeclarations that are required or provided by a SwComponentType are defined (as depicted in Figure 4.5) by means of Mod-

`eSwitchInterface`s used to type the `PortPrototype`s owned by the `SwCompo-nentType`. ⌋*(RS_SWCT_3203)*

Due to the strong interaction with the RTE for handling the mode switches, this first use case does not allow communication across ECU boundaries:

[constr_4000] Local communication of mode switches ⌈ Ports with `ModeSwitch-Interface`s cannot be connected across ECU boundaries. ⌋

[constr_2049] Different `ModeDeclarationGroup`s shall have different `short-Name`s. ⌈ A software component is not allowed to type multiple `PortPrototype`s with `ModeSwitchInterface`s where the contained `ModeDeclarationGroupPro-totype`s are referencing `ModeDeclarationGroup`s with identical `shortName`s but different `ModeDeclaration`s. ⌋

The rationale is to avoid conflicts in generated RTE files.

For instance:

Two `ModeDeclarationGroup`s with identical `shortName` "'Foo"' are defined.

`ModeDeclarationGroup` "'Foo"'
contains the `ModeDeclaration`s "'X"', "'Y"', "'Z"'

`ModeDeclarationGroup` "'Foo*"'
contains `ModeDeclaration`s "'W"', "'X"', "'Y"', "'Z"'

In this case a software component is only allowed to use either "'Foo"' or "'Foo*"'

Class	**ModeSwitchInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A mode switch interface declares a ModeDeclarationGroupPrototype to be sent and received. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
modeGrou p	ModeDeclaratio nGroupPrototyp e	1	aggr	The ModeDeclarationGroupPrototype of this mode interface.

Table 4.13: ModeSwitchInterface

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***

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 4.14: ModeDeclarationGroupPrototype

Please note that by aggregating `SwCalibrationAccessEnum` in the role `swCalibrationAccess` `ModeDeclarationGroupPrototype` gains the ability to become measurable. This implies the following constraint:

[constr_1172] Allowed values of `SwCalibrationAccessEnum` for `ModeDeclarationGroupPrototype` ⌈ The only allowed values of `swCalibrationAccess` aggregated by `ModeDeclarationGroupPrototype` are `notAccessible` and `readOnly`. ⌋

Enumeration	**SwCalibrationAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Literal	***Description***
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file.
readOnly	The element will only appear as read-only in an ASAP file.
readWrite	The element will appear in the ASAP file with both read and write access.

Table 4.15: SwCalibrationAccessEnum

[TPS_SWCT_1200] `ModeDeclarationGroupPrototype` per `ModeSwitchInterface` ⌈ The multiplicity of the aggregation of `ModeDeclarationGroupPrototype` to `ModeSwitchInterface` is pragmatically limited to 1. ⌋*(RS_SWCT_3203)*

Admittedly, there would be no technical restriction to support a 0..* multiplicity but on the other hand it does not seem as if any reasonable use case for such a scenario exists.

If somehow a `SwComponentType` would have to consider two or even more `ModeDeclarationGroupPrototype`s it is very likely that these would be part of different `ModeSwitchInterface`s.

The containment of a `ModeDeclarationGroupPrototype` in a `ModeSwitchInterface` allows for explicitly defining `SwConnector`s which communicate between `SwComponentPrototype`s and to define service interfaces for communication with `ServiceSwComponentType`s. Due to the compatibility rules of `PortInterface`s (see chapter 6) each `SwComponentType` can rely on the availability of required mode activations.

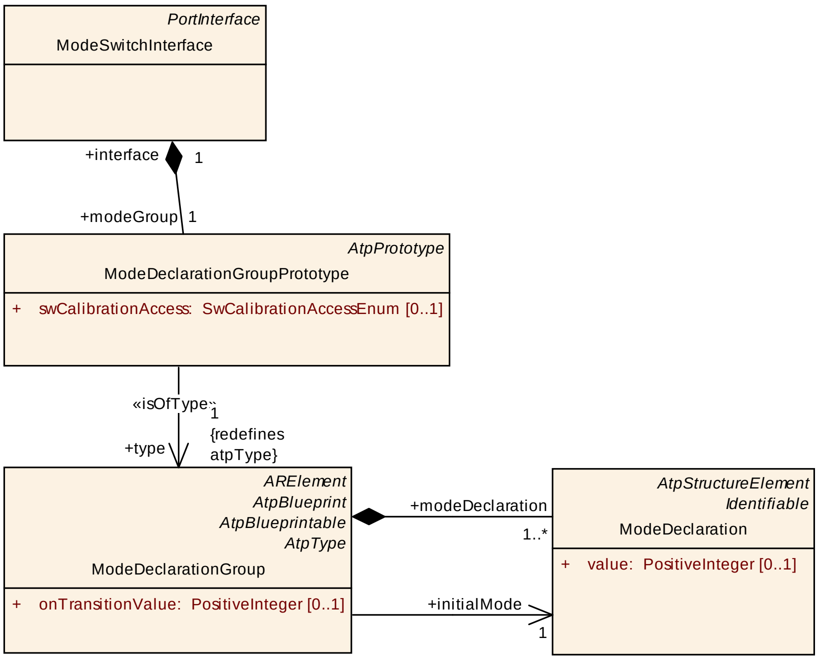

Figure 4.5: Mode Switch Interface

Please note that each `SwComponentType` can define (via their `PortPrototype`s and `ModeSwitchInterface`s) a list of required and provided `ModeDeclarationGroupPrototype`s.

[TPS_SWCT_1201] `CompositionSwComponentType` requires and provides the modes that are required or provided by its contained `SwComponentPrototypes` ⌈ Eventually, a `CompositionSwComponentType` requires and provides the modes that are required or provided by its contained `SwComponentPrototype`s. The delegation of these modes from `SwComponentPrototype`s to the enclosing `CompositionSwComponentType` is explicitly described by `DelegationSwConnector`s. ⌋*(RS_SWCT_3202, RS_SWCT_3203)*

The formal description of a software-component does not make any assumptions about the semantics of the required and provided `ModeDeclarationGroupPrototype`s. It just requires and provides the `ModeDeclarationGroupPrototype`s by name. For more information about mode declaration refer to 9.1.

[TPS_SWCT_1086] Request mode change ⌈ The ability to request a mode (i.e. the second use case) is modeled on the VFB via a `SenderReceiverInterface` and for the RTE it is like a usual communication, that means the connector can also cross ECU boundaries and the communicated `dataElement`s have to be based on `AutosarDataType`s.⌋*(RS_SWCT_3202, RS_SWCT_3203)*

However, for semantic consistency with the first use case, a communicated mode request shall also be mapped to a corresponding `ModeDeclarationGroup`. This can be defined by a mapping class as shown in figure 4.6.

The ImplementationDataType mapped to a certain ModeDeclarationGroup can then be used in a PortInterface to represent a ModeDeclaration of the associated ModeDeclarationGroup as a numerical value:

[constr_4002] Unambiguous mapping of modes to data types ⌈ Within one DataTypeMappingSet, a ModeDeclarationGroup shall not be mapped to different ImplementationDataTypes. ⌋

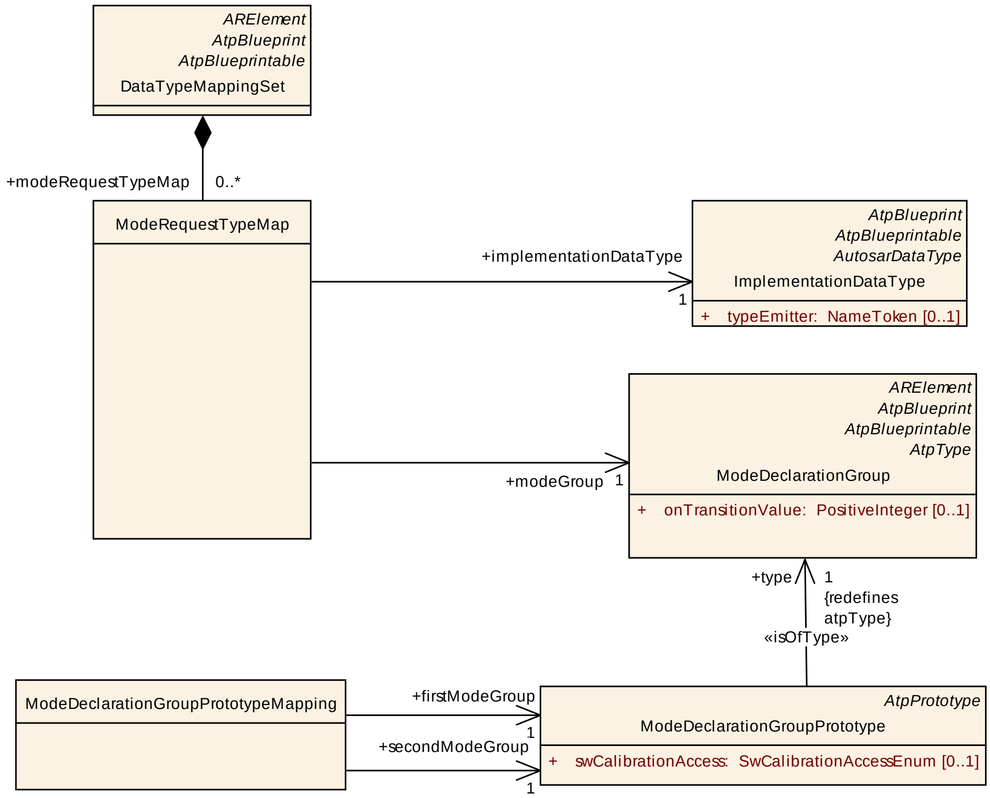

Figure 4.6: Mapping of modes to data types

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***
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 4.16: ModeRequestTypeMap

[constr_1166] Restrictions of `ModeRequestTypeMap` ⌈ For every `ModeDeclarationGroup` referenced by a `ModeDeclarationGroupPrototype` used in a `PortPrototype` typed by a `ModeSwitchInterface` 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 `SwcInternalBehavior` that is owned by the `ApplicationSwComponentType` that also owns the `PortPrototype`. ⌋

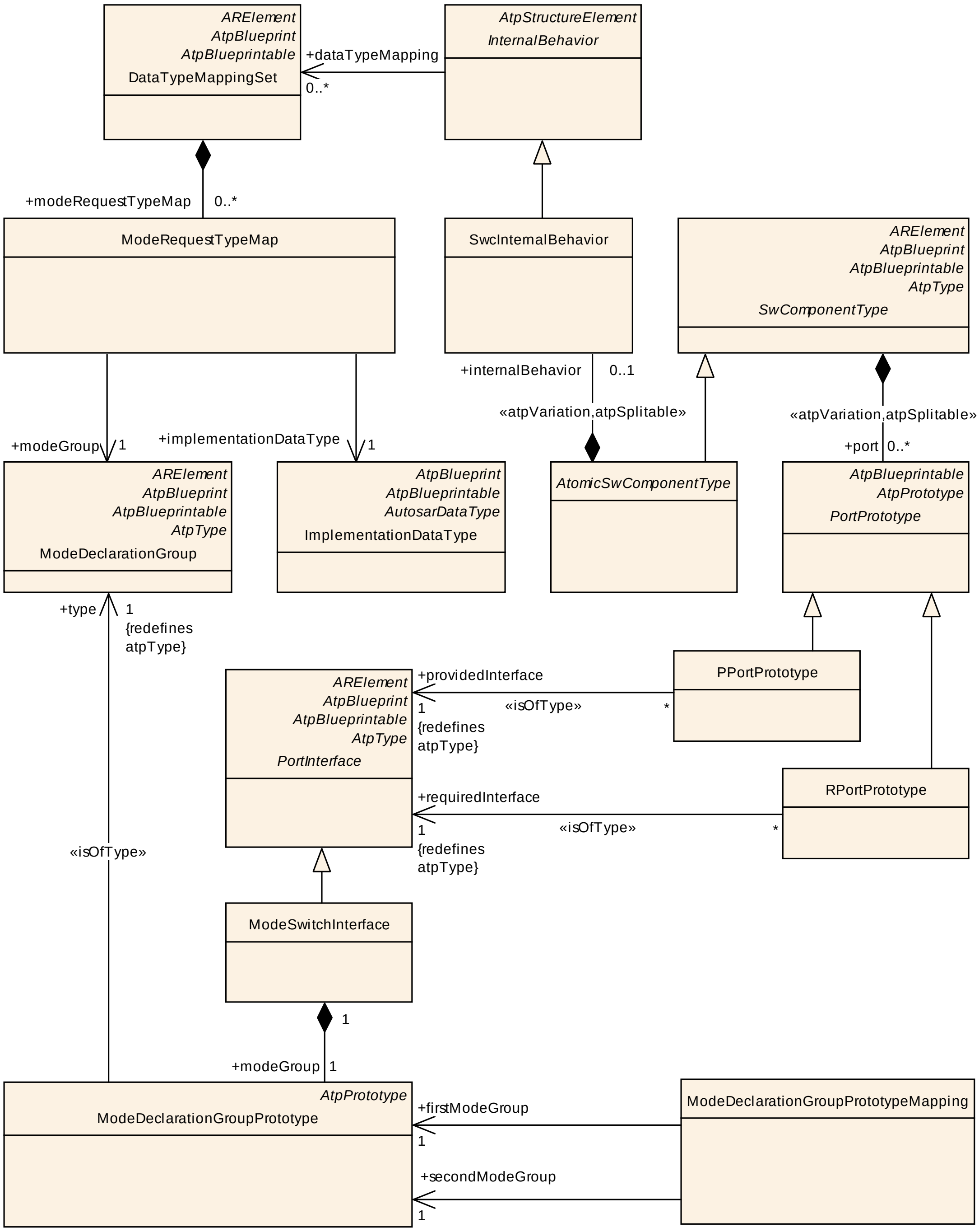

Figure 4.7: Big picture of mode declaration mapping

[constr_1167] ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType ⌈ The ImplementationDataType referenced by a ModeRequestTypeMap shall **either** be of category VALUE **or** of category TYPE_REFERENCE that in turn references an ImplementationDataType of category VALUE.

The baseType referenced by the ImplementationDataType shall have set the value of the attribute baseTypeEncoding to NONE. ⌋

[TPS_SWCT_1202] ApplicationDataType defines a subset of the values used in the ModeDeclarationGroup ⌈ Please note that the corresponding ApplicationDataType is defining a subset of the values used in the ModeDeclarationGroup and the used labels may differ from the names used for the ModeDeclarations.

It is in the responsibility of a system designer to maintain the data types and ModeDeclarationGroups according to the functional needs.

For example, a ModeRequester may only request a subset of the available Modes (via SenderReceiverInterface or ClientServerInterface). The ModeManager may additionally decide to indicate failure. ⌋*(RS_SWCT_3203)*

4.3 PortInterface Mapping and Data Scaling

In former versions of this specification, the requirements on PortInterfaces to match each others could lead to situations where PortInterfaces that were "practically" compatible would nevertheless be rejected because of formal reasons (e.g. ShortNames of dataElement do not match).

In order to also support scenarios where the developer of a CompositionSwComponentType needs to connect PortPrototypes that would match to each others but don't fulfill formal requirements the concept of "port interface mapping" has been introduced.

[TPS_SWCT_1158] Three cases for PortInterfaceMapping ⌈ In general there are three different cases, where a PortInterfaceMapping is suitable.

1. Two PortPrototypes shall be connected and the PortInterface elements are compatible except the unequal shortNames. This requires a pure logical mapping of the PortInterface elements.

2. PortInterface elements are logically equivalent but the range and resolution is differently. This requires a data conversion respectively a re-scaling of the provided data and arguments to the required data and arguments range and resolution.

3. invalidationPolicy of PortInterface elements is different. This might require the implementation of different invalidation handling strategies for the same dataElement in parallel on the same ECU.

⌋(RS_SWCT_3210)

Typically the mapping of such PortInterface is agreed once between the different component vendors and system designer in the early phase of a project.

[TPS_SWCT_1159] Mapping is described separately from the SwConnector as reusable ARElement ⌈ Therefore, the mapping is described separately from the SwConnector as reusable ARElement. A set of PortInterfaceMappings is grouped in a PortInterfaceMappingSet. ⌋*(RS_SWCT_3210)*

Class	**PortInterfaceMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies a set of (one or more) PortInterfaceMappings. **Tags:** atp.recommendedPackage=PortInterfaceMappingSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
portInterfaceMapping	PortInterfaceMapping	1..*	aggr	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).

Table 4.17: PortInterfaceMappingSet

Class	**PortInterfaceMapping (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Specifies one PortInterfaceMapping to support the connection of Ports typed by two different PortInterfaces with PortInterface elements having unequal names and/or unequal semantic (resolution or range).			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 4.18: PortInterfaceMapping

4.3.1 PortInterface Mapping

By default, the shortNames of PortInterface elements are used to identify the matching element pairs of connected PortPrototypes. In case of non-matching shortNames (this might be due to distributed development, off-the-shelves development, or reuse of software-components) it is required to explicitly specify which elements of PortInterfaces shall correlate to each others.

This definition is provided with PortInterfaceMappings.

[TPS_SWCT_1099] **`PortInterfaceMapping`** ⌈ Each `PortInterfaceMapping` describes the mapping of the `PortInterface` elements of exactly two `PortInterface`s. ⌋*(RS_SWCT_3155, RS_SWCT_3210)*

To apply the `PortInterfaceMapping` a `SwConnector` has to reference a `PortInterfaceMapping`.

[constr_1151] Applicability of `PortInterfaceMapping` ⌈ A `PortInterfaceMapping` is only applicable and valid for a `SwConnector` if the two `PortPrototype`s which are referenced by the `SwConnector` are typed by the same two `PortInterface`s which are mapped by the `PortInterfaceMapping`. ⌋

[TPS_SWCT_1100] Precedence of `PortInterfaceMapping` ⌈ The mapping via `PortInterfaceMapping` has a higher precedence than the mapping by equal `shortName`s as defined in 6. If a connector has an associated `PortInterfaceMapping` this mapping shall be strictly binding with respect to the number of mapped data elements. ⌋*(RS_SWCT_3155, RS_SWCT_3210)*

[TPS_SWCT_1101] Unmapped elements of `PortInterfaces` ⌈ Therefore unmapped `PortInterface` elements will not be connected by the referencing `SwConnector`. ⌋*(RS_SWCT_3155, RS_SWCT_3210)*

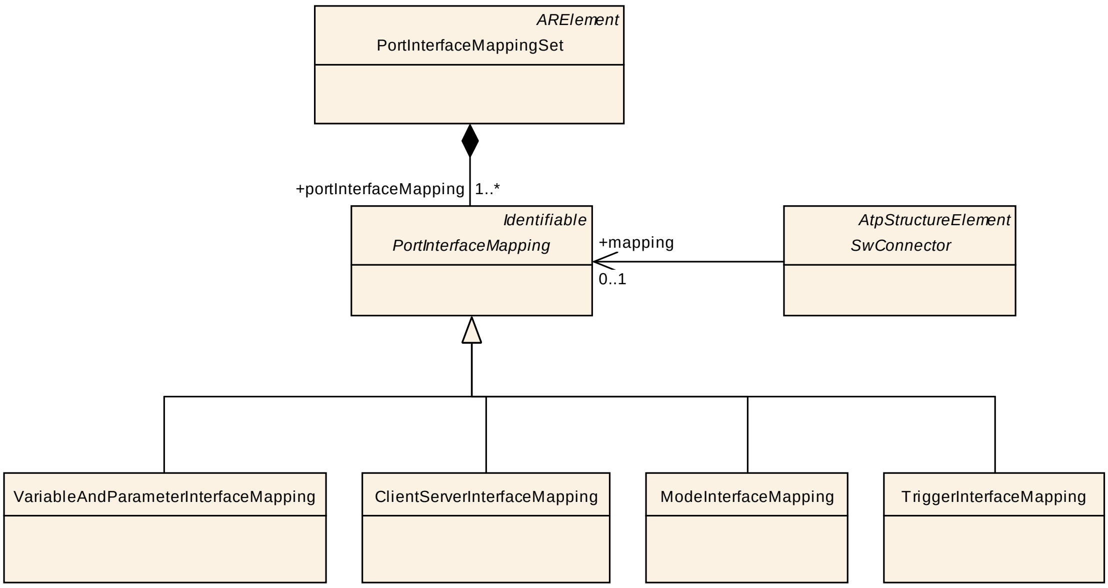

Figure 4.8: Relevant meta-classes for PortInterface element mapping

4.3.1.1 Mapping of Sender Receiver Interface, Parameter Interface and Non Volatile Data Interface Elements

[TPS_SWCT_1102] **`VariableAndParameterInterfaceMapping`** ⌈ The `VariableAndParameterInterfaceMapping` defines the correlation of `VariableDataPrototype`s and `ParameterDataPrototype`s defined in the context of `DataIn-`

terfaces, i.e. SenderReceiverInterface, NvDataInterface, or ParameterInterface. ⌋*(RS_SWCT_3155, RS_SWCT_3210, RS_SWCT_3170)*

[constr_1159] Consistency of VariableAndParameterInterfaceMapping with respect to the referenced DataInterfaces ⌈ Within one VariableAndParameterInterfaceMapping all firstDataPrototypes shall belong to one and only one DataInterface and all secondDataPrototypes shall belong to one other and only one other DataInterface. ⌋

[TPS_SWCT_1103] Mapping between different kinds of PortInterfaces ⌈ Thereby it is possible to describe the mapping between different kinds of PortInterfaces for instance an ParameterInterface and SenderReceiverInterface. ⌋*(RS_SWCT_3155, RS_SWCT_3210, RS_SWCT_3170)*

[TPS_SWCT_1104] Possible mappings are restricted by the SwImplPolicy ⌈ Nevertheless, the possible mappings of VariableDataPrototypes and ParameterDataPrototypes are restricted by the SwImplPolicy attribute. ⌋*(RS_SWCT_3155, RS_SWCT_3210, RS_SWCT_3170)*

[constr_1039] Relevance of SwImplPolicy ⌈ It is not possible to define a mapping between an element where the SwImplPolicy is set to queued and an other element where the SwImplPolicy is set differently. ⌋

This is required to fulfill the compatibility rules defined in table 6.1

[constr_1040] Conversion of SenderReceiverInterfaces ⌈ Either the AutosarDataTypes of the referred DataPrototypes are compatible as described in chapter 6.2 or a conversion of the data as described in chapter 4.3.2 is available. ⌋

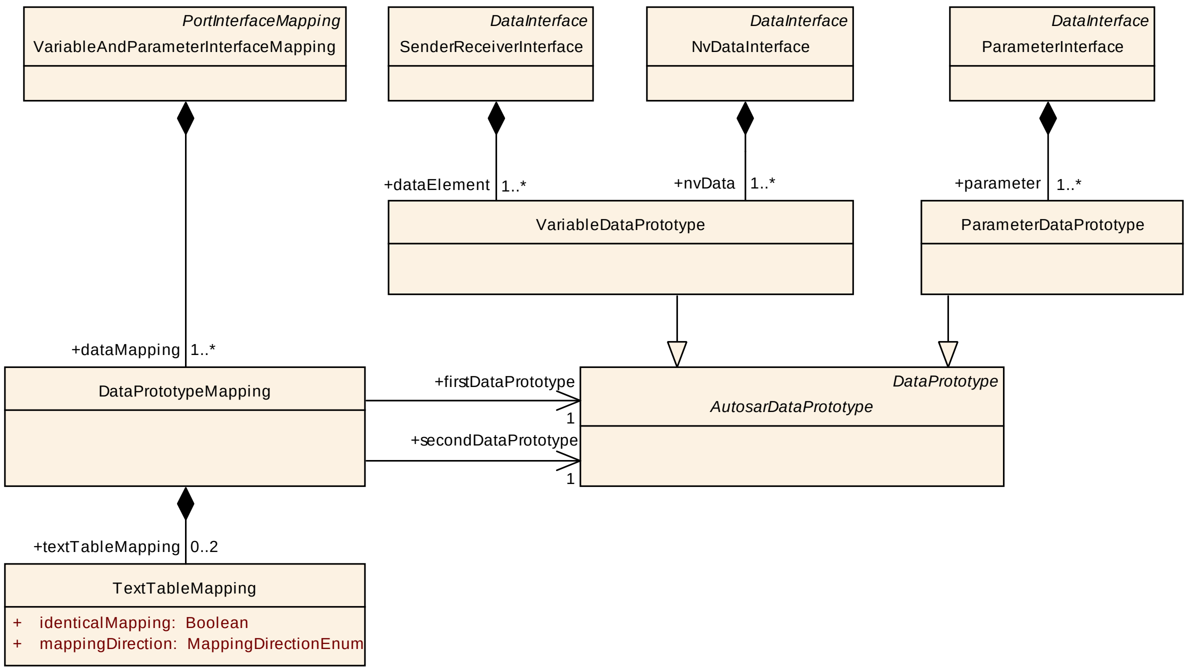

Figure 4.9: Mapping of Sender Receiver Interface, Parameter Interface and Non Volatile Data Interface elements

Class	**VariableAndParameterInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of VariableDataPrototypes or ParameterDataPrototypes in context of two different SenderReceiverInterfaces, NvDataInterfaces or ParameterInterfaces.			
Base	ARObject,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataMapping	DataPrototypeMapping	1..*	aggr	Defines the mapping of two particular VariableDataPrototypes or ParameterDataPrototypes with unequal names and/or unequal semantic (resolution or range) in context of two different SenderReceiverInterfaces, NvDataInterfaces or ParameterInterfaces

Table 4.19: VariableAndParameterInterfaceMapping

Class	**DataPrototypeMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two particular VariableDataPrototypes, ParameterDataPrototypes or ArgumentDataPrototypes with unequal names and/or unequal semantic (resolution or range) in context of two different SenderReceiverInterface, NvDataInterface or ParameterInterface or Operations. If the semantic is unequal following rules apply: The textTableMapping is only applicable if the referred DataPrototypes are typed by AutosarDataType referring to CompuMethods of category TEXTTABLE. In the case that the DataPrototypes are typed by AutosarDataType either referring to CompuMethods of category LINEAR, IDENTICAL or referring to no CompuMethod (which is similar as IDENTICAL) the linear conversion factor is calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys of the referred CompuMethods.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
firstDataPrototype	AutosarDataPrototype	1	ref	First to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, ParameterInterface or Operation.
secondDataPrototype	AutosarDataPrototype	1	ref	Second to be mapped DataPrototype in context of a SenderReceiverInterface, NvDataInterface, ParameterInterface or Operation.
subElementMapping	SubElementMapping	*	aggr	This represents the owned SubelementMapping.
textTableMapping	TextTableMapping	0..2	aggr	Applied TextTableMapping(s)

Table 4.20: DataPrototypeMapping

4.3.1.2 Mapping of Client Server Interface Elements

[TPS_SWCT_1105] **`ClientServerInterfaceMapping`** ⌈ The `ClientServerInterfaceMapping` defines the correlation of `ClientServerOperation`s defined in the context of `ClientServerInterface`s. ⌋*(RS_SWCT_3155, RS_SWCT_3210)*

[constr_1041] Conversion of `ClientServerInterfaces` ⌈ Either the `DataType`s of the referred `ArgumentDataPrototype`s are compatible as described in chapter 6.2 or a conversion of the data as described in chapter 4.3.2 is available. ⌋

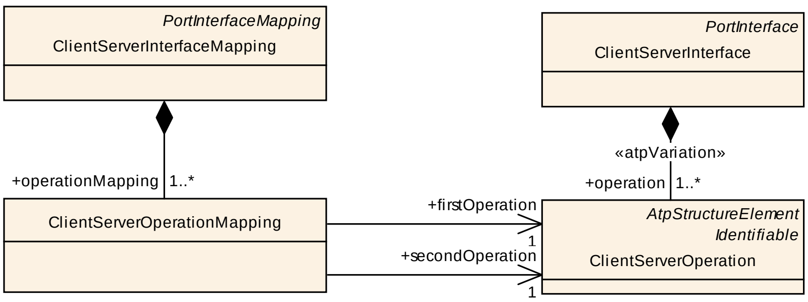

Figure 4.10: Mapping of `ClientServerInterface` elements and mapping of arguments

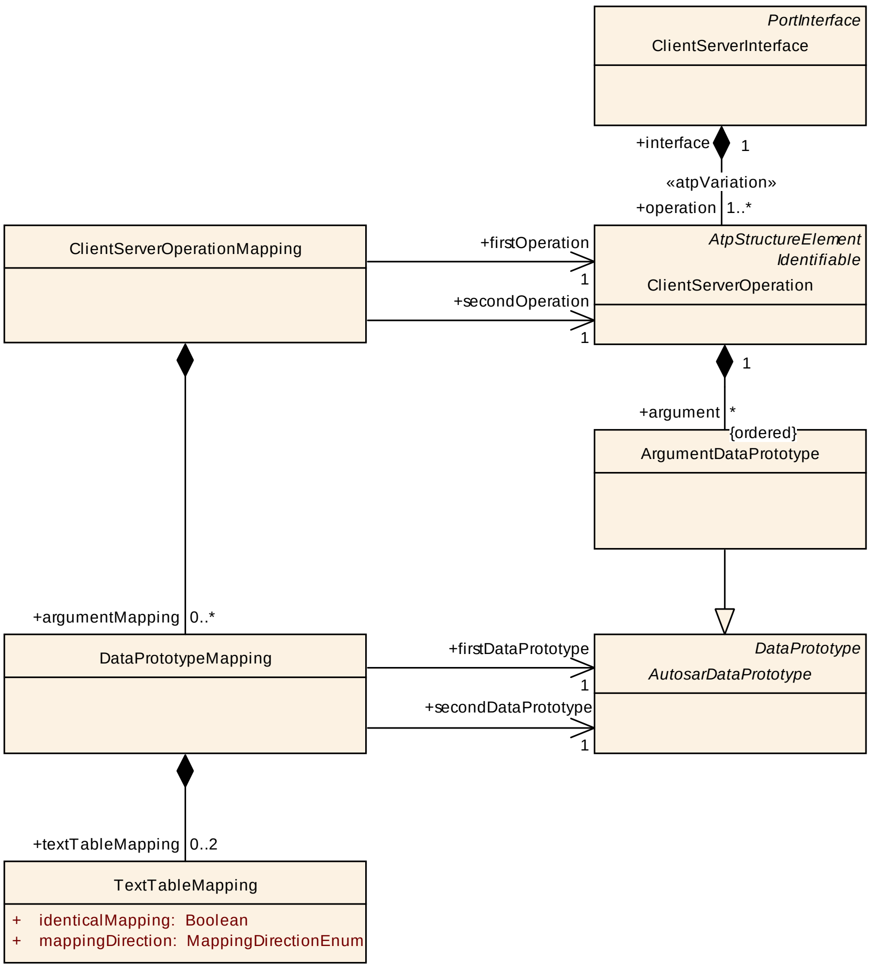

Figure 4.11: Mapping of `ArgumentDataPrototypes`

Class	ClientServerInterfaceMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of ClientServerOperations in context of two different ClientServerInterfaces.			
Base	ARObject,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	Datatype	Mul.	Kind	Note
operationMapping	ClientServerOperationMapping	1..*	aggr	Mapping of two ClientServerOperations in two different ClientServerInterfaces

Table 4.21: ClientServerInterfaceMapping

Class	ClientServerOperationMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two particular ClientServerOperations in context of two different ClientServerInterfaces.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
argumentMapping	DataPrototypeMapping	*	aggr	Defines the mapping of two particular ArgumentDataPrototypes with unequal names or unequal semantic (resolution or range) in context of Operations.
firstOperation	ClientServerOperation	1	ref	First to be mapped ClientServerOperation of a ClientServerInterface.
secondOperation	ClientServerOperation	1	ref	Second to be mapped ClientServerOperation of a ClientServerInterface.

Table 4.22: ClientServerOperationMapping

4.3.1.3 Mapping of Mode Interface Elements

[TPS_SWCT_1160] `ModeInterfaceMapping` ⌈ The `ModeInterfaceMapping` defines the correlation of `ModeDeclarationGroupPrototype`s defined in the context of `ModeSwitchInterface`s. ⌋*(RS_SWCT_3210)*

[TPS_SWCT_1167] Validity of `ModeInterfaceMapping` ⌈ The mapping of `ModeDeclarationGroupPrototype`s is only valid if these are typed by (read "refer to") compatible `ModeDeclarationGroup`s according chapter 6.7. ⌋*(RS_SWCT_3210)*

Class	ModeInterfaceMapping			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of unequal named ModeDeclarationGroupPrototypes in context of two different ModeInterfaces.			
Base	ARObject,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	Datatype	Mul.	Kind	Note
modeMapping	ModeDeclarationGroupPrototypeMapping	1	aggr	Mapping of two ModeDeclarationGroupPrototypes in two different ModeInterfaces

Table 4.23: ModeInterfaceMapping

Class	**ModeDeclarationGroupPrototypeMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Defines the mapping of two particular unequally named but otherwise compatible ModeDeclarationGroupPrototypes in the given context.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
firstModeGroup	ModeDeclarationGroupPrototype	1	ref	ModeDeclarationGroupPrototype to be mapped.
secondModeGroup	ModeDeclarationGroupPrototype	1	ref	ModeDeclarationGroupPrototype to be mapped.

Table 4.24: ModeDeclarationGroupPrototypeMapping

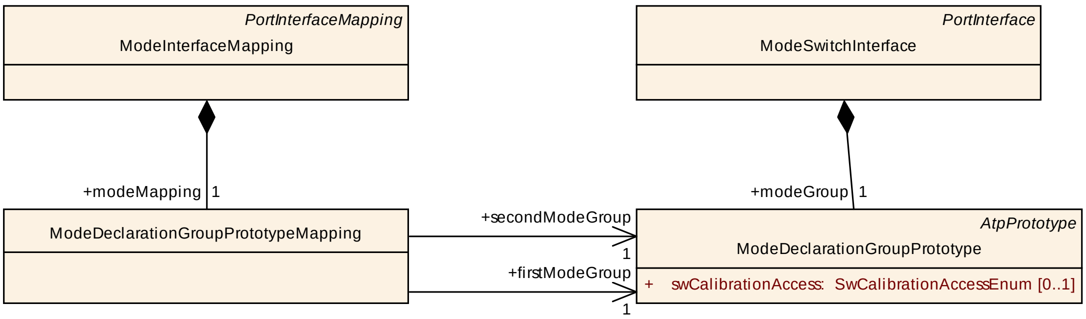

Figure 4.12: Mapping of `ModeSwitchInterface` elements

4.3.1.4 Mapping of Trigger Interface Elements

[TPS_SWCT_1161] `TriggerInterfaceMapping` ⌈ The `TriggerInterfaceMapping` defines the correlation of `Trigger`s defined in the context `TriggerInterface`s. ⌋*(RS_SWCT_3210)*

Class	**TriggerInterfaceMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of unequal named Triggers in context of two different TriggerInterfaces.			
Base	ARObject,Identifiable,MultilanguageReferrable,PortInterfaceMapping,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
triggerMapping	TriggerMapping	1..*	aggr	Mapping of two Trigger in two different TriggerInterface

Table 4.25: TriggerInterfaceMapping

Class	TriggerMapping			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	Defines the mapping of two particular unequally named Triggers in the given context.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
firstTrigger	Trigger	1	ref	A Trigger to be mapped.
secondTrigger	Trigger	1	ref	A Trigger to be mapped.

Table 4.26: TriggerMapping

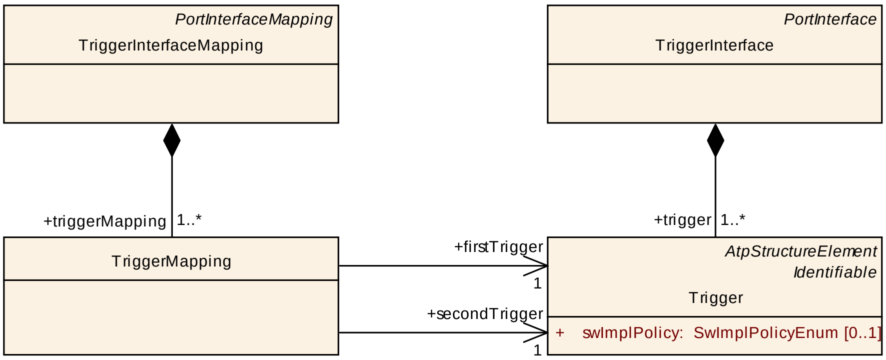

Figure 4.13: Mapping of `TriggerInterface` elements

4.3.1.5 Mapping of Elements of a composite Data Type

The mapping of elements of `PortInterface`s is not limited to mapping entire `DataPrototype`s onto each others.

[TPS_SWCT_1023] Mapping of elements of composite data types ⌈ For applications of `DataInterface`s it is also possible to formally describe the mapping of elements of `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category STRUCTURE` or `ARRAY` onto each others. ⌋*(RS_SWCT_3210, RS_SWCT_3135)*

This ability can be used if e.g. `dataElement`s on the sender and receiver side are typed by different `ApplicationRecordDataType`s.

In this case the mapping of elements of `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category STRUCTURE` or `ARRAY` onto each others allows for the definition of specific pairs of elements that fulfill the compatibility rules. Please note, however, that this does not necessarily mean that all elements on the sender side need to be mapped to elements on the receiver side to achieve compatibility. The details regarding the compatibility rules are explained in chapter 6.3.

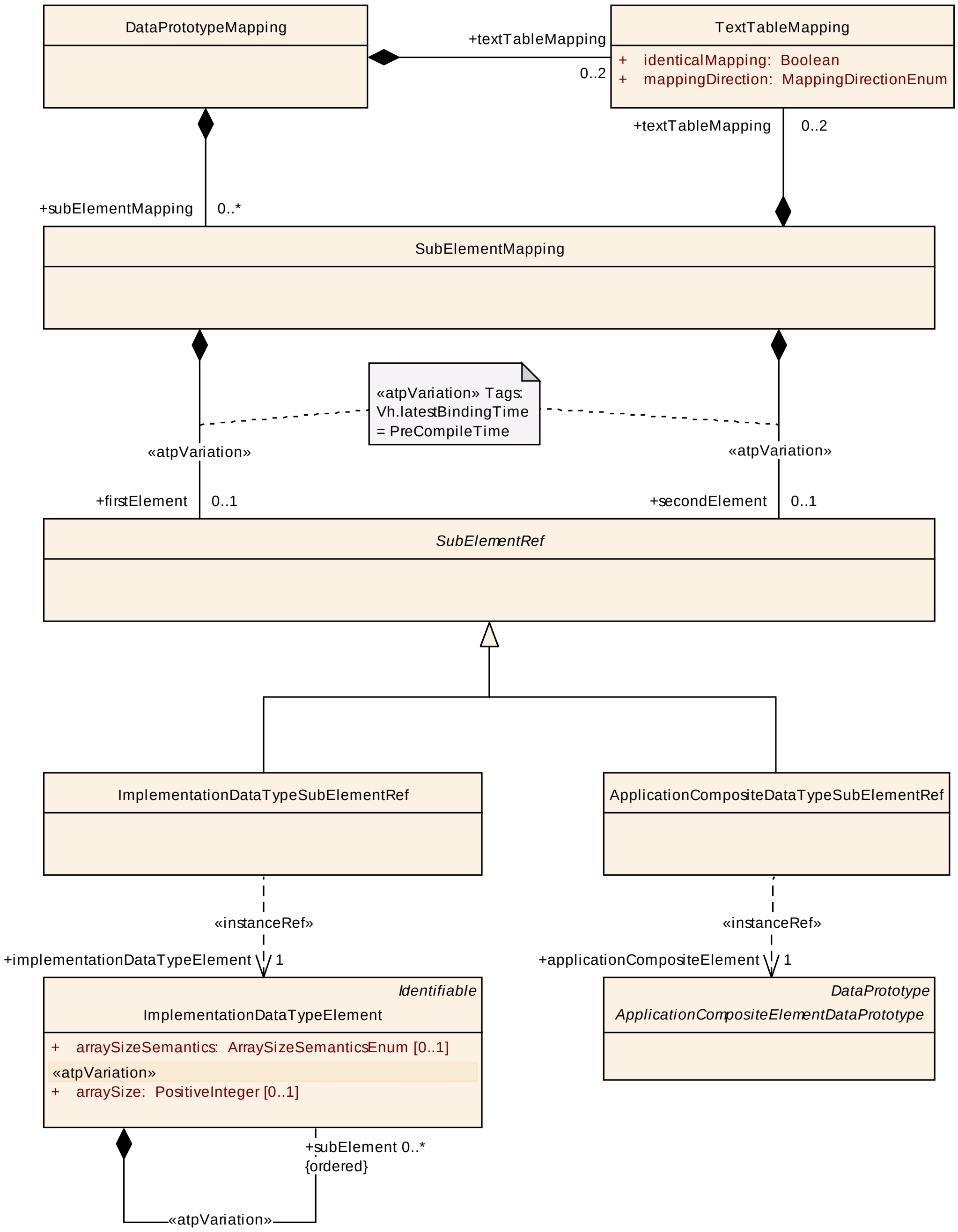

Figure 4.14: Mapping of elements of composite data types

[TPS_SWCT_1024] Combination of `ApplicationCompositeDataType` and nested `ImplementationDataType` ⌈ The mapping of elements of `ApplicationCompositeDataType`s or `ImplementationDataType`s of `category STRUCTURE` or `ARRAY` works for both `ApplicationCompositeDataType` and nested `ImplementationDataType`s and even for combinations of them, i.e. one `PortInterface` may use an `ApplicationCompositeDataType` while the other `PortInterface` uses a nested `ImplementationDataType`. ⌋*(RS_SWCT_3210, RS_SWCT_3135)*

[TPS_SWCT_1195] Mapping of composite element to primitive `DataPrototype` ⌈ It is also possible to map an element of a composite data type on the provided side to a primitive `DataPrototype` on the required side. For this purpose the multiplicity of the `firstElement` shall be set to 1 and the multiplicity of the `secondElement` shall be set to 0. ⌋*(RS_SWCT_3136)*

In general, the multiplicity of the `firstElement` can technically also be set to 0 but this case is reserved for future use.

[constr_1190] Only one mapping for composite to primitive use case ⌈ In the case described by [TPS_SWCT_1195] only one `subElementMapping` shall exist at the enclosing `DataPrototypeMapping`. ⌋

Class	**SubElementMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class allows for the definition of mappings of elements of a composite data type.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
firstElement	SubElementRef	0..1	aggr	This represents the first element referenced in the scope of the mapping. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
secondElement	SubElementRef	0..1	aggr	This represents the second element referenced in the scope of the mapping. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
textTableMapping	TextTableMapping	0..2	aggr	This allows for the text-table translation of individual elements of a composite data type.

Table 4.27: SubElementMapping

Class	**SubElementRef (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class provides the ability to reference elements of composite data type.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 4.28: SubElementRef

Class	**ImplementationDataTypeSubElementRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the specialization of SubElementMapping with respect to ImplementationDataTypes.			
Base	ARObject,SubElementRef			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
implementationDataTypeElement	ArVariableInImplementationDataInstanceRef	1	aggr	This represents the referenced implementationDataTypeElement.

Table 4.29: ImplementationDataTypeSubElementRef

Class	ApplicationCompositeDataTypeSubElementRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This meta-class represents the specialization of SubElementMapping with respect to ApplicationCompositeDataTypes.			
Base	ARObject,SubElementRef			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
applicationCompositeElement	ApplicationCompositeElementDataPrototype	1	iref	This represents the referenced ApplicationCompositeDataPrototype.

Table 4.30: ApplicationCompositeDataTypeSubElementRef

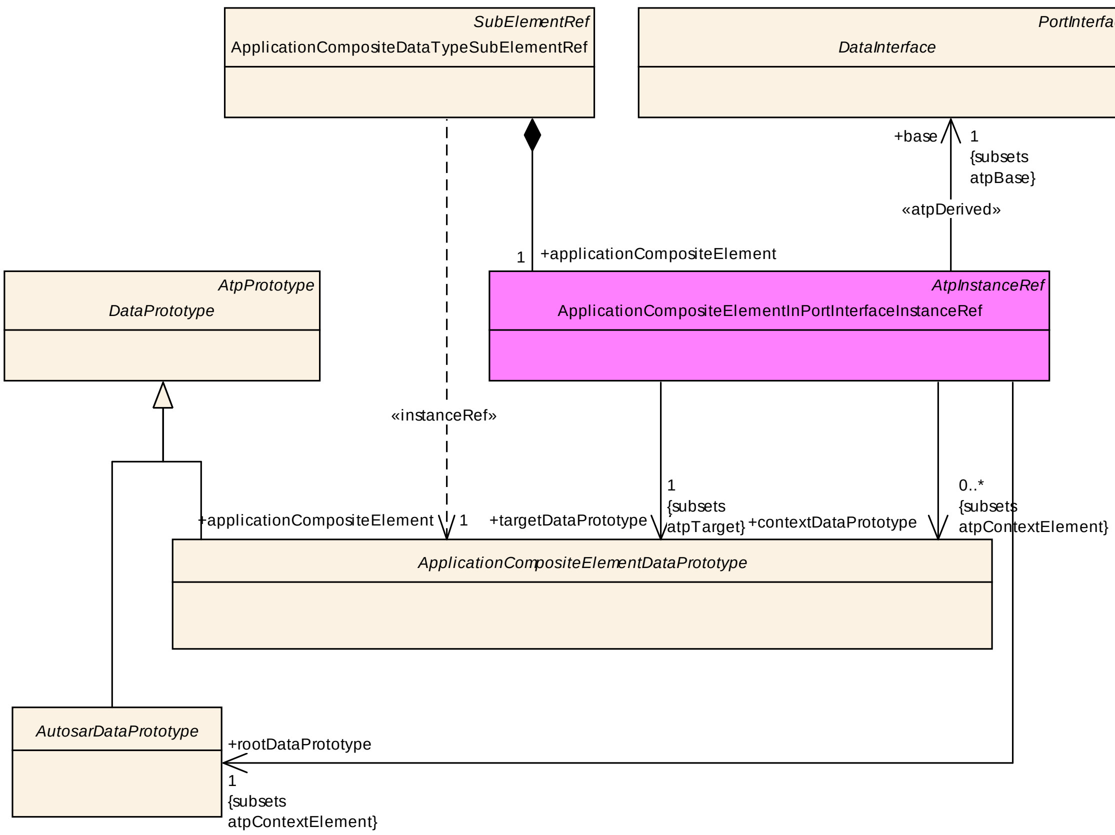

Figure 4.15: Implementation of the InstanceRef for the mapping of elements of composite application data types

[constr_1184] Consistency of `rootDataPrototype` and `base` in the context of `ApplicationCompositeElementInPortInterfaceInstanceRef` ⌈ The `rootDataPrototype` referenced by `ApplicationCompositeElementInPortInter-`

faceInstanceRef shall be owned by the applicable subclass of DataInterface referenced in the role base. This implies that the rootDataPrototype must be a ParameterDataPrototype if the base is a ParameterInterface. Otherwise the rootDataPrototype must be a VariableDataPrototype. ⌋

[constr_1185] Consistency of data types in the context of ApplicationCompositeElementInPortInterfaceInstanceRef ⌈ The definition of attributes contextDataPrototype and targetDataPrototype shall (via the type-prototype pattern) be enclosed in the context of the definition of the data type used to type rootDataPrototype. ⌋

In other words, it shall be possible to reach contextDataPrototype and targetDataPrototype by means of the type-prototype chain created by the definition of the data type used to type rootDataPrototype. And, as implied by the definition of the InstanceRef, the contextDataPrototypes shall enclose each others and, eventually, the targetDataPrototype.

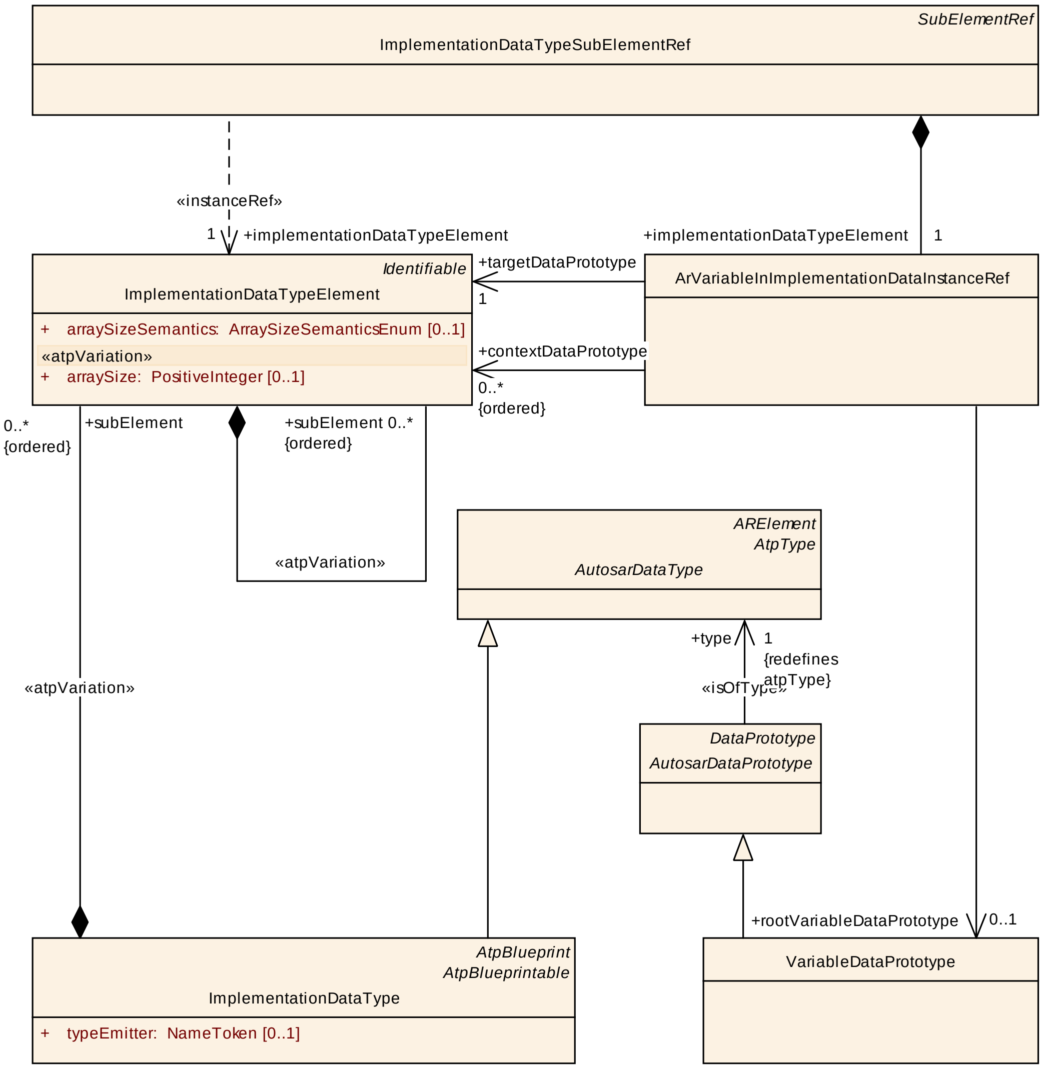

Figure 4.16: Implementation of the InstanceRef for the mapping of elements of composite implementation data types

[constr_1186] Consistency of data types in the context of ArVariableInImplementationDataInstanceRef ⌈ The definition of attributes contextDataPrototype and targetDataPrototype shall be enclosed in the context of the definition of the data type used to type rootDataPrototype. ⌋

4.3.2 Data Conversion

4.3.2.1 Linear Data Scaling

A *Linear Data Scaling* can be defined under following preconditions:

[constr_1042] Definition of a linear data scaling ⌈

- The referred AutosarDataTypes in turn refer to CompuMethods of category IDENTICAL or LINEAR.
- The CompuMethods refer either to compatible Units or to Units that in turn refer to identical definitions of PhysicalDimension (all PhysicalDimension attributes are identical)

⌋

[TPS_SWCT_1168] Linear conversion factor can be calculated ⌈ In such cases a linear conversion factor can be calculated out of the factorSiToUnit and offsetSiToUnit attributes of the referred Units and the CompuRationalCoeffs of a compuInternalToPhys of the referred CompuMethods. ⌋*(RS_SWCT_3210)*

4.3.2.2 Table Conversion

[TPS_SWCT_1162] Conditional existence of TextTableMapping ⌈ A TextTableMapping can be defined if the AutosarDataTypes refers to CompuMethods of category TEXTTABLE. ⌋*(RS_SWCT_3210)*

The TextTableMapping is defined as a table based conversion.

[TPS_SWCT_1163] Conversion from firstValue to secondValue ⌈ A firstValue of a valuePair is converted into the secondValue in case of a data flow from the firstDataPrototype to the secondDataPrototype. ⌋*(RS_SWCT_3210)*

[TPS_SWCT_1164] Conversion from secondValue to firstValue ⌈ In case of an data flow from the secondDataPrototype to firstDataPrototype the secondValue is substituted by the firstValue. ⌋*(RS_SWCT_3210)*

[TPS_SWCT_1165] Invertible mapping ⌈ If the mappingDirection attribute is set to BIDIRECTIONAL the TextTableMapping has to be invertible. This requires that the list of all firstValues and the list of all secondValues do not contain identical values inside a list. ⌋*(RS_SWCT_3210)*

[TPS_SWCT_1166] Non-invertible mapping ⌈ For non-invertible `TextTableMapping` a dedicated `TextTableMapping` for each direction can be defined.
⌋*(RS_SWCT_3210)*

Class	**TextTableMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines the mapping of two DataPrototypes typed by AutosarDataTypes that refer to CompuMethods of category TEXTTABLE.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
identicalMapping	Boolean	1	attr	If identicalMapping is set == true the values of the two referenced DataPrototypes do not need any conversion of the values.
mappingDirection	MappingDirectionEnum	1	attr	Specifies the conversion direction for which the TextTableMapping is applicable.
valuePair	TextTableValuePair	*	aggr	Defines a pair of values which are translated into each other.

Table 4.31: TextTableMapping

Enumeration	**MappingDirectionEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	Specifies the conversion direction for which the mapping is applicable.
Literal	*Description*
bidirectional	The TextTableMapping is applicable in both directions.
firstToSecond	The TextTableMapping is applicable in the direction from firstDataPrototype / firstOperationArgument referring into the PortInterface of the PPortPrototype to secondDataPrototype / secondOperationArgument referring into the PortInterface of the RPortPrototype.
secondToFirst	The TextTableMapping is applicable in the direction from secondDataPrototype / secondOperationArgument referring into the PortInterface of the PPortPrototype to firstDataPrototype / firstOperationArgument referring into the PortInterface of the RPortPrototype.

Table 4.32: MappingDirectionEnum

Class	**TextTableValuePair**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Defines a pair of text values which are translated into each other.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
firstValue	Numerical	1	attr	Value of first DataPrototype provided similar to a numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	Datatype	Mul.	Kind	Note
secondValue	Numerical	1	attr	Value of second DataPrototype provided similar to a numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 4.33: TextTableValuePair

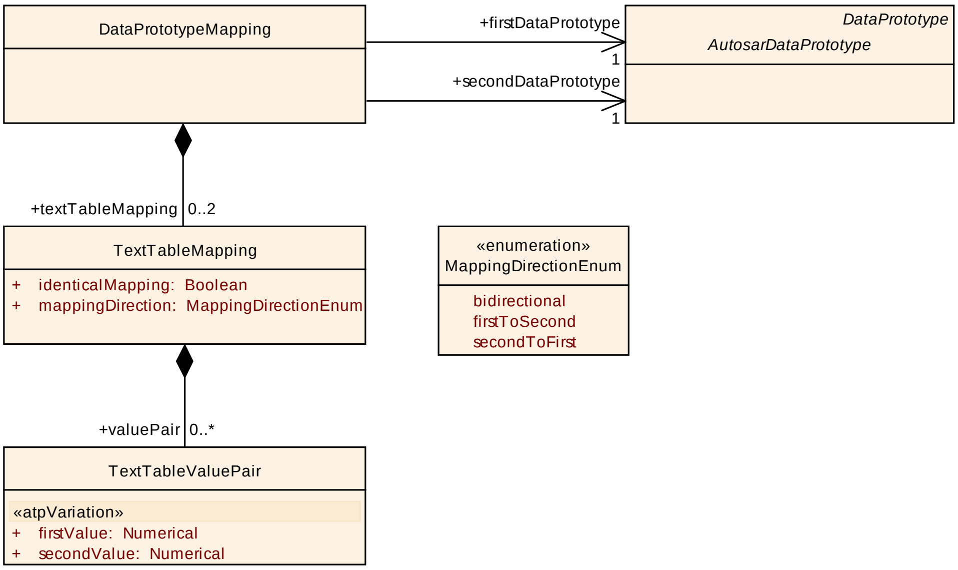

Figure 4.17: Mapping of `DataPrototypes` which `ApplicationDataTypes` referring to `CompuMethods` of category *TEXTTABLE*

4.4 Port Annotation

4.4.1 Introduction

[TPS_SWCT_1203] `PortPrototype` may own port annotations ⌈ In addition to the formal specification required to implement the communication via ports, a `PortPrototype` may own so-called port annotations (please find a summary in Figure 4.18). They do not directly influence the signature of calls via this port, but contain further information that may be useful for the application developers of the components on both sides of the connection. ⌋

[TPS_SWCT_1204] `GeneralAnnotation` ⌈ Beside formally specified attributes it is also possible to place textual information as provided in `GeneralAnnotation`. ⌋

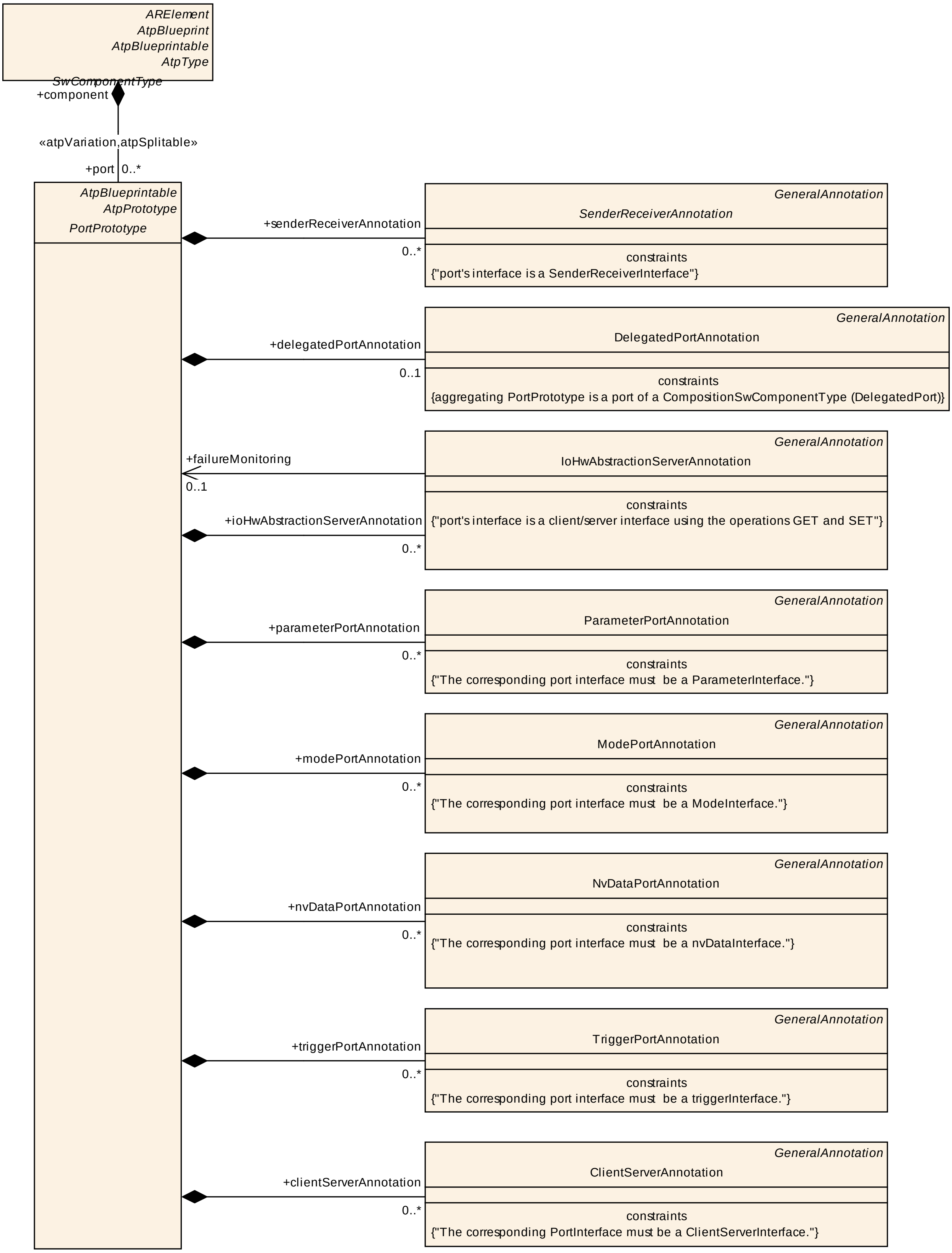

Figure 4.18: Application Level Port Annotations Overview

4.4.2 SenderReceiverAnnotation

Embedded automotive software is used to implement open-loop and closed-loop control-algorithms. Therefore, a software-component description has to accommodate typical control engineering description means which have only indirect influence of the embedded software itself.

These annotations provide the (function-) developer with a direct indication whether a certain software-component is appropriate for the control-algorithm to be designed. A typical annotation is the signal quality which is characterized by several properties. Each of the property is an annotation in its own.

[TPS_SWCT_1205] Typical annotations for sender/receiver communication ⌈ Typical annotations for sender/receiver communication are:

- **Signal Age**: this attribute expresses that the associated software-component will only work correctly given that the propagation of the signal from a sensor to a consumer can be finished within a particular time-limit. Of course, this cannot be identified on component or role level, but has to take into account the instance view as well as the actual ECU- and bus-scheduling.
- **Raw**: a raw signal is typically taken directly from the basic software modules of the ECU abstraction layer. In particular, no sensor software-component has filtered its original value. A dataElement in an RPortPrototype of a SwComponentType using this annotation indicates to the control engineer (who develops a control-algorithm for this component) that the signal has to be filtered (This relationship applies for SenderReceiverInterfaces).
- **Filtered**: this attribute indicates that a raw signal has been manipulated by some application software-components by using a certain filter.
- **Computed**: this attribute indicates that this signal is not measured directly but calculated from tentatively several other measured or calculated signals. In a vehicle, there might be alternative signals to be used from other components having a better quality, e.g. a raw signal.
- **Min**: this annotation indicates that the signal carries a minimum value. If, for example, a reference value computed in the software-component is below that value some dedicated actions (e.g. failure-mode) might have to be taken.
- **Max**: this annotation indicates that the signal carries a maximum value. If, for example, a reference value computed in the software-component is above that value some dedicated actions (e.g. failure-mode) might have to be taken.

In the meta-model this aspect is implemented by the abstract meta-class SenderReceiverAnnotation which represents the base class of both SenderAnnotation and ReceiverAnnotation. This relationship is depicted in Figure 4.19. ⌋

Class	**SenderReceiverAnnotation (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation of the data elements in a port that realizes a sender/receiver interface.			
Base	ARObject,GeneralAnnotation			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
computed	Boolean	1	attr	Flag whether this data element was not measured directly but instead was calculated from possibly several other measured or calculated values.
dataElement	VariableDataPrototype	1	ref	The instance of VariableDataPrototype annotated.
limitKind	DataLimitKindEnum	1	attr	This min or max has not to be mismatched with the min- and max for data-value in a compu-method. For example, this annotation shows when the result of the calculation performed in a RunnableEntity owned by one AtomicSwComponentType is transmitted to another AtomicSwComponentType whose RunnableEntity will use this value as a limit, e.g. the max.power which can be used by that software-component, or the current min. slip.
processingKind	ProcessingKindEnum	1	attr	This attribute controls how data is processed according to the possible values of ProcessingKindEnum.

Table 4.34: SenderReceiverAnnotation

Class	**SenderAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation of a sender port, specifying properties of data elements that don't affect communication or generation of the RTE.			
Base	ARObject,GeneralAnnotation,SenderReceiverAnnotation			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 4.35: SenderAnnotation

Class	**ReceiverAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation of a receiver port, specifying properties of data elements that don't affect communication or generation of the RTE. The given attributes are requirements on the required data.			
Base	ARObject,GeneralAnnotation,SenderReceiverAnnotation			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
signalAge	MultidimensionalTime	1	aggr	The maximum allowed age of the signal since it was originally read by a sensor. This is a requirement specified on the receiver side.

Table 4.36: ReceiverAnnotation

Enumeration	**ProcessingKindEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Kind of processing which has been applied to a data element.
Literal	***Description***
filtered	Indicates that a raw signal has been manipulated by some application software components by using filters.
none	Indicates that none of the other option apply.
raw	Specifies that a signal is taken directly from the basic software modules, i.e. from the ECU abstraction layer. It indicates to a developer that the control algorithm in the software has to provide filters.

Table 4.37: ProcessingKindEnum

Enumeration	**DataLimitKindEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	Indicates whether the data element carries a minimum or maximum value, thereby limiting the current range of another value.
Literal	***Description***
max	Limitation to maximum value
min	Limitation to minimum value
none	No limitation applicable

Table 4.38: DataLimitKindEnum

[TPS_SWCT_1206] Min and Max annotations are valid for a certain amount of time ⌈ The Min and Max annotations are valid for a certain amount of time. The value is likely to change to another valid value while the ECU is running. E.g. the maximal torque which can be requested from an engine is a typical use-case. ⌋

This value might vary depending on e.g. the status of the climate control system. Therefore, these annotations shall not be mismatched with the min and max attributes of `CompuMethods`.

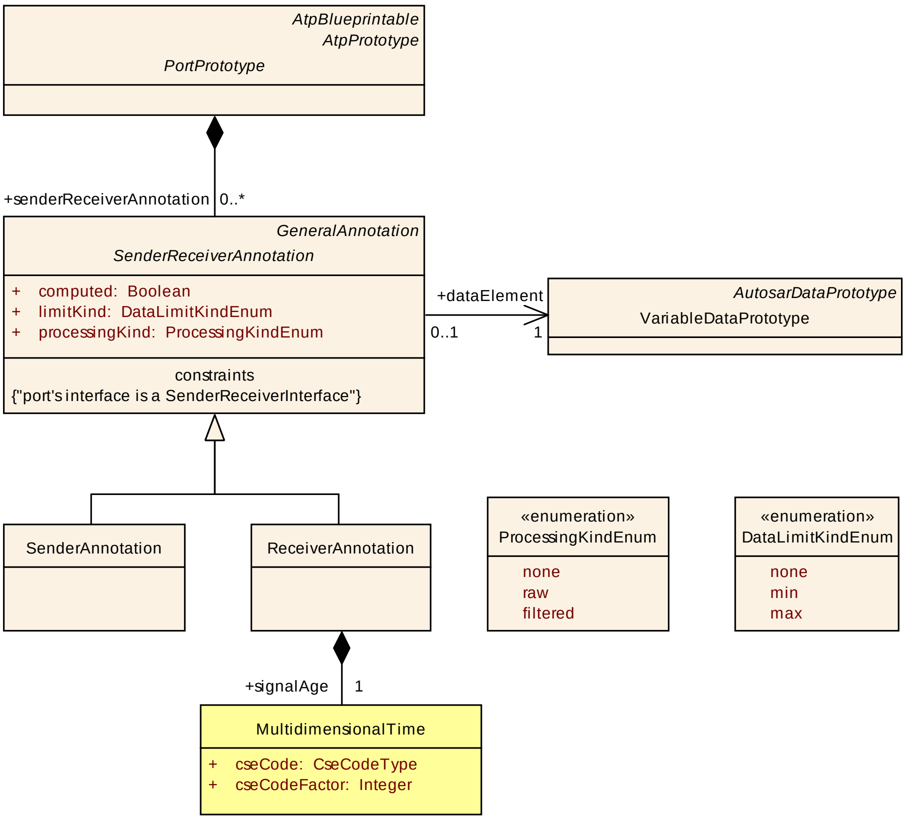

Figure 4.19: `SenderReceiverAnnotation`

The application level port annotations for sender/receiver communication have to be associated to each `dataElement` in a `PortPrototype`, e.g. there might be a "raw" `dataElement` and a "filtered" `dataElement` in the same `PortPrototype`!

[TPS_SWCT_1207] `VariableDataPrototypes` use the same application-level `SenderReceiverAnnotation` ⌈ Furthermore, if two `VariableDataPrototype`s use the same application-level `SenderReceiverAnnotation`, a reference from the annotation to the `VariableDataPrototype`s will be established by an appropriate tool. ⌋

[TPS_SWCT_1208] Grouping for `SenderReceiverAnnotation` ⌈ As shown in Figure 4.19 the `SenderReceiverAnnotation` for sender/receiver communication are grouped into

- processing type, indicating to some extend the direct quality of the signal,
- computed, which is just a flag or,
- limit type, showing the component expects an actual limit.

In the case of an `RPortPrototype`, the signal age of the value, carried by the associated `SwConnector`, can be specified. Each of these groups can be interpreted as a property of the signal-quality. ⌋

[constr_4004] Context of `SenderReceiverAnnotation` ⌈ A `SenderReceiver-Annotation` shall only be aggregated by a `PortPrototype` typed by a `Sender-ReceiverInterface`. ⌋

4.4.3 ClientServerAnnotation

[TPS_SWCT_1209] `ClientServerAnnotation` ⌈ The `ClientServerAnnotation` can be used to provide more information with respect to the `Operation` of the port. ⌋

Class	**ClientServerAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port regarding a certain Operation.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1	ref	This represents the ClientServerOperation that the ClientServerAnnotation corresponds to.

Table 4.39: ClientServerAnnotation

The main use-case is to allow define additional information related to the `Operation`.

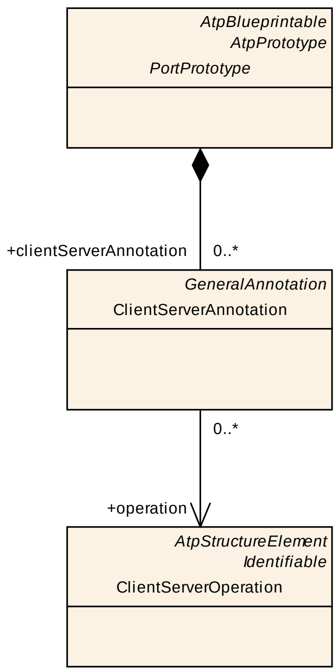

Figure 4.20: `ClientServerAnnotation`

[constr_4005] Context of `ClientServerAnnotation` ⌈ A `ClientServerAnnotation` shall only be aggregated by a `PortPrototype` typed by a `ClientServer-Interface`. ⌋

4.4.4 Annotation for the I/O Hardware Abstraction Layer

[TPS_SWCT_1210] **`IoHwAbstractionServerAnnotation`** ⌈ The attributes `bswRangeMin`, `bswRangeMax`, `bswResolution` and Unit of physical signals are currently being described by attributes of meta-class `IoHwAbstractionServerAnnotation`. ⌋

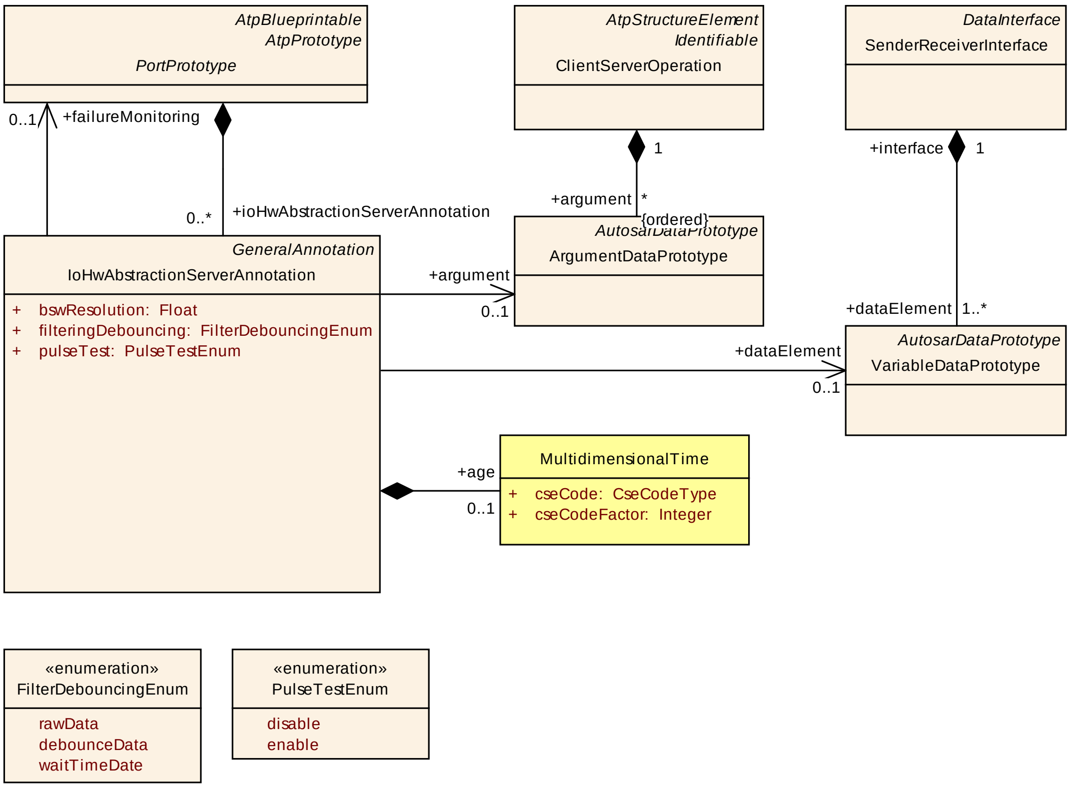

Figure 4.21: `IoHwAbstractionServerAnnotation`

Class	**IoHwAbstractionServerAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	The IoHwAbstractionPort Annotation will only be used from a sensor- or an actuator component while interacting with the IoHwAbstraction layer			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
age	MultidimensionalTime	0..1	aggr	In case of a SET operation, the age will be interpreted as Delay while in a GET operation (input) it specifies the Lifetime of the signal within the IoHwAbstraction Layer
argument	ArgumentDataPrototype	0..1	ref	Reference to the corresponding ArgumentDataPrototype. The IoHwAbstractionServerAnnotation can be applied either to sender-receiver or to client-server communication. This association only applies in the latter case

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
bswResolution	Float	1	attr	This value is determined by an appropriate combination of the range, the unit as well as the data-elements type, i.e. (ecuSignalRange.upperLimit-ecuSignalRange.lowerLimit) / (2ˆ datatypelength - 1)
dataElement	VariableDataPrototype	0..1	ref	Reference to the corresponding VariableDataPrototype. The IoHwAbstractionServerAnnotation can be applied either to sender-receiver or to client-server communication. This association only applies in the former case
failureMonitoring	PortPrototype	0..1	ref	This is only applicable in SET operations. If it is enabled, the IoHwAbstraction layer will monitor the result of the operation and issue an diagnostic signal. This means especially, that an additional client-server port has to be created. Tools can use this information to cross-check whether for each data-element in a SET operation with FailureMonitoring enabled an additional port is created The referenced port monitors a failure in the to be monitored VariableDataPrototype of the IoHwAbstraction layer. The referenced port has to be another port of the same Actuator or Sensor Component.
filteringDebouncing	FilterDebouncingEnum	1	attr	This attribute is used to indicate what kind of filtering/debouncing has been put to the signal in the IoHwAbstraction layer. rawData means that no modification of the signal has been applied. This is the default value debounceData means that the signal is a mean value waitTimeData means that the signal is delivered by a GET operation after a certain amount of time
pulseTest	PulseTestEnum	1	attr	This attribute indicates to the connected SensorActuatorSwComponentType whether the VariableDataPrototype can be used to generate pulse test sequences using the IoHwAbstraction layer

Table 4.40: IoHwAbstractionServerAnnotation

Enumeration	**FilterDebouncingEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	This enumeration defines possible values for the filter debouncing strategy.
Literal	*Description*
debounceData	The signal is a mean value
rawData	Means that no modification of the signal has been applied. This is the default value

waitTimeDate	The signal is delivered by a GET operation after a certain amount of time

Table 4.41: FilterDebouncingEnum

Enumeration	**PulseTestEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes
Note	This element indicates to the connected Actuator Software component whether the data-element can be used to generate pulse test sequences using the IoHwAbstraction layer
Literal	***Description***
disable	Disables the pulse test
enable	Enables the pulse test

Table 4.42: PulseTestEnum

Within the ECU-Abstraction Layer there are ECU-signals defined. These signals represent the electrical signals as they arrive in the micro-controller peripheral and are fetched from the registers via the MCAL. Access to the I/O Hardware Abstraction Layer is done via service interfaces, i.e. the I/O Hardware Abstraction Layer provides GET- and SET-operations at the specified service ports of a `SensorActuatorSwComponentType`.

[TPS_SWCT_1211] Assign several annotations to `ArgumentDataPrototype` ⌈The `ClientServerOperation`s provide an `ArgumentDataPrototype` where several annotations can be assigned to. They are depicted in the `IoHwAbstractionServerAnnotation` meta-class in Figure 4.21. ⌋

A detailed description of the attributes can be found in the IoHwAbstraction Layer software specification document [15]. For example, the signal age has a very dedicated meaning in this particular interface with respect to a register whereas the signal age in the `SenderReceiverAnnotation` is more generic. Especially, there is no relationship with the micro-controller peripherals.

4.4.5 Parameter Port Annotation

[TPS_SWCT_1212] `ParameterPortAnnotation` ⌈The `ParameterPortAnnotation` can be used to provide more information with respect to calibration parameter prototypes of the port. The data provided at the `PortPrototype` is calibration parameters. The `ParameterPortAnnotation` provides a reference to a particular `ParameterDataPrototype`. ⌋

Class	**ParameterPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain ParameterDataPrototype.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
parameter	ParameterData Prototype	1	ref	The instance of annotated ParameterDataPrototype.

Table 4.43: ParameterPortAnnotation

The main use-case is to allow easy access to the information which calibration parameters influence the data on the `PortPrototype`.

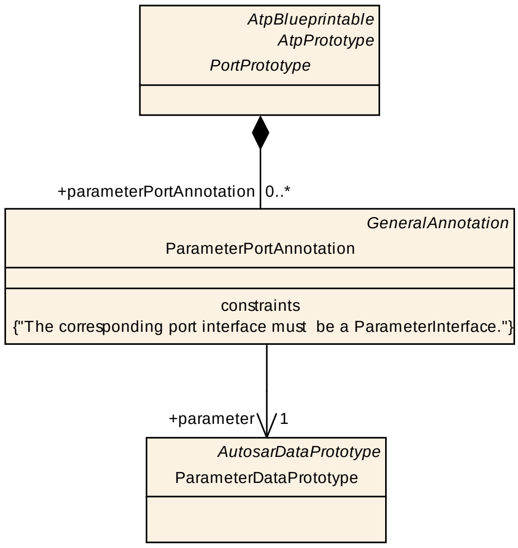

Figure 4.22: `ParameterPortAnnotation`

[constr_4006] Context of `ParameterPortAnnotation` ⌈ A `ParameterPortAnnotation` shall only be aggregated by a `PPortPrototype` owned by a `ParameterSwComponentType`. ⌋

4.4.6 Mode Port Annotation

[TPS_SWCT_1213] `ModePortAnnotation` ⌈ The `ModePortAnnotation` can be used to provide more information with respect to the mode declaration group prototype of the port. ⌋

Class	**ModePortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain ModeDeclarationGroupPrototype.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
modeGroup	ModeDeclarationGroupPrototype	1	ref	The instance of annotated ModeDeclarationGroupPrototype.

Table 4.44: ModePortAnnotation

The main use-case is to allow define additional information related to the mode declaration group prototype.

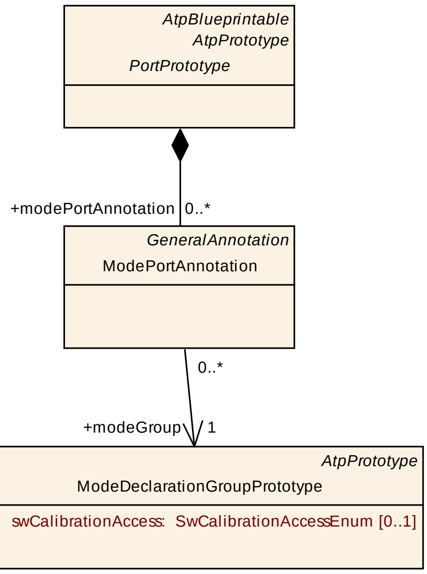

Figure 4.23: `ModePortAnnotation`

[constr_4007] Context of `ModePortAnnotation` ⌈ A `ModePortAnnotation` shall only be aggregated by a `PortPrototype` typed by a `ModeSwitchInterface`. ⌋

4.4.7 Trigger Port Annotation

[TPS_SWCT_1214] `TriggerPortAnnotation` ⌈ The `TriggerPortAnnotation` can be used to provide more information with respect to the trigger of the port. ⌋

Class	TriggerPortAnnotation			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port used for calibration regarding a certain Trigger.			
Base	ARObject,GeneralAnnotation			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
trigger	Trigger	1	ref	The instance of annotated trigger.

Table 4.45: TriggerPortAnnotation

The main use-case is to allow define additional information related to the trigger.

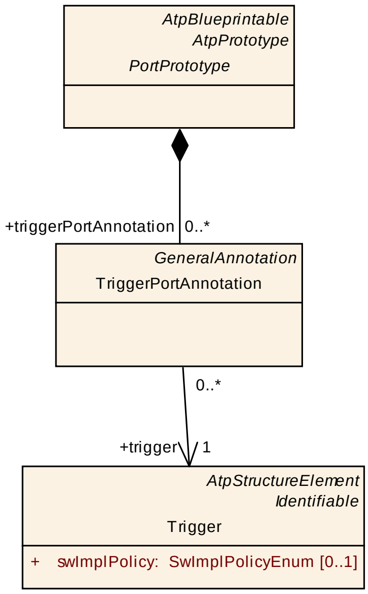

Figure 4.24: **`TriggerPortAnnotation`**

[constr_4008] Context of `TriggerAnnotation` ⌈ A `TriggerAnnotation` shall only be aggregated by a `PortPrototype` typed by a `TriggerInterface`. ⌋

4.4.8 Non Volatile Data Port Annotation

[TPS_SWCT_1215] `NvDataPortAnnotation` ⌈ The `NvDataPortAnnotation` can be used to provide more information with respect to the non volatile data of the port. ⌋

Class	**NvDataPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a port regarding a certain VariableDataPrototype.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
variable	VariableDataPrototype	1	ref	The instance of nv data annotated.

Table 4.46: NvDataPortAnnotation

The main use-case is to allow define additional information related to the non volatile data elements.

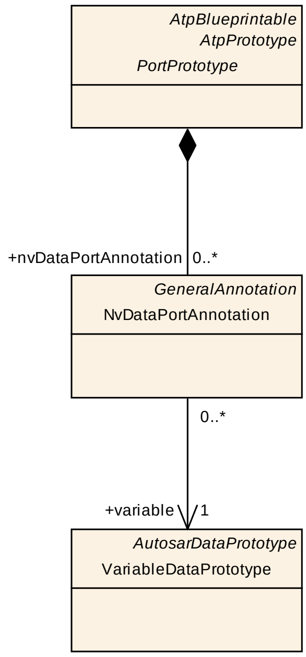

Figure 4.25: NvDataPortAnnotation

[constr_4009] Context of NvDataPortAnnotation ⌈ An NvDataPortAnnotation shall only be aggregated by a PortPrototype typed by an NvDataInterface. ⌋

4.4.9 Delegated Port Annotations

[TPS_SWCT_1216] DelegatedPortAnnotation ⌈ The DelegatedPortAnnotation is used to define the Signal Fan In or Signal Fan Out inside the CompositionSwComponentType. This information is used to pre-define and pre-check resulting communication patterns in the VFB (1:n, n:1, 1:1) if empty CompositionSwComponentTypes are used as interface definition for sub-systems. The DelegatedPortAnnotation guides either the system designer in connecting the empty CompositionSwComponentType or the sub system designer in applying communication pattern (1:n, n:1, 1:1) inside of the CompositionSwComponentType. ⌋

Class	**DelegatedPortAnnotation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::ApplicationAttributes			
Note	Annotation to a "delegated port" to specify the Signal Fan In or Signal Fan Out inside the CompositionSwComponentType.			
Base	ARObject,GeneralAnnotation			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
signalFan	SignalFanEnum	0..1	attr	Specifies the Signal Fan In or Signal Fan Out inside the Composition Type.

Table 4.47: DelegatedPortAnnotation

[TPS_SWCT_1217] Semantics of DelegatedPortAnnotation.signalFan ⌈ The attribute values have following definition:

- **single**: the internal connections in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that each dataElement present in the SenderReceiverInterfaces or operation in the ClientServerInterfaces of the outer PortPrototype is involved in a 1:1 communication pattern only.
- **nfold**: The internal connections in the CompositionSwComponentType via DelegationSwConnectors and AssemblySwConnectors are defined in a way that at least one dataElement present in the SenderReceiverInterfaces or one operation in the ClientServerInterfaces of the outer PortPrototype is involved in a 1:n or n:1 communication pattern.

⌋

[constr_4010] Context of DelegatedPortAnnotation ⌈ A DelegatedPortAnnotation shall only be aggregated by a PortPrototype aggregated by a CompositionSwComponentType. ⌋

4.4.10 General Annotation

Besides formally specified attributes it is also possible to place textual information as provided in the abstract GeneralAnnotation (see Figure 4.26 for an overview).

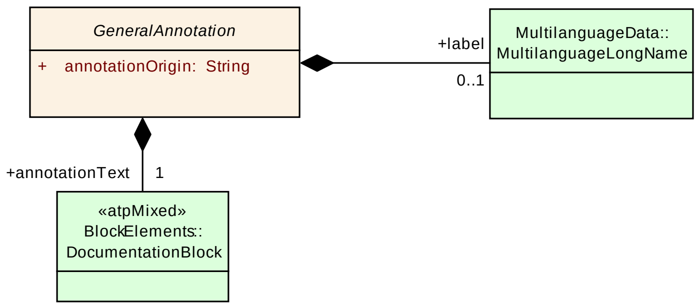

Figure 4.26: textual information in annotations

Class	GeneralAnnotation (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::General Annotation			
Note	This class represents textual comments (called annotations) which relate to the object in which it is aggregated. These annotations are intended for use during the development process for transferring information from one step of the development process to the next one. The approach is similar to the "yellow pads" ... This abstract class can be specialized in order to add some further formal properties.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
annotation Origin	String	1	attr	This attribute identifies the origin of the annotation. It is an arbitrary string since it can be an individual's name as well as the name of a tool or even the name of a process step. **Tags:** xml.sequenceOffset=30
annotation Text	Documentation Block	1	aggr	This is the text of the annotation. **Tags:** xml.sequenceOffset=40
label	MultilanguageL ongName	0..1	aggr	This is the headline for the annotation. **Tags:** xml.sequenceOffset=20

Table 4.48: GeneralAnnotation

4.5 Communication Specification

[TPS_SWCT_1218] Big picture of ComSpec ⌈ The highest level of description of information exchanged between components in an AUTOSAR system is the PortInterfaces, as shown in earlier sections. Such PortInterface however, only describes structure and does not include information about whether communication needs to be done reliably, or whether an initial value exists in case the real data is not yet available.

This information is role-specific, i.e. it shall be applied on the level of PortPrototypes rather than PortInterfaces. Therefore, most communication-relevant attributes are related to the PortPrototypes of an SwComponentType.

The communication attributes are organized in a so-called **communication specification** (in terms of the meta-model: ComSpec) classes. ⌋

Note that the communication specification is optional, i.e. its existence is not required in any case. Figures 4.27 and 4.28 provide an overview of communication specifications. The derived meta-classes are explained in the following sub-chapters.

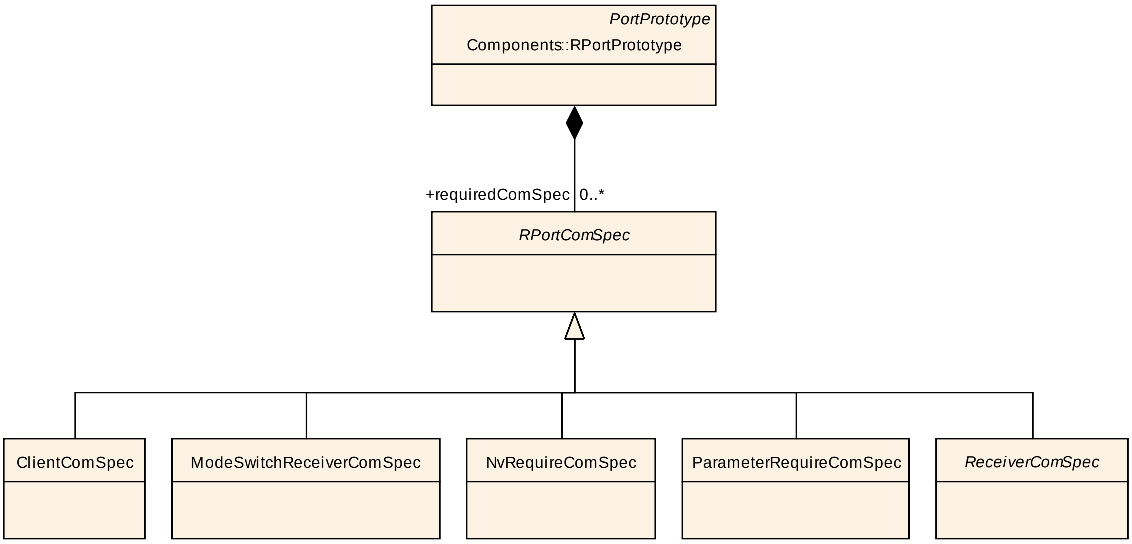

Figure 4.27: Overview of communication attributes of `RPortPrototype`

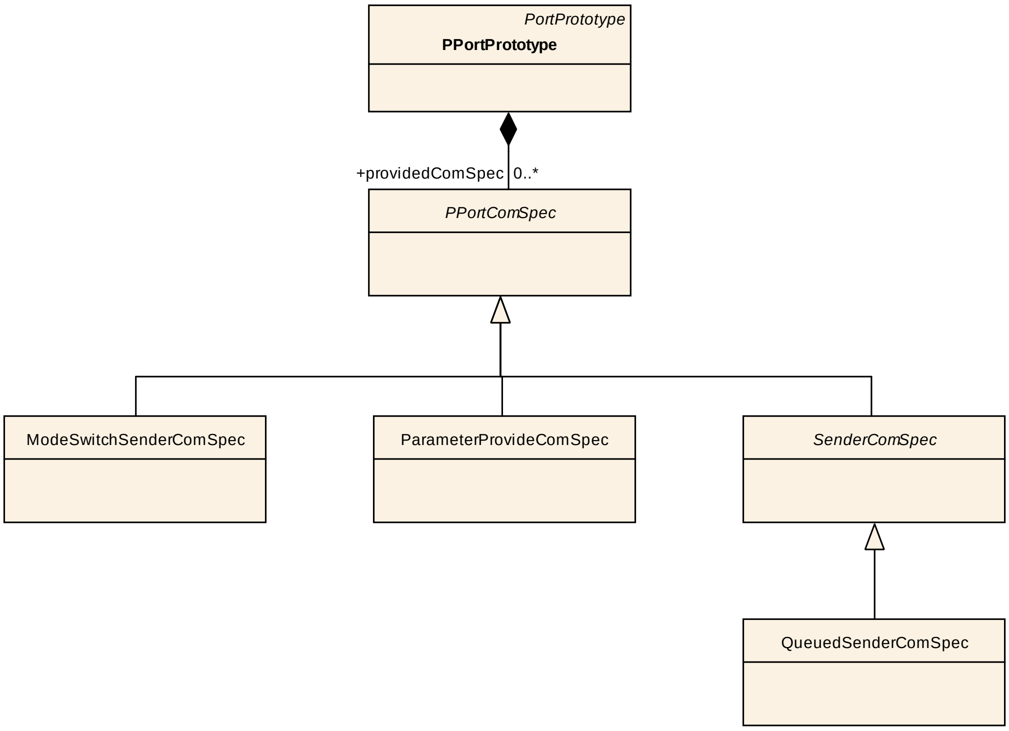

Figure 4.28: Overview of communication attributes of `PPortPrototype`

As explained before, `ComSpec` meta-classes which are required on the level of a `SwComponentType` are attached to the `PortPrototype` declarations which in turn are part of the definition of a `SwComponentType`. Nevertheless, the usage of `ComSpec`s is **not** restricted to the `PortPrototype`s of `AtomicSwComponentType`s (for more details please refer to section 2.5).

Sections 7.5.1 and 7.5.2 then explain the sender-receiver and client-server communication patterns with respect to the RTE, the RTE events and the corresponding communication attributes.

Several ComSpecs allow to define initValues in relation to the associated DataPrototype. For further details about the representation of initValues please refer to section 5.7.2.

Furthermore, semantic constraints apply such that specific subclasses of ComSpec can only be owned by PortPrototypes typed by the corresponding kind of PortInterface.

[constr_1043] PortInterface vs. ComSpec ⌈ In particular, the following correspondence applies:

PortInterface	ComSpec
SenderReceiverInterface	SenderComSpec, ReceiverComSpec
ClientServerInterface	ClientComSpec, ServerComSpec
ModeSwitchInterface	ModeSwitchComSpec
ParameterInterface	ParameterProvideComSpec, ParameterRequireComSpec
NvDataInterface	NvRequireComSpec, NvProvideComSpec

Table 4.49: PortInterface vs. ComSpec

⌋

As explained in section 2.5, there are cases where PortPrototypes owned by a CompositionSwComponentType could have initValues.

Therefore, it is possible that PortPrototypes owned by CompositionSwComponentTypes can have ComSpecs. It is *not* required that the ComSpecs defined on the composition level match the ComSpecs defined inside the CompositionSwComponentType.

If consistency would be required this constraint might be a major obstacle for integrating existing AtomicSwComponentTypes into a CompositionSwComponentType that has PortPrototypes with ComSpecs.

4.5.1 Communication Specification for Sender-Receiver Communication

Communication specification applies in different ways to specific kinds of communication. Figure 4.29 shows the meta-model of the communication attributes relevant sender-receiver communication at an RPortPrototype.

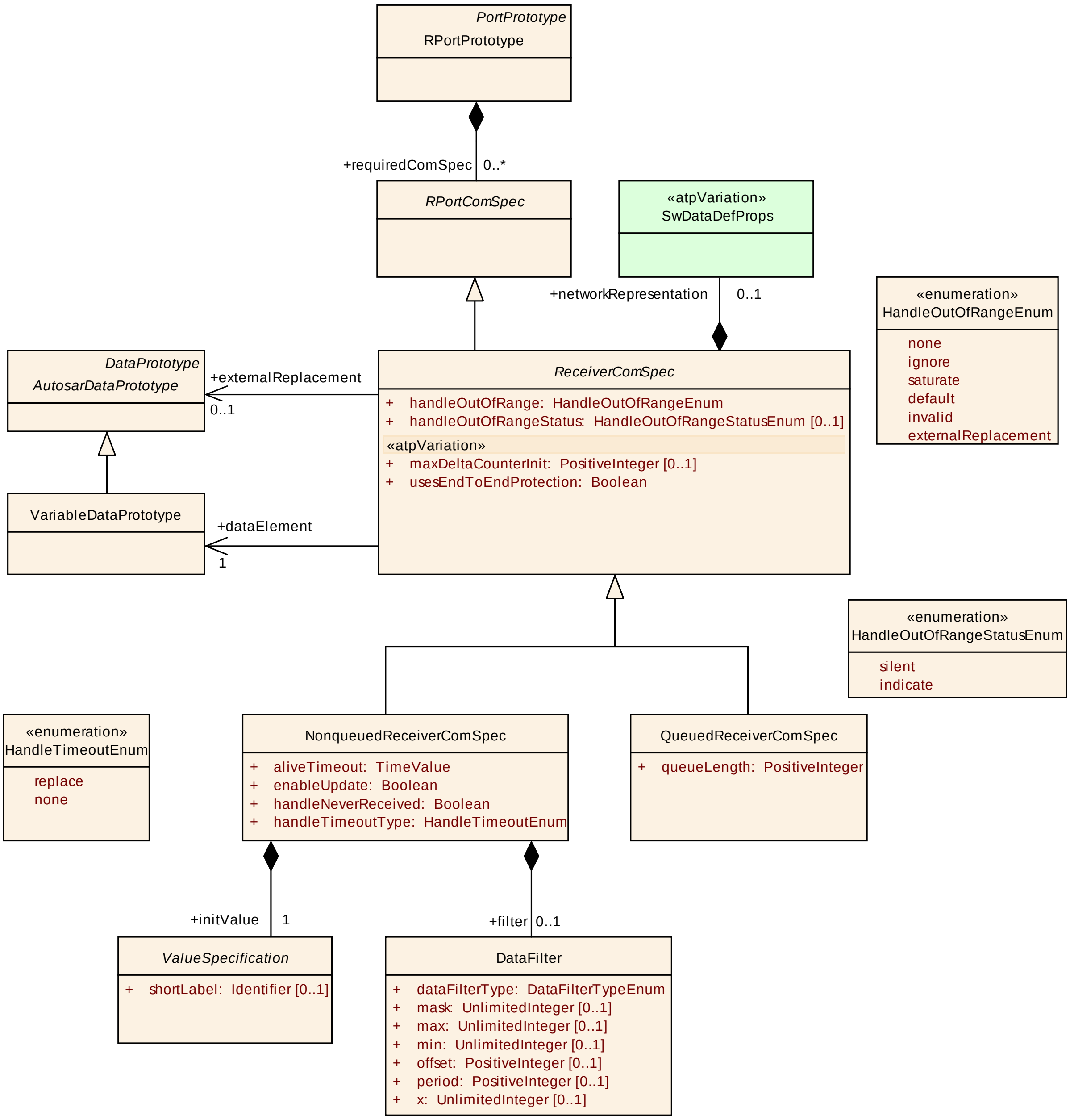

Figure 4.29: Communication attributes of `RPortPrototype` with respect to sender-receiver communication.

[TPS_SWCT_1219] `ComSpec` for queued and non-queued sender-receiver communication ⌈ Sender-receiver communication might be queued or non-queued. This aspect is reflected in the specification of the applicable `ComSpec` meta-classes. While the case of queued communication the `queueLength` attribute remains the only information item the non-queued case foresees several attributes for controlling communication behavior. ⌋

Class	**ReceiverComSpec (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Receiver-specific communication attributes (RPortPrototype typed by SenderReceiverInterface).			
Base	ARObject,RPortComSpec			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataElement	VariableDataPrototype	1	ref	Data element these attributes belong to.
externalReplacement	AutosarDataPrototype	0..1	ref	This reference is used to reference the AutosarDataPrototype to be taken for sourcing an external replacement in the out-of-range handling.
handleOutOfRange	HandleOutOfRangeEnum	1	attr	This attribute controls how values that are out of the specified range are handled according to the values of HandleOutOfRangeEnum.
handleOutOfRangeStatus	HandleOutOfRangeStatusEnum	0..1	attr	Control the way how return values are created in case of an out-of-range situation.
maxDeltaCounterInit	PositiveInteger	0..1	attr	Initial maximum allowed gap between two counter values of two consecutively received valid Data, i.e. how many subsequent lost data is accepted. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType.
usesEndToEndProtection	Boolean	1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 4.50: ReceiverComSpec

Class	**NonqueuedReceiverComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to non-queued receiving.			
Base	ARObject,RPortComSpec,ReceiverComSpec			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
aliveTimeout	TimeValue	1	attr	Specify the amount of time (in seconds) after which the software component (via the RTE) needs to be notified if the corresponding data item have not been received according to the specified timing description. If the aliveTimeout attribute is 0 no timeout monitoring shall be performed.
enableUpdate	Boolean	1	attr	This attribute controls whether application code is entitled to check whether the value of the corresponding VariableDataPrototype has been updated.
filter	DataFilter	0..1	aggr	The applicable filter algorithm for filtering the value of the corresponding dataElement.
handleNeverReceived	Boolean	1	attr	This attribute specifies whether for the corresponding VariableDataPrototype the "never received" flag is available. If yes, the RTE is supposed to assume that initially the VariableDataPrototype has not been received before. After the first reception of the corresponding VariableDataPrototype the flag is cleared. If the value of this attribute is set to TRUE the flag is required. If set to FALSE, the RTE shall not support the "never received" functionality for the corresponding VariableDataPrototype.
handleTimeoutType	HandleTimeoutEnum	1	attr	This attribute controls the behavior with respect to the handling of timeouts.
initValue	ValueSpecification	1	aggr	Initial value to be used in case the sending component is not yet initialized. If the sender also specifies an initial value the receiver's value will be used.

Table 4.51: NonqueuedReceiverComSpec

Class	**QueuedReceiverComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to queued receiving.			
Base	ARObject,RPortComSpec,ReceiverComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
queueLength	PositiveInteger	1	attr	Length of queue for received events.

Table 4.52: QueuedReceiverComSpec

Enumeration	**HandleTimeoutEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	Strategies of handling a reception timeout violation.
Literal	***Description***
none	If set to none no replacement shall take place.

replace	If set to replace, the replacement value used shall be the ComInitValue.

Table 4.53: HandleTimeoutEnum

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE; xml.xsd.type=double

Table 4.54: TimeValue

[constr_1103] `NonqueuedReceiverComSpec` and `enableUpdate` ⌈ A `NonqueuedReceiverComSpec` that has attribute `enableUpdate` set to `true` may not reference a `dataElement` that in turn is referenced by a `VariableAccess` in the role `dataReadAccess`. ⌋

[constr_1129] `swImplPolicy` and `NonqueuedReceiverComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `NonqueuedReceiverComSpec` **shall not** be set to the value `queued`. ⌋

[constr_1130] `swImplPolicy` and `NonqueuedReceiverComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `QueuedReceiverComSpec` **shall** be set to the value `queued`. ⌋

[constr_1188] Existence of `externalReplacement` ⌈ The reference `externalReplacement` shall exist if and only if the value of the attribute `handleOutOfRange` is set to `externalReplacement`. ⌋

[constr_1189] Allowed targets of `externalReplacement` ⌈ The reference `externalReplacement` shall only point to either a `VariableDataPrototype` or a `ParameterDataPrototype` ⌋

[constr_1131] `swImplPolicy` and `NonqueuedSenderComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `NonqueuedSenderComSpec` **shall not** be set to the value `queued`. ⌋

[constr_1132] `swImplPolicy` and `NonqueuedSenderComSpec` ⌈ The attribute `swImplPolicy` of a `dataElement` referenced by a `QueuedSenderComSpec` **shall** be set to the value `queued`. ⌋

[TPS_SWCT_1220] `initValue` defines an initial value that shall be taken if the corresponding `dataElement` has not yet been received ⌈ The aggregation of `ValueSpecification` in the role `initValue` defines an initial value that shall be taken if the corresponding `dataElement` has not yet been received but the application software is attempting to access its value.

This is the only relevant definition of an initial value for data transmission. That is, any `initValue` defined in the context of `VariableDataPrototype` is ignored! ⌋

The communication attributes on the sender side are sketched in Figure 4.31.

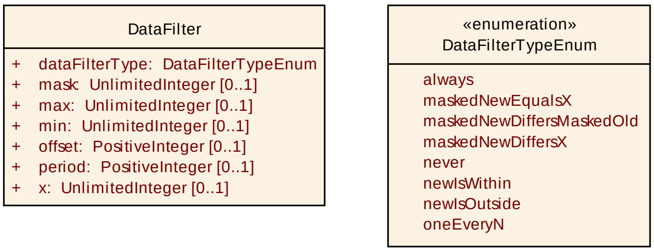

Figure 4.30: `DataFilter` and its communication attributes.

Figure 4.30 shows the model of the communication attributes relevant for defining data filters.

[TPS_SWCT_1221] `DataFilter` ⌈ For every `RPortPrototype` typed by a `SenderReceiverInterface` a `DataFilter` can be defined given that non-queued communication is foreseen. ⌋

Fifteen filter algorithms formally described by the enumeration type `DataFilterTypeEnum` in the meta-model are taken from OSEK COM 3.0.3 specification [16] that is referenced by the RTE specification [2].

[TPS_SWCT_1222] Applicability of `DataFilter` ⌈ This OSEK specification states that "filtering is only used for messages that can be interpreted as C language unsigned integer types (characters, unsigned integers and enumerations)." ⌋

[constr_1044] Applicability of `DataFilter` ⌈ According to the origin of `DataFilter`, i.e. OSEK COM 3.0.3 specification [16], `DataFilter`s can only be applied to values with an integer base type. ⌋

Class	**DataFilter**			
Package	M2::AUTOSARTemplates::CommonStructure::Filter			
Note	Base class for data filters. The type of the filter is specified in attribute dataFilterType. Some of the filter types require additional arguments which are specified as attributes of this class.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
dataFilterType	DataFilterTypeEnum	1	attr	This attribute specifies the type of the filter.
mask	UnlimitedInteger	0..1	attr	Mask for old and new value
max	UnlimitedInteger	0..1	attr	Value to specify the upper boundary
min	UnlimitedInteger	0..1	attr	Value to specify the lower boundary
offset	PositiveInteger	0..1	attr	Specifies the initial number of messages to occur before the first message is passed
period	PositiveInteger	0..1	attr	Specifies number of messages to occur before the message is passed again
x	UnlimitedInteger	0..1	attr	Value to compare with

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

Table 4.55: DataFilter

Enumeration	**DataFilterTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::Filter
Note	This enum specifies the supported DataFilterTypes.
Literal	***Description***
always	No filtering is performed so that the message always passes.
masked NewDiffers MaskedOld	Pass messages where the masked value has changed. (new_value&mask) !=(old_value&mask) new_value: current value of the message old_value: last value of the message (initialized with the initial value of the message, updated with new_value if the new message value is not filtered out)
maskedNew DiffersX	Pass messages whose masked value is not equal to a specific value x (new_value&mask) != x new_value: current value of the message
maskedNew EqualsX	Pass messages whose masked value is equal to a specific value x (new_value&mask) == x new_value: current value of the message
never	The filter removes all messages.
newIsOutside	Pass a message if its value is outside a predefined boundary. (min > new_value) OR (new_value > max)
newIsWithin	Pass a message if its value is within a predefined boundary. min <= new_value <= max
oneEveryN	Pass a message once every N message occurrences. Algorithm: occurrence % period == offset Start: occurrence = 0. Each time the message is received or transmitted, occurrence is incremented by 1 after filtering. Length of occurrence is 8 bit (minimum).

Table 4.56: DataFilterTypeEnum

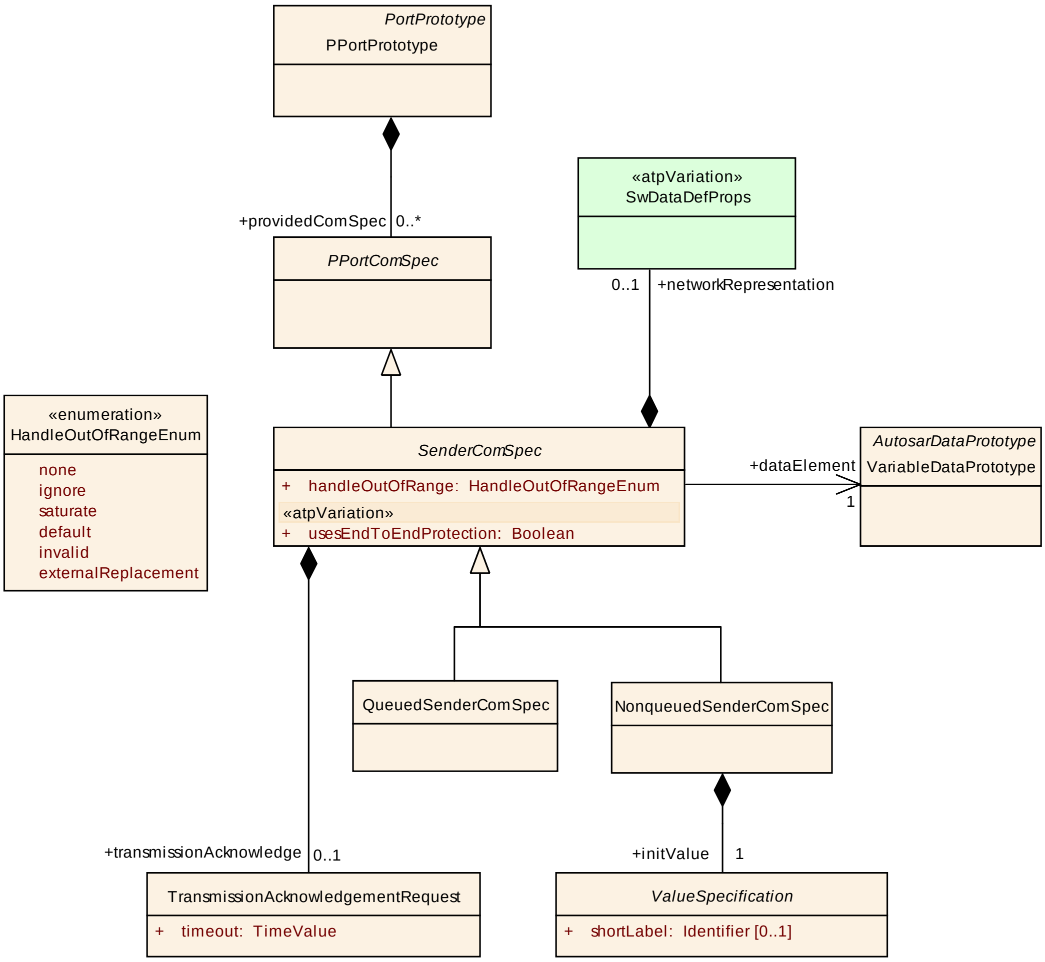

Figure 4.31: Communication attributes of `PPortPrototype` with respect to sender-receiver communication.

Class	**SenderComSpec (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a sender port (PPortPrototype typed by SenderReceiverInterface).			
Base	ARObject,PPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataElement	VariableDataPrototype	1	ref	Data element these quality of service attributes apply to.
handleOutOfRange	HandleOutOfRangeEnum	1	attr	This attribute controls how out-of-range values shall be dealt with.
networkRepresentation	SwDataDefProps	0..1	aggr	A networkRepresentation is used to define how the dataElement is mapped to a communication bus. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType.
transmissionAcknowledge	TransmissionAcknowledgementRequest	0..1	aggr	Requested transmission acknowledgement for data element.

Attribute	Datatype	Mul.	Kind	Note
usesEndToEndProtection	Boolean	1	attr	This indicates whether the corresponding dataElement shall be transmitted using end-to-end protection. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 4.57: SenderComSpec

Class	**QueuedSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes specific to distribution of events (PPortPrototype, SenderReceiverInterface and dataElement carries an "event").			
Base	ARObject,PPortComSpec,SenderComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 4.58: QueuedSenderComSpec

Class	**NonqueuedSenderComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for non-queued sender/receiver communication (sender side)			
Base	ARObject,PPortComSpec,SenderComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
initValue	ValueSpecification	1	aggr	Initial value to be sent if sender component is not yet fully initialized, but receiver needs data already.

Table 4.59: NonqueuedSenderComSpec

Class	**TransmissionAcknowledgementRequest**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests transmission acknowledgement that data has been sent successfully. Success/failure is reported via a SendPoint of a RunnableEntity.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
timeout	TimeValue	1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table 4.60: TransmissionAcknowledgementRequest

Enumeration	**HandleOutOfRangeEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication
Note	A value of this type is taken for controlling the range checking behavior of the AUTOSAR RTE.

Literal	*Description*
default	The RTE will use the initValue if the actual value is out of the specified bounds.
external Replacement	This indicates that the value replacement is sourced from the externalReplacement.
ignore	The RTE will ignore any attempt to send or receive the corresponding dataElement if the value is out of the specified range.
invalid	The RTE will use the invalidValue if the value is out of the specified bounds.
none	A range check is not required.
saturate	The RTE will saturate the value of the dataElement such that it is limited to the applicable upper bound if it is greater than the upper bound. Consequently, it is limited to the applicable lower bound if the value is less than the lower bound.

Table 4.61: HandleOutOfRangeEnum

[TPS_SWCT_1223] `networkRepresentation` defines how a specific `dataElement` is represented on a communication bus ⌈ For sender-receiver communication, it is possible to specify how `dataElement`s are represented given that the communication requires the usage of a dedicated communication bus.

That is, by means of the `networkRepresentation` it is possible to define how a specific `dataElement` is represented on a communication bus. For this purpose the `networkRepresentation` is implemented as an aggregation of `SwDataDefProps`.
⌋

[TPS_SWCT_1224] `CompuMethod`s of `dataElement` and the `networkRepresentation` are used for conversion purposes ⌈ The attached `CompuMethod`s of both the `dataElement` and the `networkRepresentation` can be used to identify the conversion between the two. The advantage of this approach is that this can also be used without any modifications in combination with a general remapping and rescaling of `dataElement`s between different `SwComponentType`s, regardless whether they are located on the same or on different ECUs. ⌋

Please note that the decision whether or not to take the `networkRepresentation` for data mapping is done in the context of the AUTOSAR System Template [11]. Please find more detailed information about this aspect in the applicable specification.

4.5.2 Communication Specification for Client-Server Communication

The communication aspects relevant for client communication are sketched in Figure 4.32.

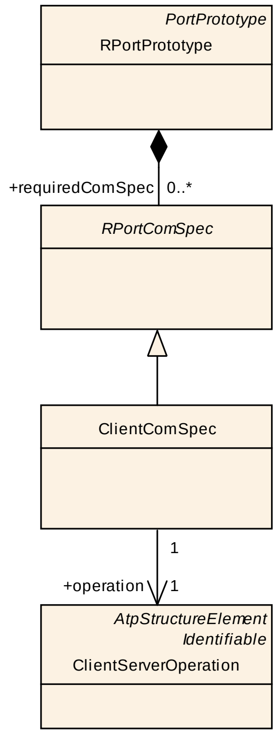

Figure 4.32: Communication attributes of `RPortPrototype` with respect to client-server communication.

Class	**ClientComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Client-specific communication attributes (RPortPrototype typed by ClientServerInterface).			
Base	ARObject,RPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1	ref	This represents the corresponding ClientServerOperation.

Table 4.62: ClientComSpec

The server side looks very similar but provides an attribute for specifying the queue length.

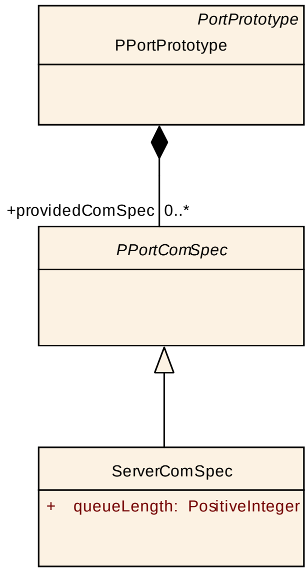

Figure 4.33: Communication attributes of PPortPrototype with respect to client-server communication.

Class	**ServerComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes for a server port (PPortPrototype and ClientServerInterface).			
Base	ARObject,PPortComSpec			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
operation	ClientServerOperation	1	ref	Operation these communication attributes apply to.
queueLength	PositiveInteger	1	attr	Length of call queue on the mode user side. The queue is implemented by the RTE. The value must be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.

Table 4.63: ServerComSpec

[TPS_SWCT_1225] `RunnableEntity` implements the functionality of two or more `ClientServerOperations` ⌈ Please note that it is technically possible to let a single `RunnableEntity` implement the functionality of two or more `ClientServerOperations`. For this purpose two or more `OperationInvokedEvents` need to reference this single `RunnableEntity`.

In this case, however, it is essential that the queue length associated with each of the `ClientServerOperations` has the same value. In other words: ⌋

[constr_1128] Queue length of `ClientServerOperations` associated with the same `RunnableEntity` ⌈ If two or more `OperationInvokedEvents` reference a single `RunnableEntity` the value of the `ServerComSpec` attribute `queueLength` shall be **identical** for all `ServerComSpecs` owned by `PPortPrototypes` of the enclosing SwComponentType that reference one of the ClientServerOperations that are also referenced by the OperationInvokedEvents. ⌋

4.5.3 Communication Specification for Mode Switch Communication

In analogy to the previous section, Figure 4.34 shows the meta-model elements relevant for a mode switch communication. On the sender side it is possible to specify that an acknowledgment is supposed to be returned that indicates the successful processing of the mode switch request.

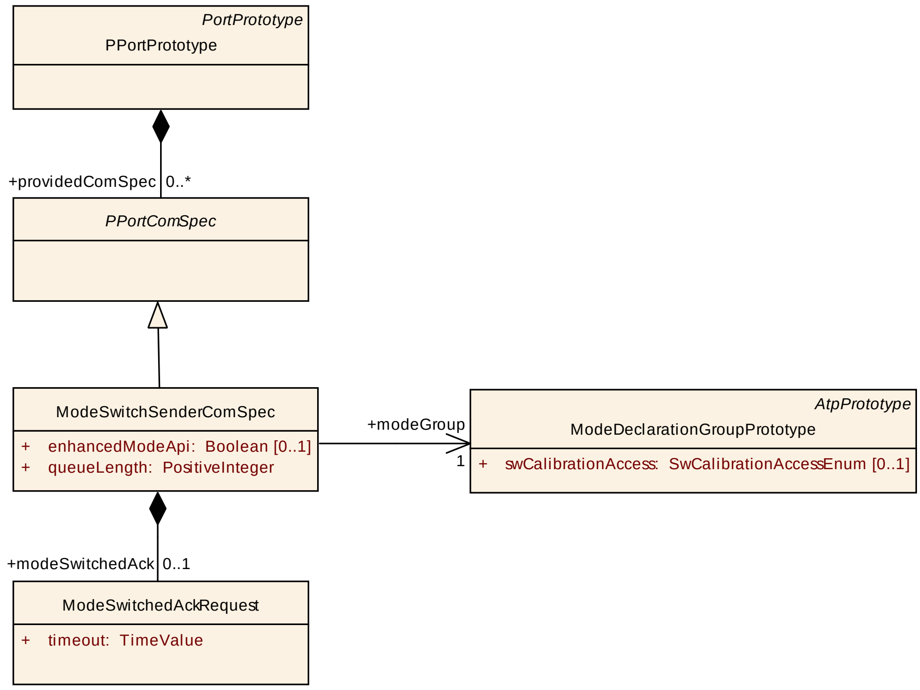

Figure 4.34: Communication attributes of PPortPrototype with respect to mode switch communication.

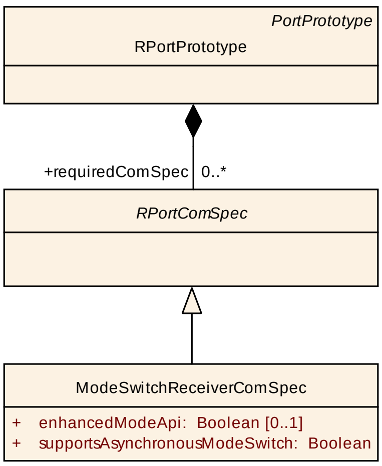

Figure 4.35: Communication attributes of PPortPrototype with respect to mode switch communication.

Class	ModeSwitchSenderComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of PPortPrototypes with respect to mode communication			
Base	ARObject,PPortComSpec			
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.
modeGroup	ModeDeclarationGroupPrototype	1	ref	Mode Declaration Group (of the same Port Interface) to which these communication attributes apply.
modeSwitchedAck	ModeSwitchedAckRequest	0..1	aggr	If this aggregation exists an acknowledgement for the successful processing of the mode switch request is required.
queueLength	PositiveInteger	1	attr	Length of call queue on the mode user side. The queue is implemented by the RTE. The value must be greater or equal to 1. Setting the value of queueLength to 1 implies that incoming requests are rejected while another request that arrived earlier is being processed.

Table 4.64: ModeSwitchSenderComSpec

Class	ModeSwitchedAckRequest			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests acknowledgements that a mode switch has been proceeded successfully			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
timeout	TimeValue	1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table 4.65: ModeSwitchedAckRequest

Class	ModeSwitchReceiverComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to mode communication			
Base	ARObject,RPortComSpec			
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.

Attribute	Datatype	Mul.	Kind	Note
supportsAsynchronousModeSwitch	Boolean	1	attr	This attribute controls the behavior of the corresponding RPortPrototype with respect to the question whether it can deal with asynchronous mode switch requests, i.e. if set to true, the RPortPrototype is able to deal with an asynchronous mode switch request.

Table 4.66: ModeSwitchReceiverComSpec

4.5.4 Communication Specification for Parameters

Granted, the definition of a `ComSpec` for `ParameterDataPrototype`s looks strange on first sight. A `ParameterDataPrototype` owned by a `PPortPrototype` typed by a `ParameterInterface` is not actually transmitted over any communication medium. Therefore, the term *communication* should in this case be taken with a grain of salt.

However, it is generally necessary to be able to define role-specific initial values for `ParameterDataPrototype`s aggregated in a `ParameterInterface`. In other words, the actual problem closely resembles the definition of initial values in the case of sender-receiver communication.

[TPS_SWCT_1226] `initValue` on the level of a `ComSpec` is relevant for connections to the corresponding `PortPrototype` ⌈ Please note that (along the example of sender-receiver communication) only the `initValue` defined in the context of a `ParameterProvideComSpec` or `ParameterRequireComSpec` is relevant for connections to the corresponding `PortPrototype`. An `initValue` defined in the scope of a `ParameterDataPrototype` is ignored. ⌋

Therefore, it is only reasonable to apply the existing and well-known pattern to the definition of initial values for `ParameterDataPrototype`s aggregated in a `ParameterInterface`. The actual modeling is sketched in Figure 4.36 for provided `ParameterDataPrototype`s and in Figure 4.37 for required `ParameterDataPrototype`s.

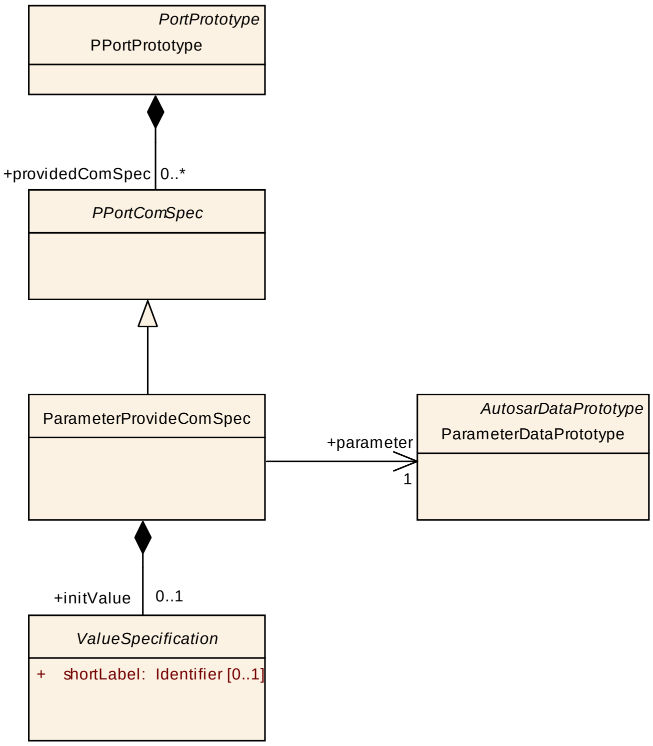

Figure 4.36: Communication attributes of `ParameterDataPrototypes` with respect to `PPortPrototype`

Class	**ParameterProvideComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the provided side of a connection.			
Base	ARObject,PPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	1	ref	The ParameterDataPrototype to which the ParameterComSpec applies.

Table 4.67: ParameterProvideComSpec

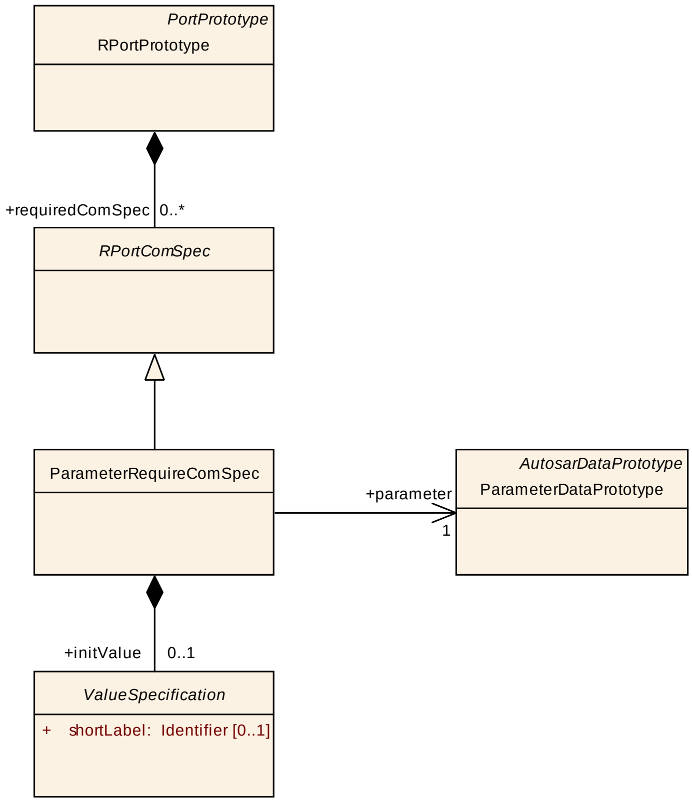

Figure 4.37: Communication attributes of `ParameterDataPrototypes` with respect to `RPortPrototype`

Class	**ParameterRequireComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	"Communication" specification that applies to parameters on the required side of a connection.			
Base	ARObject,RPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	The initial value applicable for the corresponding ParameterDataPrototype.
parameter	ParameterDataPrototype	1	ref	The ParameterDataPrototype to which the ParameterRequireComSpec applies.

Table 4.68: ParameterRequireComSpec

4.5.5 Communication Specification for NV Data

[TPS_SWCT_1141] `AtomicSwComponentType` may have `RPortPrototypes` typed by an `NvDataInterface` ⌈ An `AtomicSwComponentType` may have `RPortPrototype`s typed by an `NvDataInterface`. If such an `RPortPrototype` remains unconnected the `nvData` still need to have reasonable value[5]. ⌋*(RS_SWCT_3225)*

[5]Note that it is assumed that only a subset of meta-classes that inherit from `AtomicSwComponentType` will actually apply for the definition of initial values for `nvData`. Most likely the `Application-`

[TPS_SWCT_1227] Unconnected `RPortPrototype` typed by `NvDataInterface`
⌈ For this purpose it is possible to let the `RPortPrototype` own an `NvRequireComSpec` that in turn owns a `ValueSpecification` in the role of `initValue`.

It is therefore possible to provide an `nvData` with a reasonable value even if the corresponding `RPortPrototype` remains unconnected. ⌋*(RS_SWCT_3225)*

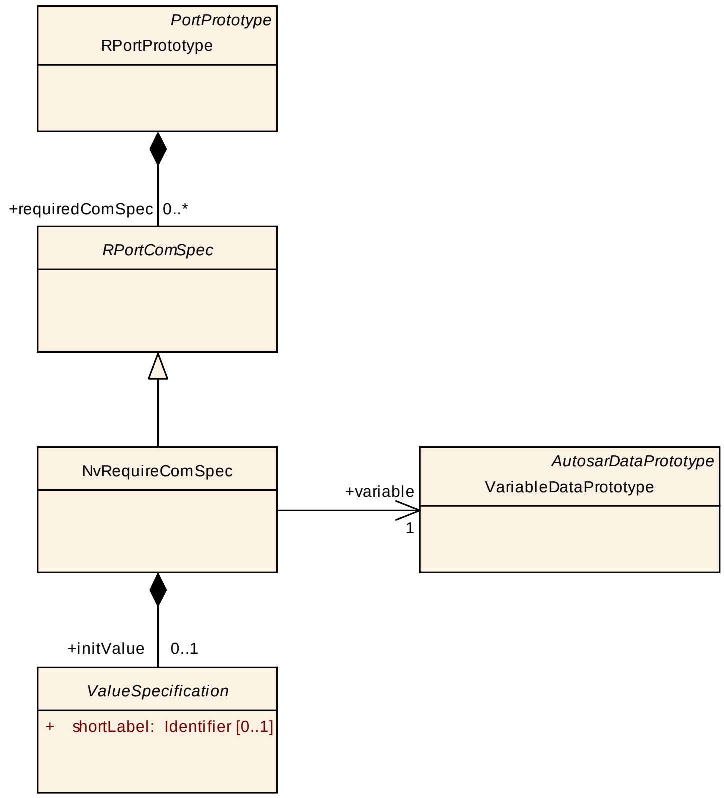

Figure 4.38: Communication attributes of a required `VariableDataPrototypes` used in the context of an `NvDataInterface`

Please note that (along the example of sender-receiver communication, see [TPS_SWCT_1226]) only the `initValue` defined in the context of a `NvRequireComSpec` is relevant for connections to the corresponding `PortPrototype`. An `initValue` defined in the scope of a `VariableDataPrototype` is ignored.

Class	**NvRequireComSpec**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to Nv data communication on the required side.			
Base	ARObject,RPortComSpec			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	The initial value owned by the NvComSpec
variable	VariableDataPrototype	1	ref	The VariableDataPrototype the ComSpec applies for.

`SwComponentType` and the `SensorActuatorSwComponentType` will be candidates for using this feature but it will obviously not be reasonable for e.g. `NvBlockComponentType`.

Attribute	Datatype	Mul.	Kind	Note

Table 4.69: NvRequireComSpec

[TPS_SWCT_1228] NvProvideComSpec ⌈ As communication with an NvBlockSwComponentType is in most cases bi-directional it is also necessary to consider role-specific communication attributes for PPortPrototypes typed by an NvDataInterface. For this purpose the NvProvideComSpec (see Figure 4.39) is defined.

The main purpose of this kind of ComSpec is the definition of initial values for the RAM block and the ROM block that corresponds to an nvData defined in the context of the NvDataInterface used to type the given PPortPrototype. ⌋*(RS_SWCT_3225)*

Note that these initial values can be taken as an input for designing an NvBlockSwComponentType, in particular the ramBlocks and romBlocks of NvBlockDescriptors owned by the NvBlockSwComponentType. Further details are explained in Figure 11.6.

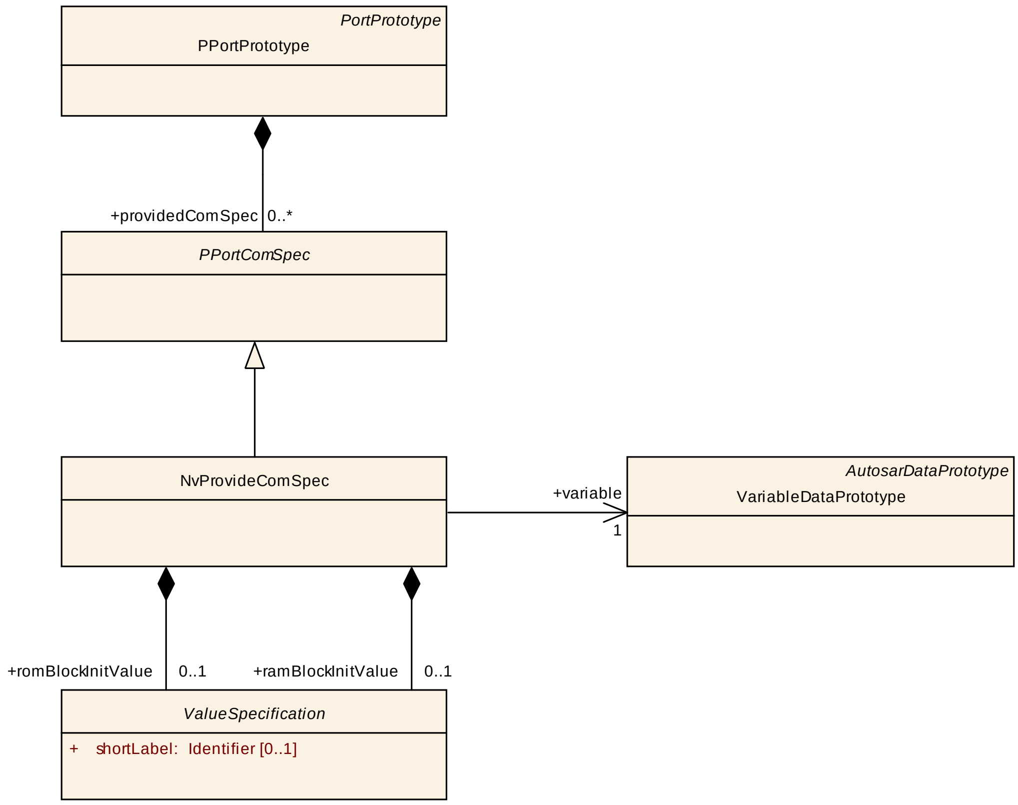

Figure 4.39: Communication attributes of a provided VariableDataPrototypes used in the context of an NvDataInterface

In other words, by means of the NvProvideComSpec the author of an ApplicationSwComponentType can express detailed requirements on the later design of a corresponding NvBlockSwComponentType.

Class	NvProvideComSpec			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Communication attributes of RPortPrototypes with respect to Nv data communication on the provided side.			
Base	ARObject,PPortComSpec			
Attribute	Datatype	Mul.	Kind	Note
ramBlockInitValue	ValueSpecification	0..1	aggr	This represents the initial value of the RAM block that corresponds to the referenced variable.
romBlockInitValue	ValueSpecification	0..1	aggr	This represents the initial value of the ROM block that corresponds to the referenced variable.
variable	VariableDataPrototype	1	ref	This represents the variable for which the ComSpec is specified.

Table 4.70: NvProvideComSpec

4.6 Port Groups within Component Types

[TPS_SWCT_1063] **`PortGroup`** ⌈ A `SwComponentType` can declare that some of its `PortPrototype`s belong to a `PortGroup`. Such a port group defines a logical grouping of `PortPrototype`s which is used as input to configure the implementation of mode managers in the basic software, for example the communication of bus signals associated with the grouped ports maybe suppressed in a certain mode. ⌋*(RS_SWCT_3200)*

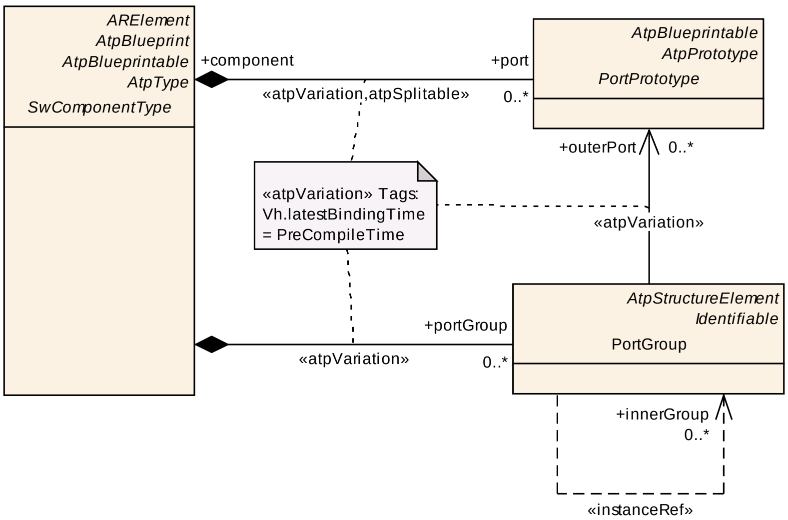

Figure 4.40: Declaration of `PortGroups`

Class	PortGroup			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	Group of ports which share a common functionality, e.g. need specific network resources. This information shall be available on the VFB level in order to delegate it properly via compositions. When propagated into the ECU extract, this information is used as input for the configuration of Services like the Communication Manager. A PortGroup is defined locally in a component (which can be a composition) and refers to the "outer" ports belonging to the group as well as to the "inner" groups which propagate this group into the components which are part of a composition. A PortGroup within an atomic SWC cannot be linked to inner groups.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
innerGroup	PortGroup	*	iref	Links a PortGroup in a composition to another PortGroup, that is defined in a component which is part of this CompositionSwComponentType.
outerPort	PortPrototype	*	ref	Outer port of this component which belongs to the group. A port can belong to several groups or to no group at all. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 4.71: PortGroup

[TPS_SWCT_1064] `PortGroups` have to be defined on the VFB level ⌈ Though the declaration `PortGroups` is not relevant for the RTE, they have to be defined on the VFB level, because they represent design decisions taken on this level. Accordingly, `PortGroups` can be defined for `CompositionSwComponentTypes` as well as for `AtomicSwComponentTypes`. ⌋ *(RS_SWCT_3200)*

[TPS_SWCT_1065] `PortPrototype` may belong to more than one `PortGroups` ⌈ A `PortPrototype` may belong to more than one `PortGroups` and `PortGroups` can be associated with the "inner" `PortGroups` of `SwComponentPrototypes` which are aggregated by the same `SwComponentType` as the `PortGroup`. By this, `PortGroups` can be locally defined but still traced down the component hierarchy. ⌋ *(RS_SWCT_3200)*

[TPS_SWCT_1066] `PortGroups` can be associated with certain `ServiceNeeds` ⌈ `PortGroups` can be associated with certain `ServiceNeeds` in order to trace the information down to the configuration of the basic software, for details see chapter 7.11.2. ⌋ *(RS_SWCT_3200)*

[constr_1147] Standardized values for the attribute `category` of meta-class `PortGroup` ⌈

The following values of the attribute `category` of meta-class `PortGroup` are reserved by the AUTOSAR standard:

- `MODE_MANAGEMENT`: This represents the usage of the `PortGroup` for the purpose of mode management

- PARTIAL_NETWORKING: This represents the usage of the PortGroup for the purpose of partial networking

⌋

4.7 End to End Protection

As described in [17] there are cases where safety-related software-components protect the data exchanged between each other. For this purpose modeling support is provided by the software-component template.

Note that several end-to-end profiles are selectable for a specific application. The specific end-to-end profile is represented by the attribute category of meta-class EndToEndDescription.

Semantically, the category value represents an identification of the specific end-to-end profile applicable for the communication of the corresponding data element. According to [17] there are two pre-defined profiles that can be used.

[TPS_SWCT_1089] end-to-end communication protection ⌈ The information specific to each profile is expressed by the set of attributes of EndToEndDescription owned by EndToEndProtection in the role endToEndProfile. ⌋*(RS_SWCT_3240)*

Class	**EndToEndDescription**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This meta-class contains information about end-to-end protection. The set of applicable attributes depends on the actual value of the category attribute of EndToEndProtection.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
category	NameToken	1	attr	The category represents the identification of the concrete E2E profile. The applicable values are specified in a semantic constraint and determine the applicable attributes of EndToEndDescription. **Tags:** xml.sequenceOffset=-100
counterOffset	PositiveInteger	0..1	attr	Bit offset of Counter from the beginning of the Array representation of the Signal Group/VariableDataPrototype (MSB order, bit numbering: bit 0 is the least important). The offset shall be a multiplicity of 4 and it should be 8 whenever possible. For example, offset 8 means that the counter will take the low nibble of the byte 1, i.e. bits 8 .. 11. If counterOffset is not present the value is defined by the selected profile. **Tags:** xml.sequenceOffset=-50

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
crcOffset	PositiveInteger	0..1	attr	Bit offset of CRC from the beginning of the Array representation of the Signal Group/VariableDataPrototype (MSB order, bit numbering: bit 0 is the least important). The offset shall be a multiplicity of 8 and it should be 0 whenever possible. For example, offset 8 means that the CRC will take the byte 1, i.e. bits 8..15. If crcOffset is not present the value is defined by the selected profile. **Tags:** xml.sequenceOffset=-60
dataId (ordered)	PositiveInteger	*	attr	This represents a unique numerical identifier. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEndProtection. **Tags:** xml.sequenceOffset=-90
dataIdMode	PositiveInteger	0..1	attr	There are three inclusion modes how the implicit two-byte Data ID is included in the one-byte CRC: • dataIDMode = 0: Two bytes are included in the CRC (double ID configuration) This is used in variant 1A. • dataIDMode = 1: One of the two bytes byte is included, alternating high and low byte, depending on parity of the counter (alternating ID configuration). For even counter low byte is included; For odd counters the high byte is included. This is used in variant 1B. • dataIDMode = 2: Only low byte is included, high byte is never used. This is applicable if the IDs in a particular system are 8 bits. **Tags:** xml.sequenceOffset=-85
dataLength	PositiveInteger	0..1	attr	This attribute represents the length of the Array representation of the Signal Group/VariableDataPrototype including CRC and Counter in bits. **Tags:** xml.sequenceOffset=-80

Attribute	Datatype	Mul.	Kind	Note
maxDeltaCounterInit	PositiveInteger	0..1	attr	Initial maximum allowed gap between two counter values of two consecutively received valid Data, i.e. how many subsequent lost data is accepted. For example, if the receiver gets Data with counter 1 and MaxDeltaCounterInit is 1, then at the next reception the receiver can accept Counters with values 2 and 3, but not 4. Note that if the receiver does not receive new Data at a consecutive read, then the receiver increments the tolerance by 1. **Tags:** xml.sequenceOffset=-70

Table 4.72: EndToEndDescription

[TPS_SWCT_1090] `EndToEndProtection` ⌈ `EndToEndProtection` is the `Identifiable` class that owns specific elements for referencing the to-be-protected data elements and signals

- `EndToEndProtectionVariablePrototype`: a specific `dataElement` owned by a specific `PortPrototype`
- `EndToEndProtectionISignalIPdu`: a specific `ISignalGroup` in the context of an `ISignalIPdu`. For more details please refer to [11]

⌋*(RS_SWCT_3240)*

[TPS_SWCT_1091] Two cases for end-to-end protection ⌈ In order to protect a VariableDataPrototype the `EndToEndProtectionVariablePrototype` shall be defined. If communication is defined between ECUs using AUTOSAR COM the `EndToEndProtectionISignalIPdu` shall be defined as well. ⌋*(RS_SWCT_3240)*

The following features apply:

- **[constr_1000] End-to-end protection is limited to sender/receive communication** ⌈ end-to-end protection applies for sender/receiver communication only ⌋
- The value of the `dataId` is assigned by a central authority rather than by the developer of the software-component.
- The information about the `dataId` shall be available at both the sender and the receiver(s).
- **[constr_1001] Value of `dataId` shall be unique** ⌈ The value of the `dataId` shall be unique within the scope of the `System`. ⌋
- End-to-end protection applies to local (i.e. within the ECU) as well as remote (i.e. ECU to ECU) communication.

[TPS_SWCT_1092] `EndToEndProtectionSet` ⌈ The meta-class `EndToEndProtectionSet` provides a container for `EndToEndProtection`. The aggregation is stereotyped ≪`atpSplitable`≫ because the information about end-to-end protection is added at a later step in the development workflow. ⌋*(RS_SWCT_3240)*

It also has the stereotype ≪`atpVariation`≫ because this allows for implementing the software-component in two variants, one that uses end-to-end protection and one that does not use it. It also might happen that the communication ends themselves are variant.

`EndToEndProtection` maintains `InstanceRef`s to one `dataElement` in the role of `sender` and to one or many `dataElement`s in the role of `receiver`. By this means it is possible to support a 1:n communication scenario.

[constr_1002] End-to-end protection does not support n:1 communication ⌈ As the n:1 communication scenario implies that probably not all senders use the same `dataId` this scenario is explicitly not supported. ⌋

[TPS_SWCT_1093] Definition of end-to-end protection is splitable ⌈ `EndToEndProtection` aggregates `EndToEndDescription` using stereotype ≪`atpSplitable`≫. By this means it is for the integrator of an ECU possible to generally specify the nature of a specific end-to-end protection but leave the actual assignment of values (e.g. for `dataId`) to a later process step. ⌋*(RS_SWCT_3240)*

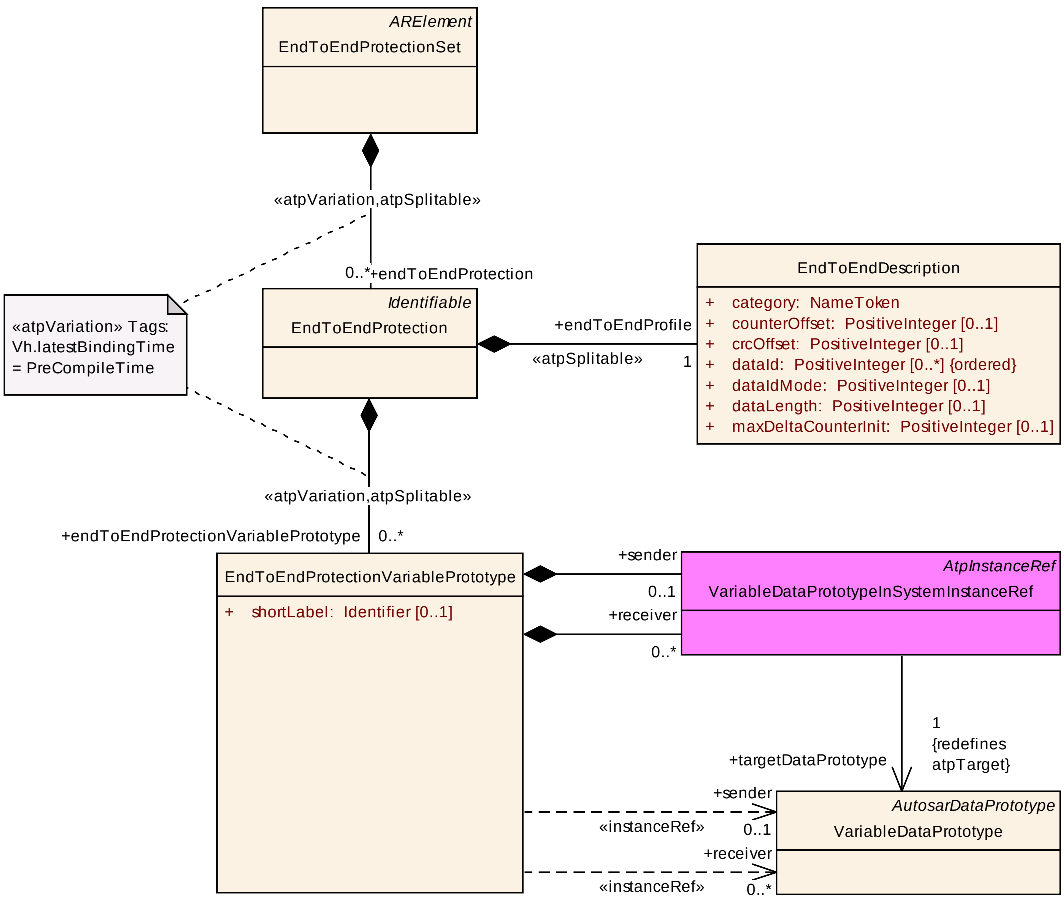

Figure 4.41: Details of the modeling of end-to-end protection

According to [17] the following constraints apply on the attributes of EndToEndProtection (note that additional M1 constraints apply as described in [17]):

[constr_1110] Value of category in EndToEndDescription ⌈ The attribute category of EndToEndDescription can have the following values:

- NONE
- PROFILE_01
- PROFILE_02

⌋

[TPS_SWCT_1094] category of EndToEndDescription ⌈ The values for the category of EndToEndDescription mentioned in [constr_1110] are standardized and reserved for being used in the way the AUTOSAR standard foresees. In addition, it is positively possible to use other than the standardized values for the category. ⌋*(RS_SWCT_3240)*

This aspect will be clarified in more detail in later revisions of the AUTOSAR standard. For the time being, it shall be noted that the usage of other than the standardized values shall not create name clashes with future standardized values. This can be achieved by using e.g. a company-specific prefix or suffix to the value of category.

The semantics of the categories is:

NONE this indicates that the E2E framework shall be enabled for the given sender/receiver respectively the given iSignalIPdu. The wrapper code shall be generated but it shall not invoke E2E library protection routines. E2E wrapper works as pass-through.

This may be used when a profile selection or profile options are not yet selected in a given system but it is required that the system can be built successfully under consideration of the E2E library. This would also be applicable for migrating from/to a system with/without E2E protection.

[TPS_SWCT_1095] category set to NONE ⌈ If attributes exist in the presence of the category being set to NONE the attributes shall be ignored. ⌋*(RS_SWCT_3240)*

PROFILE_01 This indicates that the settings of E2E profile 1 (that uses a SAE CRC8, implicit 16 bit data ID, and a 4 bit alive counter) apply.

[constr_1113] Existence of attributes in PROFILE_01 ⌈ In PROFILE_01, the following attributes shall exist:

- dataLength
- dataId

⌋

Please note that the attribute `maxDeltaCounterInit` is also part of PROFILE_01 but it does not necessarily have to exist provided that `ReceiverComSpec.maxDeltaCounterInit` exists.

[constr_1170] Interpretation of attribute `maxDeltaCounterInit` owned by `EndToEndDecription` ⌈ The value of the attribute `maxDeltaCounterInit` owned by `EndToEndDecription` shall be **ignored** if and only if a `ReceiverComSpec.maxDeltaCounterInit` exists and the `DataPrototype` referenced by this `ReceiverComSpec` in the role `dataElement` is **identical** to the `DataPrototype` owned by `EndToEndDescription` in the role `receiver`.

If the value of `cateogry` of `EndToEndDescription` is set to PROFILE_01 **and either** the described correspondence rule concerning the referenced `DataPrototype` is not fulfilled **or** `ReceiverComSpec.maxDeltaCounterInit` does not exist `EndToEndDescription.maxDeltaCounterInit` **shall exist.** ⌋

[constr_1111] Constraints of `dataId` in PROFILE_01 ⌈ In PROFILE_01, there shall be only one element in the set and the applicable range of values is [0 .. 65535]. ⌋

[constr_1112] Constraints of `dataIdMode` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `dataIdMode` is [0 .. 2]. ⌋

[constr_1114] Constraints of `crcOffset` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `crcOffset` is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies. ⌋

[constr_1115] Constraints of `counterOffset` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `counterOffset` is [0 .. 65535]. For the value of this attribute the constraint *value mod 4 = 0* applies. ⌋

[constr_1116] Constraints of `dataLength` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `dataLength` is [0 .. 240]. For the value of this attribute the constraint *value mod 8 = 0* applies. ⌋

[constr_1117] Constraints of `maxDeltaCounterInit` in PROFILE_01 ⌈ In PROFILE_01, the applicable range of values for `maxDeltaCounterInit` is [0 .. 14]. ⌋

PROFILE_02 this indicates that the settings of E2E profile 2 apply.

[constr_1118] Existence of attributes in PROFILE_02 ⌈ In PROFILE_02, only the following attributes shall exist:

- `dataLength`
- `dataId`

⌋

Please note that the attribute maxDeltaCounterInit is also part of PROFILE_01 but it does not necessarily have to exist provided that ReceiverComSpec.maxDeltaCounterInit exists.

[constr_1171] Interpretation of attribute maxDeltaCounterInit of EndToEndDecription ⌈The value of the attribute maxDeltaCounterInit owned by EndToEndDecription shall be **ignored** if and only if a ReceiverComSpec.maxDeltaCounterInit exists and the DataPrototype referenced by this ReceiverComSpec in the role dataElement is **identical** to the DataPrototype owned by EndToEndDescription in the role receiver.

If the value of cateogry of EndToEndDescription is set to PROFILE_02 **and either** the described correspondence rule concerning the referenced DataPrototype is not fulfilled **or** ReceiverComSpec.maxDeltaCounterInit does not exist EndToEndDescription.maxDeltaCounterInit **shall exist.** ⌋

[constr_1119] Constraints of dataLength in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for dataLength is [0 .. 65535]. For the value of this attribute the constraint *value mod 8 = 0* applies. ⌋

[constr_1120] Constraints of dataId in PROFILE_02 ⌈ In PROFILE_02, there shall be exactly ordered 16 elements in the set and the applicable range of values is [0 .. 255]. ⌋

[constr_1121] Constraints of maxDeltaCounterInit in PROFILE_02 ⌈ In PROFILE_02, the applicable range of values for maxDeltaCounterInit is [0 .. 15]. ⌋

Class	**EndToEndProtectionSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This represents a container for collection EndToEndProtectionInformation. **Tags:** atp.recommendedPackage=EndToEndProtectionSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
endToEnd Protection	EndToEndProte ction	*	aggr	This is one particular EndToEndProtection. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel

Table 4.73: EndToEndProtectionSet

Class	**EndToEndProtection**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	This meta-class represents the ability to describe a particular end to end protection.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
endToEnd Profile	EndToEndDescription	1	aggr	This represents the particular EndToEndDescription. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=description
endToEnd ProtectionISignalIPdu	EndToEndProtectionISignalIPdu	*	aggr	Defines to which ISignalIPdu - ISignalGroup pair this EndToEndProtection shall apply. In case several ISignalGroups are used to transport the data (e.g. fan-out in the RTE) there may exist several EndToEndProtectionISignalIPdu definitions. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=variationPoint.shortLabel
endToEnd Protection VariablePrototype	EndToEndProtectionVariablePrototype	*	aggr	Defines to which VariableDataPrototypes in the roles of one sender and one or more receivers this EndToEndprotection applies. It shall be possible to aggregate several EndToEndProtectionVariablePrototype in case additional hierarchical decompositions are introduced subsequently. In this case one particular PortPrototype is split into multiple PortPrototypes and connectors, all representing the same data entity. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortLabel, variationPoint.shortLabel

Table 4.74: EndToEndProtection

Class	**EndToEndProtectionVariablePrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::EndToEndProtection			
Note	It is possible to protect the data exchanged between software components. For this purpose, for each communication to be protected, the user defines a separate EndToEndProtection (specifying a set of protection settings) and refers to a variableDataPrototype in the role of sender and to one or many variableDataPrototypes in the role of receiver. For details, see EndToEnd Library.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
receiver	VariableDataPrototype	*	iref	This represents the receiver. Note that 1:n communication is supported for this use case.
sender	VariableDataPrototype	0..1	iref	This represents the sender. Can be optional if an ecu extract is provided and the sender is part of the extract.
shortLabel	Identifier	0..1	ref	This serves as part of the split key in case of more than one EndToEndProtectionVariablePrototype is aggregated in the bound model.

Attribute	Datatype	Mul.	Kind	Note

Table 4.75: EndToEndProtectionVariablePrototype

Please note that using end-to-end protection it is explicitly supported that one sender may correspond to one or more receivers.

[constr_1183] EndToEndProtectionVariableDataPrototypes aggregated by EndToEndProtection ⌈ All EndToEndProtectionVariableDataPrototypes aggregated by the same EndToEndProtection shall refer to the identical sender.
⌋

4.8 Partial Networking

[TPS_SWCT_1169] Support for partial networking ⌈ On the level of the Software Component Template, partial networking is supported by means of the concept of a "Virtual Function Cluster" (VFC). The latter groups all communication on the VFB with respect to a given function. However, the conceptual idea of a Virtual Function Cluster is not represented in the meta-model as such. Instead, PortGroups (see chapter 4.6) are used to specify the grouping of PortPrototypes to the higher conceptual level of a Virtual Function Cluster. ⌋*(RS_SWCT_3241)*

There are no restrictions regarding the structure of PortGroup definitions on M1. One PortPrototype may become a member of several PortGroups, thereby creating overlapping PortGroups.

[TPS_SWCT_1170] Purpose of Virtual Function Cluster ⌈ The purpose of Virtual Function Cluster within the Software Component Template mainly has three aspects:

1. assign PortPrototypes (non service related) of Sender Receiver or Client Server communication to Virtual Function Clusters.
2. control the behavior of the corresponding function in terms of whether or not it is required at a given point in time. This aspect is implemented by the concept of a **control port**. Software-components that implement control ports of a Virtual Function Cluster conceptually become **VFC Controllers**.
3. allow for the application software to retrieve the status of a given Virtual Function Cluster. This aspect is implemented by the concept of a **status port**.

⌋*(RS_SWCT_3241)*

The usage of the generic concept of PortGroups for the purpose of partial networks shall be indicated by setting the value of the attribute category of PortGroup to PARTIAL_NETWORKING.

4.8.1 VFC Control Ports

[TPS_SWCT_1171] Purpose of a control port ⌈ The purpose of a control port is to request or release a VFC. Requesting means that the VFC is actively using communication resources while *release* boils down to the VFC being inactive, i.e. the corresponding partial network may be shut down until further notice.

As the requesting and releasing semantics is implemented by means of interfacing the BSW the corresponding control ports need to be typed by a `PortInterface` that has the attribute `isService` set to `true`. ⌋*(RS_SWCT_3241)*

[TPS_SWCT_1172] Requesting and releasing partial networks ⌈ For requesting and releasing partial networks, the BSW can be interfaced in two alternative (i.e. either one or the other) ways:

- **ComM**: `ClientServerInterface` using the standardized `ComM_UserRequest.RequestComMode` [18]
- **BswM**: `SenderReceiverInterface` using the standardized `AppModeRequestInterface.requestedMode` [19]

⌋*(RS_SWCT_3241)*

[TPS_SWCT_1173] Control port shall not become a part of the `PortGroup` ⌈ Please note that the control port shall **not** become a part of the `PortGroup` that defines the particular VFC the control port is going to service. The relationship is implemented by means of a specific `SwcServiceDependency` that owns a `RoleBasedPortAssignment` to the intended control port. ⌋*(RS_SWCT_3241)*

4.8.2 VFC Status Ports

[TPS_SWCT_1175] Actively query the status of a partial network ⌈ Very much like mode management, the concept of partial networking supports the ability to actively query the status of a partial network. This can be done by means of interfacing the BSW in three alternative (as in "one of") ways:

- **ComM**: `ClientServerInterface` using the standardized `ComM_UserRequest.GetCurrentComMode` [18]
- **ComM**: `SenderReceiverInterface` using the standardized `ComM_CurrentMode.currentMode` [18]
- **BswM**: `SenderReceiverInterface` using the standardized `AppModeInterface.currentMode` [19]

⌋*(RS_SWCT_3241)*

As mentioned above, the status of the ComM can be retrieved by either a ClientServerInterface or a SenderReceiverInterface. Which of the two alternatives applies in a specific case is up to the author of a software-component[6].

When using one of the possible SenderReceiverInterfaces, the correspondence of the status port concept with mode management extends to the point that the status of the partial network is returned as an actual ModeDeclaration.

This implies that all mechanisms foreseen by the Software Component Template to react on mode changes are in place and can be used within the application software. To assure that the communication via PortPrototypes that belong to a partial network is valid the software component shall consider the status of the partial network before communicating in order to assert its activity.

[TPS_SWCT_1174] Status port shall not become a member of the PortGroup ⌈
A status port shall **not** become a member of the PortGroup that corresponds to the partial network subject to the status port. The relationship is implemented by means of a specific SwcServiceDependency that owns a RoleBasedPortAssignment to the intended status port. ⌋*(RS_SWCT_3241)*

[6] The usage of the ClientServerInterface effectively implements a “pull” approach for the mode information while the usage of the SenderReceiverInterface resembles a “push” approach if it is used in combination with a ModeSwitchEvent.

5 Data Description

5.1 Introduction

[TPS_SWCT_1229] Three different levels of abstraction regarding the definition of data types ⌈ In the context of defining data types and prototypes, the AUTOSAR concept distinguishes between three different levels of abstraction as depicted in Table 5.1.

Application Data Level
Implementation Data Level
Base Type Level

Table 5.1: Abstraction Levels for Describing Data

⌋*(RS_SWCT_3215, RS_SWCT_3216, RS_SWCT_3217)*

[TPS_SWCT_1230] Application Data Level ⌈ The ***Application Data Level*** is the common level at which `ApplicationSwComponentType`s specify a data type or prototype. This level allows to define all the data attributes which are needed from the application point of view, in order to exchange data between software components or between a software component and a measurement and calibration tool. It is possible to specify data communication of a complete `Virtual Function Bus` based on this level only.

This level includes among other things the numerical range of values, the data structure as well as the physical semantics. Data semantics (e.g. physical units) is not in the focus[1] for the RTE in order to make communication technically possible. However, it is important for a unique interpretation of data in the application software and in measurement and calibration systems. ⌋*(RS_SWCT_3216)*

In former version of this specification, this level was not clearly separated from the implementation level. These had the following drawbacks which are now solved:

- The model of primitive types (like integer, boolean, real, opaque) was anticipating implementation aspects already on a very high level of design.
- The data type model used within ports, focusing on communication via the RTE, was not sufficient to model all type-aspects of variables and parameters which are visible within an AUTOSAR system for other purposes than RTE-communication, namely NvM-data access, calibration, measurement, diagnostics, BSW-module interfaces. Using a uniform type system covering all these aspects is now favored.

[1] There are some aspects that affect the RTE, e.g. scaling of `dataElement`s

- Calibration parameters were not completely incorporated into the data type concept. Some of their attributes (especially for curves and maps) could be specified only on the level of prototypes or were not completely formalized within AUTOSAR (like `SwRecordlayout`).
- The data type system was not compatible with the usage in calibration standards like ASAM-MCD (namely the usage of categories).
- Adding implementation specific elements like a base type, was not possible without formally changing the data type used in a VFB design. A mapping mechanism that could be used in later project phases and is common in other parts of AUTOSAR (e.g. for mapping components to ECUs) was missing.
- The RTE Specification contained many default rules and assumptions on how to implement certain data types or prototypes in C. With a more formal description of all relevant implementation aspects, the generation of C-interfaces is better determined. But these aspects should be separated from the application level design.
- Since there could be many data types on the application level in a big system, the probability of name clashes in the interfaces to the RTE was rather high. Using a separate set of types to implement the RTE interfaces solves this issue.

[TPS_SWCT_1231] Application level may impose strong requirements on the design of the corresponding implementation level ⌈ It should be pointed out, that with the specification of computation methods and record layouts, the application level imposes strong requirements on the design of the corresponding implementation level (for further information see 6.2.4). It might even be the case, that when anticipating different implementations, these elements might be chosen differently.

This is due to the nature of these elements which form a bridge from the physical world to the numerical representation (and vice versa). Nonetheless we consider the specification of these elements as belonging to the application level. On the one hand, this information is required by MCD-tools and thus shall be part of a rather high-level design. On the other hand, this approach will allow to use a limited set of implementation data types. ⌋*(RS_SWCT_3215, RS_SWCT_3216, RS_SWCT_3217)*

[TPS_SWCT_1232] Implementation Data Level ⌈ The ***Implementation Data Level*** is closer to the actual code implementation in a programming language like C, though it is still an abstraction of the code. Its values correspond to the actual binary numbers handled by the programming language on the CPU. It contains concepts like pointers and unions which relate to the organization of data in memory and are not relevant for the application level.

This level also defines structure, but it can be more granular. For example, the application level may define a text to be transferred to an instrument cluster as a primitive type (if the structure is not relevant for the application), whereas on the implementation level it could be modeled as an array of bytes. ⌋*(RS_SWCT_3217)*

[TPS_SWCT_1233] Use case for the Implementation Data Level ⌈ There are several use cases for this level in AUTOSAR:

- First of all, the *Implementation Data* level can be used in the description of interfaces, and data (e.g. debug data) within the basic software, see [7] for more details on these use cases.
- `ImplementationDataType`s should also be used to describe the interfaces of libraries which operate on a purely numerical level.
- *Implementation Data* is also used for the description of interfaces between software-components and and the basic software (namely AUTOSAR Services), because these typically cover implementation aspects only.
- It is possible to define communication in a VFB system directly on this level if the physical and semantical abstraction is not of interest.
- Last not least the input for the RTE generator is defined by data descriptions on this level. This means that in case a SWC defines its data only on application level a corresponding set of implementation data types shall be created (or generated) as part of the ECU extract before the RTE can be generated.

⌋*(RS_SWCT_3217)*

[TPS_SWCT_1234] Base Level ⌈ The ***Base Type Level*** is used to describe the primitive elements in terms of bits and bytes from which the implementation data is built up. It is considered as a separate level in order to allow for reuse of the basic types defined on this level.

These base types still do not completely determine the actual implementation on a programming language, but they impose strong restrictions for this as they define for example the number of bits and bytes to be used. Depending on the use case, the base types can be defined as platform independent or can also contain platform specific attributes (namely endianess and alignment). ⌋

[TPS_SWCT_1235] Mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level ⌈ It is important to understand, that the mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level depends on the medium on which the data is transported. For example, if a physical value can be expressed with sufficient accuracy and range by a 16-bit unsigned integer, it still might look very different when sent over CAN, when seen by a software-component on a *big-endian* 32-bit machine or when seen by a software-component on a *little-endian* 16-bit processor.

Conversion between several data implementations of the same application data type might be necessary in case of communication between components on different ECUs. AUTOSAR COM [20] is responsible for this. It implies that the configura-

tion depends on the definition of the data that are transmitted between components[2].
⌋*(RS_SWCT_3215, RS_SWCT_3216, RS_SWCT_3217)*

AUTOSAR COM might need to convert a 16-bit integer between *little-endian* and *big-endian* representations; whereas an array of 16 bytes does not need to be swapped even if the endianess changes. In case of intra-ECU communication byte order conversion is not necessary, since the software-components reside on the same machine.

[TPS_SWCT_1236] Big picture of data types ⌈ Another way of approaching the concept of data types in AUTOSAR (especially with respect to the question of what "kind" of data type in related to which modeling meta-level) is to sketch the following "big picture" of data types:

ApplicationDataType: defined on **M2** - provides the meta model for data types on application level. It covers the application-relevant aspects of a data type.

An `ApplicationDataType` shall finally be mapped to an `ImplementationDatatype`.

ImplementationDataType: defined on **M2** - provides the meta-model for data types on implementation level. With respect to C source code, an `ImplementationDatatype` finally boils down to a `typedef`.

BaseType: defined on **M2** - provides the platform-dependent part of an `ImplementationDataType`. the dependency on the platform covers the following aspects:

- Definition on the level of the C language - using `nativeDeclaration`
- Technical representation on the target platform (byte order, alignment, encoding) as required for the support of MCD systems.

Platform Type: defined on **M1** - provided by AUTOSAR. Platform types shall be available on each platform on which an AUTOSAR-System can run.

The name of the platform data type and the properties with respect to the interface between modules / components is the same on every platform.

The particular representation varies from platform to platform.

Platform types shall be **modeled** using `ImplementationDataType`s.

Note that in AUTOSAR R3.x the platform types are implemented manually and could even not be expressed on ARXML model (see [rte00056]). In AUTOSAR R4.0 the platform types can be represented in the ARXML model. Subsequent releases of AUTOSAR may generate the platform types directly from the ARXML Model.

Standard Type: defined on **M1** - provided by AUTOSAR. Standard types are defined by referring to platform types.

[2]More exactly speaking, the data shall be converted to and from a so-called `SystemSignal`, see [11]for more details.

⌋*(RS_SWCT_3215, RS_SWCT_3216, RS_SWCT_3217)*

[TPS_SWCT_1237] **SwDataDefProps** ⌈The properties of data are summarized in the meta-class SwDataDefProps. This meta-class itself is the superset of all applicable properties. ⌋*(RS_SWCT_3216, RS_SWCT_3217)*

Subsets of SwDataDefProps are applicable in specific case, for a summary please refer to the following tables:

- The data categories are summarized in table 5.7.
- Properties for ApplicationDataTypes are summarized in table 5.8
- Properties for ImplementationDataTypes are summarized in table 5.17
- Properties for DataPrototypes typed by ApplicationDataTypes are summarized in table 5.33
- Properties for DataPrototypes typed by ImplementationDataTypess are summarized in table 5.34
- Applicability of SwDataDefProps is summarized in table 5.41

5.2 Data Types

5.2.1 Overview

As explained in section 5.1 it is possible to describe data provided by a software-component from the application as well as from the implementation point of view.

[TPS_SWCT_1072] **ApplicationDataType and ImplementationDataType** ⌈The common concept behind this is expressed by the abstract meta-class AutosarDataType, from which an ApplicationDataType and an ImplementationDataType is derived. ⌋*(RS_SWCT_3215, RS_SWCT_3216, RS_SWCT_3217)*

Figure 5.1 shows a summary of the basic meta-classes used for the definition of AutosarDataTypes.

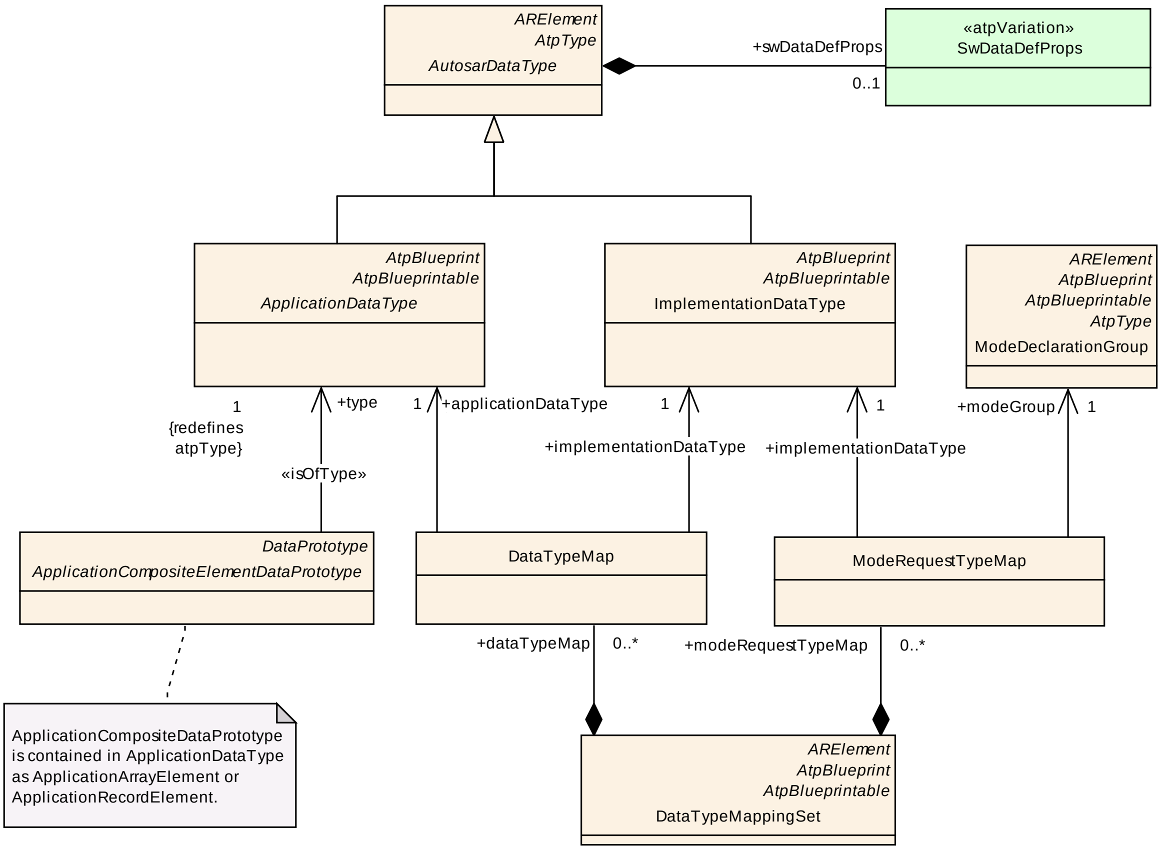

Figure 5.1: Summary of `AutosarDataType`

Class	**AutosarDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for ECU software.			
Base	ARElement,ARObject,AtpClassifier,AtpType,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swDataDef Props	SwDataDefProp s	0..1	aggr	The properties of this AutosarDataType.

Table 5.2: AutosarDataType

Class	ApplicationDataType (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationDataTypes only.			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,Autosar DataType,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	Datatype	Mul.	Kind	Note
–	–	–	–	–

Table 5.3: ApplicationDataType

Class	ImplementationDataType			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,Autosar DataType,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	Datatype	Mul.	Kind	Note
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an arrray or a struct type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table 5.4: ImplementationDataType

[TPS_SWCT_1073] Composite `ApplicationDataType` ⌈ An `ApplicationDataType` can be composed (in form of a record or an array) of elements which themselves are typed by another ApplicationDataType. ⌋*(RS_SWCT_3215, RS_SWCT_3216)*

This is expressed by the meta-class ApplicationCompositeElementDataPrototype which is shown in the figure 5.1 for completeness.

[TPS_SWCT_1074] Composite ImplementationDataType ⌈ An ImplementationDataType can also be composed of elements but in this case no type/prototype concept (see [13]) has been applied. Both concepts will be explained in the following chapters in more detail. ⌋*(RS_SWCT_3215, RS_SWCT_3217)*

5.2.2 Data Type Mapping

As explained above, the concept of application data types as well as that of implementation data types can be used to instantiate a data prototype in an M1 model. However there are use cases, especially in order to generate the RTE contract for ApplicationSoftwareComponentTypes, where it is required to consider both levels for one given data prototype.

[TPS_SWCT_1189] DataTypeMap ⌈ This is supported by the meta-class DataTypeMap by which an ApplicationDataType and an ImplementationDataType can be mapped to each others in order to describe both aspects of one dataElement. ⌋*(RS_SWCT_3216, RS_SWCT_3217, RS_SWCT_3215)*

Class	**DataTypeMap**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing ImplementationDatatype.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
applicationDataType	ApplicationDataType	1	ref	This is the corresponding ApplicationDataType
implementationDataType	ImplementationDataType	1	ref	This is the corresponding ImplementationDataType.

Table 5.5: DataTypeMap

If, for example, a dataElement in a SenderReceiverInterface is typed by an ApplicationDataType it shall additionally be associated to an ImplementationDataType in order to be able to generate the RTE.

[TPS_SWCT_1190] ModeRequestTypeMap ⌈ Another mapping class, ModeRequestTypeMap, has been introduced in order to allow the transport of mode related information via "normal" sender-receiver communication. Apart from this, mode information is not handled by the usual type system but needs special meta-classes. This is explained in more detail in chapter 4.2.5. ⌋*(RS_SWCT_3110)*

Note that the mapping classes instead of direct associations have been introduced for process reasons: It allows to maintain application and implementation types in separate M1 artifacts without direct links. For example, if a software component is moved to another hardware platform the mapping between application and implementation types might be changed in the scope of the specific component without changing the overall VFB model.

[TPS_SWCT_1191] mapped `ApplicationDataType` and `ImplementationDataType` shall be compatible ⌈ In order to set up a valid `DataTypeMap` between an `ApplicationDataType` and an `ImplementationDataType` the two types shall be compatible. This is further explained in chapter 6.2.4. Of course, if `ImplementationDataType`s are generated from existing `ApplicationDataType`s it is expected that they will be automatically compatible. ⌋*(RS_SWCT_3216, RS_SWCT_3217)*

Furthermore, the various mappings are aggregated in a container `DataTypeMappingSet` for easier maintenance in artifacts.

Class	**DataTypeMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents a list of Mappings between ApplicationDataTypes and ImplementationDataTypes. In addition, it can contain mappings between ImplementationDataTypes and ModeDeclarationGroups. **Tags:** atp.recommendedPackage=DataTypeMappingSets			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataTypeMap	DataTypeMap	*	aggr	This is one particular association between an ApplicationDataType and its ImplementationDataType.
modeRequestTypeMap	ModeRequestTypeMap	*	aggr	This is one particular association between an ModeDeclarationGroup and its ImplementationDataType.

Table 5.6: DataTypeMappingSet

Note that the meta-classes `AutosarDataType`, `ModeDeclarationGroup` and `DataTypeMappingSet` are derived from `ARElement`. This means that these and the meta-classes derived from them can be declared on the M1 level as part of an `ARPackage` and thus can be used in several different Software Component or Basic Software Module Descriptions.

How to organize `DataTypeMappingSet`s for a software system, for example whether there is a separate mapping set for each ECU or even for each software component, is considered as project specific. However, the RTE generator needs a well defined `DataTypeMappingSet` as input in relation those artifacts which might define data typed as `ApplicationDataType`s.

[TPS_SWCT_1192] Meta-classes that have an association to a DataTypeMappingSet ⌈ Therefore, the following meta-classes have an association to a DataTypeMappingSet:

- InternalBehavior, because it represents the interface between the software component's code and the RTE and all data types belonging to the particular component type have to be uniquely provided on implementation level.
- ParameterSwComponentType, for the same reason (this component type doesn't have an InternalBehavior).
- NvBlockDescriptor, because this meta-class also leads to generation of code from data types and is not associated to an InternalBehavior.

⌋

For more details about this aspect please refer to figure 5.54.

[TPS_SWCT_1193] Mappings between application and implementation types do not necessarily have to form a 1:1 relation ⌈ In general, it is not required that the sum of all mappings between ApplicationDataType and ImplementationDataType in a given system form a 1:1 relation. Depending on the use case and on the scope, 1:n as well as n:1 mappings are possible:

- Several different ApplicationDataTypes may be mapped to the same ImplementationDataType in the scope of a system, an ECU, or even a single InternalBehavior of an atomic software component. Of course, this requires that the different ApplicationDataTypes are used for different DataPrototypes and thus that the DataPrototypes are typed by them (and not by the ImplementationDataTypes). This allows to establish a more simple type system on the implementation level, than on the application model level.
- The same ApplicationDataTypes may be mapped to different ImplementationDataTypes for different ECUs. This scenario allows to chose the implementation data types according to the needs of specific ECUs.
- **[constr_1004] Mapping of ApplicationDataTypes** ⌈ The same ApplicationDataTypes may be mapped to different ImplementationDataTypes even in the scope of a single ECU (more exactly speaking, a single RTE), but not in the scope of a single atomic software component. ⌋

  This improves the portability of software components which were developed independently or are ported between ECUs.

⌋

[constr_1005] Compatibility of ImplementationDataTypes mapped to the same ApplicationDataType ⌈ It is required that ImplementationDataTypes which are taken for connecting corresponding elements of PortInterfaces and thus refer to compatible ApplicationDataTypes are also compatible among each other (so that RTE is able to cope with possible connections by converting the data accordingly). ⌋

This constraint is visualized in figure 5.2.

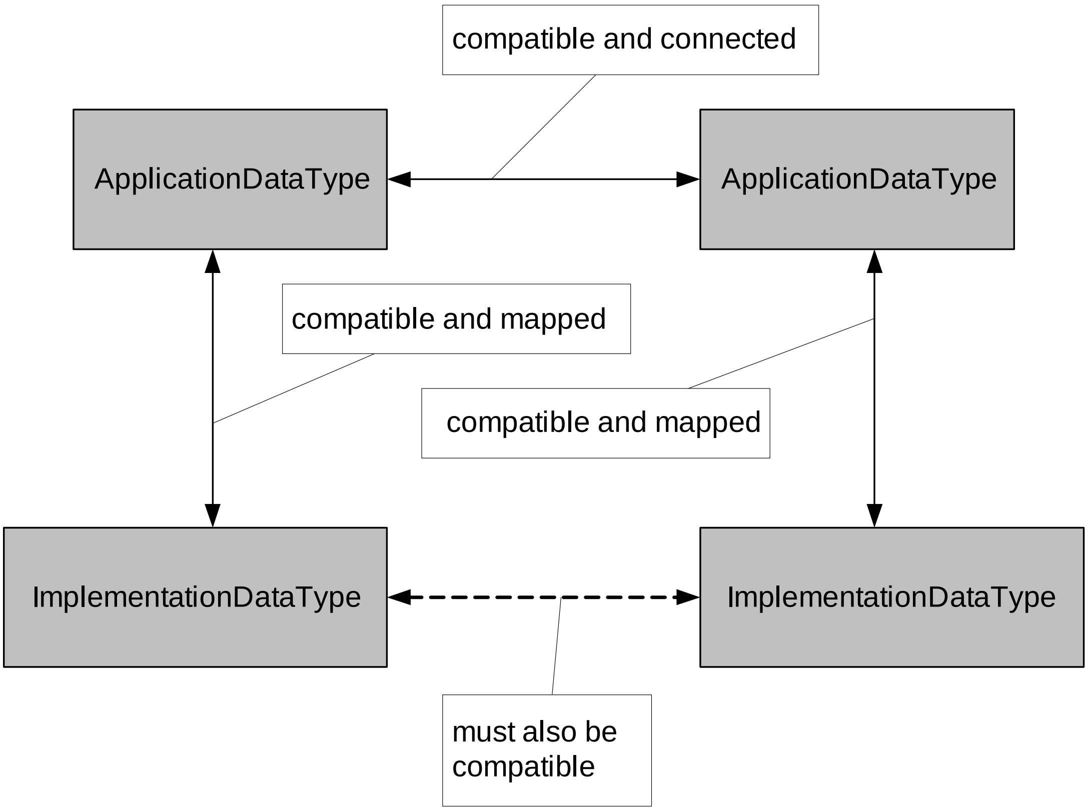

Figure 5.2: Compatibility of Data Types

5.2.3 Data Categories

An `AutosarDataType` is derived from `Identifiable`, thus having a `longName`, a `shortName`, a `category`, and several further attributes for administrative and documentation purposes (for details see [13]).

[TPS_SWCT_1238] Attribute `category` used in the context of `AutosarDataType`
⌈The `category` attribute is used to set constraints for the various properties which can be specified for an `AutosarDataType`. These properties are defined by aggregating the meta-class `SwDataDefProps` which contains several attributes and references, see detailed description in chapter 5.4 and 5.4. ⌋

[constr_1143] `category` of `AutosarDataType` shall not be extended ⌈In contrast to the general rule that `category` can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute `category` of meta-class `AutosarDataType`⌋

This approach avoids a very deep and complicated inheritance tree which otherwise would be needed on the M2 level for `AutosarDataType`. There is to some extend a redundancy between setting the `category` and defining the attributes of `Autosar-`

DataType.swDataDefProps. This redundancy is intended and allows to for a tool to rule out senseless configurations via simple rules.

In former version of this specification the categories were only used for calibration parameters. Due to several extensions the categories are now applicable for all use cases of the AutosarDataType. An overview on all categories defined for AutosarDataType is shown in table 5.7. Some of the categories are also applied to sub-elements of the type system (column "Applicable to..." in table 5.7). This is explained in more detail in the following sections.

[constr_1006] applicable data categories ⌈ Table 5.7 defines the applicable data categories depending on specific model elements related to data definition properties. ⌋

Category	Applicable to ...										Use Case				Description
	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
VALUE	x	x	x	x	x	x	x		x	x	x	x	x	x	Contains a single value.
VAL_BLK	x	x	x	x						x	x				A value block defines values stored together within one calibration parameter object. It is similar to an value array but it stores the values by means of an axis instead (only important for calibration data handling).
DATA_REFERENCE					x	x	x	x						x	Contains an address of another data prototype (whose type is given via SwDataDefProps.swPointerTargetProps)
FUNCTION_REFERENCE					x	x	x	x						x	Contains an address of a function prototype (whose signature is given via SwDataDefProps.functionPointerSignature)
TYPE_REFERENCE					x	x	x	x						x	The element is defined via reference to another data type (via SwDataDefProps.implementationDataType)
STRUCTURE	x	x	x	x	x	x	x			x	x	x	x	x	Holds one or several further elements which can have different data types. The underlying elements are defined in the same manner as normal data except for the association to SwAddrMethod: This has to be the same for all underlying elements. Corresponds to a Record if used in the application domain.
UNION					x	x	x			x	x	x		x	Can hold values of different data types. It is similar to STRUCTURE except that all of its members start at the same location in memory. A UNION data prototype can contain only one of its elements at a time. The size of the UNION is at least the size of the largest member.
ARRAY	x	x	x	x	x	x	x			x	x	x	x	x	An array of sub-elements which are of the same type.
BIT										x	x	x		x	One or several bits within a host variable, which are treated as an own data object.
HOST										x	x	x		x	A HOST data type is like a simple VALUE, but it is used for packed bit definition. That means it can host several BIT variables which have their own description and measurement access.
STRING	x	x	x	x						x	x	x	x		Contains a single value interpreted as a text string (note that it appears as a single value for the application domain; the internal representation can be an array).

Category	Applicable to ...										Use Case				Description
	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	ApplicationValueSpecification	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	SwSystemconst	McDataInstance	Calibration	Measurement	Communication Port Interfaces	RTE + BSW	
BOOLEAN	x	x	x	x						x	x	x	x		Contains one boolean state. Depending on the CPU direct addressing of single bits may not be available. So a byte or a word can be used to store only one logical state.
COM_AXIS	x		x							x	x				An axis definition as separate calibration parameter which can be referenced by any curve or map. The benefits by using a common axis is that it saves memory space, cause it is stored only one time and can be used in multiple curves or maps.
RES_AXIS	x		x							x	x				A RES AXIS (rescale axis) is also a shared axis like COM AXIS, the difference is that this kind of axis can be used for rescaling. Note that the RES AXIS is by nature a CURVE which is used to implement a non linear scaling (rescale) of the axis. In addition to saving memory space via the shared usage like a COM_AXIS, it can compress a huge range to a non-linear distributed axis points thus retaining the required accuracy.
CURVE	x	x	x	x						x	x				Calibration parameter with one input value and one output value. That means output values can be defined depending on the input value . The granularity of implemented functionality can be changed by using different number of axis points. A CURVE has always one input axis and one output axis. The output axis is a characteristic of the curve and every time present but the input axis can be defined within the curve definition or separately.
MAP	x	x	x	x						x	x				Calibration parameter with two input values and one output value. That means output values can be defined depending on the input values .The granularity of implemented functionality can be changed by using different number of axis points for y- and x-axis. A MAP has always two input axes and one output axis. The output axis is a characteristic of the map and every time present but the input axes can be defined within the map definition or separately.

Table 5.7: Usage of `Category` for Data Types

[TPS_SWCT_1239] default value for attribute `category` used in the context of `AutosarDataType` ⌈ The default value for the `category` of a `SwSystemconst` shall be `VALUE`. This has to be applied if no explicit definition of the `category` can be found.
⌋

5.2.4 Application Data Type

[TPS_SWCT_1240] Subclasses of `ApplicationDataType` ⌈ As figure 5.3 explains, the abstract meta-class `ApplicationDataType` is further derived into an `ApplicationPrimitiveDataType` and an `ApplicationCompositeDataType` which are further explained in the following sub-chapters. ⌋ *(RS_SWCT_3216)*

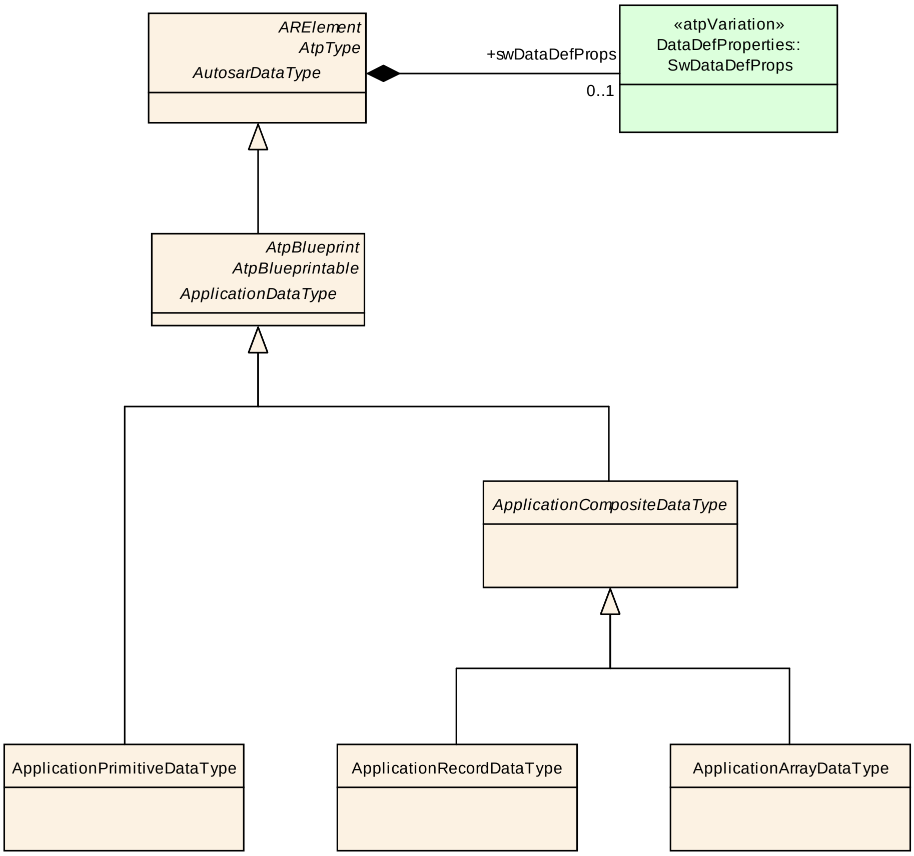

Figure 5.3: Basic Meta-Model for ApplicationDataType

	Root Element			Attribute Existence per Category									
Attributes of SwDataDefProps	ApplicationDataType	ApplicationRecordElement	ApplicationArrayElement	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP
additionalNativeTypeQualifier													
annotation	x	x	x	*	*	*	*	*	*	*	*	*	*
baseType													
compuMethod	x	x	x	0..1	0..1			0..1	0..1			0..1	0..1
dataConstr	x	x	x	0..1	0..1				0..1			0..1	0..1
displayFormat	x	x	x	0..1	0..1			0..1	0..1			0..1	0..1
implementationDataType													
invalidValue	x	x	x	0..1					0..1				
mcFunction													
swAddrMethod	x	x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment													
swBitRepresentation													
swCalibrationAccess	x			1	1	1	1	1	1	1	1	1	1
swCalprmAxisSet	x	x	x							1	1	1	1
swComparisonVariable													
swDataDependency													
swHostVariable													
swImplPolicy	x	x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution	x	x	x	0..1									
swInterpolationMethod	x	x	x	0..1						0..1	0..1	0..1	0..1
swIsVirtual													
swPointerTargetProps													
swRecordLayout	x	x	x	0..1				0..1		1	1	1	1
swRefreshTiming	x	x	x	0..1	0..1			0..1	0..1				
swTextProps	x	x	x					1					
swValueBlockSize	x	x	x		1								
unit	x	x	x	0..1	0..1							0..1	0..1
valueAxisDataType	x	x	x		0..1					0..1	0..1	0..1	0..1
Other Attributes below the Root Element													
element: ApplicationRecordElement	x	x	x			1..*							
element: ApplicationArrayElement	x	x	x				1						
ApplicationArrayElement .arraySizeSemantics	x						0..1						
ApplicationArrayElement .maxNumberOfElements	x						1						

Table 5.8: Allowed Attributes vs. `Category` for Application Data Types

Class	**ApplicationPrimitiveDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	A primitive datatype defines a set of allowed values. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.9: ApplicationPrimitiveDataType

Class	**ApplicationCompositeDataType (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for all application data types composed of other data types.			
Base	ARElement,ARObject,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.10: ApplicationCompositeDataType

[TPS_SWCT_1241] Applicable `categorys` for subclasses `ApplicationDataType` ⌈ Like any `AutosarDataType`, also the primitive and composite types on application level are characterized by its `category` and its `SwDataDefProps`. For a given `category`, only a limited set of attributes of the `SwDataDefProps` makes sense. ⌋*(RS_SWCT_3216)*

[constr_1007] Allowed attributes of `SwDataDefProps` for `ApplicationDataTypes` ⌈ The allowed attributes and their allowed multiplicities are listed as an overview in table 5.8. ⌋

This list makes use of the `SwDataDefProps` and other meta-model elements which are explained in detail in the further sections of this chapter.

[constr_1008] Applicability of categories STRUCTURE and ARRAY ⌈ The categories STRUCTURE and ARRAY correspond to `ApplicationCompositeDataType`s whereas all other categories can be applied only for `ApplicationPrimitiveDataType`s. ⌋

5.2.4.1 Application Primitive Data Types

5.2.4.1.1 Data Types for Single Values

In contrast to prior versions of this specification, the primitive application data types on M2 level are no longer specified. Instead of this, the meta-class `ApplicationPrimitiveDataType` in combination with the attached `swDataDefProps` is used to specify the details on M1 level.

[TPS_SWCT_1242] `category` characterizes the nature of a data type on application level ⌈ The `category` is used in addition to characterize the nature of a data type on application level. ⌋*(RS_SWCT_3216)*

For example, the former `IntegerType` allowed for specifying lower and upper ranges that constrained the applicable value interval. This aspect is still supported, but the meta-model is different from the former approach. Especially it is no more considered of importance to specify that an `ApplicationPrimitiveDataType` is actually represented by "integer" numbers.

Figure 5.4 provides a sketch of how limits are defined now. The key feature is the aggregation of `SwDataDefProps` at `AutosarDataType`. The meta-class `SwDataDefProps` allows for creating a reference to a `DataConstr` that in turn aggregates a `DataConstrRule`.

The latter aggregates `PhysConstr` and this meta-class finally owns two `Limit`s in the roles `lowerLimit` and `upperLimit`.

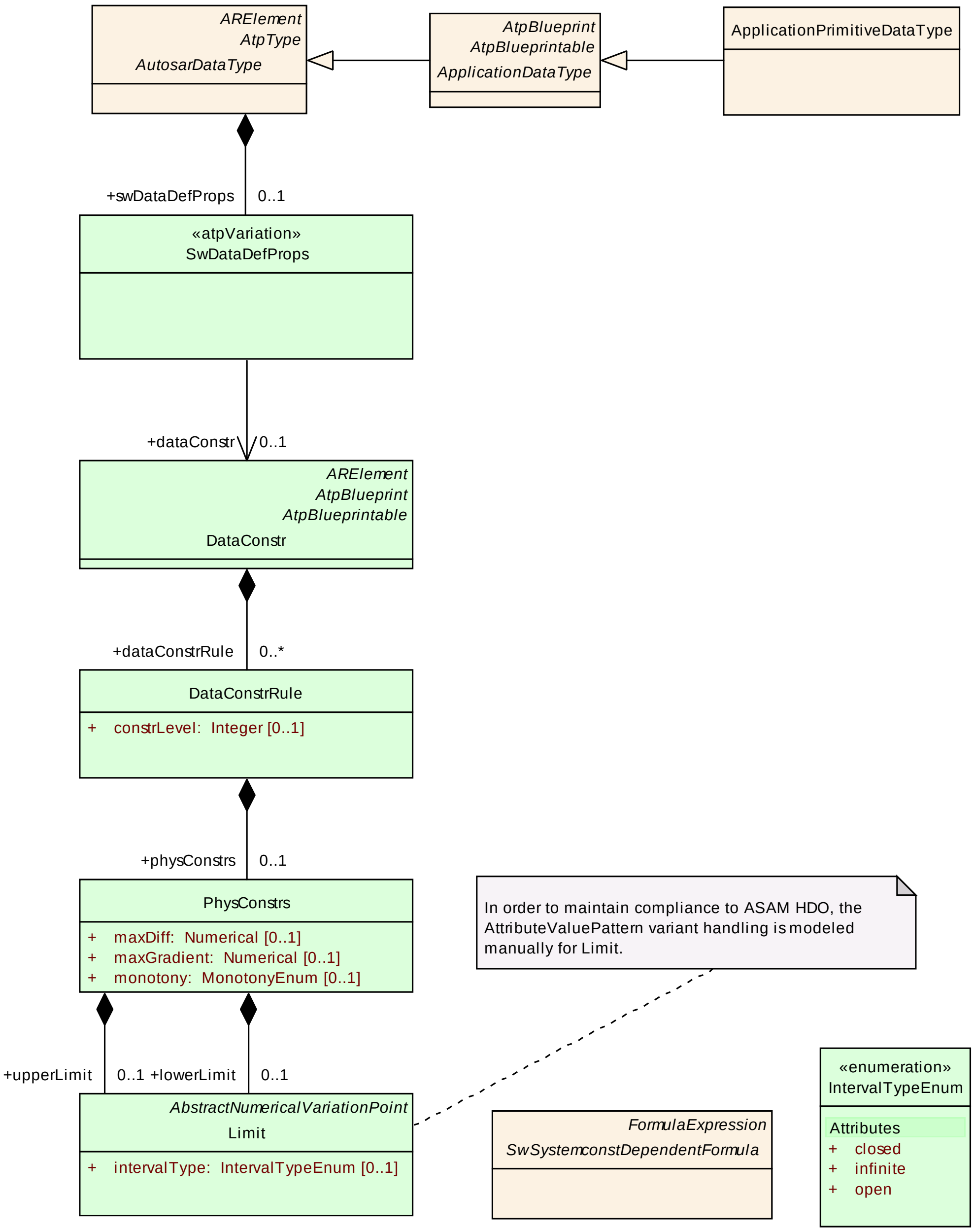

Figure 5.4: Specification of Physical Limits

Another example is shown in Figure 5.5. By making again use of `SwDataDefProps`, this figure shows how semantics in form of a `CompuMethod` and a `Unit` can be attached. Also an `initValue` can be defined which is used by the RTE in order to initialize values of `DataPrototype`s defined locally in a software-component.

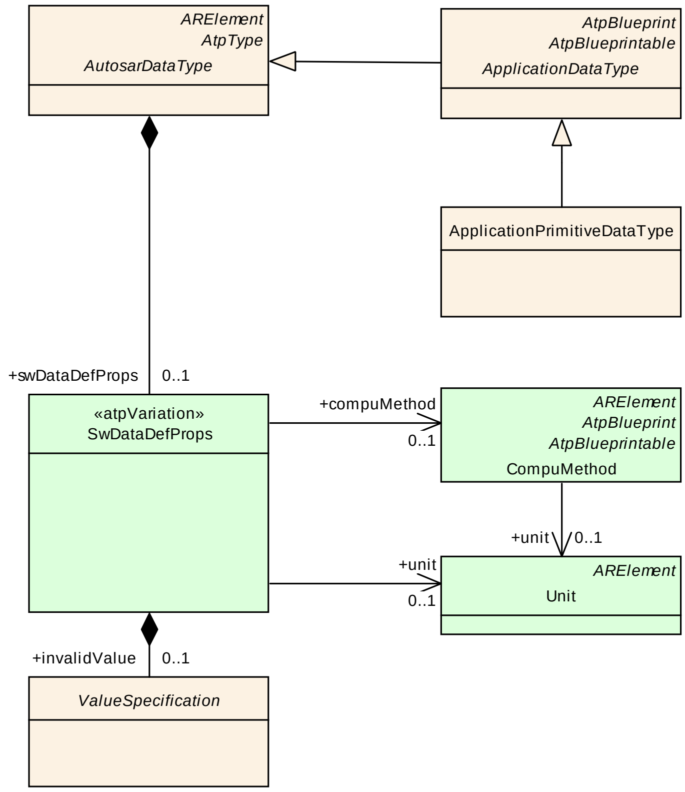

Figure 5.5: Some Properties of `ApplicationPrimitiveDataTypes`

Figure 5.6 illustrates the relationship between the data constraints for `ApplicationDataType`, `CompuMethod`, `ImplementationDataType`, `BaseType` and also the `invalidValue`.

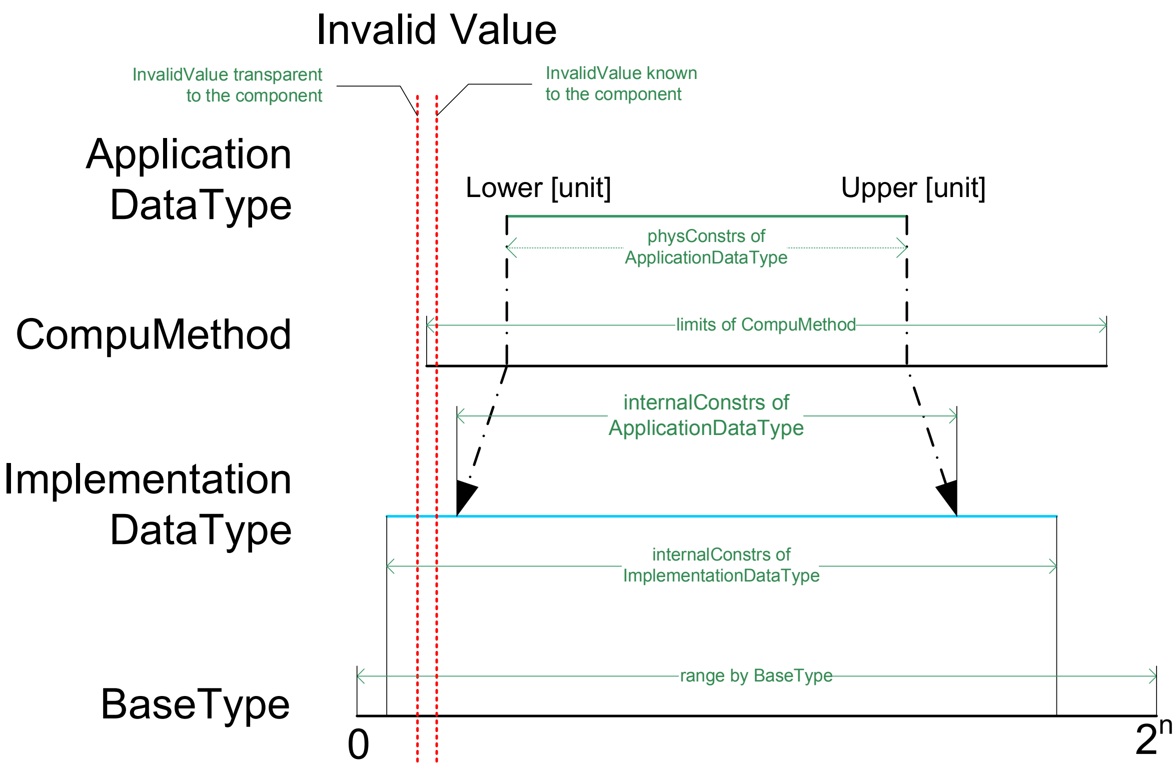

Figure 5.6: Value ranges and invalid values

[constr_2544] Limits need to be consistent ⌈

- The limits of ApplicationDataType shall be inside of the definition range of the CompuMethod

  The CompuMethod needs to be applicable for limits of an ApplicationDataType. Reason is that the internal representation of the limits for the ApplicationDataType are calculated by applying the CompuMethod.

- The such defined internal limits of the ApplicationDataType shall be within or equal the internalConstrs of the mapped ImplementationDataType.

- The limits of the ImplementationDatatype shall be within or equal to the limits defined by the size of the BaseType.

⌋

For a more detailed description of the properties that can be defined for data types (and data prototypes as well) see sections 5.4 and 5.4.2.

5.2.4.1.2 About Enumerations

[TPS_SWCT_1243] Definition of enumeration types ⌈ In the AUTOSAR meta-model, an enumeration is not implemented by means of an ApplicationCompositeDataType. Instead, a range of integer numbers can be used as a structural description for a single ApplicationPrimitiveDataType.

The mapping of the integer numbers on *labels* in the scope of the definition of an enumeration is considered as part of the semantical definition via an attached CompuMethod and not as part of the structural description. ⌋*(RS_SWCT_3216)*

Details are explained in section 5.5.1.1.

5.2.4.1.3 Data Types for Calibration Parameters

[TPS_SWCT_1244] Data types for calibration parameters are also described as primitive types ⌈ Data types for calibration parameters are from the application perspective also described as primitive types. This is obvious, if they are simple values (category VALUE). Also the category STRING is treated as a primitive type on application level.

Less obvious is the fact, that ApplicationDataTypes of the categories VAL_BLK, COM_AXIS, RES_AXIS, CURVE and MAP are not described as composite data types (as long as the application level is concerned) though they possess some kind of internal structure.

This is due to the fact, that in contrast to ApplicationCompositeDataTypes they are NOT composed in a self-similar way of other AutosarDataTypes. Instead of this, their substructure needs a special description in oder to be compatible with existing calibration techniques. ⌋

[TPS_SWCT_1245] SwDataDefProps control the structure of calibration parameters ⌈ The substructure of these types is attached to the SwDataDefProps. By this, it can also be applied on the level of prototypes or other artifacts, where the SwDataDefProps come into play. For details on these part of the SwDataDefProps see chapters 5.4.4 and 5.5.5. ⌋

5.2.4.1.4 Data Types for Textual Strings

[constr_1093] Definition of textual strings ⌈ An ApplicationPrimitiveDataType of Category STRING shall have a swTextProps which determines the arraySizeSemantics and swMaxTextSize. ⌋

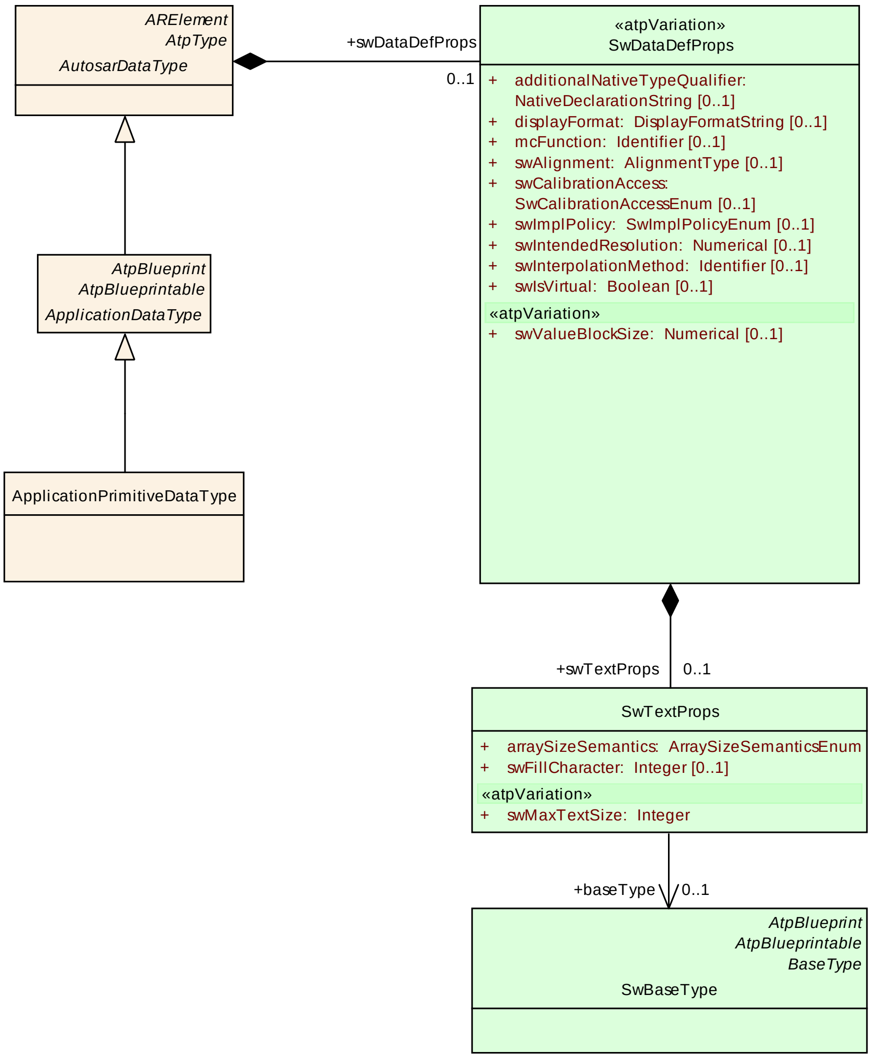

Figure 5.7: Specification of textual strings

Class	SwTextProps			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This class expresses particular properties applicable to strings in variables or calibration parameters.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
arraySizeSemantics	ArraySizeSemanticsEnum	1	attr	This attribute controls the semantics of the arraysize for the array representing the string in an ImplementationDataType. It is there to support a safe conversion between ApplicationDatatype and ImplementationDatatype, even for variable length strings as required e.g. for Support of SAE J1939. In conjunction with swFillCharacter, it provides the following options: • FixedLengthString: FixedSize - no fillcharacter • TerminatedStringFixedLengthCommunication: FixedSize - with fillcharacter • VariableLengthString: VariableSize - no fillcharacter • TerminatedStringVariableLengthCommunication: VariableSize with fillcharacter
baseType	SwBaseType	0..1	ref	This is the base type of one character in the string. In particular this baseType denotes the intended encoding of the characters in the string on level of ApplicationDataType. **Tags:** xml.sequenceOffset=30
swFillCharacter	Integer	0..1	attr	Filler character for text parameter to pad up to the maximum length swMaxTextSize. The value will be interpreted according to the encoding specified in the associated base type of the data object., e.g. 0x30 (hex) represents the ASCII character zero as filler character and 0 (dez) represents an end of string as filler character. The usage of the fill character depends on the arraySizeSemantics. **Tags:** xml.sequenceOffset=40
swMaxTextSize	Integer	1	attr	Specifies the maximum text size in characters. Note the the size in bytes depends on the encoding in the corresponding baseType. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note

Table 5.11: SwTextProps

[TPS_SWCT_1127] Byte array with variable size ⌈ SwTextProps can be used to define byte arrays of variable size. ⌋*(RS_SWCT_3182)*

[TPS_SWCT_1128] RecordLayout needed for special cases ⌈ A RecordLayout is needed for strings in special cases, i.e. the string might have a more complicated representation than an array of characters (e.g. has a separate length field). ⌋*(RS_SWCT_3128)*

[TPS_SWCT_1246] RecordLayout may be required for A2L generation ⌈ A RecordLayout may still be required for the generation of A2L if the string is part of calibration data. ⌋

The following series of XML fragments exemplifies the definition of a data type for the representation of a textual string. First, the applicable ApplicationPrimitive-DataType is defined:

Listing 5.1: Example for the definition of a string ApplicationPrimitiveDataType

```xml
<AR-PACKAGES>
  <AR-PACKAGE>
    <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
    <ELEMENTS>
      <APPLICATION-PRIMITIVE-DATA-TYPE>
        <SHORT-NAME>MyApplicationStringType</SHORT-NAME>
        <CATEGORY>STRING</CATEGORY>
        <SW-DATA-DEF-PROPS>
          <SW-DATA-DEF-PROPS-VARIANTS>
            <SW-DATA-DEF-PROPS-CONDITIONAL>
              <SW-TEXT-PROPS>
                <ARRAY-SIZE-SEMANTICS>VARIABLE-SIZE</ARRAY-SIZE-
                    SEMANTICS>
                <SW-MAX-TEXT-SIZE>50</SW-MAX-TEXT-SIZE>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">MyTextBaseType</
                    BASE-TYPE-REF>
              </SW-TEXT-PROPS>
              <INVALID-VALUE>
                <APPLICATION-VALUE-SPECIFICATION>
                  <CATEGORY>STRING</CATEGORY>
                  <SW-VALUE-CONT>
                    <SW-VALUES-PHYS>
                      <VT>MyInivalidStringValue</VT>
                    </SW-VALUES-PHYS>
                  </SW-VALUE-CONT>
                </APPLICATION-VALUE-SPECIFICATION>
              </INVALID-VALUE>
            </SW-DATA-DEF-PROPS-CONDITIONAL>
          </SW-DATA-DEF-PROPS-VARIANTS>
        </SW-DATA-DEF-PROPS>
      </APPLICATION-PRIMITIVE-DATA-TYPE>
    </ELEMENTS>
  </AR-PACKAGE>
```

Note that the category is set to the value STRING. Also the ApplicationPrimitiveDataType SwTextProps defined in the role swTextProps indicate the width of the string and also define (by means of the reference to baseType) the encoding this string data type is supposed to utilize.

Please note further that the listing also contains the definition of an invalidValue for the string data type. The next step is the definition of an ImplementationDataType that represents the string type on the implementation level:

Listing 5.2: Example for the definition of a string ImplementationDataType

```
<AR-PACKAGE>
  <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
  <ELEMENTS>
    <IMPLEMENTATION-DATA-TYPE>
      <SHORT-NAME>CharacterType</SHORT-NAME>
      <CATEGORY>VALUE</CATEGORY>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <BASE-TYPE-REF DEST="SW-BASE-TYPE">MyTextBaseType</BASE
                -TYPE-REF>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
    </IMPLEMENTATION-DATA-TYPE>
    <IMPLEMENTATION-DATA-TYPE>
      <SHORT-NAME>MyImplementationStringType</SHORT-NAME>
      <CATEGORY>STRUCTURE</CATEGORY>
      <SUB-ELEMENTS>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
          <SHORT-NAME>size</SHORT-NAME>
          <CATEGORY>TYPE_REFERENCE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-
                    DATA-TYPE">uint8</IMPLEMENTATION-DATA-TYPE-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
        <IMPLEMENTATION-DATA-TYPE-ELEMENT>
          <SHORT-NAME>string</SHORT-NAME>
          <ARRAY-SIZE>50</ARRAY-SIZE>
          <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
          <SUB-ELEMENTS>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
              <SHORT-NAME>character</SHORT-NAME>
              <CATEGORY>TYPE_REFERENCE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <IMPLEMENTATION-DATA-TYPE-REF DEST="
                        IMPLEMENTATION-DATA-TYPE">CharacterType</
                        IMPLEMENTATION-DATA-TYPE-REF>
```

```
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE-ELEMENT>
      </SUB-ELEMENTS>
    </IMPLEMENTATION-DATA-TYPE-ELEMENT>
  </SUB-ELEMENTS>
</IMPLEMENTATION-DATA-TYPE>
</ELEMENTS>
</AR-PACKAGE>
```

The interesting part about this definition is the fact that on the implementation level, it was decided to implement the string as a structure of a size element (that goes by the shortName "size") and a value element (that goes by the shortName "string") which in turn is defined as an array data type and therefore has a sub-element that goes by the shortName "character" and references the ImplementationDataType "CharacterType" which is also defined in the listing.

Please note that both the ApplicationPrimitiveDataType named "MyApplicationStringType" as well as the ImplementationDataType named "CharacterType" reference the same SwBaseType named "MyTextBaseType" which is defined in the following XML fragment:

Listing 5.3: Example for the definition of a string SwBaseType

```
<AR-PACKAGE>
  <SHORT-NAME>BaseTypes</SHORT-NAME>
  <ELEMENTS>
    <SW-BASE-TYPE>
      <SHORT-NAME>MyTextBaseType</SHORT-NAME>
      <BASE-TYPE-SIZE>16</BASE-TYPE-SIZE>
      <BASE-TYPE-ENCODING>UTF-8</BASE-TYPE-ENCODING>
    </SW-BASE-TYPE>
  </ELEMENTS>
</AR-PACKAGE>
```

The contribution of this definition of SwBaseType to the overall definition of a string data type is represented by the definition of the encoding (which is set to UTF-8). However, there ist still one important part missing, i.e. the definition of the mapping of ApplicationPrimitiveDataType to ImplementationDataType (and vice versa):

Listing 5.4: Example for the definition of the applicable DataTypeMappingSet

```
<AR-PACKAGE>
  <SHORT-NAME>DataTypeMappingSets</SHORT-NAME>
  <ELEMENTS>
    <DATA-TYPE-MAPPING-SET>
      <SHORT-NAME>theExample</SHORT-NAME>
      <DATA-TYPE-MAPS>
        <DATA-TYPE-MAP>
          <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-ARRAY-DATA-
              TYPE">MyApplicationStringType</APPLICATION-DATA-TYPE-
              REF>
```

```
          <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-
             TYPE">MyImplementationStringType</IMPLEMENTATION-DATA-
             TYPE-REF>
        </DATA-TYPE-MAP>
      </DATA-TYPE-MAPS>
    </DATA-TYPE-MAPPING-SET>
  </ELEMENTS>
</AR-PACKAGE>
```

5.2.4.2 Application Composite Data Types

[TPS_SWCT_1247] ApplicationArrayDataType and ApplicationRecordDataType ⌈ The meta-classes ApplicationArrayDataType and ApplicationRecordDataType (details are depicted in Figure 5.8) provide the means to define composite data types. Such a composite data type is required if the application software wants to have access to the individual elements of the composite as well as to do operations with the whole composite, e.g. wants to communicate the complete record or array in a single transaction.

It is possible to use a combination of ApplicationArrayDataType and ApplicationRecordDataType, so that an ApplicationArrayDataType could be defined as ApplicationRecordElement of a ApplicationRecordDataType and in the same manner a ApplicationRecordDataType could be used as the base type of an ApplicationArrayDataType. The creation of nested ApplicationCompositeDataTypes is also possible. ⌋

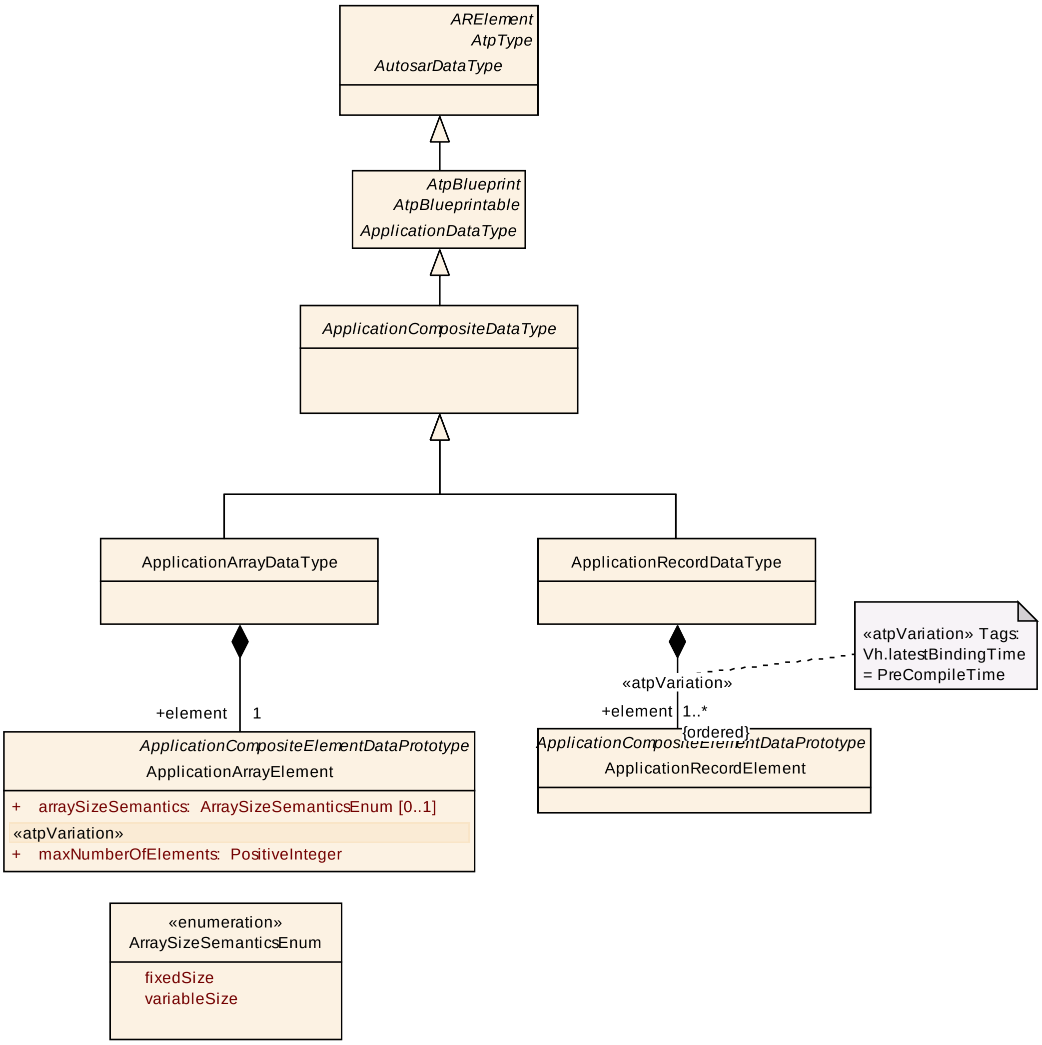

Figure 5.8: Summary of `CompositeType`

5.2.4.2.1 ApplicationArrayDataType

[TPS_SWCT_1078] Configurable array size ⌈ An `ApplicationArrayDataType` may[3] contain `maxNumberOfElements` `ApplicationArrayElement`s. Each of these `ApplicationArrayElement`s has the same data type. When referring to an element of an `ApplicationArrayDataType` within a software-component description, the element-index runs from 0 to the value of `maxNumberOfElements`-1.⌋ *(RS_SWCT_3144)*

[3]this applies although the multiplicity in the meta-model is 1. In fact, it would be possible to model `ApplicationArrayDataType` without `ApplicationArrayElement`. The latter exists only so that it can be the target of a reference within an AUTOSAR XML file

Class	**ApplicationArrayDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationCompositeDataType,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
element	ApplicationArrayElement	1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.

Table 5.12: ApplicationArrayDataType

Class	**ApplicationArrayElement**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of the elements of an application array data type.			
Base	ARObject,ApplicationCompositeElementDataPrototype,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls how the information about the array size shall be interpreted.
maxNumberOfElements	PositiveInteger	1	attr	The maximum number of elements that the array can contain. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 5.13: ApplicationArrayElement

Please note that the information about the number of elements of a specific `ApplicationArrayDataType` is not absolute but allows for further interpretation.

[TPS_SWCT_1076] Number of elements of a specific `ApplicationArrayDataType` might vary at run-time ⌈ That is, there are cases where the number of elements of a specific `ApplicationArrayDataType` might vary at run-time. To be precise, the number of elements might vary between 0 and the value denoted by `maxNumberOfElements`. For this purpose an additional attribute `arraySizeSemantics` is available that can be used to clarify the meaning of `maxNumberOfElements`.

For clarification, it might indeed happen that the actual number of elements in a specific `ApplicationArrayDataType` yields 0 simply because the respective `DataPrototype` is part of a higher-level protocol where under certain circumstances the `DataPrototype` of `ApplicationArrayDataType` is simply not required for expressing a given semantics. ⌋*(RS_SWCT_3180, RS_SWCT_3181, RS_SWCT_3144)*

Enumeration	ArraySizeSemanticsEnum
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes
Note	This type controls how the information about the number of elements in an ApplicationArrayDataType is to be interpreted.
Literal	*Description*
fixedSize	This means that the ApplicationArrayDataType will always have a fixed number of elements.
variableSize	This implies that the actual number of elements in the ApplicationArrayDataType might vary at run-time. The value of arraySize represents the maximum number of elements in the array.

Table 5.14: ArraySizeSemanticsEnum

Please note that the ability to define the semantic meaning of maxNumberOfElements is not only limited to the application data type level. The same approach also applies for ImplementationDataType.

[constr_1152] category of ApplicationArrayElement and AutosarDataType referenced in the role type shall be kept in sync ⌈ The value of category of an ApplicationArrayElement shall always be identical to the value of category of the AutosarDataType referenced by the ApplicationArrayElement. ⌋

[TPS_SWCT_1256] Definition of multi-dimensional array data types ⌈ In order to describe multi dimensional arrays an ApplicationArrayElement references again another ApplicationArrayDataType. Hereby, one ApplicationArrayDataType per dimension is required.

This multiple dimensions do have a well-defined correlation to the individual dimensions of an ImplementationDataType of category ARRAY when the ApplicationArrayDataType is mapped to an ImplementationDataType as described in section 5.2.2

The ApplicationArrayElements are mapping in the order of the ApplicationArrayElement to ApplicationArrayDataType references to ImplementationDataTypeElements in the order of first ImplementationDataTypeElement of the ImplementationDataType to leaf ImplementationDataTypeElement.

In other words the ApplicationArrayElement of the top level ApplicationArrayDataType relates to the first ImplementationDataTypeElement of the ImplementationDataType. The ApplicationArrayElement of the referenced ApplicationArrayDataTypes relates to the sub ImplementationDataTypeElements in the order of the ApplicationArrayElement -> ApplicationArrayDataType references. ⌋*(RS_SWCT_3216)*

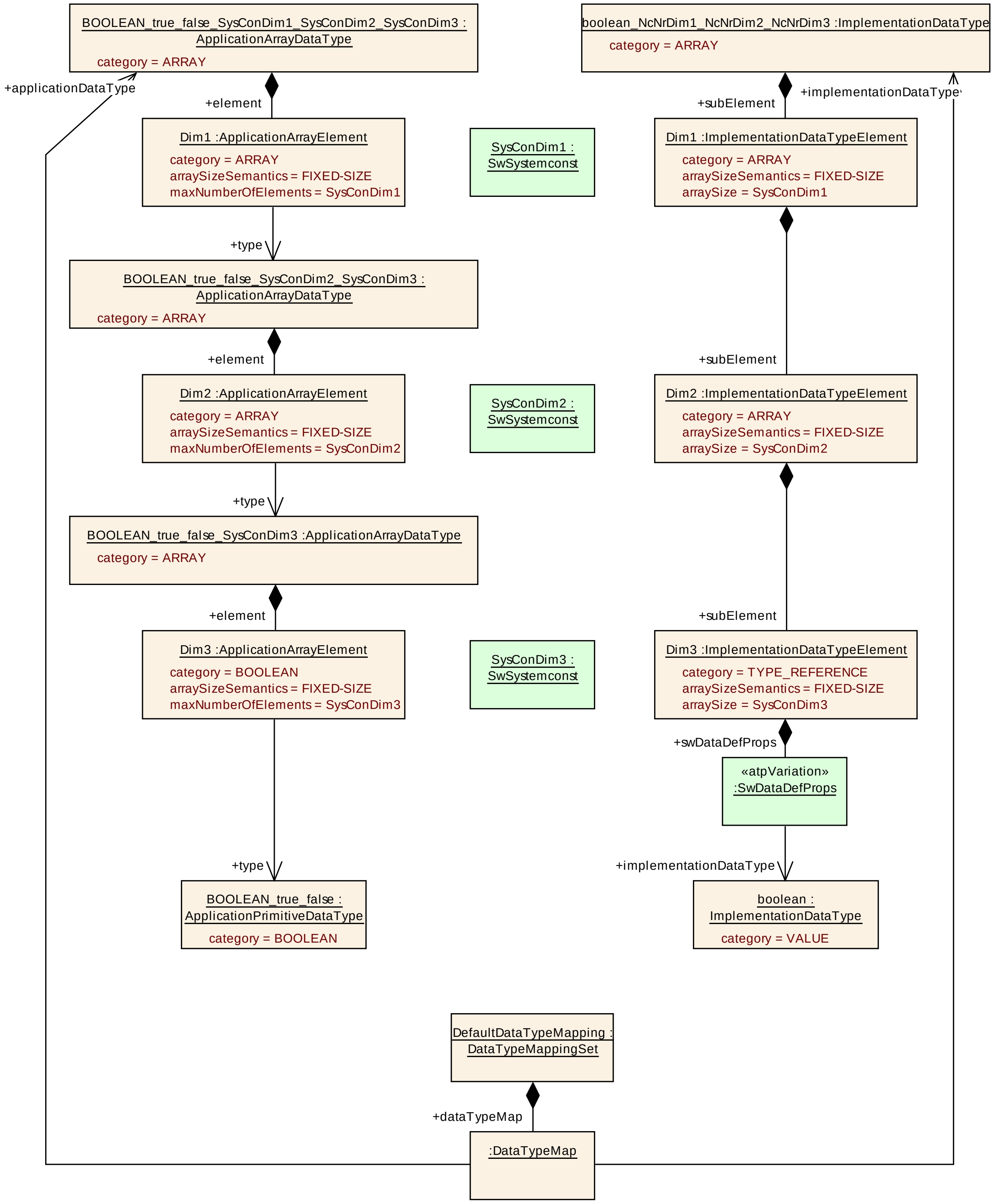

Figure 5.9: Example of a three dimensional array type

Figure 5.9 shows a three dimensional array described with a set of `ApplicationArrayDataType`s on the left hand side. The array element is typed by an `ApplicationPrimitiveDataType` of category `BOOLEAN`. On the right hand side the implementation of the three dimensional array is described with an ImplementationDataType which contains three nested ImplementationDataTypeElements.

Matching ApplicationArrayElements and ImplementationDataTypeElements are shown on the same layer. For the sake of clarity correlating maxNumberOfElements and arraySize attributes are described with the identical instance of a SwSystemconst instead of a value. Further details of variant rich M1 models are not in the scope of this example.

The data type of the array element is described by the ApplicationArrayDataType with the means of a ApplicationPrimitiveDataType of category BOOLEAN. In order to fulfill [constr_1152] the category of ApplicationArrayElement "Dim3" is set to BOOLEAN. This ApplicationPrimitiveDataType "BOOLEAN" correlates to the ImplementationDataType "boolean" of category VALUE which is typically the boolean type of the AUTOSAR Platform Types. Please note here [constr_1063].

5.2.4.2.2 ApplicationRecordDataType

[TPS_SWCT_1249] ApplicationRecordDataType ⌈ A declaration of ApplicationRecordDataType describes a non-empty set of objects, each of which has a unique identifier with respect to the ApplicationRecordDataType and each has an own ApplicationDataType. The shortName of each ApplicationRecordElement within the scope of an ApplicationRecordDataType shall be unique. ⌋*(RS_SWCT_3216)*

Class	**ApplicationRecordDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	ARElement,ARObject,ApplicationCompositeDataType,ApplicationDataType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,AutosarDataType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
element (ordered)	ApplicationRecordElement	1..*	aggr	Specifies an element of a record. The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 5.15: ApplicationRecordDataType

Class	**ApplicationRecordElement**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	ARObject,ApplicationCompositeElementDataPrototype,AtpFeature,Atp Prototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.16: ApplicationRecordElement

5.2.5 Implementation Data Type

[TPS_SWCT_1250] `ImplementationDataType` has been introduced to optimize the formal support for data type handling on the implementation level ⌈ The concept of an `ImplementationDataType` has been introduced to optimize the formal support for data type handling on the implementation level. That is, an `ImplementationDataType` conceptually corresponds to the level of (C) source code. For example, `ImplementationDataType`s have a direct impact on the contract (please find an explanation of this term in [2]) of a software-component and the RTE. ⌋ *(RS_SWCT_3217)*

	Root Element				Attribute Existence per Category						
Attributes of SwDataDefProps	**ImplementationDataType**	**ImplementationDataTypeElement**	**SwPointerTargetProps**	**SwServiceArg**	**VALUE**	**DATA_REFERENCE**	**FUNCTION_REFERENCE**	**TYPE_REFERENCE**	**STRUCTURE**	**UNION**	**ARRAY**
additionalNativeTypeQualifier	x	x	x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1
annotation	x	x	x	x	*	*	*	*	*	*	*
baseType	x	x	x	x	1						
compuMethod	x	x	x	x	0..1			0..1			
dataConstr	x	x	x	x	0..1						
displayFormat	x	x			0..1				0..1	0..1	0..1
implementationDataType	x	x	x	x				1			
invalidValue	x	x	x		0..1						
mcFunction											
swAddrMethod	x	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x				0..1	0..1	0..1		0..1	0..1	0..1
swBitRepresentation											
swCalibrationAccess	x	x			0..1				0..1	0..1	0..1
swCalprmAxisSet											
swComparisonVariable											
swDataDependency											
swHostVariable											

	Root Element				Attribute Existence per Category						
Attributes of SwDataDefProps	ImplementationDataType	ImplementationDataTypeElement	SwPointerTargetProps	SwServiceArg	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
swImplPolicy	x		x	x	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution											
swInterpolationMethod											
swIsVirtual											
swPointerTargetProps	x	x	x	x		1	1				
swPointerTargetProps .swDataDefProps	x	x	x	x		1					
swPointerTargetProps .functionPointerSignature	x	x	x	x			1				
swRecordLayout											
swRefreshTiming	x	x	x	x	0..1				0..1	0..1	0..1
swTextProps											
swValueBlockSize											
unit											
valueAxisDataType											
Other Attributes											
subElement: Implementation-DataTypeElement	x	x							1..*	1..*	1
subElement .arraySizeSemantics	x	x									0..1
subElement.arraySize	x	x									1

Table 5.17: Allowed Attributes vs. `category` for Implementation Data Types

[TPS_SWCT_1251] Limited set of values for `category` are applicable for `ImplementationDataType` ⌈ Like any `AutosarDataType`, also the data types on implementation level are characterized by its `category` and its `SwDataDefProps`. For a given `category`, only a limited set of attributes of the `SwDataDefProps` makes sense. ⌋*(RS_SWCT_3217)*

[constr_1009] `SwDataDefProps` applicable to `ImplementationDataTypes` ⌈ A complete list of the `SwDataDefProps` and other attributes and their multiplicities which are allowed for a given `category` is shown in table 5.17. ⌋

This list makes use of the `SwDataDefProps` and other meta-model elements which are explained in detail in the further sections of this chapter.

[TPS_SWCT_1252] `ImplementationDataType` can express concepts not available on application level ⌈ As a consequence of the specific focus, it is possible to express concepts with an `ImplementationDataType` that are not supported on the the application level, i.e. by `ApplicationDataType`:

- ImplementationDataType supports the definition of pointers
- It is possible to define "alias" names just as in a typedef
- It is possible to define nested ImplementationDataTypes but in contrast to the concept implemented for ApplicationDataType these implement a direct aggregation of sub-elements rather than applying the type-prototype pattern.

⌋*(RS_SWCT_3217)*

The general structure of ImplementationDataType is sketched in Figure 5.10. If a specific ImplementationDataType is supposed to define a composite data type the ImplementationDataType aggregates ImplementationDataTypeElements.

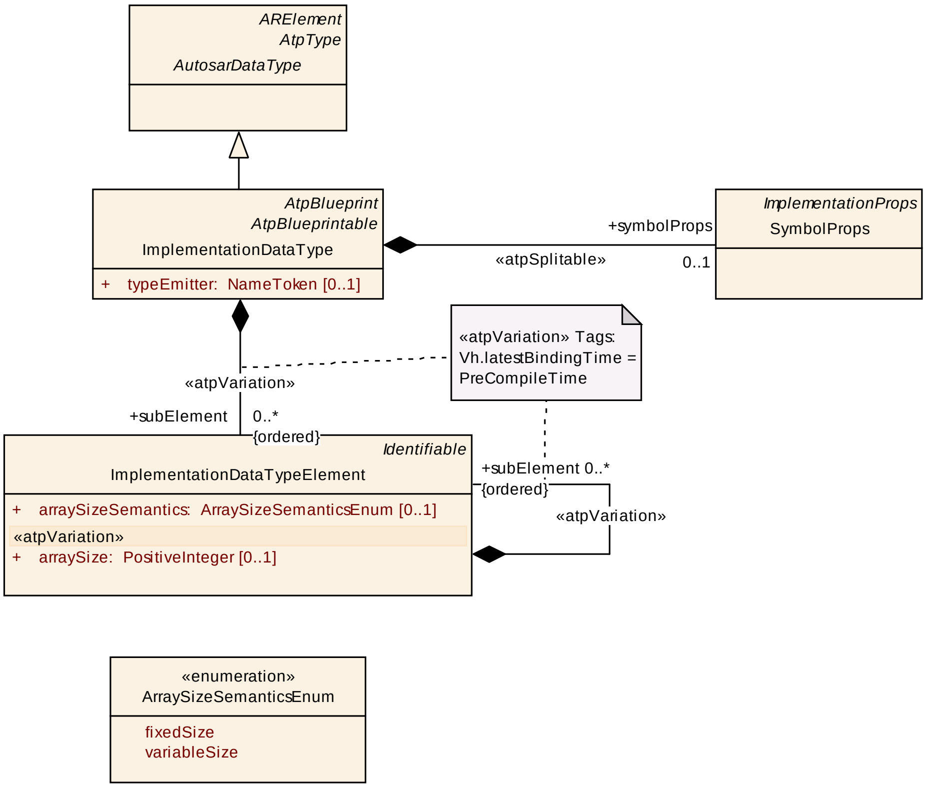

Figure 5.10: ImplementationDataType overview

Class	**ImplementationDataType**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Describes a reusable data type on the implementation level. This will typically correspond to a typedef in C-code. **Tags:** atp.recommendedPackage=ImplementationDataTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,Autosar DataType,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
subElement (ordered)	ImplementationDataTypeElement	*	aggr	Specifies an element of an arrray or a struct type. The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
symbolProps	SymbolProps	0..1	aggr	This represents the SymbolProps for the ImplementationDataType. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=shortName
typeEmitter	NameToken	0..1	attr	This attribute is used to control which part of the AUTOSAR toolchain is supposed to trigger data type definitions.

Table 5.18: ImplementationDataType

[TPS_SWCT_1253] Rules applies for the usage of the attribute `ImplementationDataType.typeEmitter` ⌈ The following set of rules applies for the usage of the attribute `ImplementationDataType.typeEmitter`:

- If the value of attribute `typeEmitter` is NOT defined **and** a `nativeDeclaration` is provided the RTE generator shall generate the corresponding data type definition[4].
- If the value of attribute `typeEmitter` is set to "RTE" **and** a `nativeDeclaration` is provided the RTE generator shall generate the corresponding data type definition.
- If the value of the attribute `typeEmitter` is set to "RTE" **and** no `nativeDeclaration` is provided the RTE generator shall issue an error message.
- If the value of attribute `typeEmitter` is set to anything else but "RTE" the RTE generator shall silently **not** generate the corresponding data type definition regardless of the existence of nativeDeclaration attribute.

⌋*(RS_SWCT_3217)*

Note that the rules listed above imply that the allowed values of the attribute `typeEmitter` are not constrained with the singular exception that the definition of the behavior in case of "RTE" is claimed by AUTOSAR. Other values can be provided; the consequences of this provision are implementation-dependent and outside the scope of the definition of the AUTOSAR standard.

[4]This rule represents the behavior before the attribute `typeEmitter` was introduced. The rule has specifically been added in order to support a backwards-compatible behavior.

[TPS_SWCT_1248] Nested definition of `ImplementationDataType` ⌈ If an `ImplementationDataTypeElement` also represents a composite data type it can aggregate `ImplementationDataTypeElements` in the role of `subElement`. Again, the type-prototype pattern does not apply in this case. ⌋*(RS_SWCT_3217)*

[constr_1106] Structure shall have at least one element ⌈ An `ImplementationDataType` or `ImplementationDataTypeElement` of `category` `STRUCTURE` shall own at least one `ImplementationDataTypeElement`. ⌋

[constr_1107] Union shall have at least one element ⌈ An `ImplementationDataType` or `ImplementationDataTypeElement` of `category` `UNION` shall own at least one `ImplementationDataTypeElement`. ⌋

Class	**ImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::CommonStructure::ImplementationDataTypes			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope, where it is aggregated. This element either consists of further subElements or it is further defined via its swDataDefProps. There are several use cases within the system of ImplementationDataTypes fur such a local declaration: • It can represent the elements of an array, defining the element type and array size • It can represent an element of a struct, defining its type • It can be the local declaration of a debug element.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arraySize	PositiveInteger	0..1	attr	The existence of this attributes (if bigger than 0) defines the size of an array and declares that this ImplementationDataTypeElement represents the type of each single array element. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
arraySizeSemantics	ArraySizeSemanticsEnum	0..1	attr	This attribute controls the meaning of the value of the array size.
subElement	ImplementationDataTypeElement	*	aggr	Element of an array or struct in case of a nested declaration (i.e. without using "typedefs"). The aggregation of ImplementionDataTypeElement is subject to variability with the purpose to support the conditional existence of elements inside a ImplementationDataType representing a structure. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
swDataDefProps	SwDataDefProps	0..1	aggr	The properties of this data type element.

Table 5.19: ImplementationDataTypeElement

[TPS_SWCT_1254] `ImplementationDataType` with array semantics ⌈ Of course, it is also possible to define an `ImplementationDataType` that provides array semantics. ⌋*(RS_SWCT_3217)*

[TPS_SWCT_1006] `arraySize` of `ImplementationDataType` shall be used to define the size of the array ⌈ The primitive attribute `arraySize` of `ImplementationDataType` shall be used to define the size of the array. ⌋

[TPS_SWCT_1007] Semantics of array index ⌈ For an `ImplementationDataType` that implements an array data type, the semantics of the array index is such that

- it shall start with the value 0
- it shall run to the value of `arraySize` -1

⌋

[constr_1105] Value of `arraySize` ⌈ The value of the attribute `arraySize` of an `ImplementationDataTypeElement` owned by an `ImplementationDataType` or `ImplementationDataTypeElement` of `category` `ARRAY` shall be greater than 0. ⌋

Please note that the array size is **not** defined as an attribute of the `ImplementationDataType` which stands for the whole array. It is actually defined as an attribute of the `ImplementationDataTypeElement` which is describing the array element (note that the same pattern is used in `ApplicationArrayDataType`).

Consequently, if a "struct" element represents an array this specific struct-element is given by an `ImplementationDataTypeElement` of category ARRAY which in turn aggregates another `ImplementationDataTypeElement` of e.g. category VALUE representing the array element and containing the size.

[TPS_SWCT_1255] Indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time ⌈ It is also possible to indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time. ⌋*(RS_SWCT_3217)*

Please find more information about this topic in section 5.2.4.2.

An `ImplementationDataType` is also allowed to have `SwDataDefProps` (this feature is inherited from `AutosarDataType`), i.e. it can define various specific structural and semantical attributes. Table 5.41 shows which `SwDataDefProps` will be typically used here.

[TPS_SWCT_1257] `ImplementationDataType` or the aggregated `ImplementationDataTypeElements` do not form closed sets ⌈ As figures 5.10 shows, an `ImplementationDataType` or the aggregated `ImplementationDataTypeElements` do not form closed sets but refer to further type definitions in one of four distinctive ways, depending on whether the type is implemented via a base type, a data or function pointer, or a reference to another implementation data type:

1. Reference to an underlying `SwBaseType`, corresponds to category VALUE.

2. Reference to `BswModuleEntry` in `SwPointerTargetProps` corresponds to category FUNCTION_REFERENCE.
3. `SwDataDefProps` in `SwPointerTargetProps` corresponds to category DATA_REFERENCE.
4. Reference to another `ImplementationDataType` corresponds to category TYPE_REFERENCE.

⌋*(RS_SWCT_3217)*

At the end, all the "leafs" of the complete tree formed by these references shall end up in `SwBaseType`s. Figure 5.11, 5.12, and Figure 5.13 illustrate more examples about Typedefs and references.

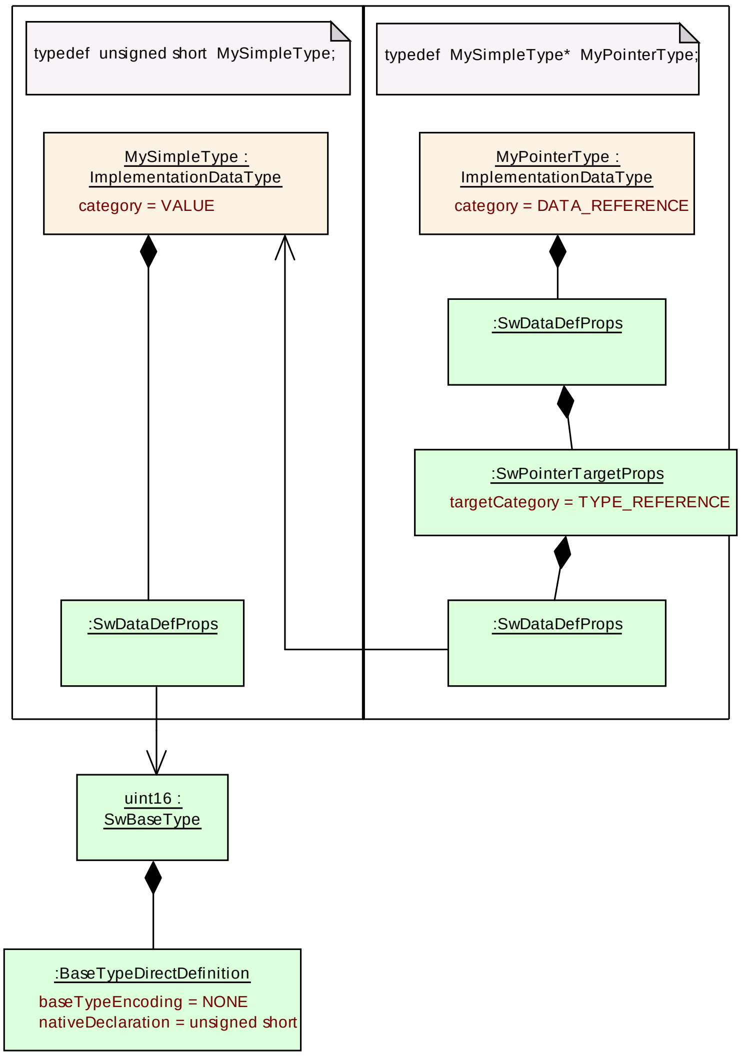

Figure 5.11: Example (1) for TypeDefs

[TPS_SWCT_1258] Definition of a pointer to data ⌈ The definition of a data pointer requires a special meta-class `SwPointerTargetProps` which aggregates another `SwDataDefProps`. This mechanism allows to describe the `Category` and properties of the pointer object itself as well as the Category and properties of its target data type. ⌋(RS_SWCT_3217)

[constr_1177] Allowed category for SwPointerTargetProps ⌈ The value of category for SwPointerTargetProps can only be one of TYPE_REFERENCE, DATA_REFERENCE, or FUNCTION_REFERENCE. The only exception from this rule applies if the swDataDefProps owned by the SwPointerTargetProps refers to a SwBaseType with native type declaration void*, in this case the value VALUE is also permitted. ⌋

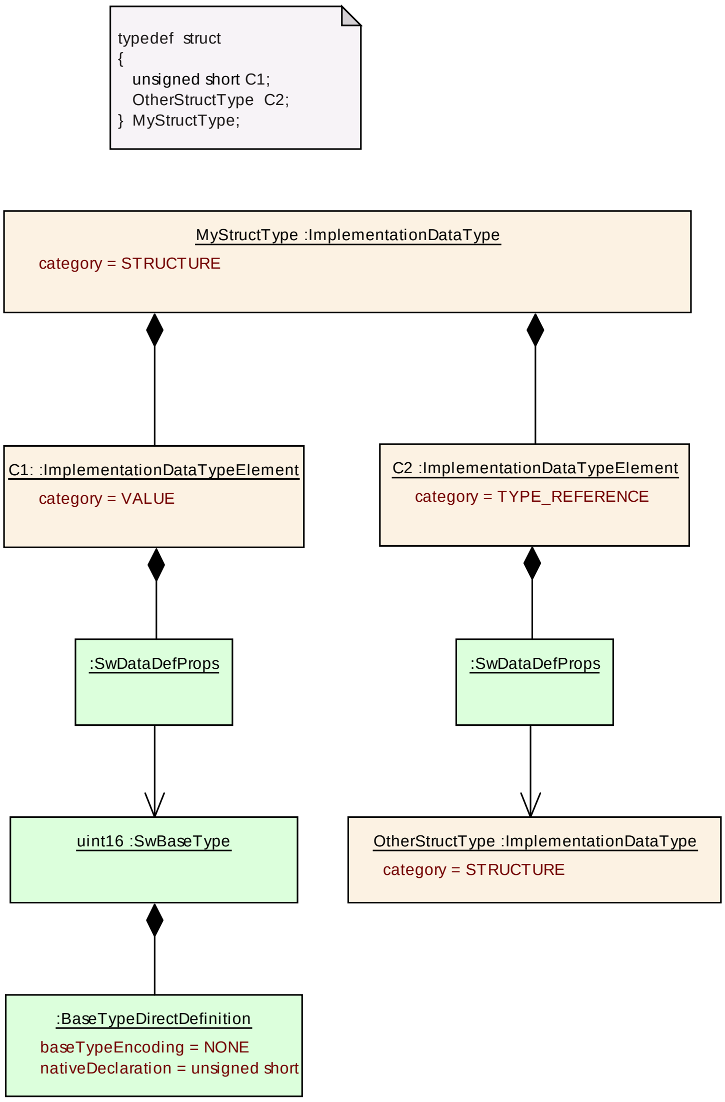

Figure 5.12: Example (2) for TypeDefs

[TPS_SWCT_1259] Definition of a pointer to a function ⌈ An Implementation-DataType or one of its sub-elements can also describe a function pointer. This completes its ability to declare all kinds of local data and of possible arguments used in library calls.

A function pointer is defined by the Category FUNCTION_REFERENCE and the association SwPointerTargetProps.functionPointerSignature that refers to a

BswModuleEntry. The latter essentially describes the signature of a function as explained in [7]. ⌋*(RS_SWCT_3217)*

Figure 5.13: Example (3) for TypeDefs

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

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
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 5.20: SwPointerTargetProps

The allowed existence and multiplicity of all the attributes of `SwDataDefProps` and other properties depend on the `category` of the `ImplementationDataType`.

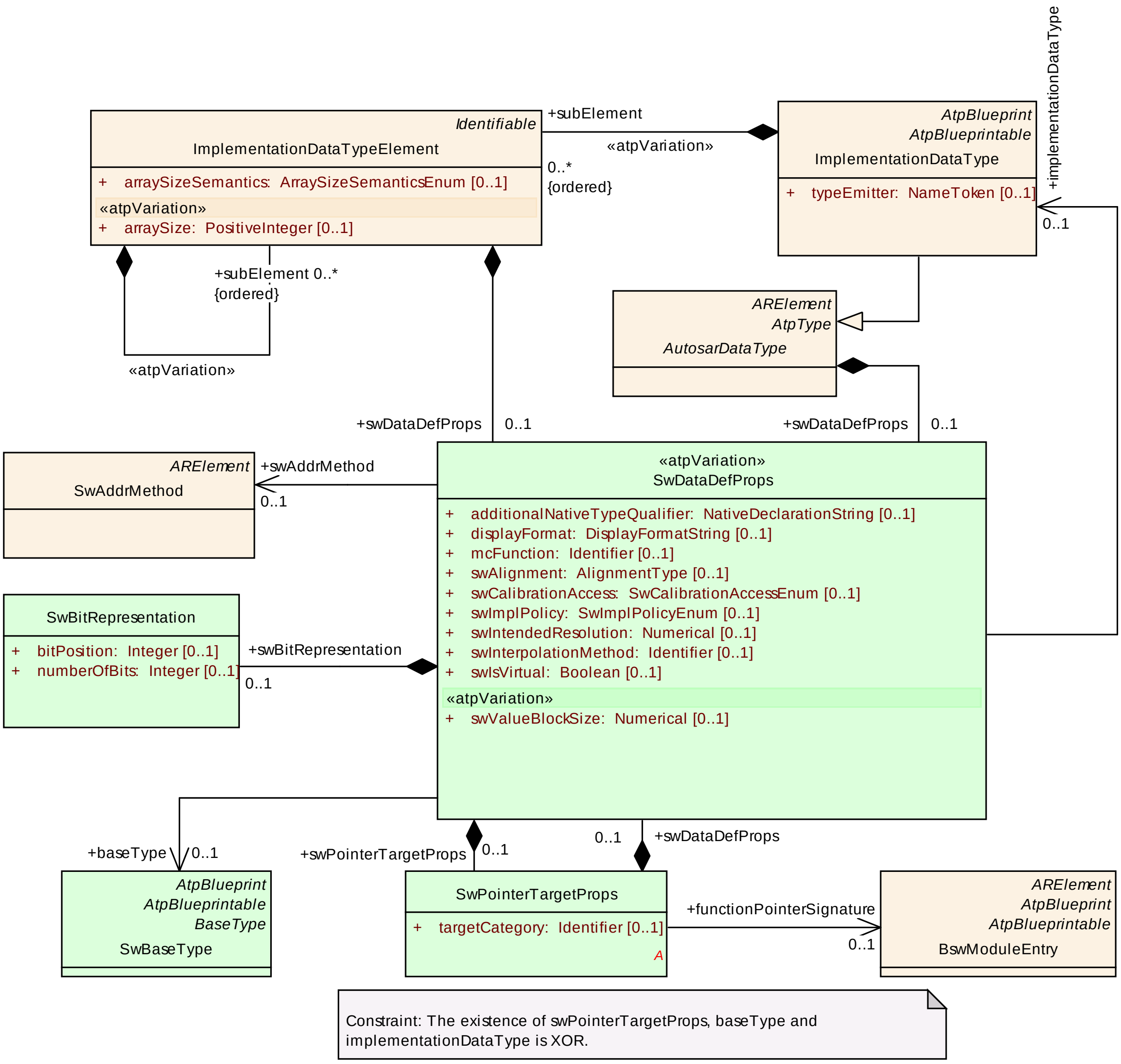

Figure 5.14: `SwDataDefProps` used in the context of `ImplementationDataType`

[constr_1178] Existence of attributes of `SwDataDefProps` in the context of `ImplementationDataType` ⌈ For the sake of removing possible sources of ambiguity, `SwDataDefProps` used in the context of `ImplementationDataType` can **only have one of**

- `baseType`
- `swPointerTargetProps`
- `implementationDataType`

⌋

Please note that an `ImplementationDataType` manifests itself in the source code of an RTE into which a `DataPrototype` typed by the `ImplementationDataType` is deployed. This implies potential naming conflicts if `ImplementationDataTypes` that have identical `shortNames` are deployed into a specific RTE.

[TPS_SWCT_1194] Symbolic name of an `ImplementationDataType` ⌈ To mitigate this potential hazard it is possible to provide the `ImplementationDataType` along with an accompanying symbolic name that can be used for resolving the name clash. The symbolic name is provided by means of the attribute `symbol` of the meta-class `SymbolProps` owned by `ImplementationDataType` in the role `symbolProps` (for more information, please refer to Figure 5.10). ⌋

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 5.21: ImplementationProps

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to attach with the symbol attribute a symbolic name that is conform to C language requirements to another meta-class, e.g. AtomicSwComponentType, that is a potential subject to a name clash on the level of RTE source code.			
Base	ARObject,ImplementationProps,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.22: SymbolProps

5.2.6 Base Type

[TPS_SWCT_1260] `SwBaseType` ⌈ `BaseType` is used to specify the basic level mentioned in chapter 5.1. In AUTOSAR, we use the meta-class `SwBaseType` which is derived from the abstract class `BaseType` due to other use cases for `BaseType` in ASAM HDO. ⌋

[TPS_SWCT_1261] Use case for `SwBaseType` ⌈ One use case for `SwBaseType` is to serve as input for the RTE generator. It will always appear at the "leaves" of data the types definitions which are relevant for RTE generation. It is used to generate the corresponding C-code typedef's in case the attribute `BaseTypeDirectDefinition.nativeDeclaration` exists. ⌋

[constr_1010] If `nativeDeclaration` does not exist ⌈ If `nativeDeclaration` does not exist in the `SwBaseType` it is required that the `shortName` (e.g. "uint8") of the corresponding `ImplementationDataType` is equal to a name of one of the Platform or Standard Types predefined in AUTOSAR code. ⌋ For more information on this refer to [21].

[TPS_SWCT_1263] Further use cases for SwBaseType ⌈ Within the basic software description, SwBaseType can be used (together with ImplementationDataTypes) for documentation or to specify variables for debugging. Furthermore, SwBaseTypes are required in the generation of support data for measurement and calibration tools. Please refer to [7] for details on these use cases. ⌋

A more detailed description of BaseTypes can also be found in *ASAM MCD 2 Harmonized Data Objects*.[5]

Class	**BaseType (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This abstract meta-class represents the ability to specify a platform dependant base type.			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
baseType Definition	BaseTypeDefini tion	1	aggr	This is the actual definition of the base type. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 5.23: BaseType

Class	**SwBaseType**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This meta-class represents a base type used within ECU software. **Tags:** atp.recommendedPackage=BaseTypes			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,BaseType,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.24: SwBaseType

Class	**BaseTypeDefinition (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This meta-class represents the ability to define a basetype.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.25: BaseTypeDefinition

[5] The definition of *Harmonized Data Objects* can be retrieved from ASAM at www.asam.net. Access is limited to ASAM members.

Class	**BaseTypeDirectDefinition**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	ARObject,BaseTypeDefinition			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
baseType Encoding	BaseTypeEnco dingString	1	attr	This specifies, how an object of the current BaseType is encoded e.g.. in an ECU in a message sequence. **Tags:** xml.sequenceOffset=90
baseType SizeDefinti onType	BaseTypeSizeD efinition	1	aggr	This aggregation is necessary to specify the exact sequence of properties in the xml-file. It represents the size of the BaseType. **Tags:** xml.roleElement=false; xml.roleWrapper Element=false; xml.sequenceOffset=40; xml.type Element=false; xml.typeWrapperElement=false
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:** xml.sequenceOffset=110
memAlign ment	Integer	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "1" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. **Tags:** xml.sequenceOffset=100

nativeDecl aration	NativeDeclarati onString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example
				
				BaseType with
				`shortName: "MyUnsignedInt"`
				`nativeDeclaration: "unsigned short"`
				
				Results in
				`typedef unsigned short MyUnsignedInt;`
				
				If the attribute is not defined the referring ImplementationDataTypes will not be generated as a typedef by RTE.
				
				If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseTypeSize. This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems.
				
				Tags: xml.sequenceOffset=120

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

Table 5.26: BaseTypeDirectDefinition

Class	**BaseTypeSizeDefinition (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This abstract class represents the possible methods of defining the size of a BaseType.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.27: BaseTypeSizeDefinition

Class	**BaseTypeAbsSize**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This is the absolute size of the basetype. In this case the BaseType is of fixed length.			
Base	ARObject,BaseTypeSizeDefinition			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
baseTypeSize	Integer	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:** xml.sequenceOffset=60

Table 5.28: BaseTypeAbsSize

Class	**BaseTypeMaxSize**			
Package	M2::AUTOSARTemplates::CommonStructure::BaseTypes			
Note	This is the maximum size of a BaseType in case of a dynamic BaseType.			
Base	ARObject,BaseTypeSizeDefinition			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
maxBaseTypeSize	Integer	0..1	attr	Describes the maximum length of the BaseType in bits **Tags:** xml.sequenceOffset=80

Table 5.29: BaseTypeMaxSize

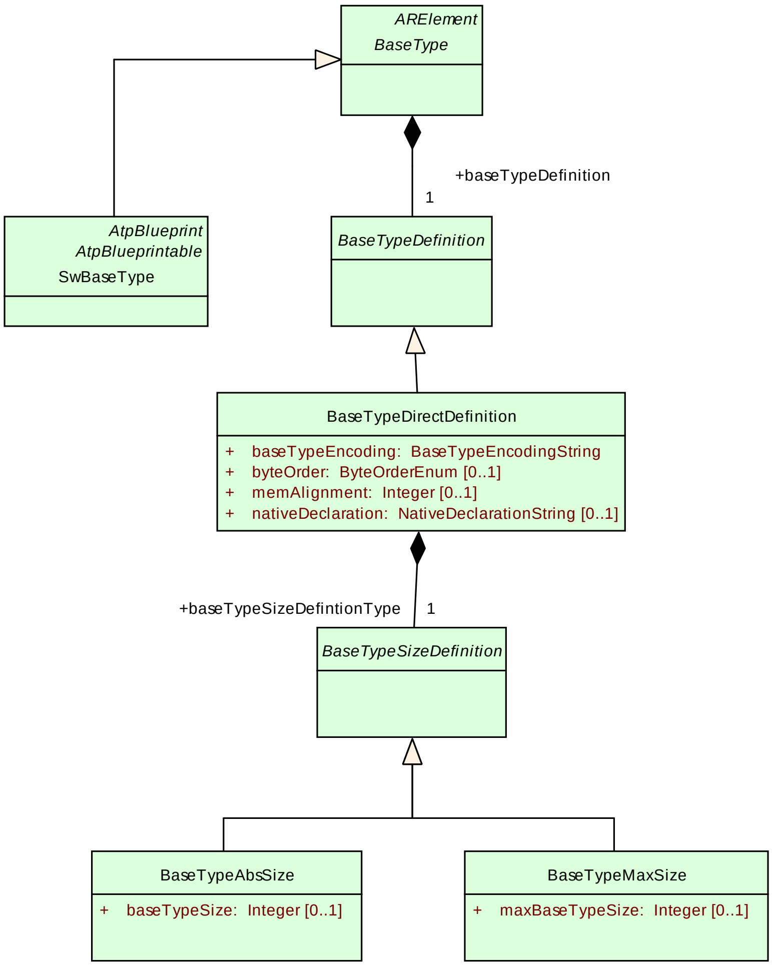

Figure 5.15: `BaseType`

Some additional hints to the properties of `SwBaseType`:

- **[constr_1011] CATEGORY of `SwBaseType`** ⌈ For `CATEGORY` only the values `FIXED_LENGTH` and `VARIABLE_LENGTH` are supported. ⌋

  [constr_1012] Value of CATEGORY is `FIXED_LENGTH` ⌈ In case of `FIXED_LENGTH` `BaseTypeAbsSize` is filled with content. ⌋

  [constr_1013] Value of CATEGORY is `VARIABLE_LENGTH` ⌈ In case of `VARIABLE_LENGTH` `BaseTypeMaxSize` is filled. ⌋

  In both cases the size is specified in bits.

- `baseTypeEncoding` specifies how the values of the base type are encoded. **[constr_1014] Supported value encodings for `SwBaseType`** ⌈ The supported values for this member are:

  - `1C`: One's complement
  - `2C`: Two's complement
  - `BCD-P`: Packed Binary Coded Decimals

- BCD-UP: Unpacked Binary Coded Decimals
- DSP-FRACTIONAL: Digital Signal Processor
- SM: Sign Magnitude
- IEEE754: floating point numbers
- ISO-8859-1: ASCII-Strings
- ISO-8859-2: ASCII-Strings
- WINDOWS-1252: ASCII-Strings
- UTF-8: UCS Transformation Format 8
- UCS-2: Universal Character Set 2
- NONE: Unsigned Integer
- VOID: corresponds to a void in C. The encoding is not formally specified here.
- BOOLEAN: This represents an unsigned integer to be interpreted as boolean. The value shall be interpreted as TRUE if the value of the unsigned integer is 1 and it shall be interpreted as FALSE if the value of the unsigned integer is 0.

  A CompuMethod shall be referenced by the corresponding AutosarDataType that implements the common sense behind the boolean concept, i.e. define a TEXTTABLE with two CompuScales: e.g. TRUE –> 1, FALSE –> 0.

⌋

- **[TPS_SWCT_1262] memAlignment and byteOrder are platform specific** ⌈ memAlignment and byteOrder are platform specific and therefore should be set only in use cases where this is really needed. These attributes shall be considered as optional. If an SwBaseType is platform specific then also the ImplementationDataType and software-component descriptions build on top of it become platform specific. ⌋

  However, there are use cases for SwBaseType where this does not matter: especially the calibration support format which is generated in ECU specific scope (and also contains SwBaseType, see [7]) could well be platform specific.

5.3 Data Prototypes

5.3.1 Overview

[TPS_SWCT_1264] Data prototypes implement a role of a data type ⌈ Generally speaking, a data prototype represents the implementation of a role of a data type within the definition of another data type, e.g. a "typed" data object declared within a software component or a port interface. This means formally that it has an is-of-type relation to a data type and is usually aggregated by another element, e.g. the internal behavior or a port interface. ⌋

In the meta-model, various kinds of data prototypes are derived from the abstract `DataPrototype` as shown in figure 5.16. The reason for the introduction of this hierarchy was the distinction between `AutosarDataPrototype` (which can be used for the application and implementation types as well) and `ApplicationCompositeElementDataPrototype` (which is restricted to be used within the application types).

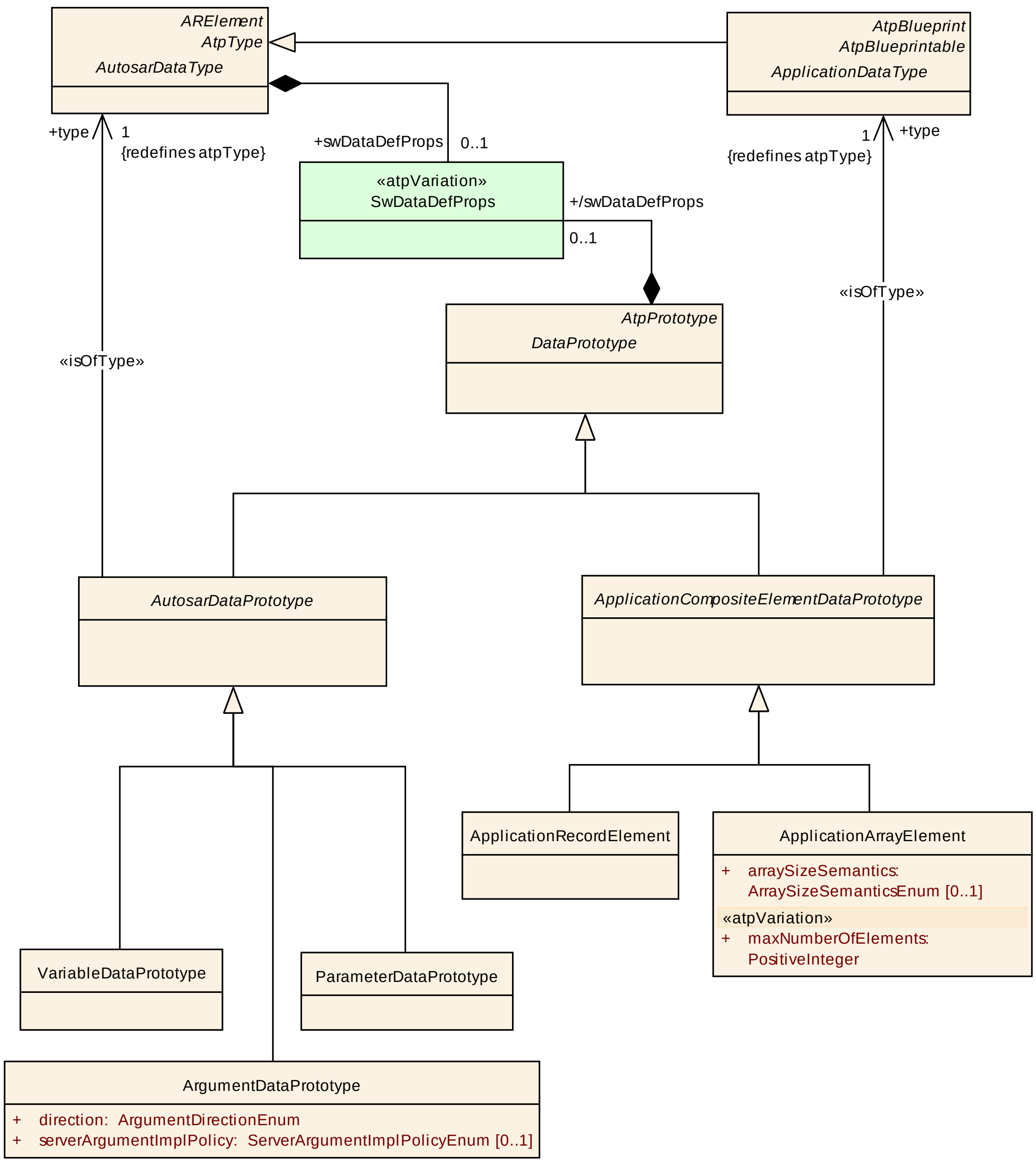

Figure 5.16: Data Prototypes Overview

Class	**DataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	ARObject,AtpFeature,AtpPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swDataDefProps	SwDataDefProps	0..1	aggr	This property allows to specify data defintion properties which apply on data prototype level.

Table 5.30: DataPrototype

Class	**AutosarDataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	ARObject,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
type	AutosarDataType	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table 5.31: AutosarDataPrototype

Class	**ApplicationCompositeElementDataPrototype (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	This class represents a data prototype which is aggregated within a composite application data type (record or array). It is introduced to provide a better distinction between target and context in instanceRefs.			
Base	ARObject,AtpFeature,AtpPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
type	ApplicationDataType	1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table 5.32: ApplicationCompositeElementDataPrototype

Because these `DataPrototype`s are modeled as own meta-classes it is possible to define own attributes for them (on M2) which (in the M1 model) could extend or constrain the attribute values already set via the corresponding data type.

[TPS_SWCT_1265] `DataPrototype` aggregates an own set of `SwDataDefProps`
⌈This mechanism is used here in the way that `DataPrototype` aggregates an own set of `SwDataDefProps`. Thus each kind of `DataPrototype` has the ability to extend or even overwrite the `SwDataDefProps` already defined by its `ApplicationDataType` or `ImplementationDataType`.

This mechanism, if carefully applied, allows for a better reuse of data types because they can be kept free of the properties which vary according to the context or are defined in later project phases. Chapter 5.4 describes more details on this. ⌋

The applicability of `SwDataDefProps` for `DataPrototype`s shall follow the same rules as for the `categories` of the corresponding `Datatype`s (see table 5.8). Further information can be found in table 5.33 and table 5.34.

Please note that table 5.33 does not include the `ApplicationRecordElement` and `ApplicationArrayElement` because these specializations of `ApplicationCompositeElementDataPrototype` are already part of table 5.8. The same applies for table 5.34 which does not include the `ImplementationDataTypeElement`.

	Root Element			Attribute Existence per Category									
Attributes of SwDataDefProps	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP
additionalNativeTypeQualifier													
annotation	x	x	x	*	*	*	*	*	*	*	*	*	*
baseType													
compuMethod													
dataConstr	x	x		0..1	0..1				0..1			0..1	0..1
displayFormat	x	x		0..1	0..1			0..1	0..1			0..1	0..1
implementationDataType													
invalidValue													
mcFunction	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAddrMethod	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swBitRepresentation													
swCalibrationAccess	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swCalprmAxisSet													
swCalprmAxisSet. swCalprmAxis /SwAxisGrouped. swCalprmRef		x	x									0..1	0..1
swCalprmAxisSet. swCalprmAxis /SwAxisIndividual. swVariableRef		x	x							0..1	0..1	0..1	0..1
swCalprmAxisSet. swCalprmAxis /SwAxisGrouped. sharedAxisType													
swCalprmAxisSet. swCalprmAxis /SwAxisIndividual. inputVariableType													
swCalprmAxisSet/ AxisIndividual/ Unit													
swCalprmAxisSet/ BaseType													
swComparisonVariable			x									0..*	0..*
swDataDependency	x	x		0..1								0..1	0..1
swHostVariable													

	Root Element			Attribute Existence per Category									
Attributes of SwDataDefProps	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	VAL_BLK	STRUCTURE	ARRAY	STRING	BOOLEAN	COM_AXIS	RES_AXIS	CURVE	MAP
swImplPolicy	x			0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1	0..1
swIntendedResolution													
swInterpolationMethod	x	x	x	0..1						0..1	0..1	0..1	0..1
swIsVirtual	x	x		0..1					0..1			0..1	0..1
swPointerTargetProps													
swRecordLayout													
swRefreshTiming	x	x		0..1	0..1			0..1	0..1				
swTextProps													
swValueBlockSize													
unit													
valueAxisDataType													

Table 5.33: Allowed Attributes vs. `Category` for `DataPrototypes` typed by Application Data Types

	Root Element			Attribute Existence per Category						
Attributes of SwDataDefProps	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
additionalNativeTypeQualifier										
annotation	x	x	x	*	*	*	*	*	*	*
baseType										
compuMethod				0..1			X			
dataConstr	x	x		0..1			X			
displayFormat	x	x		0..1			X	0..1	0..1	0..1
implementationDataType										
invalidValue				0..1						
mcFunction	x	x		0..1			X	0..1	0..1	0..1
swAddrMethod	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swAlignment	x	x		0..1	0..1	0..1	0..1	0..1	0..1	0..1
swBitRepresentation										
swCalibrationAccess	x	x		0..1			X	0..1	0..1	0..1
swCalprmAxisSet										
swComparisonVariable										
swDataDependency										
swHostVariable										
swImplPolicy	x			0..1	0..1	0..1	0..1	0..1	0..1	0..1

	Root Element			Attribute Existence per Category						
Attributes of SwDataDefProps	DataPrototype	InstantiationDataDefProps	ParameterAccess	VALUE	DATA_REFERENCE	FUNCTION_REFERENCE	TYPE_REFERENCE	STRUCTURE	UNION	ARRAY
swIntendedResolution										
swInterpolationMethod										
swIsVirtual										
swPointerTargetProps										
swPointerTargetProps .swDataDefProps										
swPointerTargetProps .functionPointerSignature										
swRecordLayout										
swRefreshTiming	x	x		0..1			X	0..1	0..1	0..1
swTextProps										
swValueBlockSize										
unit										
valueAxisDataType										

Table 5.34: Allowed Attributes vs. `Category` for `DataPrototypes` typed by `ImplementationDataTypes`

[TPS_SWCT_1266] Three non-abstract classes derived from `AutosarDataPrototype` ⌈ There are three non-abstract classes derived from `AutosarDataPrototype` which reflect the main use cases in the SWC-Template:

- Operation arguments (`ArgumentDataPrototype`) in a client-server interface.
- Variables (`VariableDataPrototype`) which are changed by the application software at runtime.
- Parameters (`ParameterDataPrototype`) which are constant (except for calibration access) from the application point of view.

⌋

Class	**ArgumentDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An argument of an operation, much like a data element, but also carries direction information and is associated with a particular operation.			
Base	ARObject,AtpFeature,AtpPrototype,AutosarDataPrototype,DataPrototype,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
direction	ArgumentDirectionEnum	1	attr	This attribute specifies the direction of the argument prototype.

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantic as if the attribute is set to useArgumentType

Table 5.35: ArgumentDataPrototype

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

Table 5.36: VariableDataPrototype

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

Table 5.37: ParameterDataPrototype

[TPS_SWCT_1267] `DataPrototype` can be aggregated in different roles ⌈ Note that even though the meta-classes `VariableDataPrototype` and `ParameterDataPrototype` already express specific use cases of the underlying data type the same `DataPrototype` can still be aggregated in different roles, e.g. in the `SwcInternalBehavior` to express different methods how to access it. ⌋

An example is the aggregation of `VariableDataPrototype` by `SwcInternalBehavior` in the roles of either `implicitInterRunnableVariable` or `explicit-`

InterRunnableVariable. Find more information concerning these use cases in chapter 7.

[TPS_SWCT_1268] Definition of initValue for a VariableDataPrototype or a ParameterDataPrototype ⌈ It is possible to assign an initValue for both a VariableDataPrototype and a ParameterDataPrototype. This aspect is sketched in 5.17. ⌋

[TPS_SWCT_1269] In PortInterfaces, initial values defined for DataPrototypes are ignored ⌈ These initValues have no meaning for DataPrototypes within PortInterfaces because in this case a more specific definition of initial values via the so-called ComSpec is required, see chapter 4.5. ⌋

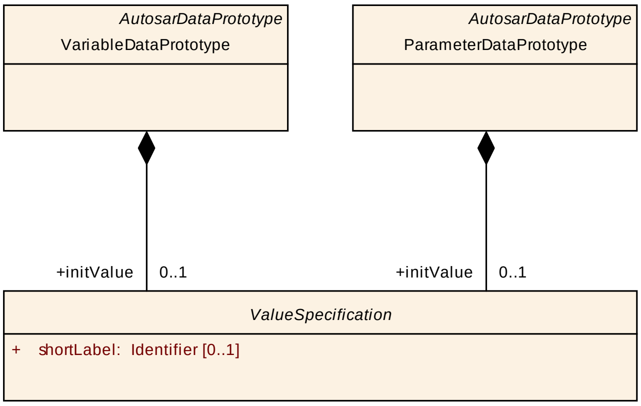

Figure 5.17: Initial value for AutosarDataPrototypes

Find more information about the interpretation of initValue in section 5.7.

5.3.2 Reference to Data Prototypes

This chapter explains the various patterns for referencing DataPrototypes. As references to a DataPrototype may or may not imply the necessity for using an instanceRef this would mean that in some places the meta-model would have to implement both variants depending on the use case.

To avoid this, AUTOSAR defines a unified reference implementation for VariableDataPrototypes and ParameterDataPrototypes.

[TPS_SWCT_1270] AutosarVariableRef ⌈ With the advent of AutosarVariableRef it is possible to implement a uniform reference to a VariableDataPrototype that covers all foreseen use cases:

- Reference to a localVariable, no AtpInstanceRef required.
- Reference to an autosarVariable (which involves an AtpInstanceRef).
- Reference to the internal structure of a VariableDataPrototype implemented using a composite ImplementationDataType.

⌋

Class	AutosarVariableRef			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Data Elements			
Note	This class represents a reference to a variable within AUTOSAR which can be one of the following use cases: localVariable: • localVariable which is used as whole (e.g. InterRunnableVariable, inputValue for curve) autosarVariable: • a variable provided via Port which is used as whole (e.g. dataAccesspoints) • an element inside of a composite local variable typed by ApplicationDatatype (e.g. inputValue for a curve) • an element inside of a composite variable provided via Port and typed by ApplicationDatatype (e.g. inputValue for a curve) autosarVariableInImplDatatype: • an element inside of a composite local variable typed by ImplementationDatatype (e.g. nvramData mapping) • an element inside of a composite variable provided via Port and typed by ImplementationDatatype (e.g. inputValue for a curve)			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
autosarVariable	DataPrototype	0..1	iref	This references a variable which is provided by a port and/or which is part of a CompositeDataType.
autosarVariableInImplDatatype	ArVariableInImplementationDataInstanceRef	0..1	aggr	This is used if the target variable is inside of variableDataPrototype typed by an ImplementationDataType.
localVariable	VariableDataPrototype	0..1	ref	This reference is used if the variable is local to the current component. It would also be possible to use the instance refence here. Such an instance ref would not have a contextElement, since the current instance is the context. But the local instance is a special case which may provide further optimization. Therefore an expcplicit reference is provided for this case.

Table 5.38: AutosarVariableRef

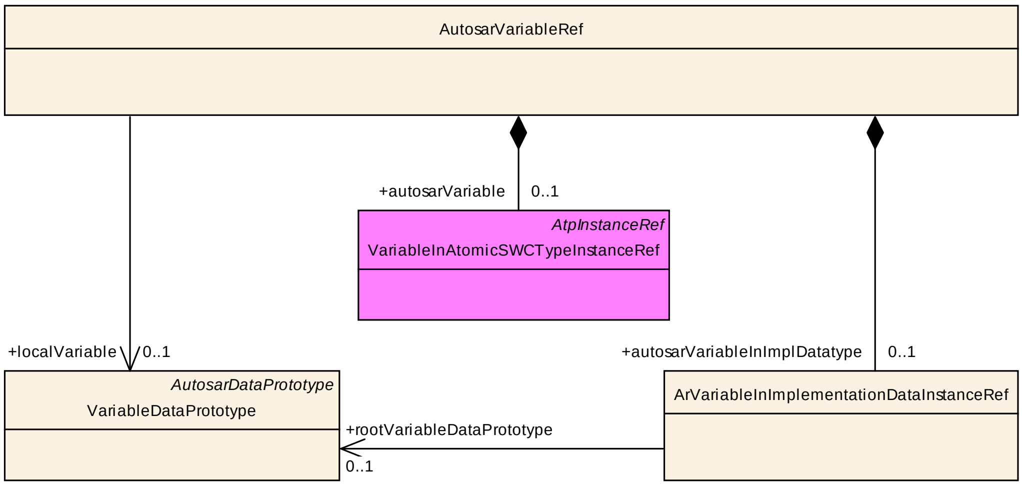

Figure 5.18: Implementation of `AutosarVariableRef`

Class	**ArVariableInImplementationDataInstanceRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents the ability to navigate into a data element inside of an VariableDataPrototype which is typed by an ImplementationDatatype. Note that it shall not be used if the target is the VariableDataPrototype itself (e.g. if its a primitive). 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***
contextDataPrototype (ordered)	ImplementationDataTypeElement	*	ref	This is a context in case there are subelements with explicit types. The reference has to be ordered to properly reflect the nested structure. **Tags:** xml.sequenceOffset=30
portPrototype	PortPrototype	0..1	ref	This is the port providing/receiving the root of the variable **Tags:** xml.sequenceOffset=10
rootVariableDataPrototype	VariableDataPrototype	0..1	ref	This refers to the variableDataPrototype which is typed by the implementationDatatype in which which the target can be found. **Tags:** xml.sequenceOffset=20
targetDataPrototype	ImplementationDataTypeElement	1	ref	This is a context in case there are subelements with explicit types. **Tags:** xml.sequenceOffset=40

Table 5.39: ArVariableInImplementationDataInstanceRef

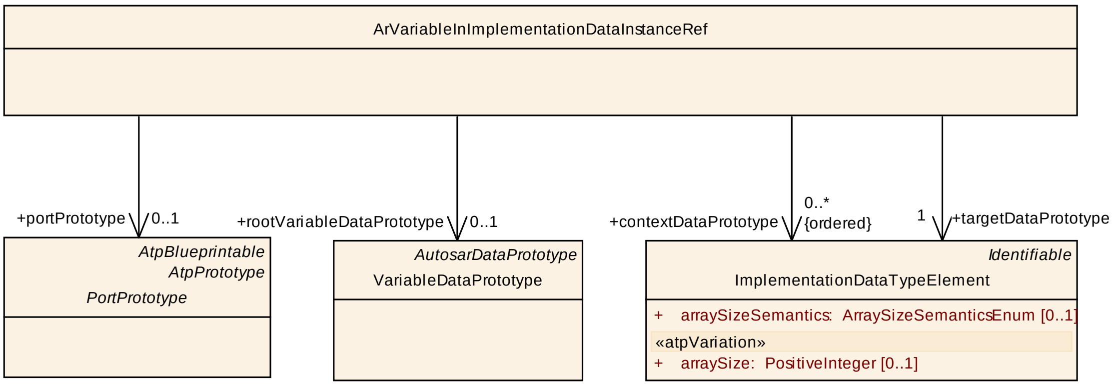

Figure 5.19: Implementation of `ArVariableInImplementationDataInstanceRef`

[constr_2536] Target of an `autosarVariable` in `AutosarVariableRef` shall refer to a variable ⌈ The target of `autosarVariable` (which in fact is an instance ref) in `AutosarVariableRef` shall either be or be nested in `VariableDataPrototype`. This means that the target shall either be a `VariableDataPrototype` or an `ApplicationCompositeElementDataPrototype` that in turn is owned by a `VariableDataPrototype`. ⌋

[TPS_SWCT_1271] `AutosarParameterRef` ⌈ With the advent of `AutosarParameterRef` it is possible to implement a uniform reference to a `ParameterDataPrototype` that covers all foreseen use cases:

- Reference to a `localParameter`, no `AtpInstanceRef` required.
- Reference to an `autosarParameter` (which involves an `AtpInstanceRef`).

⌋

Please note that there is a very limited amount of use-cases available where the `AutosarParameterRef` can (with the active consent of the AUTOSAR standard) reference a `VariableDataPrototype`.

[constr_1173] Applicability of `AutosarParameterRef` referencing a `VariableDataPrototype` ⌈ A reference from `AutosarParameterRef` to `VariableDataPrototype` is **only** applicable if the `AutosarParameterRef` is used in the context of `SwAxisGrouped`. ⌋

For example, the use case referenced in [constr_1173] applies if it is required to store a grouped axis in a variable in order to adapt the axis during run-time of the ECU by a dedicated algorithm. Note that in all cases where [constr_1173] does not apply [constr_2535] shall be fulfilled.

Class	**AutosarParameterRef**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	This class represents a reference to a parameter within AUTOSAR which can be one of the following use cases: localParameter: • localParameter which is used as whole (e.g. sharedAxis for curve) autosarVariable: • a parameter provided via PortPrototype which is used as whole (e.g. parameterAccess) • an element inside of a composite local parameter typed by ApplicationDatatype (e.g. sharedAxis for a curve) • an element inside of a composite parameter provided via Port and typed by ApplicationDatatype (e.g. sharedAxis for a curve) autosarParameterInImplDatatype: • an element inside of a composite local parameter typed by ImplementationDatatype • an element inside of a composite parameter provided via PortPrototype and typed by ImplementationDatatype			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
autosarParameter	DataPrototype	0..1	iref	This instance reference is used if the callibration parameter is either imported via a port or is part of a composite data structure.
localParameter	DataPrototype	0..1	ref	In the majority of cases this reference goes to ParameterDataPrototyoes rather than VariableDataPrototypes. Pointing the reference to a VariableDataPrototype is limited to special use cases, e.g. if the AutosarParameterRef is used in the context of an SwAxisGrouped. This reference is used if the arParameter is local to the current component. Of course, it would technically also be feasible to use an InstanceRef for this case. However, the InstanceRef would not have a contextElement (because the cureent instance is the context). Hence, the local instance is a special case which may provide further optimization. Therefore an expclicit reference is provided for this case.

Table 5.40: AutosarParameterRef

[constr_2535] Target of an `autosarParameter` in `AutosarParameterRef` shall refer to a parameter ⌈ Except for the specifically described cases where [constr_1173] applies the target of `autosarParameter` (which in fact is an instance ref) in `AutosarParameterRef` shall either be or be nested in `ParameterDataPrototype`. This means that the target shall either be a `ParameterDataPrototype` or an `ApplicationCompositeElementDataPrototype` that in turn is owned by a `ParameterDataPrototype`. ⌋

[constr_1161] Applicability of the `index` attribute of `Ref` ⌈ The `index` attribute of `Ref` is limited to a given set if use cases as there are:

- `McDataInstance.instanceInMemory`
- `AutosarVariableRef`
- `AutosarParameterRef`
- `FlatInstanceDescriptor` / `AnyInstanceRef`

⌋

The implementation of the `AtpInstanceRef`s for `AutosarVariableRef` and `AutosarParameterRef` probably needs some clarification regarding the references to `DataPrototype`s.

[TPS_SWCT_1374] Implementation of `AutosarParameterRef` ⌈ The reference to `rootParameterDataPrototype` is **not** redundant. It is required for identifying the `arParameter` itself in a `ParameterInterface` **if and only if** the `AutosarDataType` of the `arParameter` is a composite data type. If the `AutosarDataType` was a primitive data type the `target` reference is the `only` reference required. ⌋

As explained before, the implementation of `AutosarParameterRef` in a specific case is subject to [constr_1173].

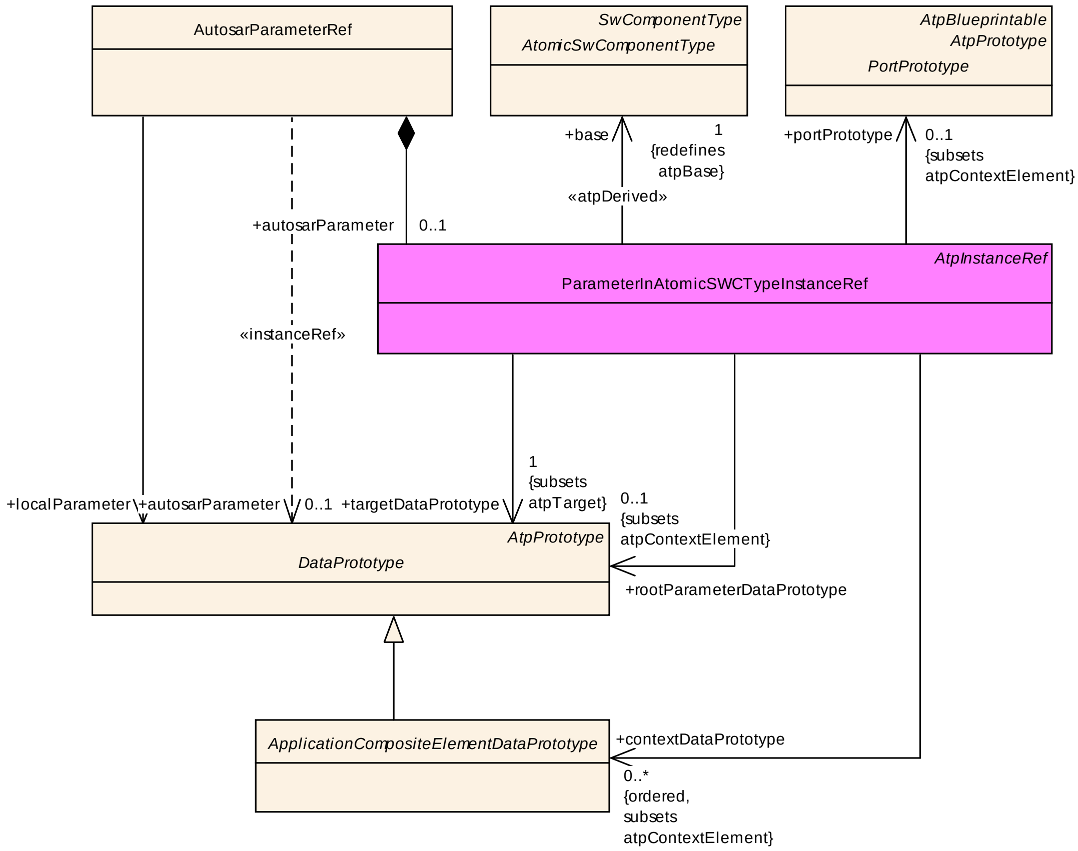

Figure 5.20: Implementation of the InstanceRef for `AutosarParameterRef`

[TPS_SWCT_1375] Implementation of `AutosarVariableRef` ⌈ The reference to `rootVariableDataPrototype` is **not** redundant. It is required for identifying the `arVariable` itself in a `SenderReceiverInterface` or `NvDataInterface` **if and only if** the `AutosarDataType` of the `arVariable` is a composite data type. If the `AutosarDataType` was a primitive data type the `target` reference is the `only` reference required. ⌋

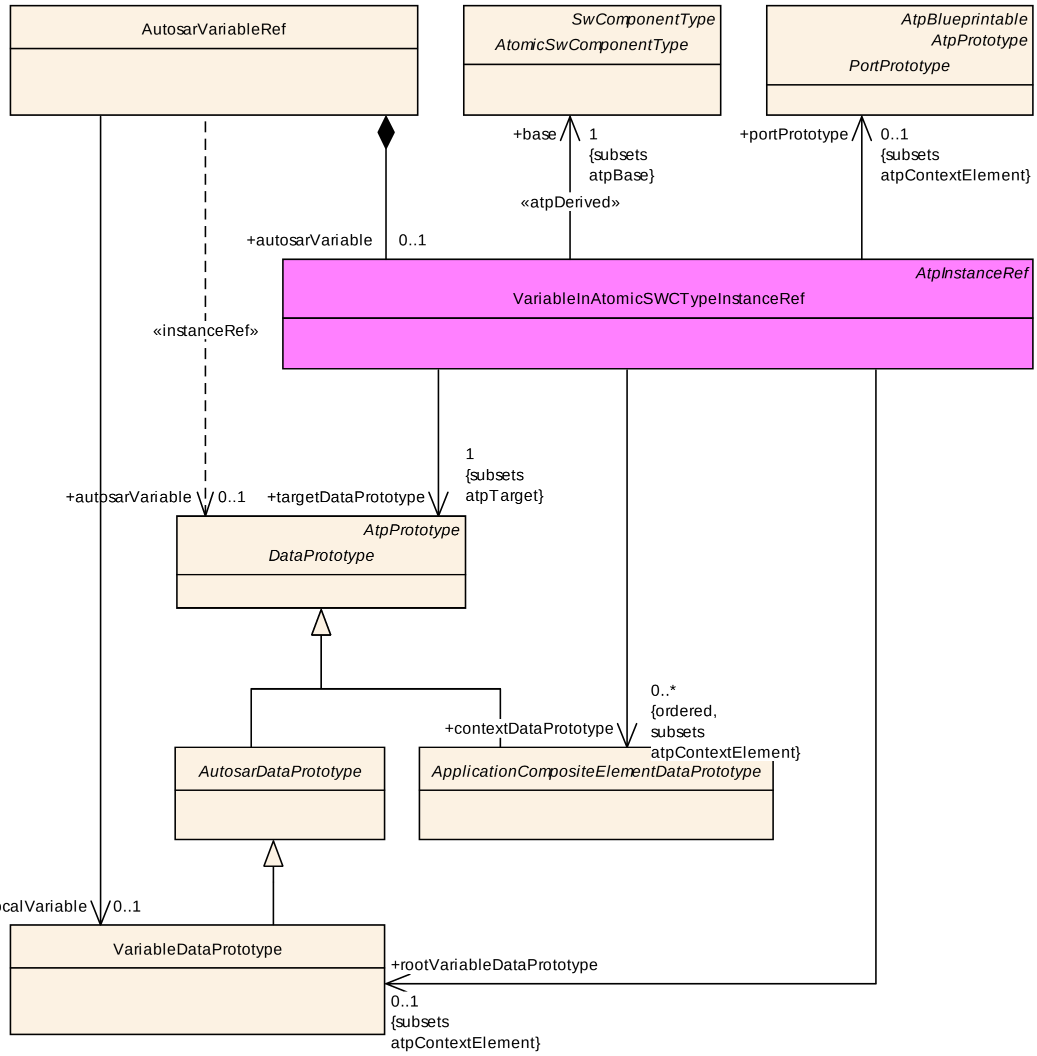

Figure 5.21: Implementation of the InstanceRef for `AutosarVariableRef`

5.4 Properties of Data Definitions

5.4.1 Overview

As it has already been shown in the previous chapters, various properties and associations can be attached to the definition of data types as well as prototypes. These are described by the meta-class `SwDataDefProps` which covers all properties of a particular data object under various aspects.

In general, the properties specified within `SwDataDefProps` may apply to all kind of data declared within the software-component template and within the basic software module description template as well, e.g. component local data, data used for communication, data used for measurement as well as for calibration. However, there are constraints for the attributes depending on the role of the data:

[constr_1015] Prioritization of `SwDatDefProps` ⌈ Prioritization and usage of `SwDataDefProps` shall follow the restrictions given in table 5.41. ⌋

	Usage For			Place of Setting										
Attributes of SwDataDefProps	**RTE**	**A2L**	**Other Usage**	**ApplicationDataType**	**ImplementationDataType**	**DataPrototype**	**InstantiationDataDefProps**	**ParameterAccess**	**ComSpec**	**SwServiceArg**	**FlatInstanceDescriptor**	**McDataInstance**	**SwSystemconst**	**PerInstanceMemory**
additionalNativeTypeQualifier	x		x	NA	D	I	NA	NA	NA	D	NA	NA	NA	NA
annotation			x	D	A	A	A	A	A	D	NA	A	D	NA
baseType	x	x	x	NA	D	I	I	I	R	D	NA	A	M	NA
compuMethod	x	x	x	D	A	I	I	NA	R	I	NA	I	D	NA
dataConstr	x	x	x	D	R	R	R	I	NA	R	NA	I	D	NA
displayFormat		x		D	A	R	R	I	NA	R	NA	I	D	NA
implementationDataType	x		x	NA	D	I	I	I	NA	D	NA	NA	NA	NA
invalidValue	x	x		D	A	I	I	NA	NA	NA	NA	I	NA	NA
mcFunction		x		NA	NA	D	R	NA	NA	NA	NA	R	NA	NA
swAddrMethod	x	x	x	D	R	R	R	NA	NA	NA	R	NA	NA	D
swAlignment	x		x	NA	D	R	R	NA	NA	NA	NA	NA	NA	NA
swBitRepresentation		x	x	NA	NA	NA	NA	NA	NA	NA	NA	D	NA	NA
swCalibrationAccess	x	x		D	R	R	R	NA	NA	R	R	I	D	NA
swCalprmAxisSet	x	x		D	NA	I	I	I	NA	NA	NA	I	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisGrouped.swCalprmRef		x		NA	NA	NA	D	R	NA	NA	NA	I	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisIndividual.swVariableRef		x		NA	NA	NA	D	R	NA	NA	NA	I	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisGrouped.sharedAxisType		x		D	NA	NA	NA	NA	NA	NA	NA	I	NA	NA
swCalprmAxisSet.swCalprmAxis /SwAxisIndividual.inputVariableType		x		D	NA	NA	NA	NA	NA	NA	NA	I	NA	NA
swCalprmAxisSet/SwAxisIndividual/Unit		optional		D	NA	I	I	I	NA	I	NA	I	NA	NA
swCalprmAxisSet/BaseType		optional		D	NA	I	I	I	NA	NA	NA	I	NA	NA
swComparisonVariable		x		NA	NA	NA	NA	D	NA	NA	NA	I	NA	NA
swDataDependency		x	x	NA	NA	D	R	NA	NA	NA	NA	I	NA	NA
swHostVariable		x	x	NA	NA	NA	NA	NA	NA	NA	NA	D	NA	NA
swImplPolicy	x		x	D	A	A	NA	NA	NA	D	NA	NA	NA	NA
swIntendedResolution			x	D[6]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
swInterpolationMethod			x	D	I	R	R	R	NA	NA	NA	I	NA	NA
swIsVirtual		x		NA	NA	D	R	NA	NA	NA	NA	I	NA	NA
swPointerTargetProps			x	NA	D	I	NA	NA	NA	D	NA	NA	NA	NA
swRecordLayout	x	x	x	D	NA	I	I	I	NA	NA	NA	I	NA	NA
swRefreshTiming		x		D	R	R	R	NA	NA	R	NA	R	NA	NA
swTextProps		x	x	D	I	I	I	I	NA	NA	NA	I	NA	NA
swValueBlockSize		x	x	D	I	I	I	I	NA	NA	NA	I	NA	NA
unit		x	x	D	I	I	I	NA	NA	I	NA	I	NA	NA
valueAxisDataType		x	x	D	I	I	I	I	NA	NA	NA	I	NA	NA

[6] `swIntendedResolution` is used only in an early phase of data type definition, especially in the context of so-called blueprints, see [1]. It can be seen as a requirement for the definition of an appropriate `CompuMethod`.

	Usage For			Place of Setting										
Attributes of SwDataDefProps	RTE	A2L	Other Usage	ApplicationDataType	ImplementationDataType	DataPrototype	InstantiationDataDefProps	ParameterAccess	ComSpec	SwServiceArg	FlatInstanceDescriptor	McDataInstance	SwSystemconst	PerInstanceMemory

Table 5.41: Usage of Attributes of `SwDataDefProps`

The following settings apply in table 5.41:

D **Define** the attribute independent from settings to the left.

R Use or **re-define** definition from the left in the scope of this element.

A **Add** attribute if not defined on the left, or as an additional information.

If the attribute has an upper multiplicity > 1 and the attribute is defined on the left then the attribute is added to the attribute defined on the left.

If the attribute has a upper multiplicity of 1 and the attribute is not defined on the left then the attribute is defined.

If the attribute has an upper multiplicity of 1 and the attribute is already defined on the left then the attribute is not redefined but this is considered as invalid configuration.

I **Inherit** the definition from the left for usage in the scope of this element.

NA Attribute is **not applicable** for usage in the scope of this element.

M Attribute is **meaningless** in the scope of this element. As it was allowed in previous versions, it declaring it as Not Applicable (N/A) would break compatibility. Tools shall ignore such an attribute without a warning.

[constr_2551] **`SwCalprmAxis.baseType` shall be ignored** ⌈ `SwCalprmAxis.baseType` is possible for schema compatibility reasons. The value shall be ignored. Tools may raise a warning in this case. ⌋

Some of the property names contain the term "variable" or "calprm", this comes from historical[7] reasons and can be taken as some hint where the property most likely applies to.

[7] In the beginning of ASAM and MSR measurements and calibration parameters (characteristics) were separated and the properties were merged over the time.

<table>
<tr><td>Class</td><td colspan="4">≪atpVariation≫ SwDataDefProps</td></tr>
<tr><td>Package</td><td colspan="4">M2::AUTOSARTemplates::CommonStructure::DataDefProperties</td></tr>
<tr><td>Note</td><td colspan="4">This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is agrregated.

Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations.

SwDataDefProps covers various aspects:
Structure of the data element for calibration use cases: Is it a single value, a curve, or a map, but also the recordLayouts which specify, how such elements are mapped/converted to the DataTypes in the programming language (or in Autosar). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet
Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier
Access policy for the MCD system, mainly expressed by swCalibrationAccess
Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue
Code generation policy provided by swRecordLayout
Tags: Vh.latestBindingTime=CodeGenerationTime</td></tr>
<tr><td>Base</td><td colspan="4">ARObject</td></tr>
<tr><td>Attribute</td><td>Datatype</td><td>Mul.</td><td>Kind</td><td>Note</td></tr>
<tr><td>additionalNativeTypeQualifier</td><td>NativeDeclarationString</td><td>0..1</td><td>attr</td><td>This attribute is used to declare native qualifiers of the prgramming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile" or "strict" of the C-language. All such declarations have to be put into one string.

Tags: xml.sequenceOffset=235</td></tr>
<tr><td>annotation</td><td>Annotation</td><td>*</td><td>aggr</td><td>This aggregation allows to add annotations (yellow pads ...) related to the current data object.

Tags: xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false</td></tr>
<tr><td>baseType</td><td>SwBaseType</td><td>0..1</td><td>ref</td><td>Base type associated with this data object.

Tags: xml.sequenceOffset=50</td></tr>
<tr><td>compuMethod</td><td>CompuMethod</td><td>0..1</td><td>ref</td><td>Computation method associated with the semantics of this data object.

Tags: xml.sequenceOffset=180</td></tr>
</table>

Attribute	Datatype	Mul.	Kind	Note
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
implementationDataType	ImplementationDataType	0..1	ref	This association denotes the implementation type of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTargetProps), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
mcFunction	Identifier	0..1	ref	Specifies the name of a "Function" (in the sense of the MC system) to which this data object belongs. This corresponds to the Function in ASAM MCD 2MC /ASAP2 which defines the characteristic resp. which provides the measurement as output. The function name is only used for support of MC systems. It can be predefined on the level of software component design. If it is not predefined, it could be filled out with a reasonable name, e.g. the component prototype name, from the ECU extract. **Tags:** xml.sequenceOffset=257

Attribute	Datatype	Mul.	Kind	Note
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrmethod, it can be specified that several data prototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced SwAddrMethod. **Tags:** xml.sequenceOffset=33
swBitRepresentation	SwBitRepresentation	0..1	aggr	Description of the binary representaion in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxisSet	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90
swComparisonVariable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170; xml.typeElement=false
swDataDependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220; xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230

Attribute	Datatype	Mul.	Kind	Note
swIntendedResolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. If however, a conversion formula is present, this can be checked for plausibility against swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolationMethod	Identifier	0..1	ref	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they must have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTargetProps	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecordLayout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290
swRefreshTiming	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120

Attribute	Datatype	Mul.	Kind	Note
swValueBlockSize	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=80
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units must be the same. **Tags:** xml.sequenceOffset=350
valueAxisDataType	ApplicationPrimitiveDataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table 5.42: SwDataDefProps

[TPS_SWCT_1272] Semantics of `swComparisonVariable` ⌈ Please note that `swComparisonVariable`s shall be displayed in the MCD system on the ordinate in a curve. By showing the input value and the comparison value the calibration engineer can see if the current working point is above or below a curve provident thresholds. For example in a curve specifying a temperature depending gear shift threshold engine speed the engine speed can be shown as "comparisonVariable".

These variables can be used to display the value of a variable on the value axis of a calibration parameter (characteristic), that is currently displayed in the MCD-System. The purpose is to compare the appropriate result from the calibration parameter in question, with a value being calculated or taken from a sensor (the comparison variable).

The sole purpose of this comparison-variable is therefore to serve the calibration process. ⌋

The meaning behind `swComparisonVariable` is depicted in Figure 5.22. Legend: t_x represents the current temperature and t_{mot} represents the motor temperature. V represents the current speed as shown in the MCD system for comparison: this is the `SwComparisonVariable`. Likewise, V_s represents the speed characteristic over the temperature.

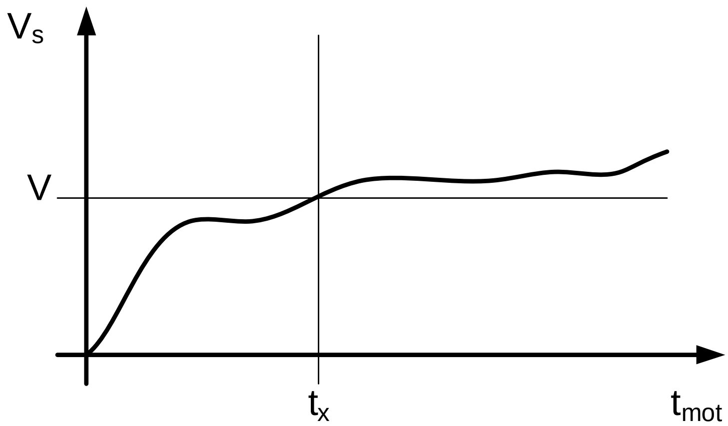

Figure 5.22: Explanation of swComparisonVariable

Enumeration	**SwCalibrationAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Literal	*Description*
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file.
readOnly	The element will only appear as read-only in an ASAP file.
readWrite	The element will appear in the ASAP file with both read and write access.

Table 5.43: SwCalibrationAccessEnum

[TPS_SWCT_1273] Precedence rules for the application of SwDataDefProps ⌈ SwDataDefProps can be specified on various levels, from type over prototype to instantiation, finally data access and calibration support after RTE generation. In general, properties specified on prototype level override the ones specified on type level.

More formally, the precedence of such properties is:

1. attributes of SwDataDefProps defined on ApplicationDataType which may be overwritten by
2. attributes of SwDataDefProps defined on ImplementationDataType which may be overwritten by
3. attributes of SwDataDefProps defined on DataPrototype which may be overwritten by
4. attributes of SwDataDefProps defined on InstantiationDataDefProps which may be overwritten by
5. attributes of SwDataDefProps defined on ParameterAccess respectively Argument which may be overwritten by
6. attributes of SwDataDefProps defined on FlatInstanceDescriptor which may be overwritten by

7. attributes of SwDataDefProps defined on McDataInstance

⌋

Note that details about applicable attributes of SwDataDefProps can be found in Table 5.41.

[TPS_SWCT_1274] SwDataDefProps used to support calibration and measurement ⌈ The last item in this list denotes that SwDataDefProps are also used as part of McSupportData which is a direct input to the generation of measurement and calibration configuration formats (so-called A2L-files). This use case is further explained in [7]. Since these data are generated by the RTE, they will use a copy of the properties according to the precedence given above.

However, even in this use case which comes after RTE generation it is possible that properties relevant for the MCD system are added which had been undefined so far. This for example applies to the attribute mcFunction that is used in the MCD system for structuring of the data but otherwise is not directly relevant for the component model in AUTOSAR.

Also, the attribute swRefreshTiming which denotes a timing information relevant for the measurement system may be set rather late in the process chain. ⌋

Obviously such an override is not applicable in all cases. In particular, the properties covering the structure shall not be redefined on DataPrototype. Implementation policy, semantics and code generation policy may be changed under consideration of compatibility rules.

Access policy for the MCD system is the most likely subject to be redefined on the DataPrototype of even on an instantiation level.

Section 5.4.3 describes how SwDataDefProps are used for measuring purposes while Section 5.4.4 describes the construction of characteristics based on the combination of SwDataDefProps with DataPrototypes.

Section 2.2.2 describes in which context calibration parameters can be defined. Finally, sections 2.2.3, 7.5.4, and 5.5.4 show how calibration parameters are used in RunnableEntitys and show the link to an actual ECU implementation.

Enumeration	**SwImplPolicyEnum**
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Specifies the implementation strategy with respect to consistency mechanisms of variables.
Literal	***Description***
const	forced implementation such that the running software within the ECU must not modify it. For example implemented with the "const" modifier in C. This can be applied for parameters (not for those in NvRam) as well as argument data prototypes.
fixed	This data element is fixed. In particular this indicates, that it might also be implemented e.g. as in place data, (#DEFINE).

measurement Point	The data element is created for measurement purposes only. The data element is never read directly within the ECU software. In contrast to a "standard" data element in an unconnected provide port is, this unconnection is guaranteed for measurementPoint data elements.
queued	The content of the data element is queued and the data element has 'event' semantics, i.e. data elements are stored in a queue and all data elements are processed in 'first in first out' order. The queuing is intended to be implemented by RTE Generator. This value is not applicable for parameters.
standard	This is applicable for all kinds of data elements. For variable data prototypes the 'last is best' semantics applies. For parameter there is no specific implementation directive.

Table 5.44: SwImplPolicyEnum

[TPS_SWCT_1275] values of the attribute `swImplPolicy` are restricted depending on the context ⌈ The values of the attribute `swImplPolicy` are restricted depending on the context. This restriction reflects the fact that not all possible implementation strategies are useful or supported for all kinds of `DataPrototype`s. ⌋

These restrictions are summarized in table 5.45 and formalized in the following constraints. Please note that the usage of `swImplPolicy` is further constraint in the combination with the attribute value `swCalibrationAccess` as described in [constr_1017].

	VariableDataPrototype							ParameterDataPrototype					Misc.	
Attribute value of SwImplPolicyEnum	**VariableDataPrototype in SenderReceiverInterface**	**VariableDataPrototype in NvDataInterface**	**VariableDataPrototype in role ramBlock**	**VariableDataPrototype in role implicitInterRunnableVariable**	**VariableDataPrototype in role explicitInterRunnableVariable**	**VariableDataPrototype in role arTypedPerInstanceMemory**	**VariableDataPrototype in role staticMemory**	**ParameterDataPrototype in ParameterInterface**	**ParameterDataPrototype in role romBlock**	**ParameterDataPrototype in role sharedParameter**	**ParameterDataPrototype in role perInstanceParameter**	**ParameterDataPrototype in role constantMemory**	**ArgumentDataPrototype**	**SwServiceArg**
const	NA	NA	NA	NA	NA	NA	NA	x	NA	NA	NA	x	NA	x
fixed	NA	NA	NA	NA	NA	NA	NA	x	NA	NA	NA	x	NA	NA
measurementPoint	x	NA	NA	NA	NA	x	x	NA	NA	NA	NA	NA	NA	NA
queued	x	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
standard	x	x	x	x	x	x	x	x	x	x	x	x	x	x
message	NA	NA	NA	NA	NA	NA	x	NA	NA	NA	NA	NA	NA	NA

	VariableDataPrototype							ParameterDataPrototype					Misc.	
Attribute value of SwImplPolicyEnum	VariableDataPrototype in SenderReceiverInterface	VariableDataPrototype in NvDataInterface	VariableDataPrototype in role ramBlock	VariableDataPrototype in role implicitInterRunnableVariable	VariableDataPrototype in role explicitInterRunnableVariable	VariableDataPrototype in role arTypedPerInstanceMemory	VariableDataPrototype in role staticMemory	ParameterDataPrototype in ParameterInterface	ParameterDataPrototype in role romBlock	ParameterDataPrototype in role sharedParameter	ParameterDataPrototype in role perInstanceParameter	ParameterDataPrototype in role constantMemory	ArgumentDataPrototype	SwServiceArg

Table 5.45: Allowed attributes values for `SwImplPolicy` vs. `DataPrototype`s and their roles

The following settings apply in table 5.45:

x Attribute is applicable for usage in the scope of this element.

NA Attribute is **not** applicable for usage in the scope of this element.

[constr_2035] `swImplPolicy` for `VariableDataPrototype` in `SenderReceiverInterface` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in `SenderReceiverInterface` shall be `standard`, `queued` or `measurementPoint`. ⌋

[constr_2036] `swImplPolicy` for `VariableDataPrototype` in `NvDataInterface` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in `NvDataInterface` shall be `standard`. ⌋

[constr_2037] `swImplPolicy` for `VariableDataPrototype` in the role `ramBlock` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in the role `ramBlock` shall be `standard`. ⌋

[constr_2038] `swImplPolicy` for `VariableDataPrototype` in the role `implicitInterRunnableVariable` ⌈ The overriding `swImplPolicy` attribute value of a `VariableDataPrototype` in the role `implicitInterRunnableVariable` shall be `standard`. ⌋

[constr_2039] `swImplPolicy` for `VariableDataPrototype` in the role `explicitInterRunnableVariable` ⌈ The overriding `swImplPolicy` attribute value of a

VariableDataPrototype in the role explicitInterRunnableVariable shall be standard. ⌋

[constr_2040] swImplPolicy for VariableDataPrototype in the role arTypedPerInstanceMemory ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role arTypedPerInstanceMemory shall be standard or measurementPoint. ⌋

[constr_2041] swImplPolicy for VariableDataPrototype in the role staticMemory ⌈ The overriding swImplPolicy attribute value of a VariableDataPrototype in the role staticMemory shall be standard, measurementPoint or message. ⌋

[constr_2042] swImplPolicy for ParameterDataPrototype in ParameterInterface ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in ParameterInterface shall be standard, const or fixed. ⌋

[constr_2043] swImplPolicy for ParameterDataPrototype in the role staticMemory ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role romBlock shall be standard. ⌋

[constr_2044] swImplPolicy for ParameterDataPrototype in the role sharedParameter ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role sharedParameter shall be standard. ⌋

[constr_2045] swImplPolicy for ParameterDataPrototype in the role perInstanceParameter ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role sharedParameter shall be standard. ⌋

[constr_2046] swImplPolicy for ParameterDataPrototype in the role constantMemory ⌈ The overriding swImplPolicy attribute value of a ParameterDataPrototype in the role sharedParameter shall be standard, const or fixed. ⌋

[constr_2047] swImplPolicy for ArgumentDataPrototype ⌈ The overriding swImplPolicy attribute value of a ArgumentDataPrototype shall be standard. ⌋

[constr_2048] swImplPolicy for SwServiceArg ⌈ The overriding swImplPolicy attribute value of a SwServiceArg shall be standard or const. ⌋

[TPS_SWCT_2000] Default value for attribute swImplPolicy ⌈ If the attribute swImplPolicy is not explicitly set at any of the locations listed in "'Place of Setting"' for SwDataDefProps mentioned in table 5.41 the default value standard applies. ⌋

5.4.2 Invalid Value

The diagram 5.5 shows that in addition to the semantics defined through the compuMethod (explained below in chapter 5.5.1), also an invalidValue can be specified. This is a requirement of the VFB [3], allowing to express which specific value is used to indicate invalidation.

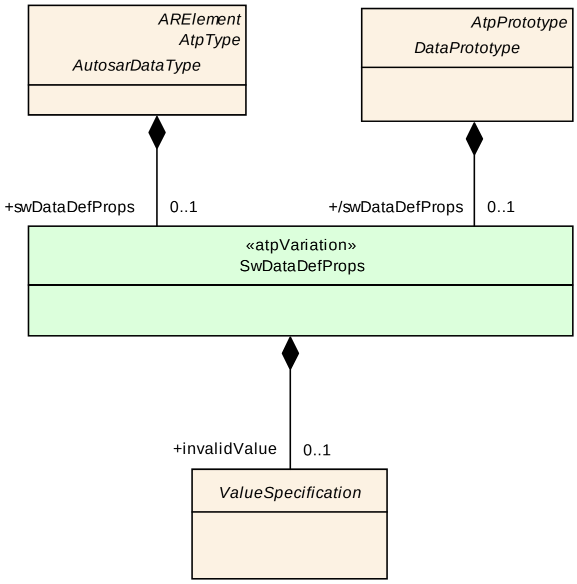

Figure 5.23: Invalid value

The `invalidValue` can be used in different flavors (also illustrated in Figure 5.6:

- On the one hand it is possible to keep the `invalidValue` transparent to the sending and receiving software components. In this case the invalidation API of the RTE on the sender side has to be used.

  The receiving software component can either use the data receive status or the `DataReceiveErrorEvent` respectively `DataReceivedEvent` to decide about the validity of the received data or the receiving software component can rely on the reception of an `initValue` as a default value in case of data invalidation.

  In this case the invalid value should (and usually will) be outside of the range limits defined by the `compuMethod`.

- On the other hand it is possible that the communicating software components do have knowledge about the `invalidValue` and the `invalidValue` is visible for them. This is in particular the case if the sender and receiver are calculating a checksum over a larger data structure to implement an end to end communication protection. To ensure the integrity of the checksums it is required to set invalid values by the sending component directly and to receive invalid values unchanged.

  In this case the invalid value should (and usually will) be inside of the range limits defined by the `compuMethod`.

- Further on it is possible that in case of 1:n communication different receivers requiring a different handling of data invalidation depending on the criticality of its functionality. For instance, one receiver applies the checksum based end to end communication protection and another receiver relies on the substitution of invalid values by initValues.

Of course, an invalidValue can also be specified without setting a compuMethod.

Figure 5.6 illustrates the relationship between ApplicationDatatype, CompuMethod, ImplementationDataType, invalidValue, BaseType.

[constr_2545] invalidValue shall fit in the specified ranges ⌈ The invalidValue shall be in the range of the ImplementationDatatype. ⌋

Please note that the invalidValue is a ValueSpecification. Of course, it would technically be possible to use any subclass of ValueSpecification at this place.

[constr_1016] invalidValue is restricted ⌈ invalidValue is restricted to to be either a compatible NumericalValueSpecification, TextValueSpecification or a ConstantReference that in turn points to a compatible ValueSpecification. ⌋

Invalidation of composite types shall be handled by the RTE as follows: For reasons of efficiency the sending RTE should set the first leaf element (recursively identify this where applicable) of a composite data type to the invalid value in order to indicate that the entire value of the composite data type is invalid.

The receiving RTE would then interpret the invalidation status of the first leaf element and use this as an invalidation status of the entire composite data type. This works for arrays and record types as well, because record elements are ordered in the formal description.

[constr_1140] Combination of initValue with the attribute handleInvalid ⌈ The combination of setting the attribute handleInvalid of the meta-class InvalidationPolicy owned by SenderReceiverInterface to value replace **and** of setting the value of the attribute initValue owned by a corresponding NonqueuedReceiverComSpec effectively to the value of the invalidValue (owned by a corresponding SwDataDefProps) is not supported. ⌋

The term "corresponding" (as utilized in [constr_1140]) refers to the fact that information regarding the fulfillment of [constr_1140] is factually distributed over different areas of the meta-model. For clarification, the following relationship should be considered:

The SenderReceiverInterface defines how to deal with an invalid value by means of the attribute handleInvalid on the basis of individual dataElements. The SenderReceiverInterface is taken for typing a RPortPrototype that in turn owns a ReceiverComSpec. [constr_1140] applies if the particular ReceiverComSpec is actually a NonqueuedReceiverComSpec that refers to the same dataElement.

In this case the invalidValue owned by the SwDataDefProps that in turn is owned by the respective dataElement is relevant for the fulfillment of [constr_1140]. The "big picture" of this relationship is sketched in Figure 5.24.

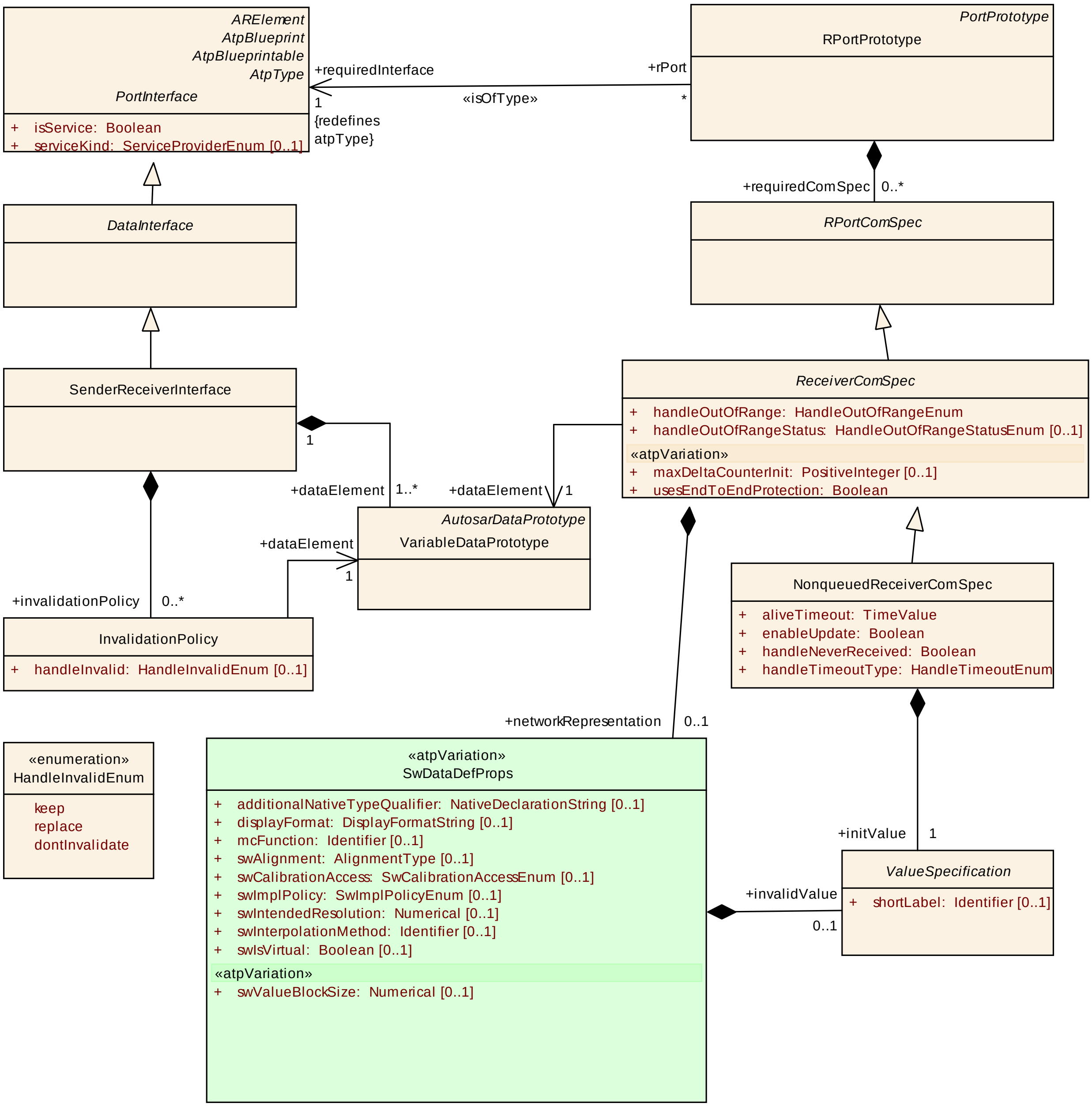

Figure 5.24: Relationships required to consider the `invalidValue`

5.4.3 Properties for Measurement

In embedded automotive software design, measurement means access to memory locations in an ECU and transferring its contents to the measurement & calibration system. While in classical software design, variables abstract the memory locations in the code, AUTOSAR provides for this purpose the `DataPrototype` with its various specializations:

- VariableDataPrototype of a SenderReceiverInterface or NvDataInterface used in a PortPrototype (of a SwComponentPrototype), to capture sender-receiver and non volatile data communication between SwComponentPrototypes
- ArgumentDataPrototype of a ClientServerOperation in a ClientServerInterface to capture client-server communication between SwComponentPrototypes.
- VariableDataPrototype in the context of an SwcInternalBehavior to
  - capture communication between RunnableEntitys within a SwComponentPrototype
  - handle data in a non volatile memory block
  - provide pure software component internal memory which has to be accessible for a MCD system

Note that that the ability of being measured is not restricted to primitive data (category VALUE) but can also be applied to composite data (category STRUCTURE or ARRAY).

The following semantical and structural features from SwDataDefProps are relevant (among other purposes) for the measurement system:

- swCalibrationAccess
- swImplPolicy
- compuMethod
- unit (if not specified by compuMethod)
- baseType
- swAddrMethod

[TPS_SWCT_1130] Measurement and calibration access to model elements is defined by swCalibrationAccess ⌈ The ability to be accessed by e.g. a calibration tool is given by setting the swCalibrationAccess attribute. ⌋*(RS_SWCT_3152)*

The following table shows all valid settings of swCalibrationAccess:

Enumeration	**SwCalibrationAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties
Note	Determines the access rights to a data object w.r.t. measurement and calibration.
Literal	***Description***
notAccessible	The element will not be accessible via MCD tools, i.e. will not appear in the ASAP file.
readOnly	The element will only appear as read-only in an ASAP file.
readWrite	The element will appear in the ASAP file with both read and write access.

Table 5.46: SwCalibrationAccessEnum

[constr_1017] Supported combinations of SwImplPolicy and SwCalibrationAccess ⌈ The table 5.47 defines the supported combinations of SwImplPolicy and SwCalibrationAccess attribute setting. ⌋

SwImplPolicy	SwCalibrationAccess		
	notAccessible	readOnly	readWrite
fixed	yes	not supported	not supported
const	yes	yes	not supported
standard	yes	yes	yes
queued	yes	not supported	not supported
measurementPoint	not supported	yes	not supported

Table 5.47: Supported combinations of SwImplPolicy and SwCalibrationAccess

[constr_1018] measurementPoint shall not be referenced in DataReadAccess ⌈ Due to the nature of data elements characterized as measurementPoint, such data elements shall not be referenced in DataReadAcess. ⌋

5.4.4 Properties of Curves and Maps

A characteristic table is defined by setting the category of the corresponding AutosarDataType or DataPrototype to CURVE respectively MAP. Its SwDataDefProps determine an axis description. The type of the functional values is given by the attached SwBaseType and the CompuMethod.

The axis description itself is defined by the meta-model element SwCalprmAxisSet aggregating the appropriate number of SwCalprmAxisTypeProps. This is the base class for a so called "individual axis" SwAxisIndividual or a "grouped axis" SwAxisGrouped. The latter is used to share axis points by several characteristic tables. Figure 5.25 shows an overview on the relevant meta-model elements.

The type of the functional values is given by the attached SwBaseType and the CompuMethod or by the referenced ApplicationDataType. If an ApplicationDataType is referenced (via valueAxisDataType) this supersedes CompuMethod, Unit, and BaseType if these are defined in parallel.

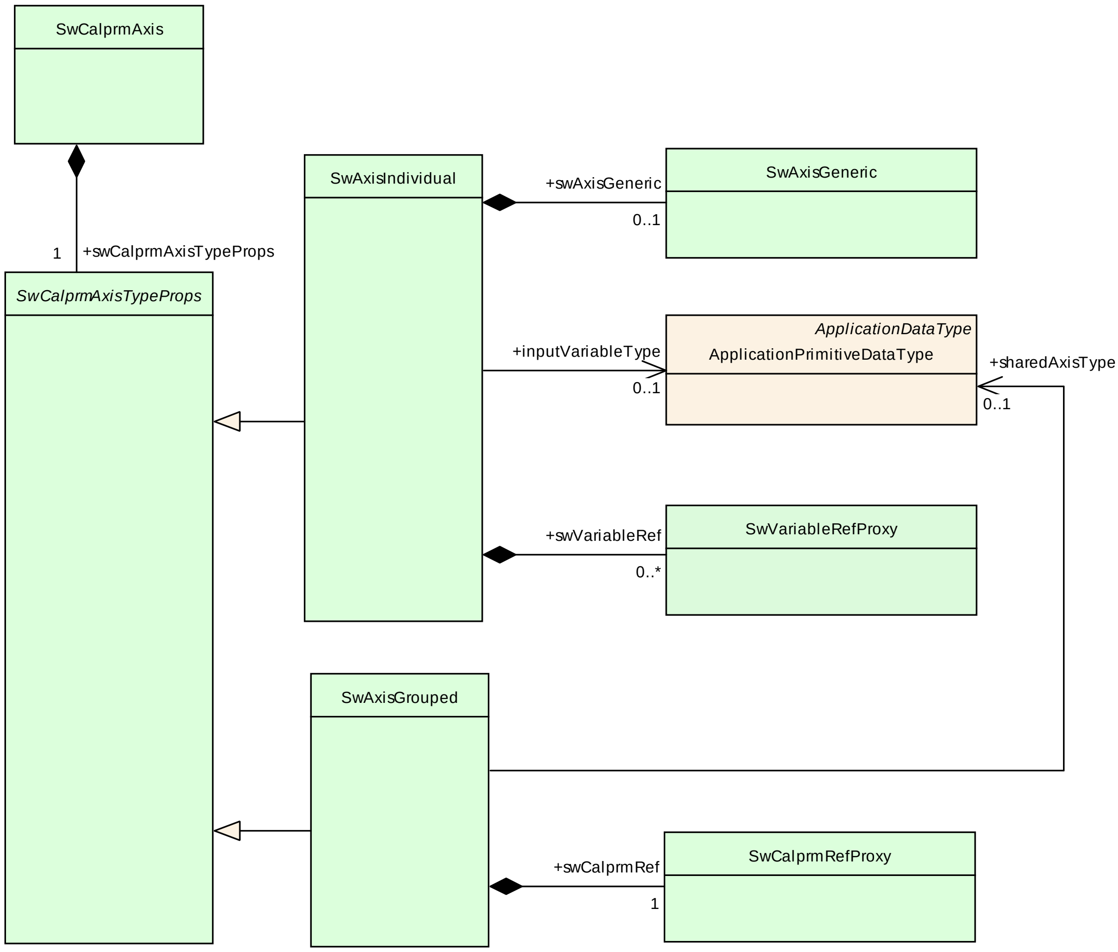

Figure 5.25: Overview on the Meta-Model for Axis Description

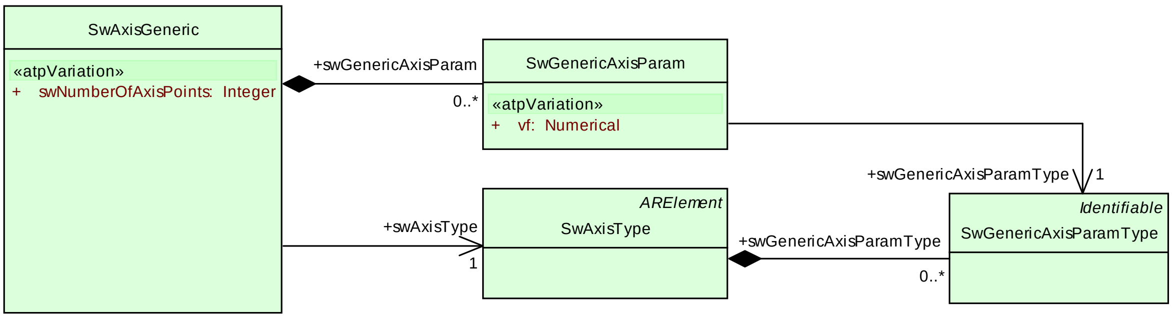

Figure 5.26: Overview on a Generic Axis

Figure 5.27 shows how an individual axis is represented by the meta-model. The corresponding M1 Model is illustrated in Figure 5.28. The `SwAxisIndividual` references value-models to account the minimum and the maximum number of axis values as well as the number of axis points.

Hence, the size of the structure to hold the functional values is determined by the number of axis values for all axes. The type of the axis values is determined when the type of the referenced input value (`swVariableRef`) has been set. For further details see 5.4.5.

[TPS_SWCT_1107] `swMinAxisPoints` and `swMaxAxisPoints` represent variation points ⌈ The value of attributes `swMinAxisPoints` and `swMaxAxisPoints` is subject to variant handling. ⌋*(RS_SWCT_3148)*

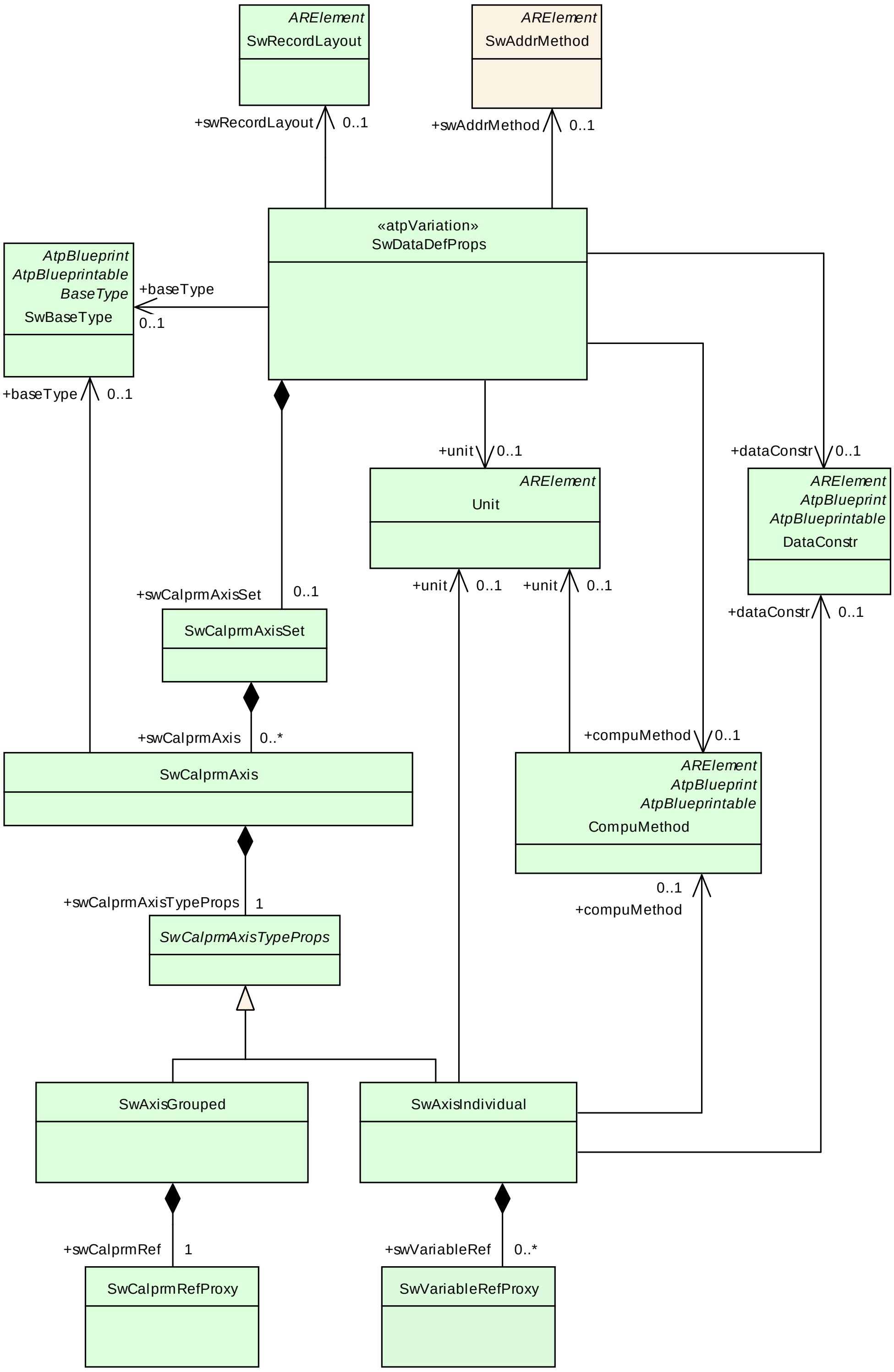

Figure 5.27: Meta-Model Elements used for a Curve

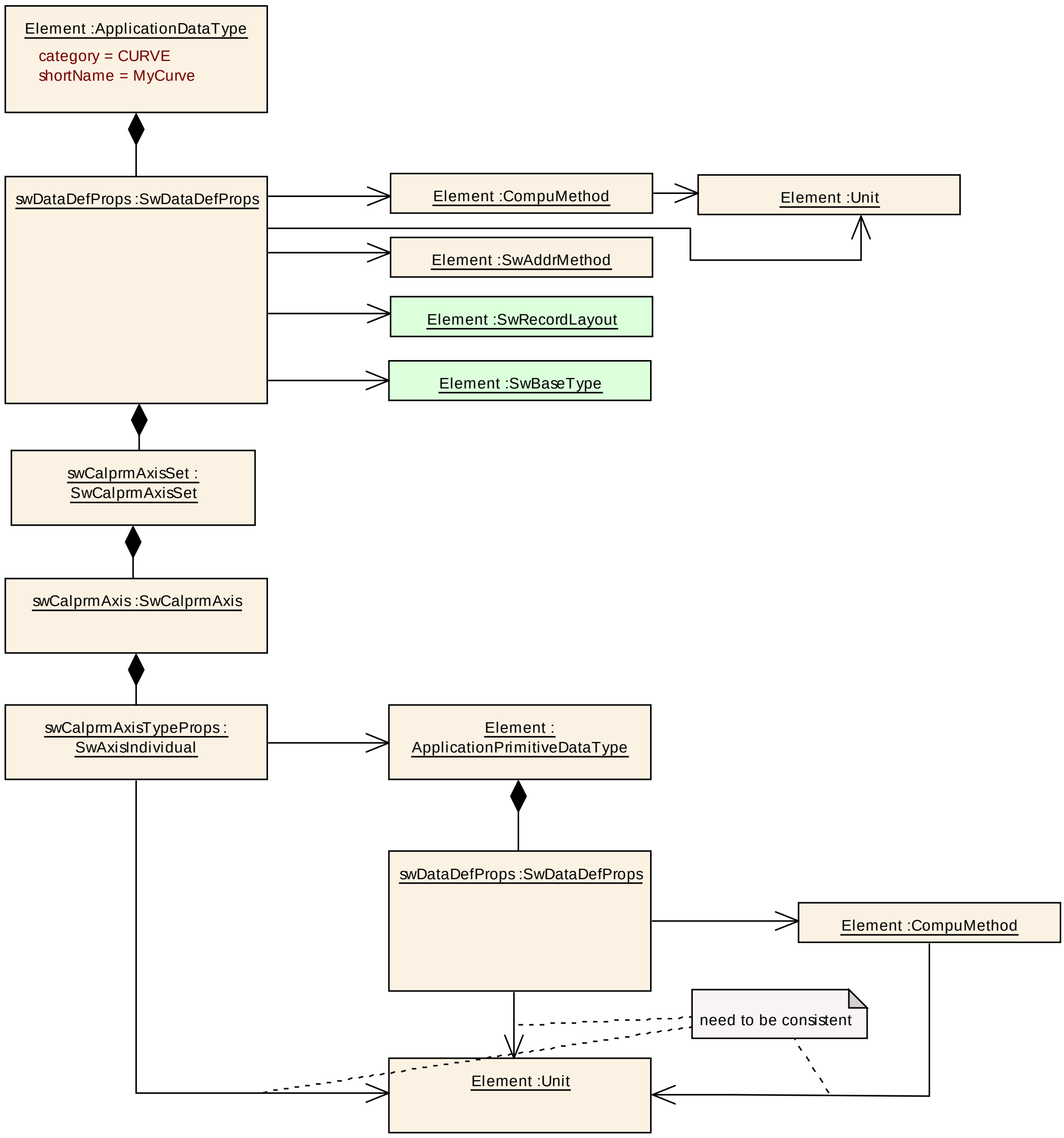

Figure 5.28: Illustration of a Curve in M1

Class	**SwCalprmAxisSet**			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationParameter			
Note	This element specifies the input parameter axes (abscissas) of parameters (and variables, if these are used adaptively).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swCalprm Axis	SwCalprmAxis	*	aggr	One axis belonging to this SwCalprmAxisSet **Tags:** xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=20; xml.type Element=false; xml.typeWrapperElement=false

Table 5.48: SwCalprmAxisSet

Class	**SwCalprmAxis**			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationParameter			
Note	This element specifies an individual input parameter axis (abscissa).			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
category	CalprmAxisCategoryEnum	0..1	attr	This property specifies the category of a particular axis. **Tags:** xml.sequenceOffset=30
baseType	SwBaseType	0..1	ref	The SwBaseType to be used for the axis. Note that this is not applicable for ApplicationDataTypes. The value shall be ignored. **Tags:** atp.Status=obsolete xml.sequenceOffset=110
displayFormat	DisplayFormatString	0..1	attr	This property specifies how the axis values shall be displayed e.g. in documents or in measurement and calibration tools. **Tags:** xml.sequenceOffset=100
swAxisIndex	AxisIndexType	0..1	attr	This attribute specifies which axis is specified by the containing SwCalprmAxis. For example in a curve this is usually "1". In a map this is "1" or "2". **Tags:** xml.sequenceOffset=20
swCalibrationAccess	SwCalibrationAccessEnum	0..1	attr	Describes the applicability of parameters and variables. **Tags:** xml.sequenceOffset=90
swCalprmAxisTypeProps	SwCalprmAxisTypeProps	1	aggr	specific properties depending on the type of the axis. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=40; xml.typeElement=true; xml.typeWrapperElement=false

Table 5.49: SwCalprmAxis

Enumeration	**CalprmAxisCategoryEnum**
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationParameter
Note	This enum specifies the possible values of the category property within SwCalprmAxis.
Literal	**Description**
comAxis	COM_AXIS is equal to an STD_AXIS, the difference is, that a COM_AXIS is an shared axis, that means this axis can be used multiple times by different curves or maps. **Tags:** xml.name=COM_AXIS

comAxis_O	COM-AXIS is equal to an STD_AXIS, the difference is, that a COM-AXIS is an shared axis, that means this axis can be used multiple times by different curves or maps. This value is obsolete. **Tags:** atp.Status=obsolete xml.name=COM-AXIS
curveAxis	CURVE_AXIS uses a separate CURVE to rescale the axis. The referenced CURVE is used to lookup an axis index, and the index value is used by the controller to determine the operating point in the CURVE or MAP. **Tags:** xml.name=CURVE_AXIS
curveAxis_O	CURVE-AXIS uses a separate CURVE to rescale the axis. The referenced CURVE is used to lookup an axis index, and the index value is used by the controller to determine the operating point in the CURVE or MAP. This value is obsolete. **Tags:** atp.Status=obsolete xml.name=CURVE-AXIS
fixAXIS	FIX_AXIS means that the input axis is not stored. The axis is calculated using parameters and so on it is also not possible to modify the axis points. **Tags:** xml.name=FIX_AXIS
fixAXIS_O	FIX-AXIS means that the input axis is not stored. The axis is calculated using parameters and so on it is also not possible to modify the axis points. This value is obsolete. **Tags:** atp.Status=obsolete xml.name=FIX-AXIS
resAxis	RES_AXIS is also an shared axis like COM_AXIS, the difference is that this kind of axis can be used for rescaling. **Tags:** xml.name=RES_AXIS
resAxis_O	RES-AXIS is also an shared axis like COM_AXIS, the difference is that this kind of axis can be used for rescaling. This value is obsolete. **Tags:** atp.Status=obsolete xml.name=RES-AXIS
stdAxis	STD_AXIS means that input and output axis definition are stored within this CURVE. There is no shared or calculated axis. **Tags:** xml.name=STD_AXIS
stdAxis_O	STD-AXIS means that input and output axis definition are stored within this CURVE. There is no shared or calculated axis. This value is obsolete. **Tags:** atp.Status=obsolete xml.name=STD-AXIS

Table 5.50: CalprmAxisCategoryEnum

Class	**SwCalprmAxisTypeProps (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationParameter			
Note	Base class for the type of the calibration axis. This provides the particular model of the specialization. If the specialization would be the directly from SwCalPrmAxis, the sequence of common properties and the specializes ones would be different.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.51: SwCalprmAxisTypeProps

Class	**SwAxisIndividual**			
Package	M2::AUTOSARTemplates::CommonStructure::Axis			
Note	This element describes an axis integrated into a parameter (field etc.). The integration makes this individual to each parameter. The so-called grouped axis represents the counterpart to this. It is conceived as an independent parameter (see class SwAxisGrouped).			
Base	ARObject,SwCalprmAxisTypeProps			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
compuMethod	CompuMethod	0..1	ref	This is the compuMethod which is expected for the axis. It is used in early stages if the particular input-value is not yet available. **Tags:** xml.sequenceOffset=30
dataConstr	DataConstr	0..1	ref	Refers to constraints, e.g. for plausibility checks. **Tags:** xml.sequenceOffset=80
inputVariableType	ApplicationPrimitiveDataType	0..1	ref	This is the datatype of the input value for the axis. This allows to define e.g. a type of curve, where the input value is finalized at the access point.
swAxisGeneric	SwAxisGeneric	0..1	aggr	this specifies the properties of a generic axis if applicable. **Tags:** xml.sequenceOffset=90
swMaxAxisPoints	Integer	1	attr	Maximum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=60
swMinAxisPoints	Integer	1	attr	Minimum number of base points contained in the axis of a map or curve. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=70

Attribute	Datatype	Mul.	Kind	Note
swVariableRef	SwVariableRefProxy	*	aggr	Refers to input variables of the axis. It is possible to specify more than one variable. Here the following is valid: • The variable with the highest priority must be given first. It is used in the generation of the code and is also displayed first in the application system. • All variables referenced must be of the same physical nature. This is usually detected in that the conversion formulae affected refer back to the same SI-units. In AUTOSAR this ensured by the constraint, that the referenced input variables must use a type compatible to "inputVariableType". • This multiple referencing allows a base point distribution for more than one input variable to be used. One example of this are the temperature curves which can depend both on the induction air temperature and the engine temperature. These variables can be displayed simultaneously by MCD systems (adjustment systems), enabling operating points to be shown in the curves. **Tags:** xml.roleElement=false; xml.roleWrapperElement=true; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
unit	Unit	0..1	ref	This represents the physical unit of the input value of the axis. It is provided to support the case that the particular input variable is not yet known. **Tags:** xml.sequenceOffset=40

Table 5.52: SwAxisIndividual

Class	SwAxisGeneric			
Package	M2::AUTOSARTemplates::CommonStructure::Axis			
Note	This element defines a genric axis. In a generic axis the axispoints points are calculated in the ECU. The ECU is equipped with a fixed calculation algorithm. Parameters for the algorithm can be stored in the data component of the ECU. Therefore these paramters are specified in the data declaration, not in the calibration data.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	Datatype	Mul.	Kind	Note
swAxisType	SwAxisType	1	ref	Associated axis calculation strategy. **Tags:** xml.sequenceOffset=20
swGenericAxisParam	SwGenericAxisParam	*	aggr	Specific parameter of a generic axis. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false
swNumberOfAxisPoints	Integer	1	attr	The number of base points to be calculated for this axis. This element exists to enable the number of axis points to be stored explicitly, although it could also be described as swGenericAxisParam. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=30

Table 5.53: SwAxisGeneric

Class	**SwAxisGrouped**			
Package	M2::AUTOSARTemplates::CommonStructure::Axis			
Note	An SwAxisGrouped is an axis which is shared between multiple calibration parameters.			
Base	ARObject,SwCalprmAxisTypeProps			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
sharedAxisType	ApplicationPrimitiveDataType	0..1	ref	This is the datatype of the calibration parameter providing the shared axis.
swAxisIndex	AxisIndexType	0..1	attr	Describes which axis of the referenced calibration parameter provides the values for the group axis. The index satisfies the following convention: • 0 = value axis. in this case, the interpolation result of the referenced parameter is used as a base point index. This means that the A2L keyword CURVE_AXIS_REF can be supported. • The index should only be specified if the parameter under swCalprm contains more than one axis. It is standard practise for the axis index of parameters with more than one axis, to be set to 1, if data has not been assigned to swAxisIndex. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
swCalprm Ref	SwCalprmRefProxy	1	aggr	This property specifes the calibration parameter which serves as the input axis. In AUTOSAR, the type of the referenced Calibration parameter must be compatible to the type specified by sharedAxisType. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.54: SwAxisGrouped

5.4.5 Setting an Axis Input Value

When an interpolation routine is called, an input value has to be provided to find the appropriate axis entry in the implementation of a `RunnableEntity`. However, this input value cannot be arbitrarily chosen but only be selected from available `VariableDataPrototype` assigned to it.

In an axis definition attached to an `ApplicationPrimitiveDataType`, it is possible to specify the `inputVariableType` for the input values.

[constr_1019] Compatibility of input value and axis ⌈ The `SwDataDefProps` the input variable shall be compatible to the `datatype` resp. `compuMethod` resp. `unit` of the `SwAxisIndividual`. ⌋

Every `ParameterDataPrototype` then allows to specify zero or more input values (being type compatible to `inputVariableType`) in its axis description.

This means that at the specification time of an `SwcInternalBehavior` a list of input values has to be specified where the implementor of a `RunnableEntity` can choose of. The input values are `DataPrototype` entities either being

- a `VariableDataPrototype` in a `SenderReceiverInterface` or `NvDataInterface` of a `PortPrototype`, of the `AtomicSwComponentType` where the `SwcInternalBehavior` is associated to, or an `ArgumentDataPrototype` in a `ClientServerOperation` of a `ClientServerInterface` in a `PortPrototype` of the `AtomicSwComponentType` where the `InternalBehavior` is associated to, or
- an `VariableDataPrototype` within the `SwcInternalBehavior`.

To achieve this, `SwAxisIndividual` is referencing a `SwVariableRefProxy`. This proxy is an abstract class being refined in AUTOSAR style by a `DataPrototypeRefProxy` entity as shown in Figure 5.29. This `DataPrototypeRefProxy` has an `instanceRef` to a `DataPrototype` in the appropriate context.

Originally, MSRSW uses a `SwVariableRef` to set the input value of an axis appropriately. In AUTOSAR, this has been extended by first introducing a `SwVari-`

ableRefProxy. This will then be derived in DataPrototypeRef (AUTOSAR style) or SwVariableRef (MSR style).

As shown in Figure 5.29, this approach is also used to represent a DataPrototype-Ref in all roles, e.g. the result of an interpolation routine applied to an axis, the input value determination, a list of dependent parameters, and swDataDependency.

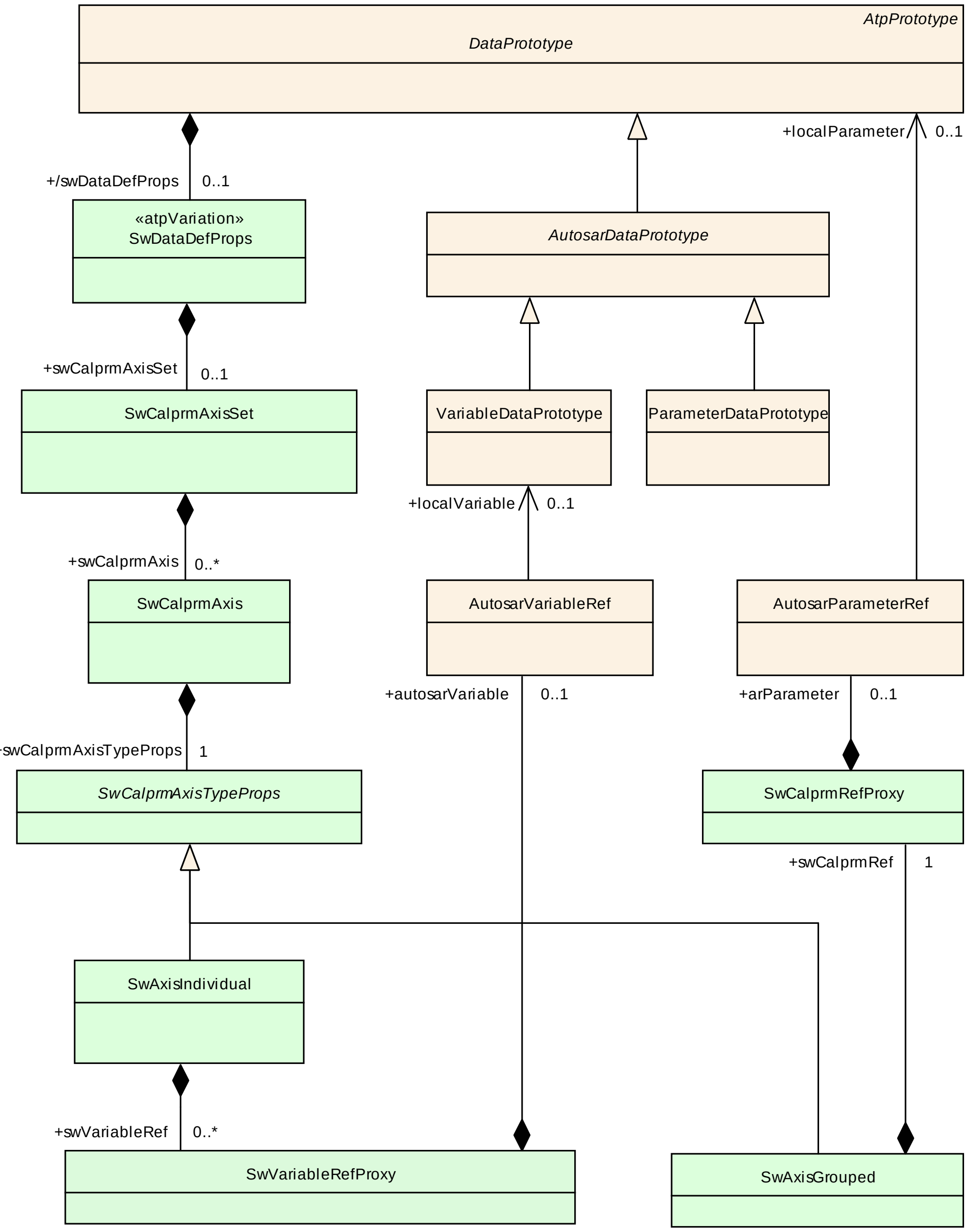

Figure 5.29: Extended Axis Elements and Input Variable Reference

Grouped curves share the same axis definition. In MSRSW, this is shown by referencing the SwCalprm, representing an individual curve, from a SwAxisGrouped.

Note that this does not describe which axis shall be taken from a reference SwCalprm acting as a shared axis. This would be done in SwAxisGrouped.axisIndex.

AUTOSAR applies a similar proxy approach for parameters as for the variables. Therefore, an `SwCalprmRefProxy` has been introduced in MSRSW, and is aggregated by the `SwAxisGrouped` element.

The `SwCalprmProxy` aggregates an `AutosarParameterRef` providing an association to a `ParameterDataPrototype`, representing a curve with an axis. When defining the data type of a parameter the type of the shared axis is defined in `sharedAxisType`.

[constr_1020] `ParameterDataPrototype` needs to be of compatible data type as referenced in `sharedAxisType` ⌈ Finally, the `ParameterDataPrototype` assigned in `swCalprmRef` shall be typed by data type compatible to `sharedAxisType`.
⌋

The AUTOSAR-style is shown in the upper left part of Figure 5.29, while in the upper middle the MSRSW style is shown, referencing the `SwCalprm`.

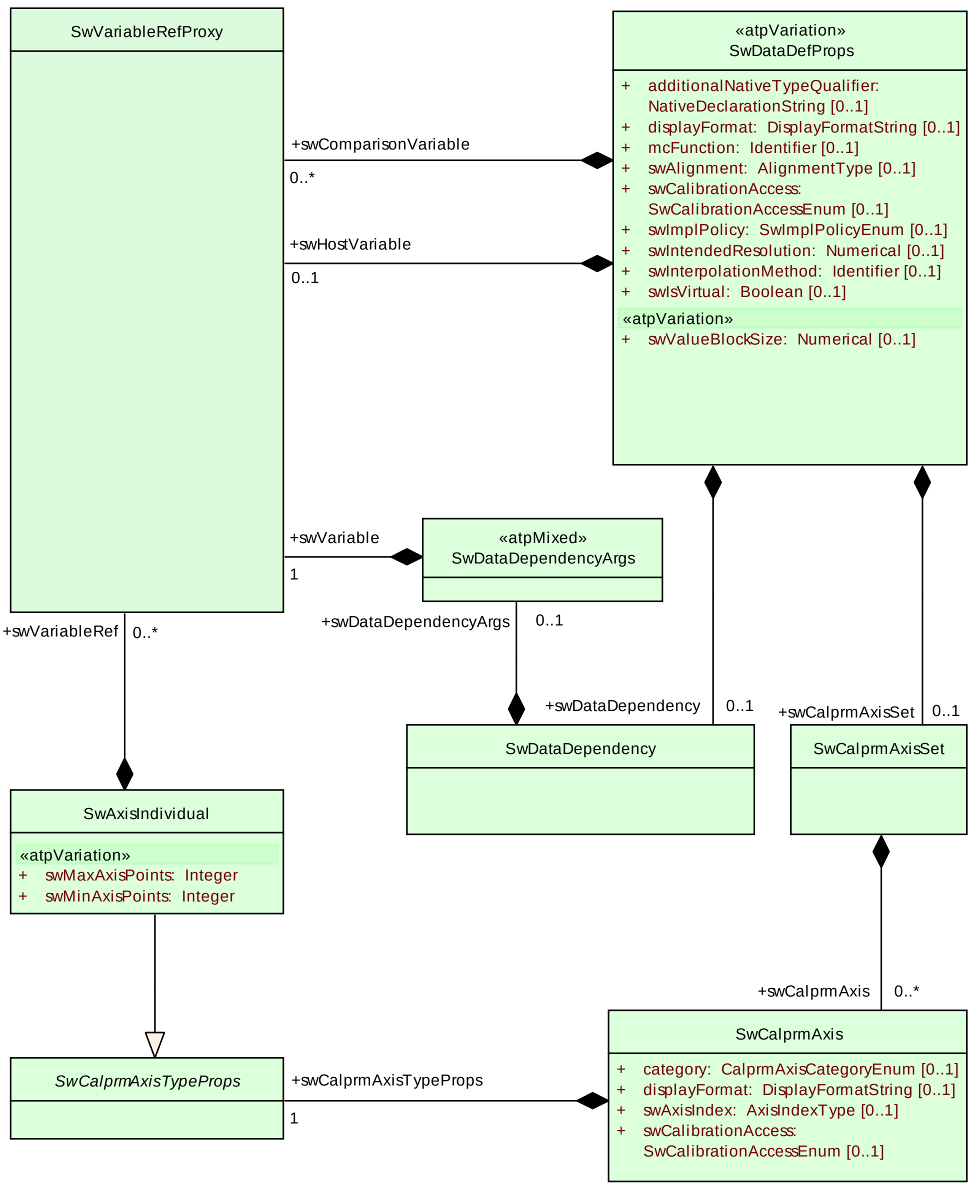

Figure 5.30: Applying Proxy Variable Reference Mechanism

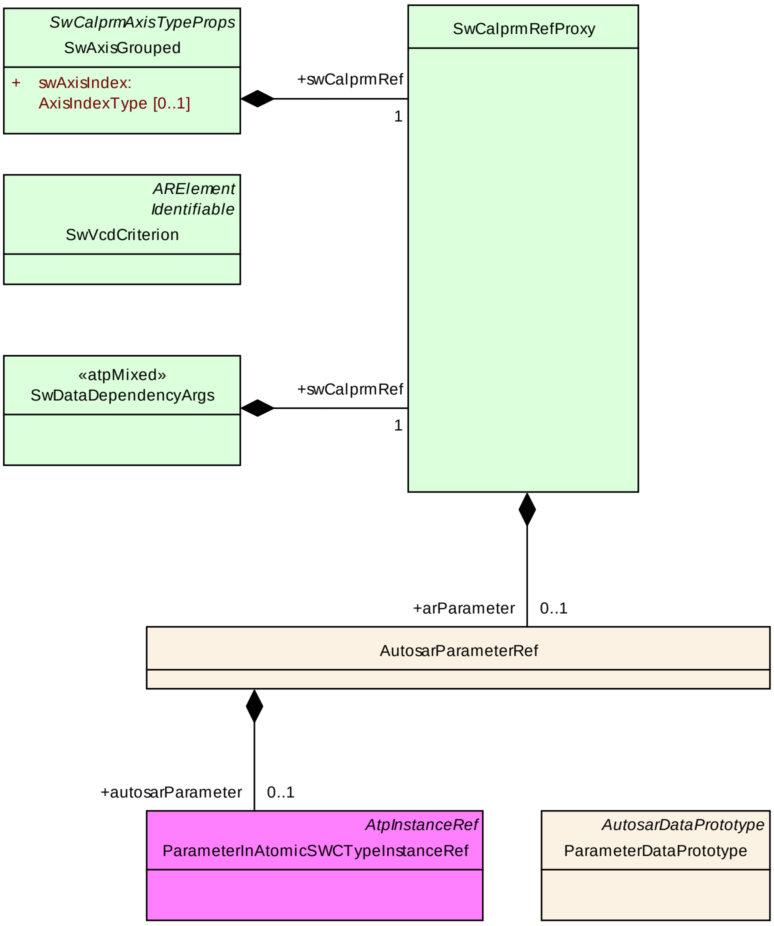

Figure 5.31: Applying Proxy Parameter Reference Mechanism

Class	**SwCalprmRefProxy**			
Package	M2::AUTOSARTemplates::CommonStructure::DatadictionaryProxies			
Note	Wrapper class for different kinds of references to a calibration parameter.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arParameter	AutosarParameterRef	0..1	aggr	This represents a Parameter within Autosar. Note that the Datatype of the referenced ParameterDataPrototype must be an ApplicationDataType of category VALUE.
mcDataInstance	McDataInstance	0..1	ref	This reference is used in the McSupport file to express the final instance of group axis etc. It is not allowed to use this outside of an McDataInstance. The referenced mcDataInstance must be origninated from a ParameterDataPrototype.

Table 5.55: SwCalprmRefProxy

Class	**SwVariableRefProxy**			
Package	M2::AUTOSARTemplates::CommonStructure::DatadictionaryProxies			
Note	Parent class for several kinds of references to a variable.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
autosarVariable	AutosarVariableRef	0..1	aggr	This represents the reference to a Variable in an Autosar system. Note that the Target of the reference within AutosarrVariableRef must be of primitiveType
mcDataInstanceVar	McDataInstance	0..1	ref	This reference is used in the McSupport file to express the final instance of input values etc. It is not allowed to use this outside of an McDataInstance. The referenced mcDataInstance must be origninated from a VaraibleDataPrototype.

Table 5.56: SwVariableRefProxy

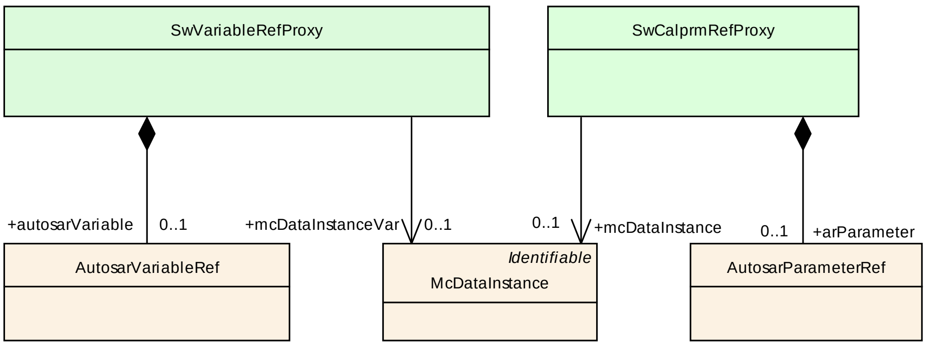

Figure 5.32: Proxy reference classes

The basic patterns for referencing `DataPrototype`s are explained in section 5.3.2. In the context of this chapter it is worth to remark that the definition of access to calibration parameters is implemented in the context of a `RunnableEntity` (see Figure 7.3).

As the definition of a calibration parameter may involve the definition of several axes the necessity to provide this amount of information might become cumbersome and (to some extent) redundant and difficult to maintain if the same calibration parameter is accessed from within several `RunnableEntity`s. In other words: in this case it would be necessary to repeat the more or less complex set of information for each `RunnableEntity`.

To avoid this unnecessary level of complexity for the definition of access to calibration parameters, it is possible to define the access to the calibration parameter on the level of `InstantiationDataDefProps` which have been defined to facilitate this kind of re-use (for more information please refer to section 7.5.4). This ability is also documented in Table 5.41.

5.4.6 Specifying Data Dependencies

`SwDataDependency` allows dependent data elements to be specified. For example other `ParameterDataPrototype`s can be combined into one `ParameterDataElement` whose consistent value is automatically derived by the measurement and calibration system. Upon adjusting one of the parameters, the dependent parameter is then also automatically adjusted according to the chosen formula.

Consider for example a rectangular triangle with a hypotenuse of length 1, where the length of the other sides are the parameter A and B. When adjusting A the parameter B has to be adjusted accordingly to $B = \sqrt{(1 - A * A)}$. Also other parameters might depend on B, e.g. $B\_AREA = B * B$ or $TRIANGULAR\_AREA = (A * B)/2$. This example is shown in listing 5.5.

A dependent parameter should not be adjustable by itself. The only way to influence its value is through the adjustment of a parameter it depends on.

Listing 5.5: Data Dependency

```
<PER-INSTANCE-PARAMETERS>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>A</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The independent Parameter</L-2>
    </DESC>
    <CATEGORY>VALUE</CATEGORY>
  </PARAMETER-DATA-PROTOTYPE>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>B</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The dependent Parameter</L-2>
    </DESC>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-DATA-DEPENDENCY>
            <SW-DATA-DEPENDENCY-FORMULA>SQRT( X1 * X1)</SW-
                DATA-DEPENDENCY-FORMULA>
            <SW-DATA-DEPENDENCY-ARGS>
              <AR-PARAMETER>
                <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-
                    PROTOTYPE">/DataDependency/foo/bar/A</
                    LOCAL-PARAMETER-REF>
              </AR-PARAMETER>
            </SW-DATA-DEPENDENCY-ARGS>
          </SW-DATA-DEPENDENCY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
  </PARAMETER-DATA-PROTOTYPE>
  <PARAMETER-DATA-PROTOTYPE>
    <SHORT-NAME>B_AREA</SHORT-NAME>
    <DESC>
      <L-2 L="DE">The dependent Parameter</L-2>
```

```
            </DESC>
            <SW-DATA-DEF-PROPS>
              <SW-DATA-DEF-PROPS-VARIANTS>
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                  <SW-DATA-DEPENDENCY>
                    <SW-DATA-DEPENDENCY-FORMULA>X1 * X1</SW-DATA-
                       DEPENDENCY-FORMULA>
                    <SW-DATA-DEPENDENCY-ARGS>
                      <AR-PARAMETER>
                        <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-
                           PROTOTYPE">/DataDependency/foo/bar/B</
                           LOCAL-PARAMETER-REF>
                      </AR-PARAMETER>
                    </SW-DATA-DEPENDENCY-ARGS>
                  </SW-DATA-DEPENDENCY>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
              </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
          </PARAMETER-DATA-PROTOTYPE>
          <PARAMETER-DATA-PROTOTYPE>
            <SHORT-NAME>TRIANGULAR_AREA</SHORT-NAME>
            <DESC>
              <L-2 L="DE">The dependent Parameter</L-2>
            </DESC>
            <SW-DATA-DEF-PROPS>
              <SW-DATA-DEF-PROPS-VARIANTS>
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                  <SW-DATA-DEPENDENCY>
                    <SW-DATA-DEPENDENCY-FORMULA>(X1 * X2) / 2</SW-
                       DATA-DEPENDENCY-FORMULA>
                    <SW-DATA-DEPENDENCY-ARGS>
                      <AR-PARAMETER>
                        <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-
                           PROTOTYPE">/DataDependency/foo/bar/A</
                           LOCAL-PARAMETER-REF>
                      </AR-PARAMETER>
                      <AR-PARAMETER>
                        <LOCAL-PARAMETER-REF DEST="PARAMETER-DATA-
                           PROTOTYPE">/DataDependency/foo/bar/B</
                           LOCAL-PARAMETER-REF>
                      </AR-PARAMETER>
                    </SW-DATA-DEPENDENCY-ARGS>
                  </SW-DATA-DEPENDENCY>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
              </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
          </PARAMETER-DATA-PROTOTYPE>
        </PER-INSTANCE-PARAMETERS>
      </SWC-INTERNAL-BEHAVIOR>
    </INTERNAL-BEHAVIORS>
```

Class	**SwDataDependency**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This element describes the interdependencies of data objects, e.g. variables and parameters. Use cases: • Calculate the value of a calibration parameter (by the MCD system) from the value(s) of other calibration parameters. • Virtual data - that means the data object is not directly in the ecu and this property describes how the "virtual variable" can be computed from the real ones (by the MCD system).			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swDataDependencyArgs	SwDataDependencyArgs	0..1	aggr	Specifies the arguments used in the data dependency. Note that this is 0..1 since the aggregated class is a container (atpMixed). **Tags:** xml.sequenceOffset=40
swDataDependencyFormula	CompuGenericMath	0..1	aggr	This element describes the formula with which the dependencies between the participating objects are defined. **Tags:** xml.sequenceOffset=30

Table 5.57: SwDataDependency

Class	≪atpMixed≫ **SwDataDependencyArgs**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This element specifies the elements used in a SwDataDependency.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
swCalprmRef	SwCalprmRefProxy	1	aggr	Specifies a calibration parameter as an input argument to the dependency. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=60; xml.typeElement=false; xml.typeWrapperElement=false
swVariable	SwVariableRefProxy	1	aggr	Specifies a variable as an input argument to the dependency. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=70; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.58: SwDataDependencyArgs

5.4.7 Precedence of data properties with respect to data elements, axis elements, computation methods, units

There similar attributes defined in SwDataDefProps as well as in CalprmAxis as well as in CompuMethod. Therefore we need to define which attribute value wins in the overall process from SWC-Description to MC-Support to ASAM-A2l.

Figure 5.33 illustrates the fact that some attributes in SwDataDefProps can also be expressed in subelements respectively in referenced elements.

The general precedence rule is that

- SwDataDefProps wins over valueAxisDataType (exception: compuMethod and unit)
- SwDataDefProps wins over compuMethod
- SwDataDefProps wins over swCalprmAxis
- SwDataDefProps.swCalprmAxis wins over swCalprmAxis.compuMethod resp. SwAxisIndividual.inputVariableType
- SwAxisIndividual.inputVariableType wins over SwAxisIndividual.CompuMethod, SwAxisIndividual.unit, but **not** over SwAxisIndividual.dataConstr

The following examples illustrate particular cases. The highest precedence comes first.

unit of value axis uses the following precedence

- SwDataDefProps.valueAxisDataType.unit
- SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod.unit
- SwDataDefProps.unit
- SwDataDefProps.compuMethod.unit

[constr_2550] Units of value axis shall be consistent ⌈ The units specified in the context of value axis shall be the same, even if there is a precedence rule. ⌋

[constr_2550] reflects the fact that unit may be specified in different phases of the development process but finally need to be consistent.

data constraints of value axis uses the following precedence

- SwDataDefProps.dataConstr
- SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr

[constr_2548] Data constraint of value axis shall match ⌈ The values compliant to SwDataDefProps.dataConstr shall be also be compliant to SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr.

In other words SwDataDefProps.dataConstr win over but are not allowed to relax SwDataDefProps.valueAxisDataType.swDataDefProps.dataConstr but are not allowed ⌋

compu method of value axis uses the following precedence

- SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod
- SwDataDefProps.compuMethod

display format of value axis uses the following precedence

- SwDataDefProps.displayFormat
- SwDataDefProps.valueAxisDataType.swDataDefProps.displayFormat
- SwDataDefProps.valueAxisDataType.swDataDefProps.compuMethod.displayFormat
- SwDataDefProps.compuMethod.displayFormat

Note that this deviates from the general rule since displayFormat is not an essential property. The last item in the list above is the consequence of the fact that if there is a valueAxisDataType it supersedes the compuMethd

calibration access of value axis uses the following precedence

- SwDataDefProps.calibrationAccess
- SwDataDefProps.valueAxisDataType.swDataDefProps.calibrationAccess

Note that this deviates from the general rule since calibrationAccess is not such an essential property.

unit of input axis uses the following precedence

- SwAxisIndividual.unit
- SwAxisIndividual.compuMethod.unit
- SwAxisIndividual.inputVariableType.swDataDefProps.unit
- SwAxisIndividual.swVariableRef.type.swDataDefProps.unit
- SwAxisIndividual.swVariableRef.type.swDataDefProps.compuMethod.unit

[constr_2549] Units of input axis shall be consistent ⌈ The units specified in the context of an input axis shall be compatible, even if there is a precedence rule. ⌋

[constr_2549] reflects the fact that unit may be specified in different phases of the development process but finally need to be consistent.

data constraint of input axis uses the following precedence

- SwAxisIndividual.dataConstr

- SwAxisIndividual.inputVariableType.swDataDefProps.dataConstr
- SwAxisIndividual.swVariableRef.type.swDataDefProps.dataConstr

Note that `SwAxisIndividual .inputVariableType .swDataDefProps .dataConstr` represent the input value, not the axis itself. For this reason there is no specific constraint that the `dataConstr` need to match.

display format of input axis uses the following precedence

- SwCalprmAxis.displayFormat
- SwCalprmAxis.swCalprmAxisTypeProps.compuMethod.displayFormat
- SwCalprmAxis.swCalprmAxisTypeProps.inputVariableType.swDataDefProps .displayFormat
- SwCalprmAxis.swCalprmAxisTypeProps.inputVariableType.swDataDefProps .compuMethod.displayFormat
- SwCalprmAxis.swCalprmAxisTypeProps.swVariableRef.type.swDataDefProps .displayFormat
- SwCalprmAxis.swCalprmAxisTypeProps.swVariableRef.type.swDataDefProps .compuMethod.displayFormat

Note that `SwAxisIndividual .inputVariableType .swDataDefProps .dataConstr` represent the input value, not the axis itself. For this reason there is no specific constraint that `displayFormat` needs to match.

SwCalibrationAccess of the input axis uses following precedence

- SwDataDefProps.swCalibrationAccess
- SwCalprmAxis.swCalibrationAccess

Note that the `swCalibrationAccess` defined on a compound primitive reflects the entire curve or map. Therefore, if the entire curve or map cannot be accessed by the measurement calibration diagnostic system (MCD-System), the axis can also not be accessed. On the other hand it might be that access is granted for the value axis only but not for the axis points.

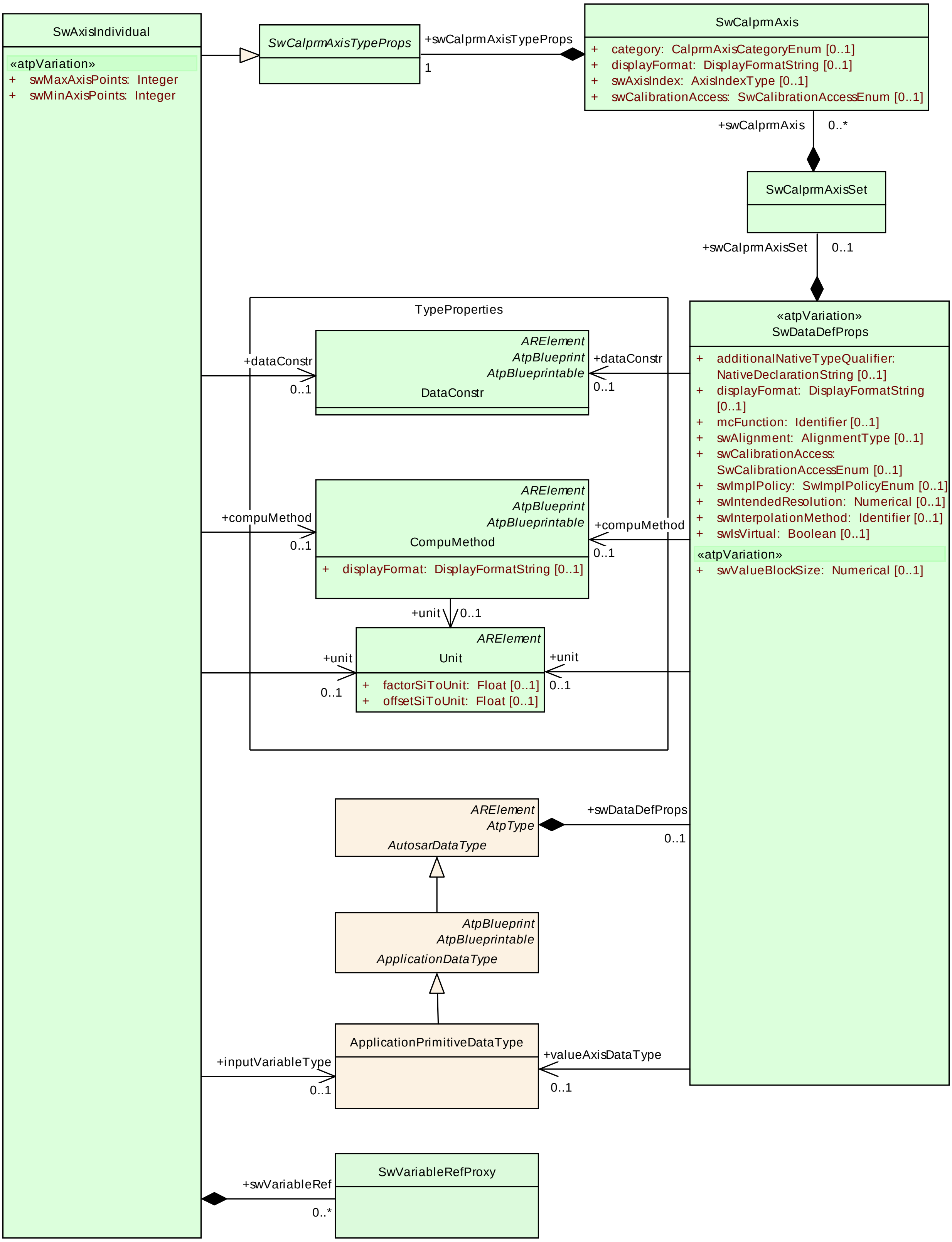

Figure 5.33: Various Attributes in the Context of `SwDataDefProps`

5.5 Elements used in Properties of Data Definitions

This section describes further elements which are attached to `SwDataDefProps` via associations.

5.5.1 Computation Methods

[TPS_SWCT_1276] Computation methods ⌈ An important part of semantics is the specification of a so-called computation method which specifies the conversion between the physical and the internal representation of data. This usually makes sense only for primitive data types. ⌋

An `ApplicationCompositeDataType` cannot be given a particular semantic meaning as a whole but it is obviously possible to specify the semantics of all or a part of the contained elements, i.e. the `ApplicationPrimitiveDataType`s.

Class	**CompuMethod**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express the relationship between a physical value and the mathematical representation. Note that this is still independent of the technical implementation in data types. It only specifies the formula how the internal value corresponds to its physical pendant. **Tags:** atp.recommendedPackage=CompuMethods			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuInter nalToPhys	Compu	0..1	aggr	This specifies the computation from internal values to physical values. **Tags:** xml.sequenceOffset=80
compuPhy sToInternal	Compu	0..1	aggr	This represents the computation from physical values to the internal values. **Tags:** xml.sequenceOffset=90
displayFor mat	DisplayFormatS tring	0..1	attr	This property specifies, how the physical value shall be displayed e.g. in documents or measurement and calibration tools. **Tags:** xml.sequenceOffset=20
unit	Unit	0..1	ref	This is the physical unit of the Physical values for which the CompuMethod applies. **Tags:** xml.sequenceOffset=30

Table 5.59: CompuMethod

This meta-class `CompuMethod` was actually taken from the *ASAM* standard's *harmonized data objects*. This is also indicated by the green color of the meta-classes in the diagram.

[constr_1142] `category` of `compuMethod` shall not be extended ⌈ In contrast to the general rule that `category` can be extended by user-specific values it is **not allowed** to extend the meaning of the attribute `category` of meta-class `CompuMethod` ⌋

[TPS_SWCT_1277] Computation methods are used for the conversion of *internal* values into their *physical* representation and vice versa ⌈ `CompuMethod`s (see Figure 5.34) are used for the conversion of *internal* values into their *physical* representation and vice versa. The direction of the conversion depends on the origin of the value to be converted:

- If the value is provided by the ECU then the conversion direction is from internal to physical.
- If a physical value is provided by the tester it is converted to internal values before being sent to the ECU

⌋

[TPS_SWCT_1278] `CompuMethods` can also be used to assign symbolic names to internal values ⌈ `CompuMethod`s can also be used to assign symbolic names to internal values (like an enumeration in C) or to ranges of internal values or to single bits (like a bitfield in C). This is also considered as a conversion between internal numbers and a semantical representation. Some examples are given below. ⌋

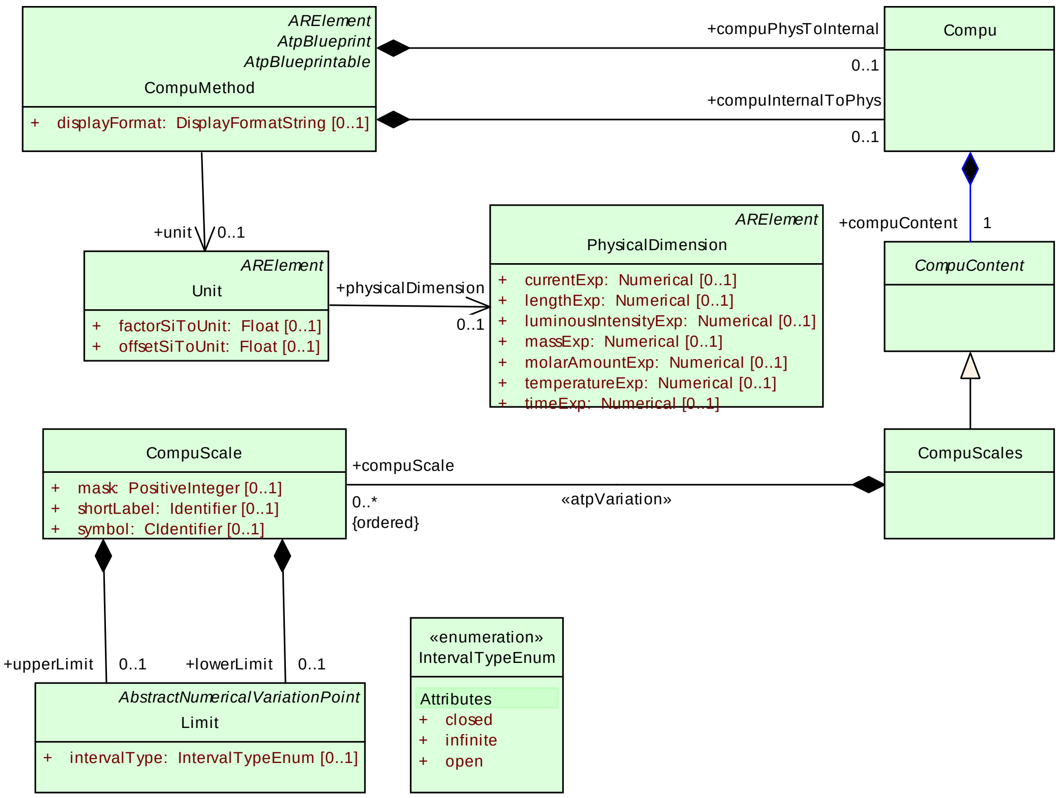

Figure 5.34: A `CompuMethod` and its attributes define data semantics

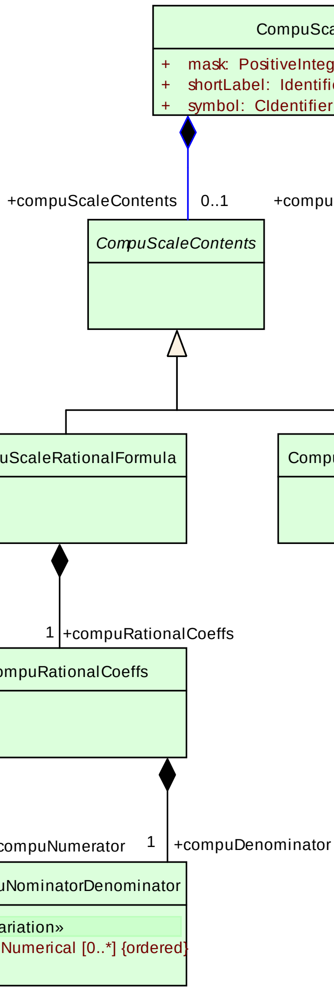

Figure 5.35: A `CompuScale` and its attributes define data semantics

[TPS_SWCT_1279] Preferred conversion direction depends on the use case ⌈
The preferred conversion direction depends on the use case. The physical-to-internal direction is suitable for calibration while the internal-to-physical direction is preferred for diagnostic purposes. ⌋

In the following, the internal-to-physical conversion direction is used as the default. Usually a `CompuMethod` is defined for one conversion direction only even if it is used in both directions.

For simple functions like identical (1:1 conversion) or linear functions this is sufficient because the inverse function can be derived quite easily from the defined function. In this case also the limits for the reverse direction can be gained by applying the forward function to the forward limits.

For more complex functions (e.g. rational functions) it is usually not possible to compute the inverse function automatically. More seriously, the inversion yields ambiguous results if the function is not monotonic. To deal with such possible ambiguities in a direct way an inverse value can be provided explicitly for the function or for each of its parts respectively.

[constr_1021] A `CompuMethod` shall specify instructions for both directions ⌈ The forward and inverse direction shall always be clearly determined either by

- explicitly specifying both directions
- automatically inverting the `CompuMethod` if applicable

⌋

[constr_1022] Limits shall be defined for each direction of `CompuMethod` ⌈ In case that both domains are specified in the `CompuMethod` both shall have explicitly defined limits. ⌋

[TPS_SWCT_1280] `CompuMethod` applied to values outside of its limits ⌈ If a `CompuMethod` is applied to values outside of its limits, it is up to the MCD-tool (Measurement, Calibration, Diagnostic tool) to indicate this to the user. In this case the `CompuMethod` shall not be applied at all. ⌋

[constr_1175] Depending on its `category`, `CompuMethod` shall refer to a `unit` ⌈ As a `CompuMethod` specifies the conversion between the physical world and the numerical values they shall refer to a `unit` unless the `CompuMethod`'s `category` is one of `TEXTTABLE`, `BITFIELD_TEXTTABLE`, or `IDENTICAL`. ⌋

[constr_1175] does *not* imply that `CompuMethod`s where the `category` is one of `TEXTTABLE`, `BITFIELD_TEXTTABLE`, or `IDENTICAL` are not *allowed* to refer to a unit. They may still refer to a `unit`, but according to [constr_1175] this relation is not *mandated*.

A further implication is that the unit itself may not have a dimension, i.e. all exponents of SI units are 0.

Figure 5.34 sketches a conceptual overview of `CompuMethod`. It consists of the following attributes:

- **[TPS_SWCT_1281] `Unit` associated with a `PhysicalDimension`** ⌈ A unit (described in next section) can be associated with a `PhysicalDimension`. ⌋

  Note that quantities like "%" are not derived from SI units. However, they have a meaning in the physical world and need to be represented in form of data types. Therefore, a CompuMethod also applies in those cases.

- **[TPS_SWCT_1430] Conversion specification from internal to physical values as well as the reverse conversion** ⌈ A conversion specification from internal to physical values, as well as the reverse conversion. Both of them in turn consist of an abstract `CompuContent`. Derived classes allow the specification of a conversion formula in two different ways. ⌋

  [constr_1024] Stepwise definition of `CompuMethods` ⌈ Within AUTOSAR only the stepwise definition (`CompuScales`) is used. ⌋

- **[TPS_SWCT_1282] Number of intervals in which a given conversion applies** ⌈`CompuScales` is a number of intervals (called `CompuScale`) within which a certain conversion applies. The respective interval is given in terms of upper and lower limit. Limits are explained in more detail in chapter 5.2.4.1.

Within each CompuScale we have the abstract CompuScaleContent. To deal with possible ambiguities in a direct way an inverse value can be provided explicitly for that particular scale (compuInverseValue). ⌋

- As the diagram shows, CompuScaleContent is an abstract meta-class. A number of derived meta-classes allow the specification of a conversion formula in a variety of ways, including:
  - mapping the whole interval to a constant (CompuConst)
  - providing rational coefficients of the conversion formula (CompuRationalCoeffs)
- **[TPS_SWCT_1283] Rational function** ⌈The rational function is specified as rational coefficients for the numerator (compuNumerator) and the denominator (compuDenominator). CompuNominatorDenominator can have as many *V* elements as needed for the rational function.

  The sequence of the values *V* carries the information for the exponents, that means the first *V* is the coefficient for x0, the second *V* is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented. ⌋

  [constr_1025] Avoid division by zero in rational formula ⌈ The rational formula shall not yield any division by zero. ⌋

[TPS_SWCT_1284] `CompuScale` might require a representation in the generated RTE C code ⌈ A CompuScale might require a representation in the generated RTE C code. For this purpose it is necessary to identify a property that controls how to symbol used for the CompuScale in the C code is created. The symbol itself can be created out of different sources according to a standardized precedence schema. ⌋

[constr_1145] Finding the symbol for the representation of a `CompuScale` in C code ⌈

In general, the value of the attributes symbol, vt, and shortLabel can be taken as a the source for naming the symbol that represents the CompuScale in the C code. The following rule applies (lower values indicate higher priority):

1. Take the value of symbol if this attribute exists.
2. Take the value of vt if it makes a valid C identifier.
3. Take the value of shortLabel if it exists.

Fail if none of the possible options apply.

⌋

[constr_1146] Applicability of a symbol for a `CompuScale` in C code ⌈ The `symbol` attribute shall only be provided for `CompuScale`s where the `category` of the enclosing `CompuMethod` is one of the following:

- SCALE_LINEAR_AND_TEXTTABLE
- SCALE_RATIONAL_AND_TEXTTABLE
- TEXTTABLE
- TAB_NOINTP
- BITFIELD_TEXTTABLE

⌋

Class	**Compu**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express one particular computation.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuContent	CompuContent	1	aggr	This specifies the details of the computation. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false
compuDefaultValue	CompuConst	0..1	aggr	This property can be used to specify an output value for a conversion formula, if the value to be converted lies outside the plausibility limit. Although this is possible for all conversion formulae, it is especially valid for variables with tabular conversion formulae. **Tags:** xml.sequenceOffset=70

Table 5.60: Compu

Class	**CompuContent (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This abstract meta-class represents the various definition means of a computation method.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 5.61: CompuContent

Class	**CompuScale**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to specify one segment of a segmented computation method.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** xml.sequenceOffset=30
compuInverseValue	CompuConst	0..1	aggr	This is the inverse value of the constraint. This supports the case that the scale is not reversible per se. **Tags:** xml.sequenceOffset=60
compuScaleContents	CompuScaleContents	0..1	aggr	This represents the computation details of the scale. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=70; xml.typeElement=false; xml.typeWrapperElement=false
lowerLimit	Limit	0..1	aggr	This element specifies the lower limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=40
mask	PositiveInteger	0..1	attr	In difference to all the other computational methods every COMPU-SCALE will be applied including the bit MASK. Therefore it is allowed for this type of COMPU-METHOD, that COMPU-SCALES overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the COMPU-SCALE elements. **Tags:** xml.sequenceOffset=35
shortLabel	Identifier	0..1	ref	This element specifies a short name for the particular scale. The name can for example be used to derive a programming language identifier. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
symbol	CIdentifier	0..1	ref	The symbol, if provided, is used by code generators to get a C identifier for the CompuScale. The name will be used as is for the code generation, therefore it needs to be unique within the generation context. **Tags:** xml.sequenceOffset=25
upperLimit	Limit	0..1	aggr	This element specifies the upper limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=50

Table 5.62: CompuScale

Class	**CompuScales**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to stepwise express a computation method.			
Base	ARObject,CompuContent			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
compuScale (ordered)	CompuScale	*	aggr	This represents one scale within the compu method. Note it is variation in oder to support bluprints of enumerations. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=BlueprintDerivationTime xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.63: CompuScales

Class	**CompuScaleContents (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This abstract meta-class represents the content of one particular scale.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 5.64: CompuScaleContents

Class	**CompuRationalCoeffs**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the ability to express a rational function by specifying the coefficients of nominator and denominator.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
compuDenominator	CompuNominatorDenominator	1	aggr	This is the denominator of the expression. **Tags:** xml.sequenceOffset=30
compuNumerator	CompuNominatorDenominator	1	aggr	This is the numerator of the rational expression. **Tags:** xml.sequenceOffset=20

Table 5.65: CompuRationalCoeffs

Class	**CompuConst**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the fact that the value of a computation method scale is constant.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
compuConstContentType	CompuConstContent	1	aggr	This is the actual content of the constant compu method scale. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=10; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.66: CompuConst

For a detailed description of `compuMethods`, please refer to the *ASAM MCD 2 Harmonized Data Objects*.

ASAM Category	*Meaning*	*Specific properties*
IDENTICAL	This `CompuMethod` just hands over the internal value with an optional unit.	Only the base elements are allowed and `unit`, `physConstr` and `internalConstr` are optional. This is the simplest type of a `CompuMethod`.
LINEAR	A linear conversion can be performed in two steps: The internal value is multiplied with a factor; after that, an offset is added to the result of the multiplication.	Exactly one `CompuScale`, with two `v` in `compuNominator` and one `v` in `compuDenominator`.
SCALE_LINEAR	Used for a piecewise linear conversion	more than one `compuScale` can be defined. Additionally there have to be the `upperLimit` and `lowerLimit` elements which define the region of validity for the linear function. The boundaries of the regions shall not overlap.

ASAM Category	*Meaning*	*Specific properties*
SCALE_LINEAR_AND_TEXTTABLE	Used for piecewise definition of one linear and several texttable scales.	Properties depend on the used scale function. For details see definition of SCALE_LINEAR and TEXTTABLE. The scales shall each provide lowerLimit and upperLimit definitions.
RAT_FUNC	The rational function type is similar to the linear type without the restrictions for the compuNumerators and compuDenominators.	It can have as many v elements as needed for the rational function. The sequence of the values *v* carries the information for the exponents, that means the first *v* is the coefficient for x0, the second *v* is the coefficient for x1, etc. With this sequence the values of the exponents can be entirely represented. A rational function is only applicable for conversions in the direction that it is defined for, i.e. the automatic calculation of the inverse function is not supported by the MCD system.
SCALE_RAT_FUNC	Used for piecewise defined rational conversion.	
SCALE_RATIONAL_AND_TEXTTABLE	Used for piecewise definition of one rational and several texttable scales.	Properties depend on the used scale function. For details see definition of SCALE_RAT_FUNC and TEXTTABLE. The scales shall each provide lowerLimit and upperLimit definitions.
TEXTTABLE	The type TEXTTABLE is used for transformations of the internal value into textual elements.	**[constr_1134] Allowed structure of TEXTTABLE** ⌈ physConstr is not allowed. compuInternalToPhys shall exist with compuScales consisting of upperLimit and lowerLimit. ⌋ The result is placed in the vt member of compuConst. The compuDefaultValue is optional. If the reverse calculation is needed then for each scale the compuInverseValue can be used to define the reverse calculation result. If no inverse value is explicitly defined then the smallest possible value of the scale will be used as result of the reverse calculation.
TAB_NOINTP	Similar to TEXTTABLE but for numerical values.	The values per scale are defined in compuConst.

ASAM Category	*Meaning*	*Specific properties*
BITFIELD_TEXTTABLE	Similar to TEXTTABLE but for bit fields	BITFIELD_TEXTTABLE is derived from TEXTTABLE. The main difference is that TEXTTABLE results to a single value while BITFIELD_TEXTTABLE results to a concatenated value set. **[constr_1135] Limit of `vt` in BITFIELD_TEXTTABLE** ⌈ The separator is "|" and is forbidden in `vt` therefore. ⌋ In difference to all the other computational methods **every** `CompuScale` will be applied including the bit mask specified in `Mask`. Therefore it is allowed for this type of `CompuMethod`, that `CompuScale`s overlap. To calculate the string reverse to a value, the string has to be split and the according value for each substring has to be summed up. The sum is finally transmitted. The processing has to be done in order of the `CompuScale` elements.

Table 5.67: ASAM compuMethod

[TPS_SWCT_1429] [constr_1135] only applies for `BITFIELD_TEXTTABLE` ⌈ Note that [constr_1135] only applies for `BITFIELD_TEXTTABLE`. It does **not** apply to the definition of `vt` in the context of an `ApplicationValueSpecification`. ⌋

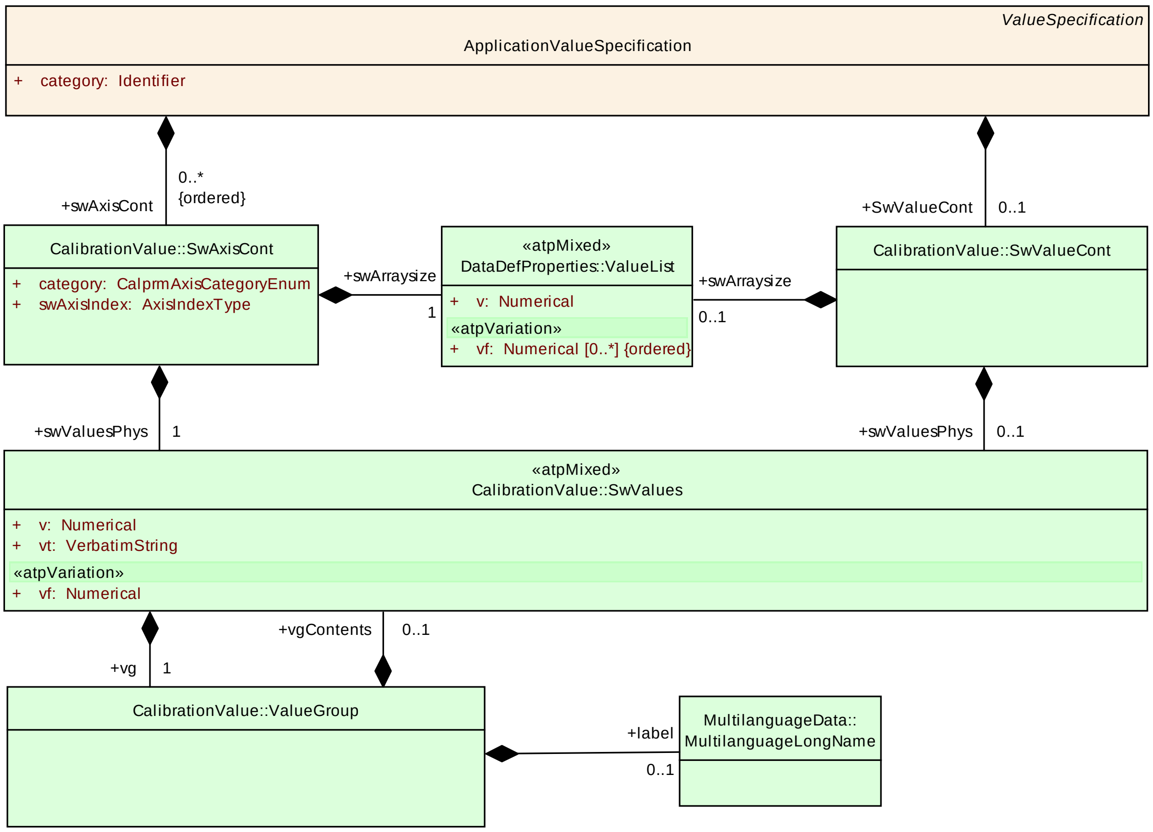

Figure 5.36: Definition of an `ApplicationValueSpecification`

Class	**CompuScaleRationalFormula**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the fact that the computation in this scale is represented as rational term.			
Base	ARObject,CompuScaleContents			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuRationalCoeffs	CompuRationalCoeffs	1	aggr	This specifies the coefficients of the rational fomula. **Tags:** xml.sequenceOffset=110

Table 5.68: CompuScaleRationalFormula

Class	**CompuScaleConstantContents**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This meta-class represents the fact that a particular scale of the computation method is constant.			
Base	ARObject,CompuScaleContents			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
compuConst	CompuConst	1	aggr	This represents the fact that the scale is a constant. The use case is mainly a non interplolated scale. It is a simplification of the fact that a constant scale can also be expressed as Rational Function of oder 0. **Tags:** xml.sequenceOffset=90

Table 5.69: CompuScaleConstantContents

Class	**CompuNominatorDenominator**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::ComputationMethod			
Note	This class represents the ability to express a polynomial either as Nominator or as Denominator.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
v (ordered)	Numerical	*	attr	this is the list of polynomial factors. Note that the first vf represents the power=0. The polynomial is v * xˆ 0 + v * xˆ 1 ... **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.roleElement=true; xml.roleWrapperElement=false; xml.sequenceOffset=20; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.70: CompuNominatorDenominator

5.5.1.1 Example for Enumeration

The following example illustrates how an enumeration is specified using CompuMethod.

Listing 5.6: example for enumeration

```
<COMPU-METHOD>
  <SHORT-NAME>boolean</SHORT-NAME>
  <CATEGORY>TEXTTABLE</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>false</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>true</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

5.5.1.2 Example for Linear Conversion

The following examples illustrates how a linear conversion is specified using CompuMethod.

$F_{[kmh]} = 30_{[kmh]} + 2_{[kmh]} * x$

Listing 5.7: example for linear CompuMethod

```
<COMPU-METHOD>
  <SHORT-NAME>linear</SHORT-NAME>
  <CATEGORY>LINEAR</CATEGORY>
  <UNIT-REF DEST="UNIT">kmh</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>30</V>
            <V>2</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
```

```
          </COMPU-RATIONAL-COEFFS>
        </COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
```

5.5.1.3 Example for Linear Conversion with texttable

The following example illustrates how a linear conversion with a texttable is specified using CompuMethod.

Listing 5.8: example for linear and texttable CompuMethod

```
<COMPU-METHOD>
  <SHORT-NAME>linear</SHORT-NAME>
  <CATEGORY>SCALE_LINEAR_AND_TEXTTABLE</CATEGORY>
  <UNIT-REF DEST="UNIT">kmh</UNIT-REF>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">300</UPPER-LIMIT>
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>30</V>
            <V>2</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>1</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">350</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">350</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>SensorError</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">351</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">351</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>SignalNotAvailable</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-INTERNAL-TO-PHYS>
</COMPU-METHOD>
```

5.5.1.4 Example for conversion specified by a rational function

The semantics of rational function is:

$$Internal = \frac{v_0 * phys^0 + v_1 * phys^1 + v_2 * phys^2 + ...}{v_0 * phys^0 + v_1^* phys^1 + v_2 * phys^2 + ...}$$

The following example illustrates a reciprocal conversion.

$$I = \frac{1000}{60 + 2_{[K-1]} * P_{[K]}}$$

Listing 5.9: example for rational CompuMethod

```
<COMPU-METHOD>
  <SHORT-NAME>rational</SHORT-NAME>
  <CATEGORY>RAT_FUNC</CATEGORY>
  <UNIT-REF DEST="UNIT">Kelvin</UNIT-REF>
  <COMPU-PHYS-TO-INTERNAL>
    <COMPU-SCALES>
      <COMPU-SCALE>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">-29</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="INFINITE" />
        <COMPU-RATIONAL-COEFFS>
          <COMPU-NUMERATOR>
            <V>1000</V>
          </COMPU-NUMERATOR>
          <COMPU-DENOMINATOR>
            <V>60</V>
            <V>2</V>
          </COMPU-DENOMINATOR>
        </COMPU-RATIONAL-COEFFS>
      </COMPU-SCALE>
    </COMPU-SCALES>
  </COMPU-PHYS-TO-INTERNAL>
</COMPU-METHOD>
```

5.5.1.5 Example for BITFIELD_TEXTTABLE

The following example shows how a BITFIELD_TEXTTABLE can be used to assign a special meaning to each bit of an AutosarDataType of category VALUE:

Bit 0	front left	0(0) = no, 1(1) = yes
Bit 1	front right	0(0) = no, 1(2) = yes
Bit 2	rear left	0(0) = no, 1(4) = yes
Bit 3	rear right	0(0) = no, 1(8) = yes
Bit 4-5	problem	00(0) = flat tire 01(16) = low pressure 10(32) = unbalanced 11(48) = unknown
All Bits	error	11111111 = invalid value

Table 5.71: Example Bitfield

Note that this example is somehow tricky. Bit 6+7 are not used for valid data, but are part of the mask. By this the error can safely be masked out.

Internal: 28

```
28 = 0b0001_1100
Bit    7654 3210
```

Physical:

"problem = low pressure | rear right = yes | rear left = yes | front right = no | front left = no"

Listing 5.10: example for bit field text table CompuMethod

```
<COMPU-METHOD>
  <SHORT-NAME>Texttable</SHORT-NAME>
  <CATEGORY>BITFIELD_TEXTTABLE</CATEGORY>
  <COMPU-INTERNAL-TO-PHYS>
    <COMPU-SCALES>
      <!-- problem -->
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>flat tire</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">16</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">16</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>low pressure</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">32</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">32</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>unbalanced</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
        <SHORT-LABEL>problem</SHORT-LABEL>
        <MASK>0b11110000</MASK>
        <LOWER-LIMIT INTERVAL-TYPE="CLOSED">48</LOWER-LIMIT>
        <UPPER-LIMIT INTERVAL-TYPE="CLOSED">48</UPPER-LIMIT>
        <COMPU-CONST>
          <VT>unknown</VT>
        </COMPU-CONST>
      </COMPU-SCALE>
      <COMPU-SCALE>
```

```
    <SHORT-LABEL>problem</SHORT-LABEL>
    <MASK>0b11111111</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">255</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">255</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>invalid</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
  <!-- rear right -->
  <COMPU-SCALE>
    <SHORT-LABEL>rearRight</SHORT-LABEL>
    <MASK>0b11001000</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>no</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
  <COMPU-SCALE>
    <SHORT-LABEL>rearRight</SHORT-LABEL>
    <MASK>0b11001000</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">8</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">8</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>yes</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
  <!-- rear left -->
  <COMPU-SCALE>
    <SHORT-LABEL>rearLeft</SHORT-LABEL>
    <MASK>0b11000100</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>no</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
  <COMPU-SCALE>
    <SHORT-LABEL>rearLeft</SHORT-LABEL>
    <MASK>0b11000100</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">4</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">4</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>yes</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
  <!-- front right -->
  <COMPU-SCALE>
    <SHORT-LABEL>frontRight</SHORT-LABEL>
    <MASK>0b11000010</MASK>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
    <COMPU-CONST>
      <VT>no</VT>
    </COMPU-CONST>
  </COMPU-SCALE>
```

```
<COMPU-SCALE>
  <SHORT-LABEL>frontRight</SHORT-LABEL>
  <MASK>0b11000010</MASK>
  <LOWER-LIMIT INTERVAL-TYPE="CLOSED">2</LOWER-LIMIT>
  <UPPER-LIMIT INTERVAL-TYPE="CLOSED">2</UPPER-LIMIT>
  <COMPU-CONST>
    <VT>yes</VT>
  </COMPU-CONST>
</COMPU-SCALE>
<!-- front left -->
<COMPU-SCALE>
  <SHORT-LABEL>frontLeft</SHORT-LABEL>
  <MASK>0b11000001</MASK>
  <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
  <UPPER-LIMIT INTERVAL-TYPE="CLOSED">0</UPPER-LIMIT>
  <COMPU-CONST>
    <VT>no</VT>
  </COMPU-CONST>
</COMPU-SCALE>
<COMPU-SCALE>
  <SHORT-LABEL>frontLeft</SHORT-LABEL>
  <MASK>0b11000001</MASK>
  <LOWER-LIMIT INTERVAL-TYPE="CLOSED">1</LOWER-LIMIT>
  <UPPER-LIMIT INTERVAL-TYPE="CLOSED">1</UPPER-LIMIT>
  <COMPU-CONST>
    <VT>yes</VT>
  </COMPU-CONST>
</COMPU-SCALE>
      </COMPU-SCALES>
    </COMPU-INTERNAL-TO-PHYS>
  </COMPU-METHOD>
```

Note that a constraint applies concerning the values within a `CompuMethod` of category `TextTable`. It it not allowed that the same textual value appears more than one time in the table, i.e. “error”, “OK”, “error” is considered a violation of constraint [constr_1133].

[constr_1133] Values shall be unique ⌈ In a `CompuMethod` of category `TextTable` the values of all involved `Vt` within the collection of `CompuScale`s shall be unique. ⌋

5.5.2 Physical Units, Physical Dimensions and Unit Groups

[TPS_SWCT_1285] Physical dimension ⌈ Another important part of the semantics associated with a data type is its physical dimension. Units are used to augment the value with additional information like *m/s* or *liter*. This is necessary for a correct interpretation of the physical value for input and output processes.

The conversion of values into other units like *km/h* into *miles/h* is also possible. Therefore the unit involves information about its physical dimensions. ⌋

[TPS_SWCT_1056] Physical dimension ⌈ The substructure of physical dimensions defines all used quantities in the SI-System[8] (e.g. velocity as length/time corresponds to m/s). ⌋*(RS_SWCT_2100)*

[TPS_SWCT_1057] Unit references one physical dimension ⌈ The unit references one physical dimension. If the physical dimensions of two units are identical, a conversion between them is basically possible. ⌋*(RS_SWCT_2100)*

[TPS_SWCT_1058] `UnitGroup` ⌈ The `UnitGroup`s determine if such a conversion is appropriate. ⌋*(RS_SWCT_2100)*

Figure 5.37 depicts the concept how units are defined.

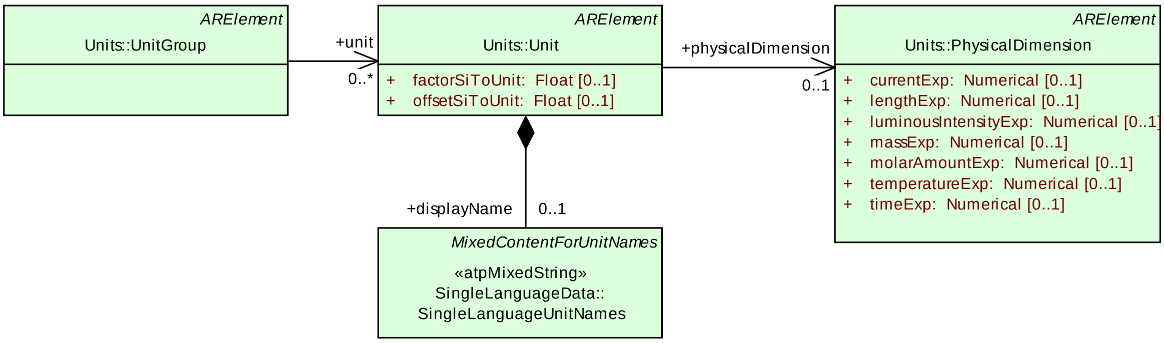

Figure 5.37: Definition of SI based units

For a detailed description of these elements please refer to the [22]. Standard units are already predefined for AUTOSAR in form of a description file.

<table>
<tr><td>Class</td><td colspan="4">Unit</td></tr>
<tr><td>Package</td><td colspan="4">M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units</td></tr>
<tr><td>Note</td><td colspan="4">This is a physical measurement unit. All units that might be defined should stem from SI units. In order to convert one unit into another factor and offset are defined. For the calculation from SI-unit to the defined unit the factor (factorSiToUnit) and the offset (offsetSiToUnit) are applied:

unit = siUnit * factorSiToUnit + offsetSiToUnit

For the calculation from a unit to SI-unit the reciprocal of the factor (factorSiToUnit) and the negation of the offset (offsetSiToUnit) are applied:

siUnit = (unit - offsetSiToUnit) / factorSiToUnit

Tags: atp.recommendedPackage=Units</td></tr>
<tr><td>Base</td><td colspan="4">ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable</td></tr>
<tr><td>Attribute</td><td>Datatype</td><td>Mul.</td><td>Kind</td><td>Note</td></tr>
</table>

[8]For the definition of what SI units are, see http://physics.nist.gov/cuu/Units/

Attribute	Datatype	Mul.	Kind	Note
displayName	SingleLanguageUnitNames	0..1	aggr	This specifies how the unit shall be displayed in documents or in user interfaces of tools.The displayName corresponds to the Unit.Display in an ASAM MCD-2MC file. **Tags:** xml.sequenceOffset=20
factorSiToUnit	Float	0..1	attr	This is the factor for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=30
offsetSiToUnit	Float	0..1	attr	This is the offset for the conversion from and to siUnits. **Tags:** xml.sequenceOffset=40
physicalDimension	PhysicalDimension	0..1	ref	This association represents the physical dimension to which the unit belongs to. Note that only values with units of the same physical dimensions might be converted. **Tags:** xml.sequenceOffset=50

Table 5.72: Unit

[TPS_SWCT_1059] Exponent for each of the seven fundamental dimensions ⌈ For basing a new unit directly upon SI units an exponent for each of the seven fundamental dimensions and its corresponding SI unit needs to be specified. ⌋*(RS_SWCT_2100)*

[TPS_SWCT_1060] Negative exponents ⌈ Negative exponents are allowed. ⌋*(RS_SWCT_2100)*

Note that quantities like "%" are not derived from SI units and therefore have no association to a physical dimension.

Class	PhysicalDimension			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units			
Note	This class represents a physical dimension. If the physical dimension of two units is identical, then a conversion between them is possible. The conversion between units is related to the definition of the physical dimension. Note that the equivalence of the exponents does not per se define the convertibility. For example Energy and Torque share the same exponents (Nm). Please note further the the value of an exponent does not necessarily have to be an integer number. It is also possible that the value yields a rational number, e.g. to compute the square root of a given physical quantity. In this case the exponent value would be a rational number where the numerator value is 1 and the denominator value is 2. **Tags:** atp.recommendedPackage=PhysicalDimensions			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
currentExp	Numerical	0..1	attr	This attribute represents the exponent of the physical dimension "electric current". **Tags:** xml.sequenceOffset=50
lengthExp	Numerical	0..1	attr	The exponent of the physical dimension "length". **Tags:** xml.sequenceOffset=20
luminousIntensityExp	Numerical	0..1	attr	The exponent of the physical dimension "luminous intensity". **Tags:** xml.sequenceOffset=80
massExp	Numerical	0..1	attr	The exponent of the physical dimension "mass". **Tags:** xml.sequenceOffset=30
molarAmountExp	Numerical	0..1	attr	The exponent of the physical dimension "quantity of substance". **Tags:** xml.sequenceOffset=70
temperatureExp	Numerical	0..1	attr	The exponent of the physical dimension "temperature". **Tags:** xml.sequenceOffset=60
timeExp	Numerical	0..1	attr	The exponent of the physical dimension "time". **Tags:** xml.sequenceOffset=40

Table 5.73: PhysicalDimension

[constr_1026] Compatibility of `Units` ⌈ For data types or prototypes, units should be referenced from within the associated `CompuMethod`. But if it is referenced from within `SwDataDefProps` (for exceptional use cases) it shall be compatible to the ones referenced from the referred `CompuMethod`. ⌋

Class	UnitGroup			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Units			
Note	This meta-class represents the ability to specify a logical grouping of units.The category denotes the unit system that the referenced units are associated to. In this way, e.g. country-specific unit systems (CATEGORY="COUNTRY") can be defined as well as specific unit systems for certain application domains. In the same way a group of equivalent units, can be defined which are used in different countries, by setting CATEGORY="EQUIV_UNITS". KmPerHour and MilesPerHour could such be combined to one group named "vehicle_speed". The unit MeterPerSec would not belong to this group because it is normally not used for vehicle speed. But all of the mentioned units could be combined to one group named "speed". Note that the UnitGroup does not ensure the physical compliance of the units. This is maintained by the physical dimension. **Tags:** atp.recommendedPackage=UnitGroups			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	Datatype	Mul.	Kind	Note
unit	Unit	*	ref	This represents one particular unit in the UnitGroup. **Tags:** xml.sequenceOffset=20

Table 5.74: UnitGroup

[TPS_SWCT_1068] `Units` can be grouped with the help of `UnitGroup` ⌈ `Units` can be grouped with the help of `UnitGroup`. This grouping is intended as a logical grouping which allows for example an MCD (Measurement Calibration Diagnostic) device to present different unit systems to the user such that he can chose the most appropriate one. ⌋*(RS_SWCT_2100)*

According to [22] the following two `category`s are recommended:

- `COUNTRY` collects units which are common in a particular country, denoted by the `shortName` / `longName` of the `UnitGroup`
- `EQUIV_UNITS` define a group of equivalent units, which are used for example in different countries.

Additional categories may be mutually agreed between the stakeholders.

In the example shown in Figure 5.38, `Units` are classified by country and use.

[TPS_SWCT_1061] Conversion of units ⌈ If a unit has to be converted according to the chosen country code the `physicalDimension` of both units shall be the same. If another unit shares the same `UnitGruop` with a `category` of `EQUIV_UNITS` it is preferred as target of the conversion. ⌋*(RS_SWCT_2100)*

Assume "MilesPerHour" should be converted to a European unit: Based on the physicalDimension a conversion to "MeterPerSec" as well as "MilesPerHour" is possible. In this case "KmPerHour" is preferred because "MilesPerHour" and "KmPerHour" are both members of the UnitGroup named "VehicleSpeed". In contrast to this "MeterPerSec" is not considered as appropriate for "VehicleSpeed".

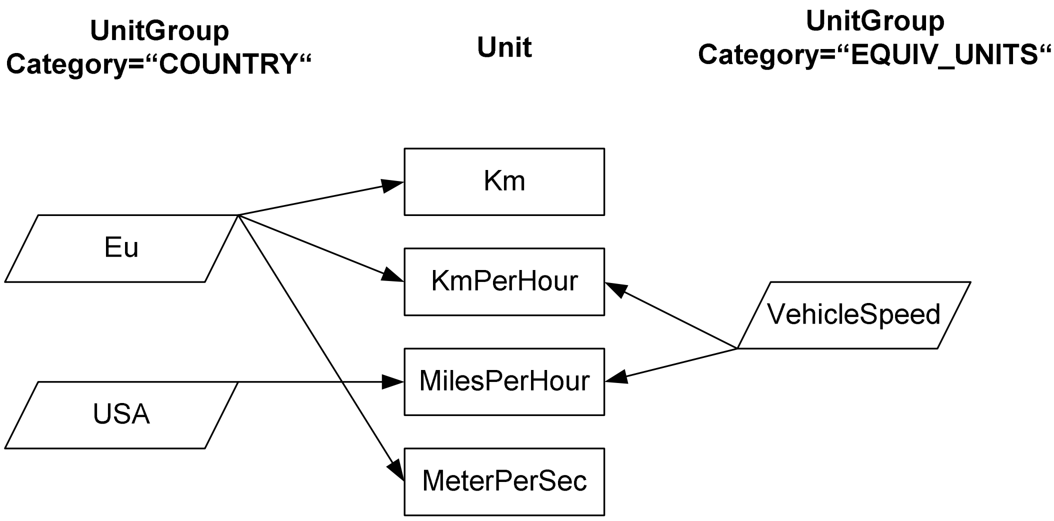

Figure 5.38: Example for units and unit groups

5.5.3 Data Constraints

Section 5.2.4.1 already shows an example on how to define constraints for the physical range of a data type, see Figure 5.4.

[TPS_SWCT_1286] `DataConstr` ⌈ In general, the meta-class DataConstr can be aggregated (via SwDataDefProps.dataConstr to define various constraints for the possible values of a data type. This includes limits for the physical and internal range, as well as special constraints (monotony) for the setup of axis definition. ⌋

Figure 5.39 and the following class tables show the meta-classes involved in the definition of constraints.

A more detailed documentation of these meta-classes can be found in in [22]. As refinement of these definitions, the following values apply for constraintLevel:

[constr_2561] Application of `DataConstrRule.constrLevel` ⌈ DataConstrRule.constrLevel is limited to

0: This represents so called "hard limits". They shall always be specified.

1: This represents so called "soft limits". Soft limits may be violated after confirmation by the user of an MCD-System.

Other values may exist, but the semantics is outside of the AUTOSAR scope.

⌋

[TPS_SWCT_1287] Standard limits and extended limits in the ASAM-MCD2 (ASAP2) specification ⌈ The ASAM-MCD2 (ASAP2) specification [23] defines standard limits and extended limits. If extended limits exist, the standard limits may be violated upon user confirmation. Note that in consequence, of this definition, the following approach applies for A2L generation:

- If only one `DataConstrRule` with `constrLevel` set to **0** is specified, it represents the standard limits in A2L. No extended limits are generated.
- If two `DataConstrRule` exist, then:
  - the one with `constrLevel` set to **0** represents to the extended limits
  - the one with `constrLevel` set to **1** represents to the standard limits

Note that even if this is somehow counterintuitive (since the one with `constrLevel` set to 0 changes its role), it matches the best to the definitions in ASAM-MCD2. ⌋

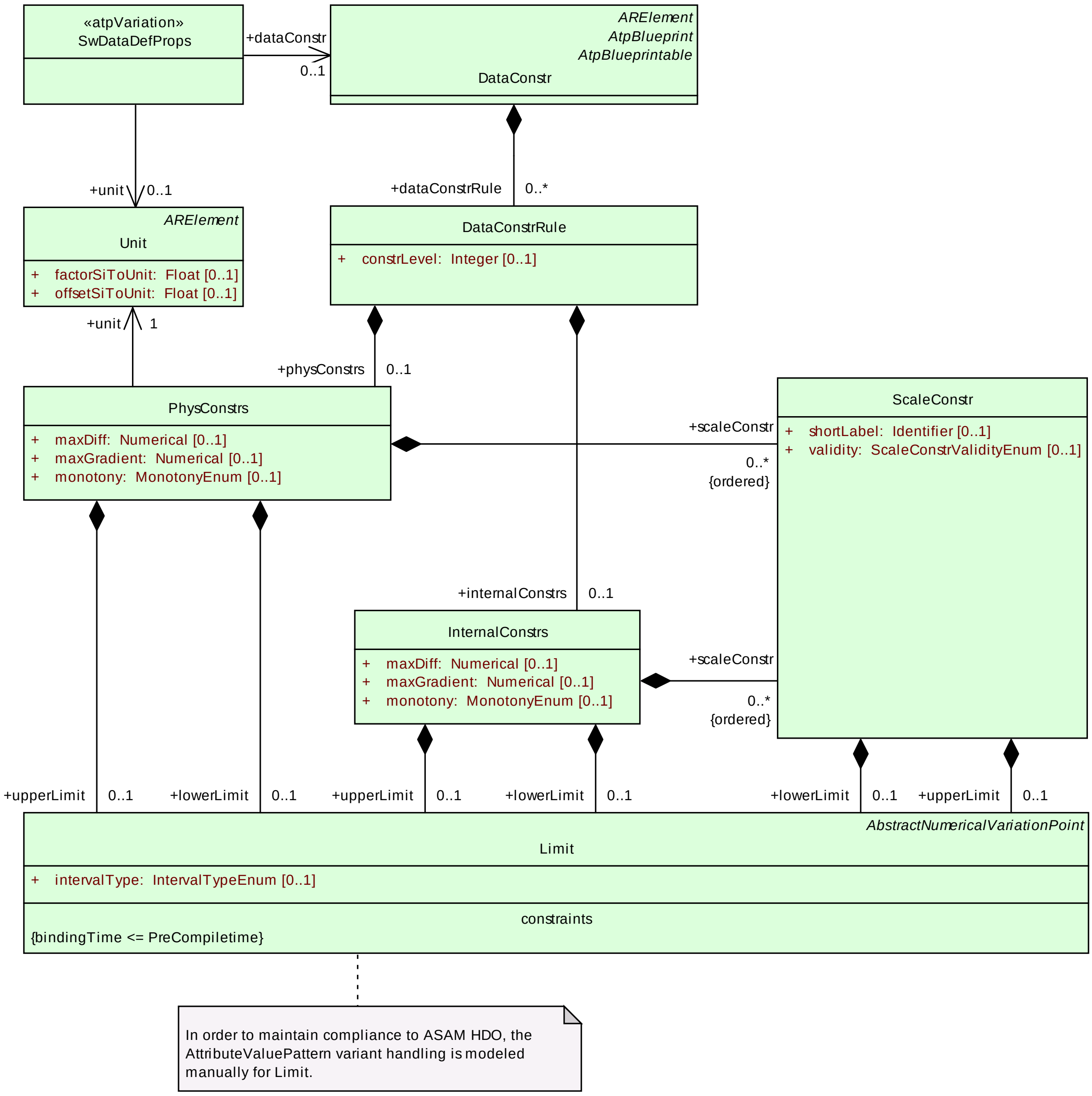

Figure 5.39: Meta-model for defining Data Constraints

Class	**DataConstr**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints on data. **Tags:** atp.recommendedPackage=DataConstrs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataConstr Rule	DataConstrRule	*	aggr	This is one particular rule within the data constraints. **Tags:** xml.roleElement=true; xml.roleWrapper Element=true; xml.sequenceOffset=30; xml.type Element=false; xml.typeWrapperElement=false

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

Table 5.75: DataConstr

Class	**DataConstrRule**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to express one specific data constraint rule.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
constrLevel	Integer	0..1	attr	This attribute describes the category of a constraint. One of its functions is in the area of constraint violation, where it can be used from a certain level, to produce error messages. The lower the level, the more stringent the check. Used to distinguish hard or soft limits; **Tags:** xml.sequenceOffset=20
internalConstrs	InternalConstrs	0..1	aggr	Describes the limitiations applicapble on the internal domain (as opposed to the physical domain). **Tags:** xml.sequenceOffset=40
physConstrs	PhysConstrs	0..1	aggr	Describes the limitiations applicapble on the physical domain (as opposed to the internal domain). **Tags:** xml.sequenceOffset=30

Table 5.76: DataConstrRule

Class	**PhysConstrs**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to express physical constraints. Therefore it has (in opposite to InternalConstrs) a reference to a Unit.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
lowerLimit	Limit	0..1	aggr	This element specifies the lower limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=20
maxDiff	Numerical	0..1	attr	Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. **Tags:** xml.sequenceOffset=60

Attribute	Datatype	Mul.	Kind	Note
maxGradient	Numerical	0..1	attr	This element specifies the maximum slope that may be used in curves and maps. **Tags:** xml.sequenceOffset=50
monotony	MonotonyEnum	0..1	attr	This specifies the monotony constraints on the data object. Note that this applies only to curves and maps. **Tags:** xml.sequenceOffset=70
scaleConstr (ordered)	ScaleConstr	*	aggr	This is one particular scale in which contributes to the data constraints. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false
unit	Unit	1	ref	This is the unit to which the physical contraints relate to. In particular it is the physical unit of the specified limits. **Tags:** xml.sequenceOffset=80
upperLimit	Limit	0..1	aggr	This element specifies the upper limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=30

Table 5.77: PhysConstrs

Class	**InternalConstrs**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to express internal constraints.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
lowerLimit	Limit	0..1	aggr	This element specifies the lower limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=20
maxDiff	Numerical	0..1	attr	Maximum difference that is permitted between two consecutive values if the constraint is applied to an axis. **Tags:** xml.sequenceOffset=60
maxGradient	Numerical	0..1	attr	This element specifies the maximum slope that may be used in maps and curves. **Tags:** xml.sequenceOffset=50

Attribute	Datatype	Mul.	Kind	Note
monotony	MonotonyEnum	0..1	attr	This element specifies the monotony characteristics of the current internal or physical limits. The following table shows the monotony characteristics which are to be filled through the corresponding values. If the element has no contents or if it is omitted, "no-monotony" is the default content. **Tags:** xml.sequenceOffset=70
scaleConstr (ordered)	ScaleConstr	*	aggr	This is one particular scale in which contributes to the data constraints. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=40; xml.typeElement=false; xml.typeWrapperElement=false
upperLimit	Limit	0..1	aggr	This element specifies the upper limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=30

Table 5.78: InternalConstrs

Class	**ScaleConstr**			
Package	M2::AUTOSARTemplates::CommonStructure::GlobalConstraints			
Note	This meta-class represents the ability to specify constraints as a list of intervals (called scales).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
desc	MultiLanguageOverviewParagraph	0..1	aggr	<desc> represents a general but brief description of the object in question. **Tags:** xml.sequenceOffset=30
lowerLimit	Limit	0..1	aggr	This element specifies the lower limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=40
shortLabel	Identifier	0..1	ref	This element specifies a short name for the scaleConstr. This can for example be used to create more specific messages of a constraint checker. The constraints cannot be associated in the meta-model, therefore shortLabel is somehow a substitute for shortName. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
upperLimit	Limit	0..1	aggr	This element specifies the upper limit of a closed, half-open or open interval. It can also be set to infinity by setting the attribute INTERVAL-TYPE to INFINITE. No value has to be set in the case of an infinite interval. **Tags:** xml.sequenceOffset=50
validity	ScaleConstrValidityEnum	0..1	attr	Specifies if the values defined by the scales are considered to be valid. If the attribute is missing then the default value is "VALID". **Tags:** xml.attribute=true

Table 5.79: ScaleConstr

Class	≪**atpMixedString**≫ **Limit**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::LocalConstraints			
Note	This class represents the ability to express a numerical limit. Note that this is in fact a NumericalValuationPoint but has the additional attribute intervalType.			
Base	ARObject,AbstractNumericalVariationPoint,AttributeValueVariationPoint,FormulaExpression,SwSystemconstDependentFormula			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
intervalType	IntervalTypeEnum	0..1	attr	This specifies the type of the interval. If the attribute is missing the interval shall be considered as "CLOSED". **Tags:** xml.attribute=true

Table 5.80: Limit

[TPS_SWCT_1288] Interpretation of `PhysConstrs` and `InternalConstrs` by tools ⌈ `DataConstr` is an `ARElement` which can be reused by several data type specifications. Especially an implementation and an application data type which are mapped to each other, can refer to the same constraints or they can define their own constraints.

To avoid conflicts, in both cases `PhysConstrs` shall be interpreted by tools only with respect to application data types while `InternalConstrs` shall be interpreted only with respect to implementation data types. If either a physical or internal constraint is missing an existing `CompuMethod` can be used to calculate the missing information. ⌋

[TPS_SWCT_1289] Semantics of `Limit` ⌈ Technically, a `Limit` specifies a boundary of the interval of valid values for a given context (i.e. a data type). Please note that the boundary might or might not be part of the interval itself, i.e. the interval might be open or closed. From the formal point of view, the range represents all real numbers defined by:

$$range \;=\; \{x \in \Re \,\|\, lowerLimit.value < x < upperLimit.value\}$$

$$\cup\{lowerLimit.value \parallel lowerLimit.intervalType == \text{“}CLOSED''\}$$
$$\cup\{upperLimit.value \parallel upperLimit.intervalType == \text{“}CLOSED''\}$$

⌋

Enumeration	**IntervalTypeEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::LocalConstraints
Note	This enumerator specifies the type of an interval.
Literal	***Description***
closed	The area is limited by the value given. The value itself is included.
infinite	The area is unlimited. (- or + depending on lower or higher Limit). Note that in this case the numerical value specified in the limit has no relevance.
open	The area is limited by the value given. The value itself is not included.

Table 5.81: IntervalTypeEnum

5.5.4 Addressing Methods

In an ECU there might be various methods to access a particular object (e.g measurement or calibration parameter) according to a given address. This variety might come from different kind of memory (near, far, ...) but also from indirections which are introduced by the compiler.

[TPS_SWCT_1290] `SwAddrMethod` ⌈ In order to allow a measurement and calibration system to access such objects `SwAddrMethod`s are specified. Another purpose of this feature is to support the definition of abstract memory sections, i.e. to specify which variables shall be put together in the same sections in case of generated code (especially for data allocated by the RTE).

`SwAddrMethod` will be used to group data, for example, to cover the fact that sometimes it is required that one or more calibration parameters out of the overall collection of calibration parameters of a `SwComponentPrototype` respectively an AUTOSAR software component shall be placed in another memory location than the other parameters of the `SwComponentPrototype` respectively the AUTOSAR software component. ⌋

[TPS_SWCT_1291] Association of `MemorySection` with `SwAddrMethod` ⌈ In `Implementation` the particular `MemorySection` is associated with the `SwAddrMethod`. This association indicates that all objects of the associated addressing method shall be placed in the given memory section. ⌋

[TPS_SWCT_1294] Missing `SwDataDefProps.swAddrMethod` ⌈ If the association `SwDataDefProps.swAddrMethod` is missing the object can be placed anywhere without restriction, e.g. using a default behavior of the RTE generator. Contradicting specifications (e.g. two different component types request different associations for one particular `SwAddrMethod`) shall be flagged as an error. ⌋

[TPS_SWCT_1292] Usage of `SwAddrMethod` in the context of a `DataPrototype`
⌈ Figure 5.40 illustrates the usage of `SwAddrMethod` in the context of a `DataPrototype`. Note that the software component which defines the `DataPrototype` will in general not be the same to which the `Implementation` that actually contains the description of the `MemorySection` belongs.

The reason for this is that the resources for data allocated by the RTE will be described in the `Implementation` of the RTE. The indirection via `SwAddrMethod` makes this possible. ⌋

[TPS_SWCT_1293] RTE Generator has to derive the Memory Allocation Keyword
⌈ Please note that the RTE Generator has to derive the Memory Allocation Keyword used for `RunnableEntity`s and `BswSchedulableEntity`s from the `shortName` of the `SwAddrMethod` only because the alignment defined in `MemorySection` is not known at contract phase. ⌋

[constr_2034] `SwAddrMethod` referenced by `RunnableEntitys` or `BswSchedulableEntitys` ⌈ `RunnableEntity`s and `BswSchedulableEntity`s shall not reference a `SwAddrMethod` which attribute `memoryAllocationKeywordPolicy` is set to `AddrMethodShortNameAndAlignment`. ⌋

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 5.82: SwAddrMethod

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 5.83: 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 5.84: MemorySectionType

Enumeration	**MemoryAllocationKeywordPolicyType**
Package	M2::AUTOSARTemplates::CommonStructure::AuxillaryObjects
Note	Enumeration to specify the name pattern of the Memory Allocation Keyword.
Literal	***Description***
AddrMethod ShortName	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.
AddrMethod ShortName AndAlign-ment	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 5.85: MemoryAllocationKeywordPolicyType

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 5.86: AlignmentType

For more information on the specification of the `MemorySection` refer to [7].

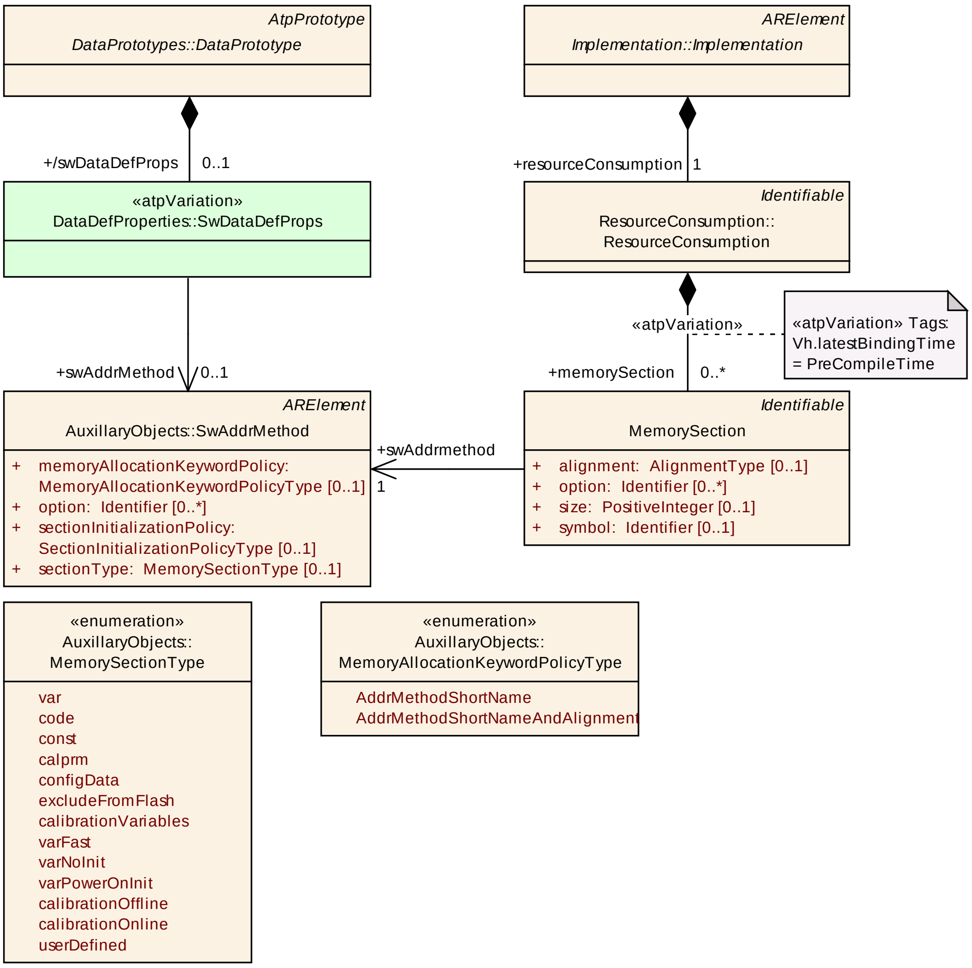

Figure 5.40: Assigning an address method to a memory section

5.5.5 Record Layouts

[TPS_SWCT_1295] **`RecordLayout`** ⌈ The `RecordLayout` describes how data is serialized in the memory of an ECU. This aspect is important with respect to the following aspects:

- to inform a measurement and calibration system how the data is serialized in the memory of an ECU
- to make sure that the software development results in the intended data structures
- to identify the proper interpolation routines

Via the `SwDataDefProps` a record-layout can be associated to a data entity. On the one hand, if the very same serialization approach is used for multiple `ApplicationDataType`s all of these may refer to the same `RecordLayout` even if the size of the data is different. ⌋

5.5.5.1 Specifying Record Layouts

As mentioned above, the purpose of record layout is to specify how an object (e.g. a calibration parameter) is serialized in memory of an ECU. The canonical approach for this is to define nested groups (SwRecordLayoutGroup).

These groups indicate the structure of the corresponding ImplementationDataType. The serialization is then executed by iterating over the axes of a curve, a map, or iterating along a string. The contents of such a record layout group (SwRecordLayoutGroupContent) is a mixture of (thus nested) groups and values (SwRecordLayoutV).

These values refer to particular properties of the object (e.g. value, count, . . .). By application of this pattern, the serialization of any complex object can be specified.

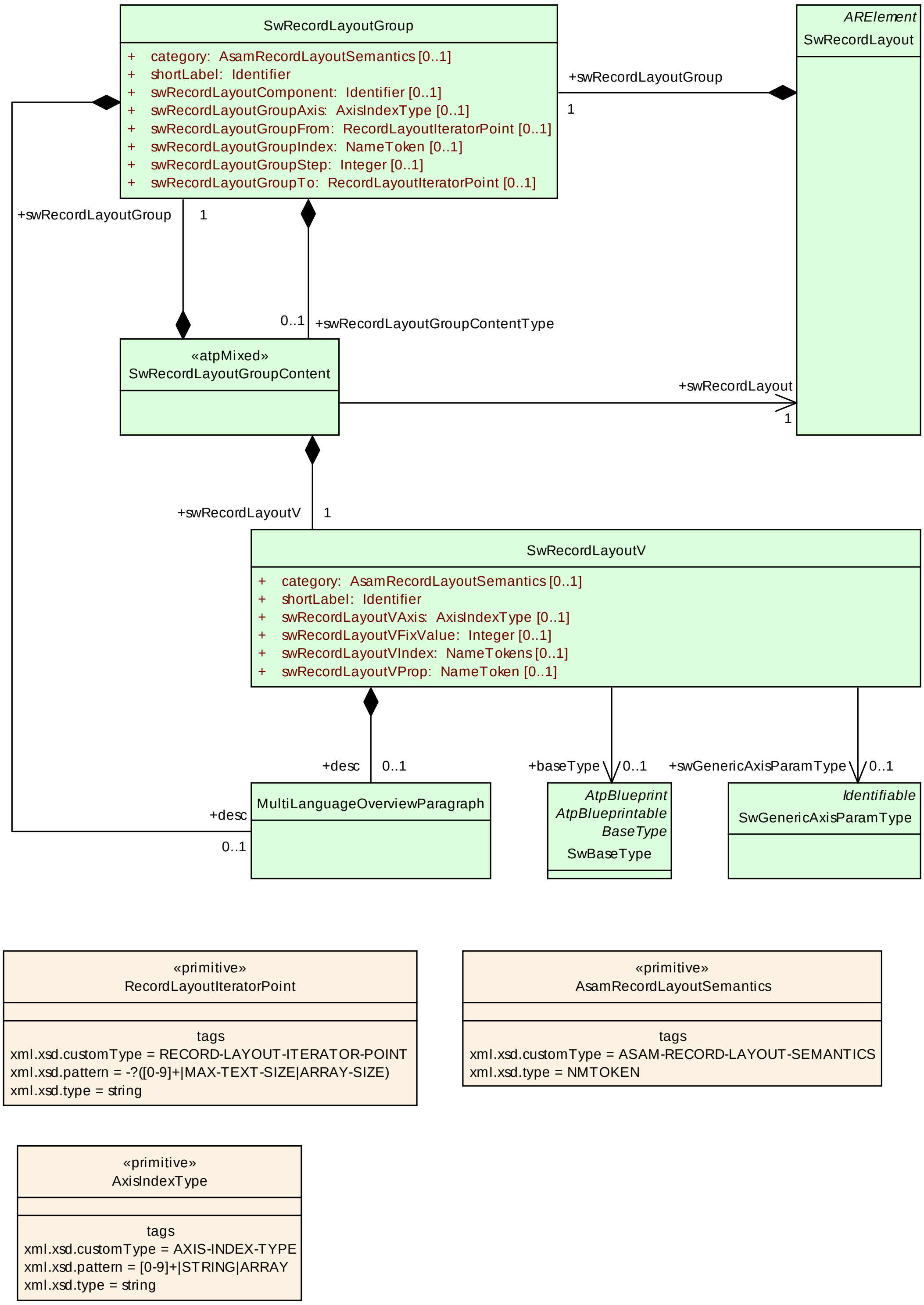

Figure 5.41: Specification of a record layout

Class	**SwRecordLayoutV**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout			
Note	This element specifies which values are stored for the current SwRecordLayoutGroup. If no baseType is present, the SwBaseType referenced initially in the parent SwRecordLayoutGroup is valid. The specification of swRecordLayoutVAxis gives the axis of the values which shall be stored in accordance with the current record layout SwRecordLayoutGroup. In swRecordLayoutVProp one can specify the information which shall be stored.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
desc	MultiLanguageOverviewParagraph	0..1	aggr	this property allows a brief description about the particular record layout value which can help to identify the entry. In depth documentation should go to introduction of the surrounding record layout. **Tags:** xml.sequenceOffset=20
category	AsamRecordLayoutSemantics	0..1	attr	This attribute denotes the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2l keywords. It is possible to express the specifc semantics of A2l recordlayout keywords in swRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. **Tags:** xml.sequenceOffset=5
baseType	SwBaseType	0..1	ref	This allows to refer to a base type in case a specific encoding is intended. If no base type is referred, the base type referenced initially in the corresponding DataPrototype is to be used. **Tags:** xml.sequenceOffset=30
shortLabel	Identifier	1	ref	This attribute specifies a name which can be used e.g. when ECU code is generated from the record layout value. **Tags:** xml.sequenceOffset=3
swGenericAxisParamType	SwGenericAxisParamType	0..1	ref	This association supports the case that a value from a generic axis definition shall be stored. This value is denoted by a particular generic axis parameter type. **Tags:** xml.sequenceOffset=70

Attribute	Datatype	Mul.	Kind	Note
swRecordLayoutVAxis	AxisIndexType	0..1	attr	This attribute gives the index of the axis of which values that are stored in the record. swRecordVIndex refers to the symbolic names of the iterators for which the axis value shall be stored in the recrod. In case of nested iterators (mainly for multidimensional objects) the iterator names are specified as whitespace separated names. These symbolic names relate to swRecordLayoutGroupIndex. The iterators are processed from left to right in such a manner that they symbolize the loop index from the outside to the inside. It is an error if more components are specified than axis are there in the related ApplicationDataType. **Tags:** xml.sequenceOffset=40
swRecordLayoutVFixValue	Integer	0..1	attr	This attribute specifies the filler character for the current record layout, in the form of hex digits. It is also used to specify the fix value for e.g. FIXRIGHTDIFF. **Tags:** xml.sequenceOffset=80
swRecordLayoutVIndex	NameTokens	0..1	attr	The symbolic value for iteration, or the symbolic values separated by white-spaces, refer to the symbolic values given in swRecordLayoutGroupIndex . The iterators are processed from left to right, in such a manner that they symbolize the loop index from the outside to the inside. It is an error if the record layout is referenced by an entity which has less number of axis than index names referenced here. **Tags:** xml.sequenceOffset=60
swRecordLayoutVProp	NameToken	0..1	attr	This attribute describes the kind of values to be stored. More details see below. **Tags:** xml.sequenceOffset=50

Table 5.87: SwRecordLayoutV

Class	**SwRecordLayoutGroup**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout			
Note	Specifies how a record layout is set up. Using SwRecordLayoutGroup it recursively models iterations through axis values. The subelement swRecordLayoutGroupContentType may reference other SwRecordLayouts, SwRecordLayoutVs and SwRecordLayoutGroups for the modeled record layout.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
desc	MultiLanguageOverviewParagraph	0..1	aggr	This property allows a brief description about the particular record layout group which can help to identify the entry. In depth documentation should go to introduction of the surrounding record layout. **Tags:** xml.sequenceOffset=20
category	AsamRecordLayoutSemantics	0..1	attr	This attribute denotes the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2l keywords. It is possible to express the specific semantics of A2l recordlayout keywords in swRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. **Tags:** xml.sequenceOffset=5
shortLabel	Identifier	1	ref	This attribute specifies a name which can be used e.g. when ECU code is generated from the record layout group. **Tags:** xml.sequenceOffset=3
swGenericAxisParamType	SwGenericAxisParamType	0..1	ref	This association allows to specify record layout groups to iterate over generic axis parameters. For example, if the generic axis parameter is an array, the record layout group will iterate over this array. Obviously, the axis referred to by swRecordLayoutGroupAxis must be a generic axis in which the referenced SwGenericAxisType is aggregated. **Tags:** xml.sequenceOffset=50
swRecordLayoutComponent	Identifier	0..1	ref	is used to denote the component to which the group in question applies. Thus, the record layout supports structured objects.This secures independence from the sequence of components, because they can be referred to via name. **Tags:** xml.sequenceOffset=90

Attribute	Datatype	Mul.	Kind	Note
swRecordLayoutGroupAxis	AxisIndexType	0..1	attr	This attribute specifies the iteration axis number for a SwRecordLayoutGroup. The current record layout group then refers exactly to the axis with this number. This means that the values are taken by iterating along the thus referenced axis. **Tags:** xml.sequenceOffset=30
swRecordLayoutGroupContentType	SwRecordLayoutGroupContent	0..1	aggr	This is the contents of the recordLayout which is produced for every step of iteration. **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=100; xml.typeElement=false; xml.typeWrapperElement=false
swRecordLayoutGroupFrom	RecordLayoutIteratorPoint	0..1	attr	This element specifies the iterator index for the point in the axis from which a record layout group is commenced. Negative values are also possible, i.e. the value -4 counts from the fourth value from the end. If this property is missing, the iteration starts with '1'. **Tags:** xml.sequenceOffset=60
swRecordLayoutGroupIndex	NameToken	0..1	attr	This element attributes a symbolic name to the iterator of the superimposed record layout group. This can be referenced as a loop index in contained SwRecordLayoutV elements. **Tags:** xml.sequenceOffset=40
swRecordLayoutGroupStep	Integer	0..1	attr	This property specifies the step width for the iterator index that is used for the current record layout group. Note that negative values are also possible, in case of the starting point is higher than the endpoint. If the property is missing, the step width is "1". **Tags:** xml.sequenceOffset=80
swRecordLayoutGroupTo	RecordLayoutIteratorPoint	0..1	attr	This element specifies the end point for the iteration. Negative values are also possible, i.e. the value -4 counts up to the fourth value from the end. If this property is not there, the iteration ends at "-1" which is the last element. Note that depending on the arraySizeSemantics of SwTextProps the iteration ends at the value specified in swMaxTextSize. **Tags:** xml.sequenceOffset=70

Table 5.88: SwRecordLayoutGroup

[constr_2533] Iteration along output axis is only supported for VALUE and VAL_BLK ⌈ `swRecordLayoutVIndex` in `SwRecordLayoutV` cannot be 0 for any data category other than VALUE and VAL_BLK. ⌋

For CURVE, MAP etc. the iteration shall be performed along the input axis.

Primitive	**AxisIndexType**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout
Note	This type specifies an axis in a curve/map data object. The index satisfies the following convention: • 0 output "axis" • 1 input axis 1 (x input axis e.g. of a curve) • 2 input axis 2 (y input axis e.g. of a map) • 3 input axis 3 (z input axis e.g. of a cuboid) • 4.. 9 etc. The output "axis" provides access to the output value of the parameter. Note that this access is usually performed via an index according to the input axis. In addition to this, the Values STRING and ARRAY support specific iterations. **Tags:** xml.xsd.customType=AXIS-INDEX-TYPE; xml.xsd.pattern=[0-9]+|STRING|ARRAY; xml.xsd.type=string

Table 5.89: AxisIndexType

Primitive	**RecordLayoutIteratorPoint**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout
Note	This primitive denotes a start / endpoint for the iteration of a recordLayoutGroup. It can be an integer or one of the keywords MAX-TEXT-SIZE|ARRAY-SIZE. Note that negative numbers are counted backwards. Therefore e.g. -1 refers to the last value. **Tags:** xml.xsd.customType=RECORD-LAYOUT-ITERATOR-POINT; xml.xsd.pattern=-?([0-9]+|MAX-TEXT-SIZE|ARRAY-SIZE); xml.xsd.type=string

Table 5.90: RecordLayoutIteratorPoint

`swRecordLayoutVProp` in `SwRecordLayoutV` describes the type of values to be stored. The following values for `swRecordLayoutVProp` are permitted:

Property	Description
VALUE	The value of the axis for the current iterator point. This is e.g. the particular point on an input-axis, but also the particular character in a string.
COUNT	The amount of values of the axis
LEFTDIFF	The difference to the previous axis point
RIGHTDIFF	The difference to the next axis point
DIST	The distance value of this axis in case of a fixed axis with distance specification
SHIFT	The shift value of this axis in case of a fixed axis with shift/offset
OFFSET	The offset value of this axis in case of a fixed axis with shift/offset
SOURCE-ADR	The address of the source of this axis (Note that this does not apply to the value axis)

RESULT-ADR	The address of the result for this axis (note that this does not apply to input axis)
ADDRESS	The address of the axis point
FILL	Fill with the hex value specified as contents of `swRecordLayoutFixValue`
FIXLEFTDIFF	Difference between this and a fixed left-hand value specified in `swRecordLayoutFixValue`
FIXRIGHTDIFF	Difference between this and a fixed right-hand value specified in `swRecordLayoutFixValue`

Table 5.91: swRecordLayoutVProp

Figure 5.42 and Figure 5.43 illustrate most of these properties.

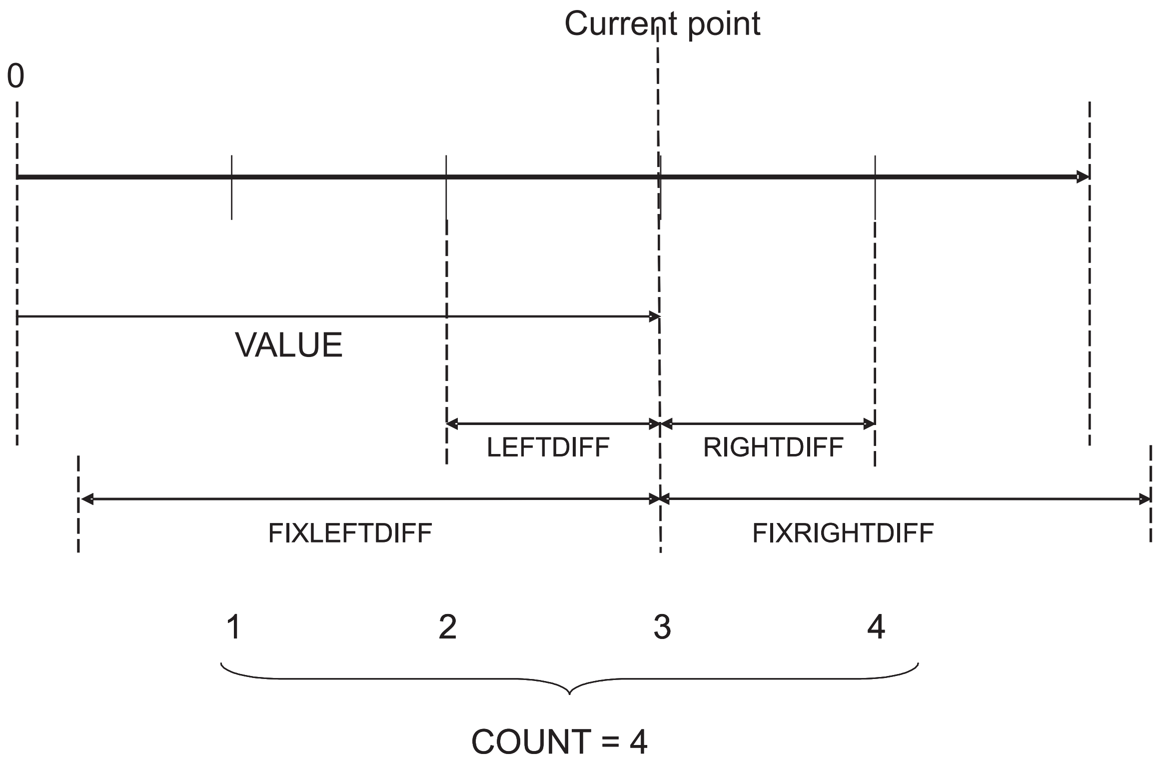

Figure 5.42: Values for swRecordLayoutVProp for individual axis

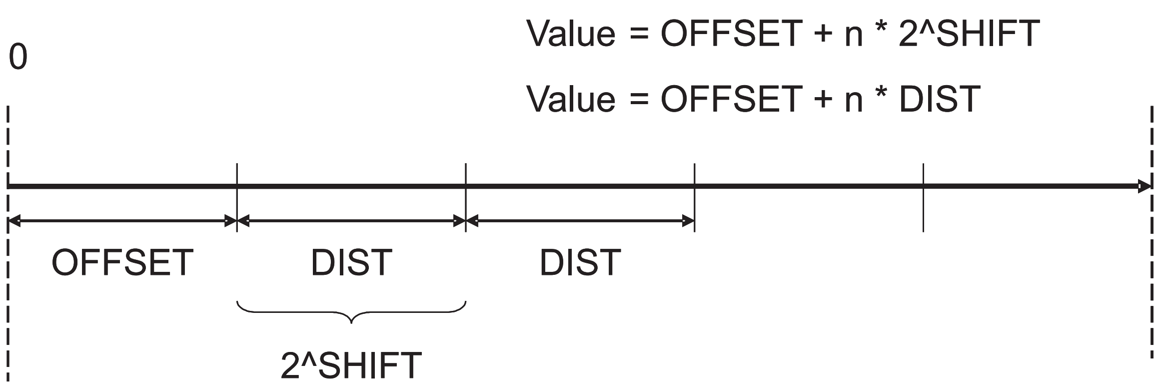

Figure 5.43: Values for swRecordLayoutVProp for fixed axis

[TPS_SWCT_1296] Different approaches of ASAM MCD-2MC and AUTOSAR with respect to RecordLayout ⌈ ASAM MCD-2D specification (also known as A2L, resp. ASAP) uses keywords in record layouts where MSR/AUTOSAR uses the more generic approach specified here. It may happen that this generic approach cannot always be safely mapped to the A2L keywords. Therefore swRecordLayoutV as well as swRecordLayoutGroup can have a category which can assist the conversion to the current A2L format. ⌋

Primitive	**AsamRecordLayoutSemantics**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::RecordLayout
Note	This primitive is used to denote the semantics in particular in terms of the corresponding A2L-Keyword. This is to support the mapping of the more general record layouts in AUTOSAR/MSR to the specific A2L keywords. It is possible to express the specifc semantics of A2l recordlayout keywords in swRecordlayoutGroup but not always vice versa. Therefore the mapping is provided in this optional attribute. It is specified as NMTOKEN to reduce the direct dependency of ASAM an AUTOSAR standards. **Tags:** xml.xsd.customType=ASAM-RECORD-LAYOUT-SEMANTICS; xml.xsd.type=NMTOKEN

Table 5.92: AsamRecordLayoutSemantics

The values for category can, for example, be taken from the ASAM MCD 2D specification provided in [23]. Examples are such as INDEX_INCR, INDEX_DECR, COLUMN_DIR, ROW_DIR, ALTERNATE_WITH_X, ALTERNATE_WITH_Y, ALTERNATE_CURVES. The consistency of these categories with the structure of the SwRecordLayout shall be ensured by the author of the record layout.

Note that there are keywords in A2L bound to a calibration parameter which in MSR/AUTOSAR are represented by the SwRecordLayout (GUARD_RAILS, DEPOSIT etc.).

The following XML fragment provides an example for a SwRecordLayout for a curve. Note that in this case recognizing the patterns represented by the A2L-Keywords (shown in XML-Comment) is pretty straight forward, even if the keywords were not provided in the category.

Listing 5.11: example for RecordLayout of a curve

```
<SW-RECORD-LAYOUT>
  <SHORT-NAME>RecordLayoutCurve</SHORT-NAME>
  <SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-V><!--  SRC_ADDR_X  -->
      <SHORT-LABEL>srcAdr</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-PROP>SOURCE-ADR</SW-RECORD-LAYOUT-V-PROP>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-V><!--  NO_AXIS_PTS_X -->
      <SHORT-LABEL>noOfAxisPts</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-PROP>COUNT</SW-RECORD-LAYOUT-V-PROP>
      <SW-RECORD-LAYOUT-V-INDEX>1</SW-RECORD-LAYOUT-V-INDEX>
```

```
      </SW-RECORD-LAYOUT-V>
      <SW-RECORD-LAYOUT-GROUP><!-- AXIS_PTS_X  -->
        <SHORT-LABEL>xPts</SHORT-LABEL>
        <CATEGORY>AXIS_PTS_X:INDEX_INCR</CATEGORY>
        <SW-RECORD-LAYOUT-GROUP-AXIS>1</SW-RECORD-LAYOUT-GROUP-AXIS>
        <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
        <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
        <SW-RECORD-LAYOUT-V>
          <SHORT-LABEL>xPt</SHORT-LABEL>
          <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS> <!--
             AXIS_PTS_X  -->
          <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
        </SW-RECORD-LAYOUT-V>
      </SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP>
        <SHORT-LABEL>values</SHORT-LABEL><!-- FNC_VALUES -->
        <CATEGORY>FNC_VALUES:COLUMN_DIR</CATEGORY>
        <SW-RECORD-LAYOUT-GROUP-AXIS>0</SW-RECORD-LAYOUT-GROUP-AXIS>
        <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
        <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
        <SW-RECORD-LAYOUT-V>
          <SHORT-LABEL>value</SHORT-LABEL>
          <SW-RECORD-LAYOUT-V-AXIS>0</SW-RECORD-LAYOUT-V-AXIS><!--
             FNC_VALUES  -->
          <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
        </SW-RECORD-LAYOUT-V>
      </SW-RECORD-LAYOUT-GROUP>
    </SW-RECORD-LAYOUT-GROUP>
  </SW-RECORD-LAYOUT>
```

The following XML fragment depicts an example for a `SwRecordLayout` for a curve using guard rails. Note that in this example it is also possible to recognize the pattern of guard rails:

Guard rails are represented by a group which consists of three groups referring the same axis where the first group iterates from 1 to 1, the second group iterates from 1 to -1 and the third group iterates from -1 to -1.

Listing 5.12: example for RecordLayout of a curve using guard rails

```
<SW-RECORD-LAYOUT>
  <SHORT-NAME>CurveWithGuardRails</SHORT-NAME>
  <SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-V><!--  SRC_ADDR_X  -->
      <SHORT-LABEL>srcAdr</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-PROP>SOURCE-ADR</SW-RECORD-LAYOUT-V-PROP>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-V><!--  NO_AXIS_PTS_X -->
      <SHORT-LABEL>noOfAxisPts</SHORT-LABEL>
      <SW-RECORD-LAYOUT-V-PROP>COUNT</SW-RECORD-LAYOUT-V-PROP>
      <SW-RECORD-LAYOUT-V-INDEX>1</SW-RECORD-LAYOUT-V-INDEX>
    </SW-RECORD-LAYOUT-V>
    <SW-RECORD-LAYOUT-GROUP>
      <CATEGORY>AXIS_PTS_X:GUARD_RAILS</CATEGORY>
      <SW-RECORD-LAYOUT-GROUP>
        <SW-RECORD-LAYOUT-GROUP-AXIS>1</SW-RECORD-LAYOUT-GROUP-AXIS>
```

```
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP-AXIS>1</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP-AXIS>1</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>-1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
  </SW-RECORD-LAYOUT-GROUP>
  <SW-RECORD-LAYOUT-GROUP>
    <CATEGORY>FNC_VALUES:GUARD_RAILS</CATEGORY>
    <SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP-AXIS>0</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP-AXIS>0</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
    <SW-RECORD-LAYOUT-GROUP>
      <SW-RECORD-LAYOUT-GROUP-AXIS>0</SW-RECORD-LAYOUT-GROUP-AXIS>
      <SW-RECORD-LAYOUT-GROUP-FROM>-1</SW-RECORD-LAYOUT-GROUP-FROM>
      <SW-RECORD-LAYOUT-GROUP-TO>-1</SW-RECORD-LAYOUT-GROUP-TO>
      <SW-RECORD-LAYOUT-V>
        <SW-RECORD-LAYOUT-V-AXIS>1</SW-RECORD-LAYOUT-V-AXIS>
        <SW-RECORD-LAYOUT-V-PROP>VALUE</SW-RECORD-LAYOUT-V-PROP>
      </SW-RECORD-LAYOUT-V>
    </SW-RECORD-LAYOUT-GROUP>
  </SW-RECORD-LAYOUT-GROUP>
```

```
    </SW-RECORD-LAYOUT-GROUP>
  </SW-RECORD-LAYOUT>
```

5.5.5.2 RecordLayouts and DataTypes

[constr_1027] Types for record layouts ⌈ Because `ParameterDataPrototype`s have a `isOfType`-relation to `ApplicationDataType`s or `Implementation-DataType`s the related data types shall properly match to the details as specified in `swDataDefProps`. ⌋

This is exemplified in figure 5.44.

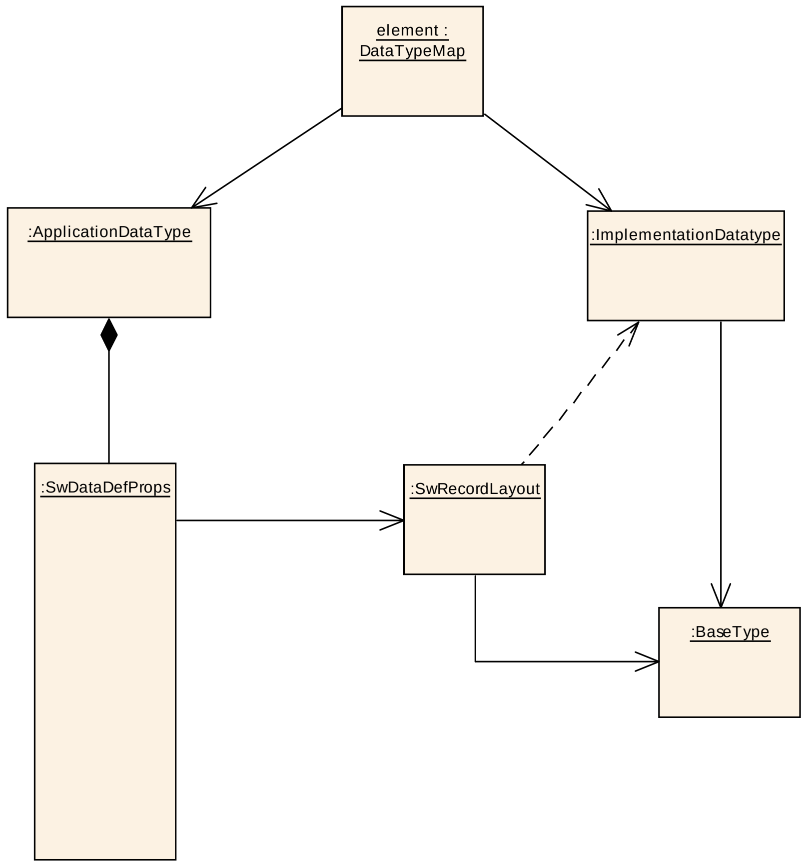

Figure 5.44: Dependency of `DataTypes` and `RecordLayouts`

[TPS_SWCT_1297] Compliance of `ApplicationDataTypes` or `Implementa-tionDataTypes` to `swDataDefProps` ⌈ In order to maintain this compliance the following options exist

- Manually create `ImplementationDataType`s from corresponding `Applica-tionDataType`s and the referenced `RecordLayout`s
- Automatically create `ImplementationDataType`s according to the `Record-Layout`s. This could be performed by a model transformation according to the algorithm shown below.

⌋

[TPS_SWCT_1298] Computing SwRecordLayout from ImplementationDataTypes is not possible ⌈ Note that computing SwRecordLayouts from ImplementationDataTypes is not really possible because the particular semantics of the components is not available (swRecordLayoutVProp). ⌋

Figure 5.45 and figure 5.46 illustrate how data types can be derived from SwRecordLayouts.

The "blue" data types are derived from the record layout. These diagrams illustrate in particular the fact that on the level of ApplicationDataType even complex entities such as curves and maps appear as somehow primitive. The inner details of such entities are handled e.g. by service libraries.

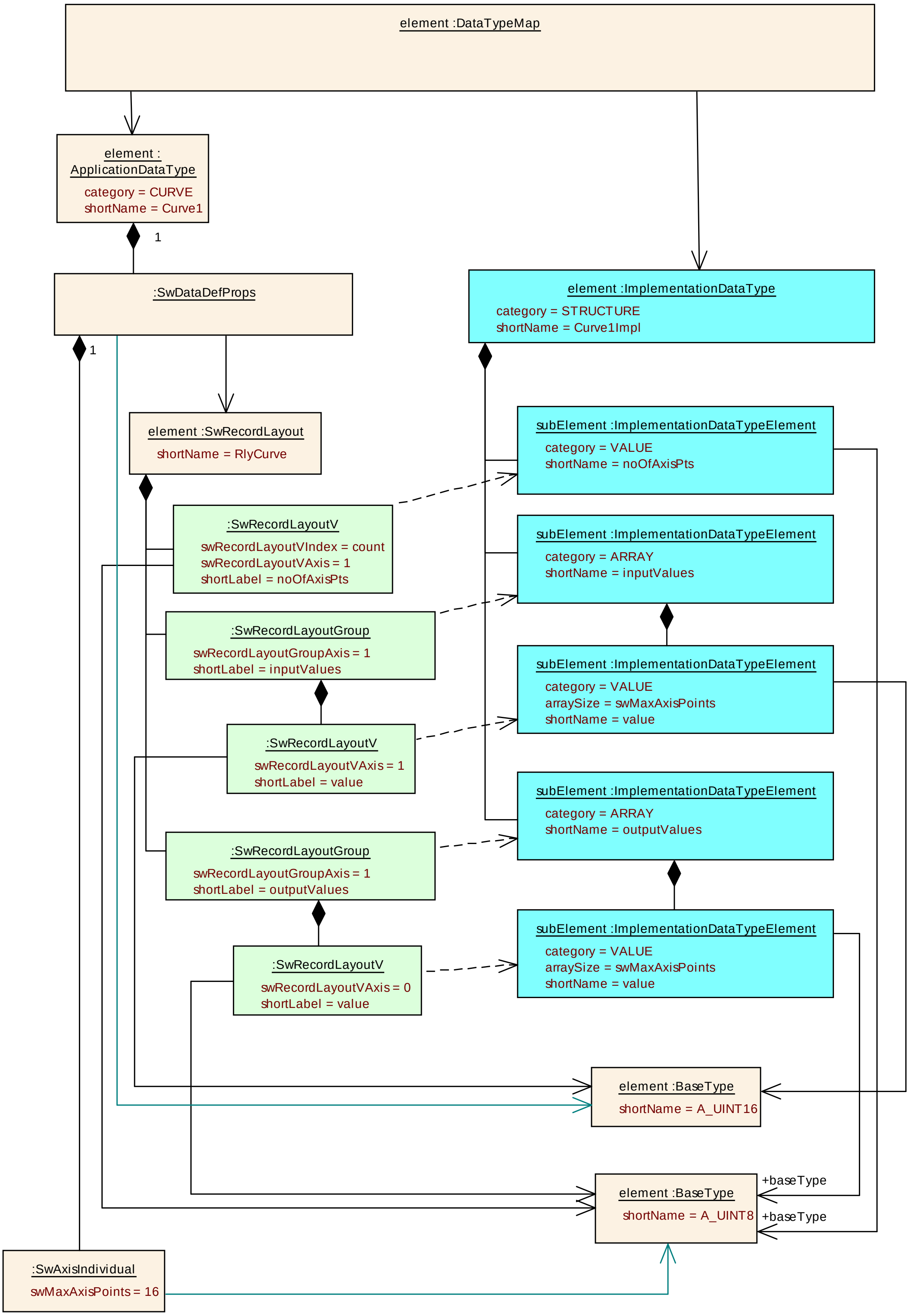

Figure 5.45: Curve implemented as two consecutive arrays

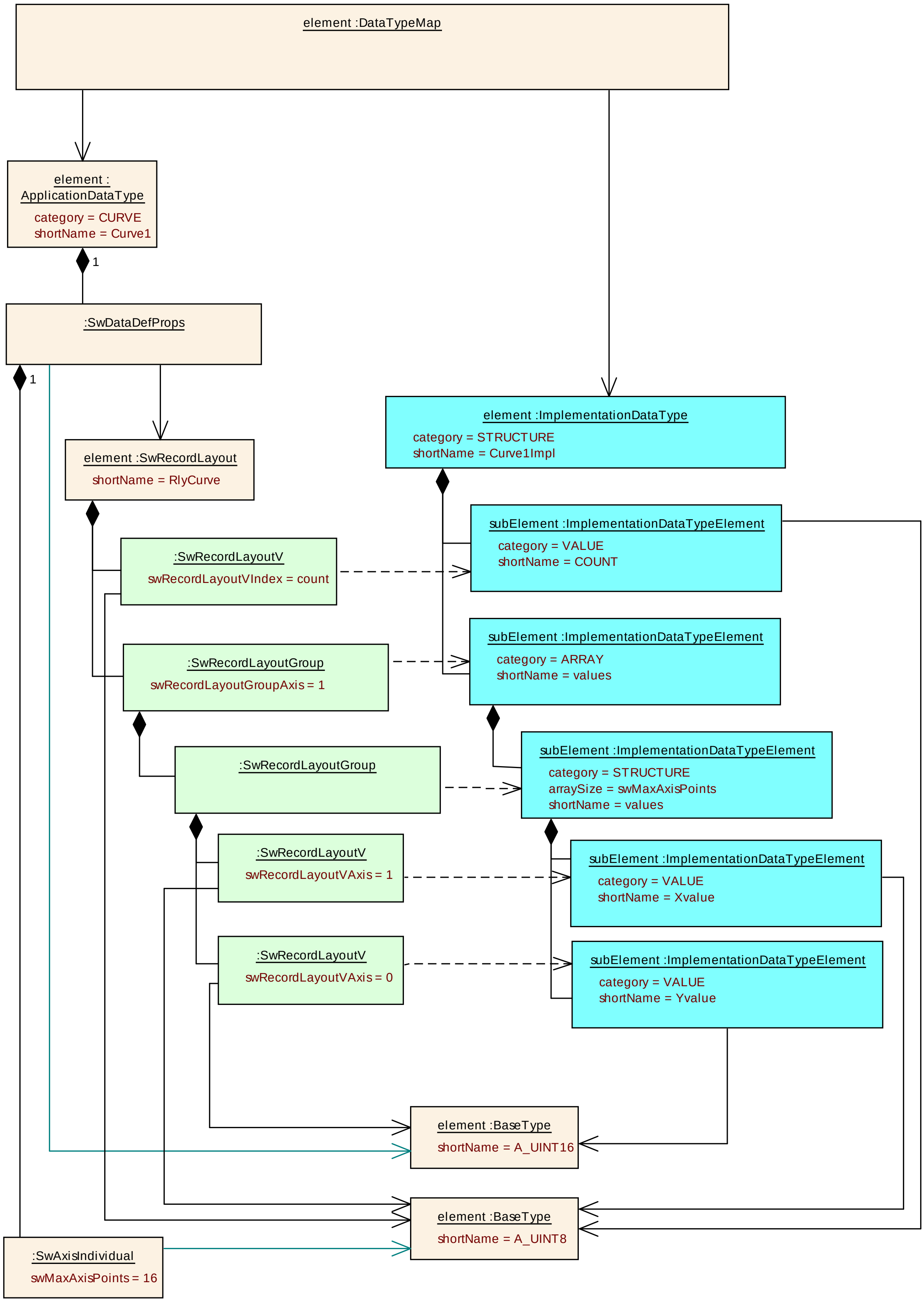

Figure 5.46: Curve implemented as array of value pairs

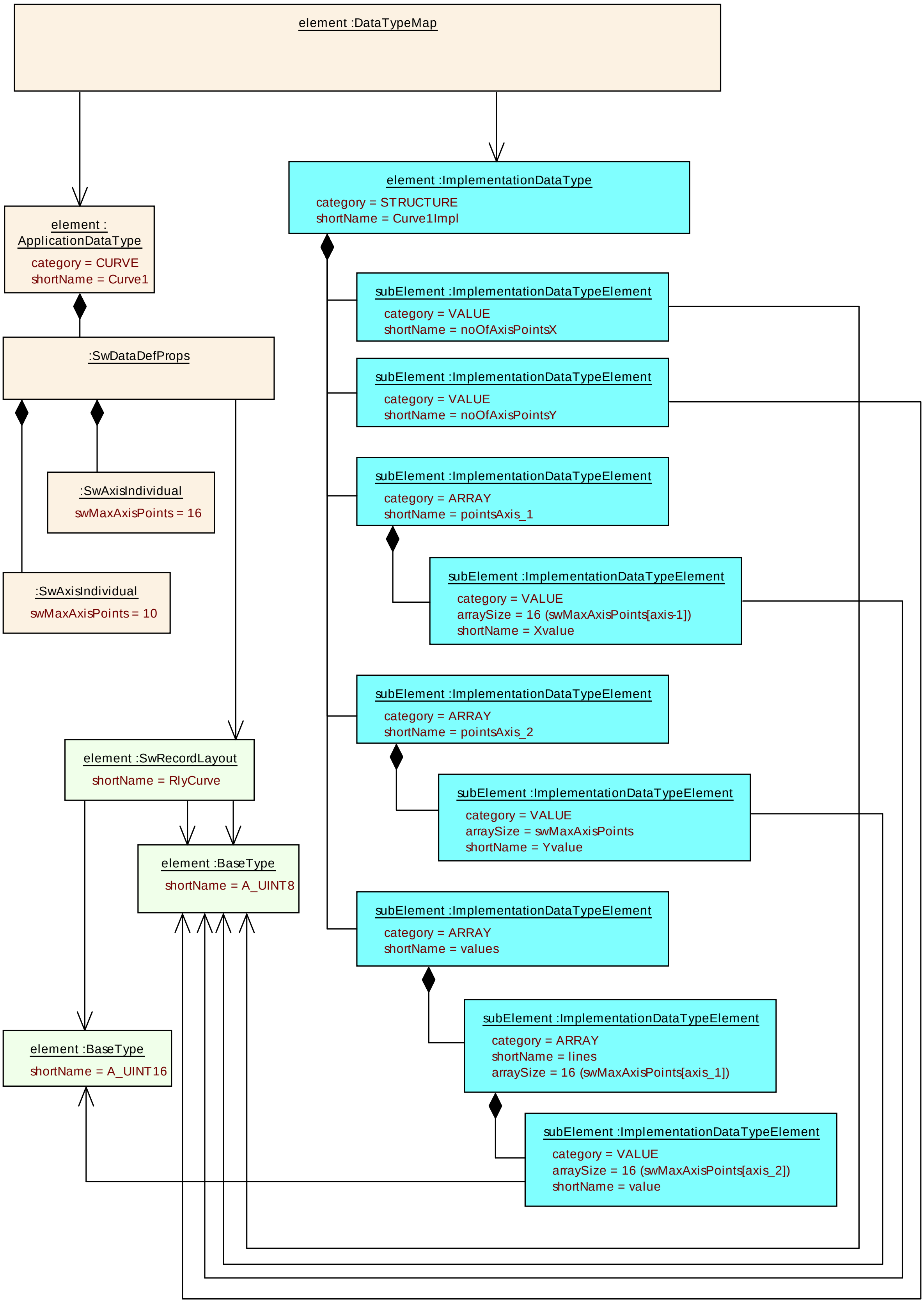

Figure 5.47: Record layout and data type for a map

The algorithm to generate the desired data types is illustrated in the following two diagrams. We create an ImplementationDataType for each ApplicationDataType. Figure 5.48 illustrates how to map the details.

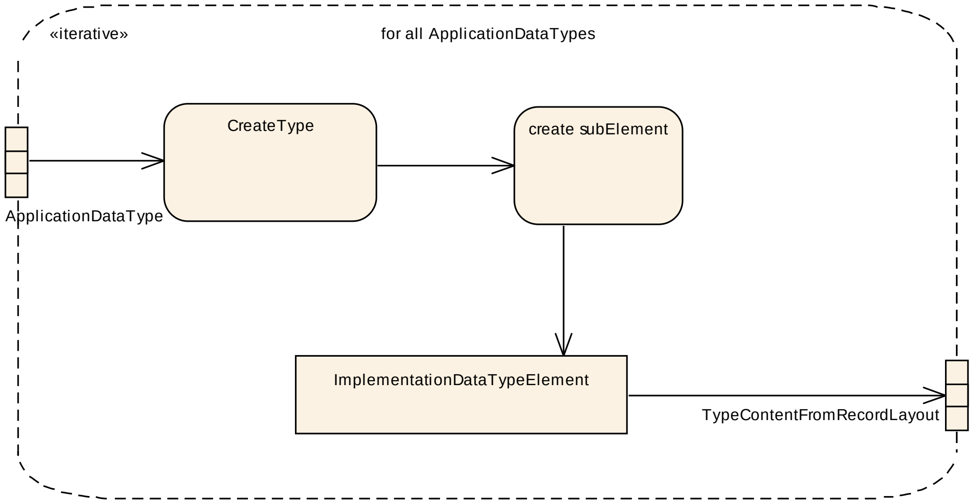

Figure 5.48: algorithm to map the details of an application data type to the corresponding implementation data type according to the record layout

[TPS_SWCT_1299] Relation of `swRecordLayoutGroup` to `subElement` ⌈ For each swRecordLayoutGroup an appropriate subElement shall be created.

This sub element is then refined according to the approach sketched in figure 5.49. The algorithm shall be recursively applied applied to the newly crated ImplementationDataTypeElements. As the record layout groups are nested, this recursion yields the complete structure in the ImplementationDataType. ⌋

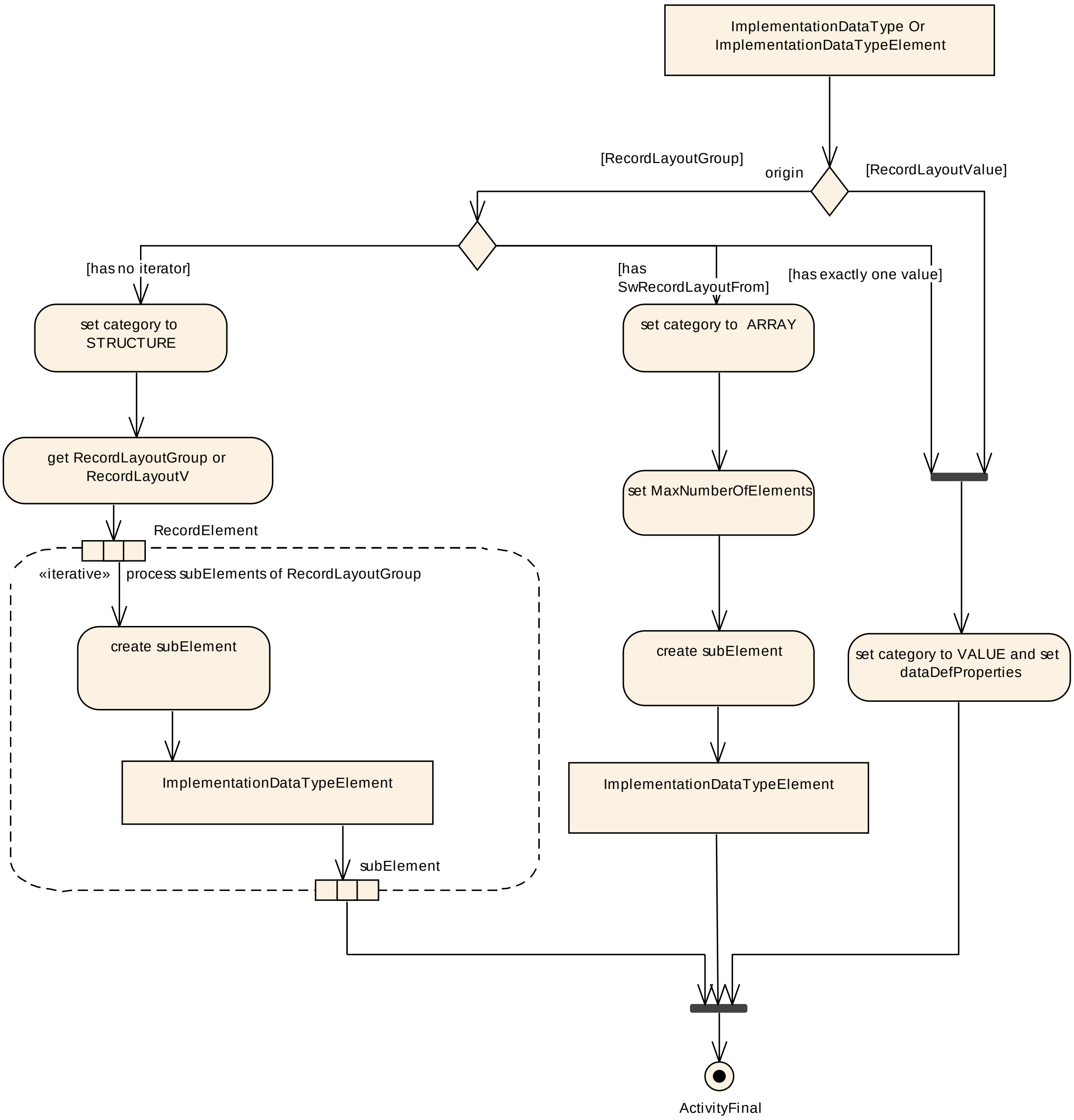

Figure 5.49: refining subElements

5.5.5.3 Record Layouts and Interpolation Routines

[TPS_SWCT_1300] Relationship between record layouts and interpolation routines ⌈ The relationship between record layouts and interpolation routines can be specified in `InterpolationRoutineMappingSet`. The interpolation routine is represented as `BswModuleEntry` and implements a particular interpolation method which is denoted in `shortLabel` of `InterpolationRoutine`. The intended interpolation method is denoted in `interpolationMethod` of `SwDataDefProps`. ⌋

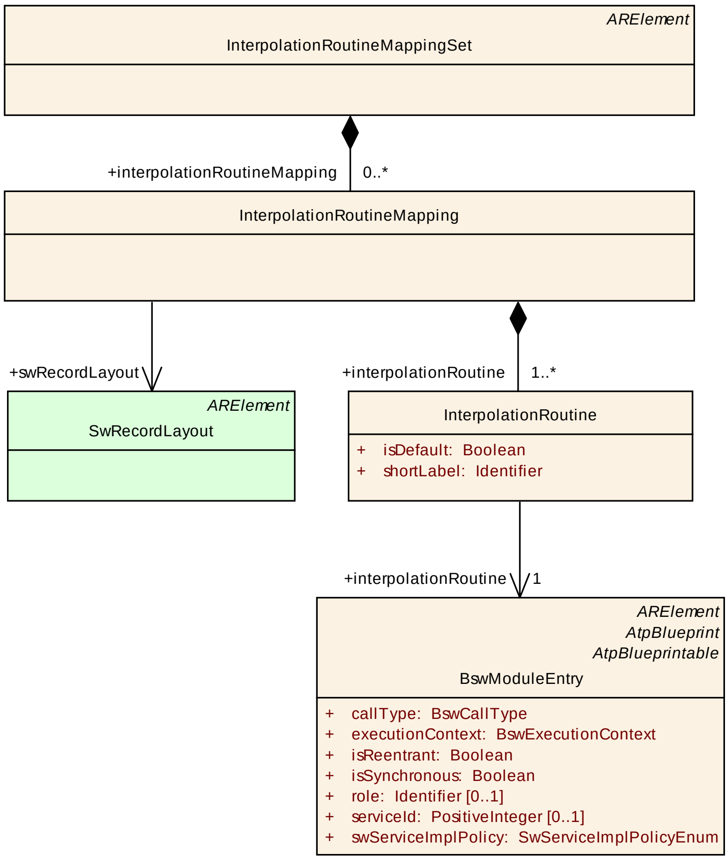

Figure 5.50: Mapping of Record Layouts and Interpolation Routines

Class	**InterpolationRoutineMappingSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutineMappingSet			
Note	This meta-class specifies a set of interpolation routine mappings. **Tags:** atp.recommendedPackage=InterpolationRoutineMappingSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
interpolationRoutineMapping	InterpolationRoutineMapping	*	aggr	This specifies one particular mapping of recordlayout and its matching interpolationRoutines.

Table 5.93: InterpolationRoutineMappingSet

Class	**InterpolationRoutineMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutineMappingSet			
Note	This meta-class provides a mapping between one record layout and its matching interpolation routines. This allows to formally specify the semantics of the interpolation routines. The use case is such that the curves/Maps define an interpolation method. This mapping table specifies which interpolation routine implements methods for a particular record layout. Using this information, the implementer of an SWC can select the appropriate interpolation routine.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
interpolationRoutine	InterpolationRoutine	1..*	aggr	This is one particular interpolation routine which is mapped to the record layout.
swRecordLayout	SwRecordLayout	1	ref	This refers to the record layout which is mapped to interpolation routines.

Table 5.94: InterpolationRoutineMapping

Class	**InterpolationRoutine**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::InterpolationRoutineMappingSet			
Note	This represents an interpolation routine taken to evaluate the contents of a curve or map against a specific input value.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
interpolationRoutine	BswModuleEntry	1	ref	This specifies a BswModuleEntry which implements the current interpolation method for the given record layout. **Tags:** xml.sequenceOffset=30
isDefault	Boolean	1	attr	This specifies if the current interpolationMethod is the default for the referenced record layout. **Tags:** xml.sequenceOffset=20
shortLabel	Identifier	1	ref	This is the name of the interpolation method which is implemented by the referenced bswModuleEntry. It corresponds to swInterpolationMethod in SwDataDefProps. **Tags:** xml.sequenceOffset=10

Table 5.95: InterpolationRoutine

5.6 Specification of Constant Values

[TPS_SWCT_1177] Assignment of constant values ⌈ Constant values can be assigned to a meta-class by aggregating the meta-class `ValueSpecification`. This aggregation can be used in two ways:

1. by referencing to a reusable `ConstantSpecification` which contains another `ValueSpecification`
2. or through an inline aggregation of a value specification of various kind.

⌋*(RS_SWCT_3175)*

Class	**ConstantSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specification of a constant that can be part of a package, i.e. it can be defined stand-alone. **Tags:** atp.recommendedPackage=ConstantSpecifications			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
valueSpec	ValueSpecification	1	aggr	Specification of an expression leading to a value for this constant.

Table 5.96: ConstantSpecification

Class	**ValueSpecification (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Base class for expressions leading to a value which can be used to initialize a data object.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
shortLabel	Identifier	0..1	ref	This can be used to identify particular value specifications for human readers, for example elements of a record type.

Table 5.97: ValueSpecification

[TPS_SWCT_1178] Specialized subclasses of `ValueSpecification` ⌈ Figure 5.52 shows the specialized subclasses of `ValueSpecification` which allow to define values for different use cases:

- Reference to a constant (which is actually a reusable value specification)
- `TextualValueSpecification`
- `NumericalValueSpecification`
- `ArrayValueSpecification`

- RecordValueSpecification
- ApplicationValueSpecification: this can be used to specify the value of compound primitive types such as curves and maps. It is also possible to use this in general (e.g. for a primitive calibration value) for the specification of a value of a DataPrototype typed by an ApplicationDataType (see section 5.7.4).

  Note that ApplicationValueSpecification is modeled along the example of ASAM CDF (for more information please refer to [24]).
- reference to a DataPrototype: this can be used to describe initial values for pointer variables in the basic software. One use case is the exchange of data descriptions used to access calibration data for software emulation methods (see [7] for details).

⌋*(RS_SWCT_3175)*

[TPS_SWCT_1179] Compound primitive type ⌈ For clarification, a "compound primitive type" is an ApplicationPrimitiveDataType of category CURVE, MAP, COM_AXIS, RES_AXIS, and VAL_BLK. This implies the existence of a swRecordLayout owned by the swDataDefProps of the ApplicationPrimitiveDataType that defines the mapping to a corresponding ImplementationDataType.

The main characteristic of the "compound primitive" is that with respect to the application data type layer its data type is considered a primitive data type but when it comes to the implementation data type layer the type is implemented as a composite data type according to the applicable SwRecordLayout. ⌋*(RS_SWCT_3216)*

Note the specific meaning of ConstantReference: it passes the definition of the value on to a ConstantSpecification that is defined as part of an AUTOSAR Package.

Note that ValueSpecification does not inherit from any data type. This would cause a redundancy[9] in the meta-model since the intended data type of a ValueSpecification is already determined by the context in which it is aggregated.

Nonetheless the intended data type imposes a certain constraint on the content of a ValueSpecification:

[constr_4035] ValueSpecification shall fit into data type ⌈ An instance of ValueSpecification which is used to assign a value to a software object typed by an AutosarDataType shall fit into this AutosarDataType without losing information. ⌋

For example, it is not allowed to assign the numerical value "1.5" as initial value to a data prototype typed by an ImplementationDataType which has an integer base type.

[9] For example, “1” can be taken as a constant value for many data types. If the ValueSpecification would refer to a specific Datatype it would be necessary to define a “1” for every single Datatype this value is supposed to be used in combination with.

[TPS_SWCT_1180] Maximum possible size of compound primitive type ⌈ Note that if the size of the "compound primitive" (curve/map) is defined using an attributeValueVariationPoint (in other words swMaxAxisPoints, swNumberOfAxisPoints, swValueBlockSize dependend on the value of SwSystemconst) the initValue shall provide the maximum possible amount of values. ⌋*(RS_SWCT_3216)*

In this case it is the responsibility of model author to ensure that the size of the specified init values matcht the range of the involved system constants.

[constr_1160] Size of "compound primitive" is variant ⌈ For "compound primitives" where the size is subject to variation the size of the specified initValues shall match the range of the involved SwSystemconst. ⌋

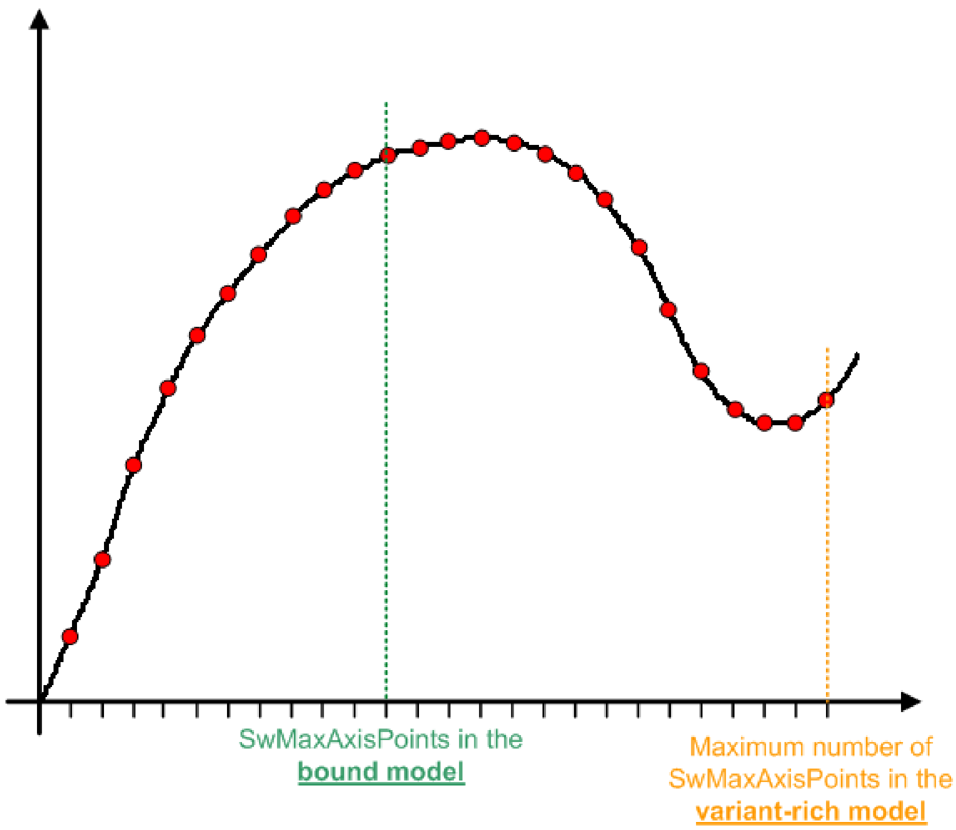

Figure 5.51: Explanation of SwmaxAxisPoints

[TPS_SWCT_1181] Bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst ⌈ The processing tools shall take the lower part of the initValues in case the bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst. ⌋*(RS_SWCT_3216, RS_SWCT_3148)*

[constr_2050] Mandatory information of a SwAxisCont ⌈ If SwAxisCont is defined for an ApplicationValueSpecification the SwAxisCont shall define one swAxisIndex value and one swArraySize value per dimension, even in the case when the owning ApplicationValueSpecification defines only the content of a single dimensional object like a CURVE. ⌋

[constr_2051] Mandatory information of a SwValueCont ⌈ If SwValueCont is defined for an ApplicationValueSpecification the SwValueCont shall define always the attribute swArraySize if the ApplicationValueSpecification is of category CURVE, MAP, COM_AXIS, RES_AXIS, CURVE_AXIS, VAL_BLK, STRING or ARRAY. ⌋

Please note that for multidimensional compound primitives (e.g. MAP) it is necessary to know the dimensions in order to be able to process the SwValues. [constr_2050] and [constr_2051] shall support a consistent handling of single and multidimensional compound primitives.

[constr_2052] Values of swArraySize and the number of values provided by swValuesPhys shall be consistent. ⌈ swValuesPhys shall define as many numbers of values as the swArraySize defines. If several swArraySize values are provided these have to be multiplied in order to get the total number of swValuesPhys values. ⌋

Please note that case of "‘compound primitives"’ typically the attribute swValuesPhys defines more than one value. [constr_2051] and [constr_2052] shall enable a consistent handling of the swValuesPhys values regardless how many dimensions the related compound primitive defines. If the ApplicationValueSpecification defines values for a compound primitive with more than one input axis the swArraySize gets mandatory to ensure the correct processing of the swValuesPhys values independent from of the existence of vgs.

[TPS_SWCT_2001] Values of SwAxisCont with the CATEGORY CURVE_AXIS, COM_AXIS, RES_AXIS are for display only ⌈ In case of ApplicationValueSpecifications of category MAP or CURVE it is possible that the SwAxisCont of axes can be omitted if the axis is of category COM_AXIS or RES_AXIS or CURVE_AXIS. If SwAxisCont values exists in such cases for the axes these are for display purpose only because the related DatePrototype of the MAP or CURVE does not hold the values of such axes. These are properties of the DatePrototype of the COM_AXIS or RES_AXIS or CURVE_AXIS. ⌋

Hence values of the COM_AXIS itself are described by SwValueCont.

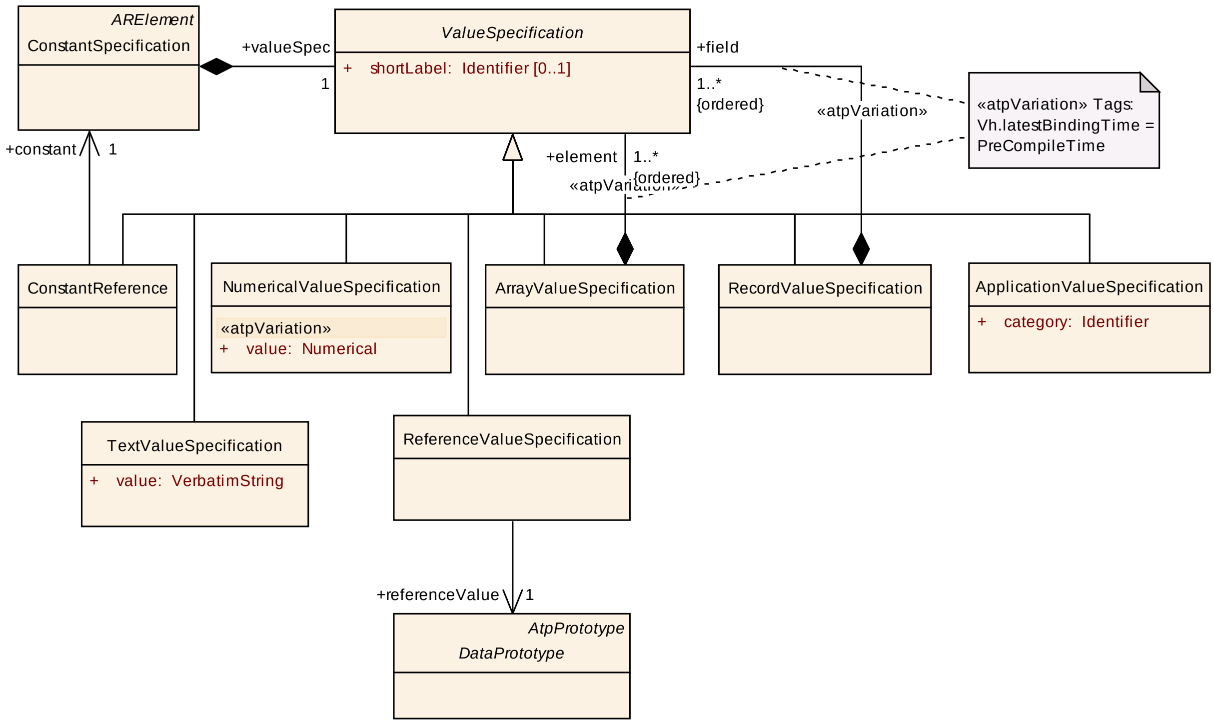

Figure 5.52: Summary of `ValueSpecification`

Class	**ApplicationValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents values for DataPrototypes typed by ApplicationDataTypes (this includes in particular compound primitives). For further details refer to ASAM CDF 2.0. This meta-class corresponds to some extent with SW-INSTANCE in ASAM CDF 2.0.			
Base	ARObject,ValueSpecification			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
category	Identifier	1	ref	Specifies to which category of ApplicationDataType this ApplicationValueSpecification can be applied (e.g. as an initial value), thus imposing constraints on the structure and semantics of the contained values.
SwValueCont	SwValueCont	0..1	aggr	This represents the values of a compoundPrimitive.
swAxisCont (ordered)	SwAxisCont	*	aggr	This represents the axis values of a compound primitive (curve or map) The first swAxisCont describes the x-axis, the second the y-axis, the third the z-axis. In addition to this, the axis can be denoted in swAxisIndex.

Table 5.98: ApplicationValueSpecification

Class	**ArrayValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies the values for an array.			
Base	ARObject,ValueSpecification			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
element (ordered)	ValueSpecification	1..*	aggr	The value for a single array element. All ValueSpecifications aggregated by ArrayValueSpecification must have the same structure. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 5.99: ArrayValueSpecification

Class	**ConstantReference**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Instead of defining this value inline, a constant is referenced.			
Base	ARObject,ValueSpecification			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
constant	ConstantSpecification	1	ref	The referenced constant.

Table 5.100: ConstantReference

Class	**NumericalValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	A numerical ValueSpecification which is intended to be assigned to a Primitive data element. Note that the numerical value is a variant, it can be computed by a formula.			
Base	ARObject,ValueSpecification			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
value	Numerical	1	attr	This is the value itself. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 5.101: NumericalValueSpecification

Class	**RecordValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies the values for a record.			
Base	ARObject,ValueSpecification			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	Datatype	Mul.	Kind	Note
field (ordered)	ValueSpecification	1..*	aggr	The value for a single record field. This could also be mapped explicitly to a record element of the data type using the shortName of the ValueSpecification. But this would introduce a relationship to the data type that is too strong. As of now, it is only important that the structure of the data type matches the structure of the ValueSpecification indepenently of the shortNames. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 5.102: RecordValueSpecification

Class	ReferenceValueSpecification			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	Specifies a reference to a data prototype to be used as an initial value for a pointer in the software.			
Base	ARObject,ValueSpecification			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
referenceValue	DataPrototype	1	ref	The referenced data prototype.

Table 5.103: ReferenceValueSpecification

Class	SwAxisCont			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationValue			
Note	This represents the values for the axis of a compound primitive (curve, map). For standard and fix axis, SwAxisCont contains the values of the axis directly. The axis values of SwAxisCont with the CATEGORY CURVE_AXIS, COM_AXIS, RES_AXIS are for display only. For editing and processing, only the values in the related GroupAxis are binding..			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
category	CalprmAxisCategoryEnum	1	attr	this category specifies the particular axis types: • FIX_AXIS • STD_AXIS • COM_AXIS • CURVE_AXIS (swArraysize necessary) • RES_AXIS (swArraysize necessary) **Tags:** xml.sequenceOffset=20

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swArraysize	ValueList	1	aggr	For multidimensional compound primitivies (curve, map ...) it is necessary to know the dimensions.They are specified using swArraySize. • RES_AXIS • CURVE_AXIS **Tags:** xml.sequenceOffset=70
swAxisIndex	AxisIndexType	1	attr	This property allows to explicitly assign the axis contents to a particular axis. It is specified by numbers where 1 corresponds to the x-axis. It is also possible to derive the axis association from the sequence of the parent. **Tags:** xml.sequenceOffset=50
swValuesPhys	SwValues	1	aggr	swValuesPhys represents the values in the physical domain. **Tags:** xml.sequenceOffset=80
unit	Unit	1	ref	This represents the physical unit of the provided values. **Tags:** xml.sequenceOffset=30
unitDisplayName	SingleLanguageUnitNames	0..1	aggr	This represents the display name which is used for the physical unit of the axis. **Tags:** xml.sequenceOffset=40

Table 5.104: SwAxisCont

Class	**SwValueCont**			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationValue			
Note	This metaclass represents the content of one particular SwInstance.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
swArraysize	ValueList	0..1	aggr	For multidimensional compound primitivies (curve, map ...) it is necessary to know the dimensions.They are specified using swArraySize. swArraysize is importand for VAL_BLK. **Tags:** xml.sequenceOffset=40
swValuesPhys	SwValues	0..1	aggr	swValuesPhys represents the values in the physical domain. **Tags:** xml.sequenceOffset=50
unit	Unit	1	ref	This represents the physical unit of the provided values. **Tags:** xml.sequenceOffset=20

Attribute	Datatype	Mul.	Kind	Note
unitDisplay Name	SingleLanguage UnitNames	0..1	aggr	This specifies how the physical units of the current value set shall be displayed in documents or in user interfaces of tools. **Tags:** xml.sequenceOffset=30

Table 5.105: SwValueCont

Class	≪atpMixed≫ SwValues			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationValue			
Note	This metaclass represents a list of values. These values can either be the input values of a curve (abscissa values) or the associated values (ordinate values). In case of multdimensional structures, the values are ordered such that the lowest index runs the fastest. In particular for maps and cuboids etc. the resulting long value list can be subsectioned using ValueGroup. But the processing needs to be done as if vg is not there. Note that numerical values and textual values should not be mixed.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
v	Numerical	1	attr	This is a non variant Value. It is provided for sake of Compatibility to ASAM CDF. **Tags:** xml.sequenceOffset=40
vf	Numerical	1	attr	This allows to specify the value as Variationpoint. It is distinugished to non variant for sake of compatibility to ASAM CDF 2.0. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.sequenceOffset=20
vg	ValueGroup	1	aggr	This allows to have intersections in the values in order to support specific rendeing (eg. using stylesheets). For tools it is important that V the values are always processed in the same (flattened) order and the tool is able to interpret it without respecting VG. **Tags:** xml.sequenceOffset=50
vt	VerbatimString	1	attr	This represents the values of textual data elements (Strings). Note that vt uses the | to separate the values for the different bitfield masks in case that the semantics of the related DataPrototype is described by means of a BITFIELD_TEXTTABLE in the associated CompuMethod. **Tags:** xml.sequenceOffset=30

Table 5.106: SwValues

Class	**TextValueSpecification**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	A text (string) ValueSpecification which is intended to be assigned to a Primitive data element.			
Base	ARObject,ValueSpecification			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
value	VerbatimString	1	attr	This is the value itself.

Table 5.107: TextValueSpecification

Class	**ValueGroup**			
Package	M2::AUTOSARTemplates::CommonStructure::CalibrationValue			
Note	This element enables valules to be grouped. It can be used to perform row and column-orientated groupings, so that these can be rendered properly e.g. as a table.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
label	MultilanguageLongName	0..1	aggr	This label allows to give the valueGroup a partilucular name. It can be usel if the Values are rendered as a table. **Tags:** xml.sequenceOffset=20
vgContents	SwValues	0..1	aggr	This is the contents of the value group **Tags:** xml.roleElement=false; xml.roleWrapperElement=false; xml.sequenceOffset=30; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.108: ValueGroup

Class	≪atpMixed≫ **ValueList**			
Package	M2::AUTOSARTemplates::CommonStructure::DataDefProperties			
Note	This is a generic list of numerical values.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
v	Numerical	1	attr	This is a particular numerical value without variation. **Tags:** xml.sequenceOffset=30
vf (ordered)	Numerical	*	attr	This is one entry in the list of numerical values **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime xml.roleElement=true; xml.roleWrapperElement=false; xml.typeElement=false; xml.typeWrapperElement=false

Table 5.109: ValueList

5.6.1 Example for Constant Specification for CURVE

The following example illustrates how an ConstantSpecification is specified for a CURVE. Please note, that in this example the vf attribute is used for the swArraysize as well as for the swValuesPhys. The basic intention of vf is the usage for variant rich models but it is valid as well if vf contains invariant values.

Listing 5.13: Example for Constant Specification for CURVE

```
<CONSTANT-SPECIFICATION>
 <SHORT-NAME>PhysInitValuesOfCurve</SHORT-NAME>
 <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a
     CURVE where the axis is a STD_AXIS</L-2>
 </DESC>
 <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
   <CATEGORY>CURVE</CATEGORY>
   <SW-AXIS-CONTS>
    <SW-AXIS-CONT>
     <CATEGORY>STD_AXIS</CATEGORY>
     <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
     <SW-ARRAYSIZE>
      <VF>4</VF>
     </SW-ARRAYSIZE>
     <SW-VALUES-PHYS>
      <VF>0</VF>
      <VF>1</VF>
      <VF>2</VF>
      <VF>3</VF>
     </SW-VALUES-PHYS>
    </SW-AXIS-CONT>
   </SW-AXIS-CONTS>
   <SW-VALUE-CONT>
    <UNIT-REF DEST="UNIT">/AUTOSAR/AISpecification/Units/NwtMtr</
       UNIT-REF>
    <SW-ARRAYSIZE>
     <VF>4</VF>
    </SW-ARRAYSIZE>
    <SW-VALUES-PHYS>
     <VF>00.000</VF>
     <VF>10.000</VF>
     <VF>20.000</VF>
     <VF>30.000</VF>
    </SW-VALUES-PHYS>
   </SW-VALUE-CONT>
  </APPLICATION-VALUE-SPECIFICATION>
 </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

5.6.2 Example for Constant Specification for MAP

The following example illustrates how an ConstantSpecification is specified for a MAP. In this case one axis of the MAP is a STD_AXIS and the second one is a COM_AXIS. Please note, that in this example the v attribute is used for the swArraysize as well as for the swValuesPhys. This is possible because the example contains only invariant values.

Listing 5.14: Example for Constant Specification for MAP

```
<CONSTANT-SPECIFICATION>
 <SHORT-NAME>PhysInitValuesOfMap</SHORT-NAME>
 <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a MAP
      where the first axis is a STD_AXIS and the second axis is a
      COM_AXIS</L-2>
 </DESC>
 <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
   <CATEGORY>MAP</CATEGORY>
   <SW-AXIS-CONTS>
    <SW-AXIS-CONT>
     <CATEGORY>STD_AXIS</CATEGORY>
     <SW-AXIS-INDEX>1</SW-AXIS-INDEX>
     <SW-ARRAYSIZE>
      <V>4</V>
     </SW-ARRAYSIZE>
     <SW-VALUES-PHYS>
      <V>0</V>
      <V>1</V>
      <V>2</V>
      <V>3</V>
     </SW-VALUES-PHYS>
    </SW-AXIS-CONT>
   </SW-AXIS-CONTS>
   <SW-VALUE-CONT>
    <UNIT-REF DEST="UNIT">/AUTOSAR/AISpecification/Units/NwtMtr</
       UNIT-REF>
    <SW-ARRAYSIZE>
     <V>4</V>
     <V>2</V>
    </SW-ARRAYSIZE>
    <SW-VALUES-PHYS>
     <VG>
      <LABEL>
       <L-4 L="EN">Values for axis index 2 equals 0</L-4>
      </LABEL>
      <V>00</V>
      <V>10</V>
      <V>20</V>
      <V>30</V>
     </VG>
     <VG>
      <LABEL>
       <L-4 L="EN">Values for axis index 2 equals 1</L-4>
```

```
        </LABEL>
        <V>01</V>
        <V>11</V>
        <V>21</V>
        <V>31</V>
      </VG>
    </SW-VALUES-PHYS>
   </SW-VALUE-CONT>
  </APPLICATION-VALUE-SPECIFICATION>
 </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

5.6.3 Example for Constant Specification for COM_AXIS

The following example illustrates how an ConstantSpecification is specified for a COM_AXIS.

Listing 5.15: Example for Constant Specification for COM_AXIS

```
<CONSTANT-SPECIFICATION>
 <SHORT-NAME>PhysInitValuesOfComAxis</SHORT-NAME>
 <DESC>
  <L-2 L="EN">This example shows a ConstantSpecification for a
     COM_AXIS</L-2>
 </DESC>
 <VALUE-SPEC>
  <APPLICATION-VALUE-SPECIFICATION>
   <CATEGORY>COM_AXIS</CATEGORY>
   <SW-VALUE-CONT>
    <UNIT-REF DEST="UNIT">/AUTOSAR/AISpecification/Units/Rpm</UNIT-
       REF>
    <SW-ARRAYSIZE>
     <V>6</V>
    </SW-ARRAYSIZE>
    <SW-VALUES-PHYS>
     <V>0</V>
     <V>500</V>
     <V>1000</V>
     <V>1500</V>
     <V>3000</V>
     <V>5000</V>
    </SW-VALUES-PHYS>
   </SW-VALUE-CONT>
  </APPLICATION-VALUE-SPECIFICATION>
 </VALUE-SPEC>
</CONSTANT-SPECIFICATION>
```

5.7 Initial Values

5.7.1 Overview

[TPS_SWCT_1301] Importance of initial values ⌈ If the value of the VariableDataPrototype/ParameterDataPrototype has not been set by a piece of software it can still happen that another piece of software tries to access the value of the VariableDataPrototype/ParameterDataPrototype.

For various reasons it is therefore advised to be able to specify an initial value for a VariableDataPrototype/ParameterDataPrototype in case the value has not been assigned in a controlled manner. However, the definition of an initial value in many cases depends on a context in which the value is accessed. ⌋

Therefore, the AUTOSAR standard foresees means for defining initial values for VariableDataPrototypes/ParameterDataPrototypes on different conceptual levels. That is, although defined for the same VariableDataPrototype/ParameterDataPrototype, an initial value defined on one conceptual level can "supersede" the definition of another initial value on a different conceptual level provided that the priority of the first is higher than the priority of the latter.

The meaning of "supersede" in this context is that that the definition of an initial value on a specific conceptual level is the only relevant definition of an initial value on that level. That is, any initial value defined in the context of a conceptual level of lower priority is ignored!

[TPS_SWCT_1182] Conceptual levels for the definition of initial values ⌈ The following conceptual levels for the definition of initial values exist:

1. It is possible to aggregate an initValue directly at the definition of any VariableDataPrototype/ParameterDataPrototype.
2. It is possible to aggregate an initValue at the level of a ComSpec, namely:
   - NonqueuedSenderComSpec
   - NonqueuedReceiverComSpec
   - ParameterProvideComSpec
   - ParameterRequireComSpec
   - NvRequireComSpec
3. It is possible to aggregate a implInitValue and an appInitValue at the definition of a CalibrationParameterValue.

The priority of one definition of an initial value over another is reflected by the numerical order of the above enumeration, e.g. a definition on level 2 supersedes a definition on level 1. ⌋

5.7.2 Initial Value Representation

[TPS_SWCT_1183] Actual value of an `initValue` shall be interpreted according to the `AutosarDataType` ⌈ A `DataPrototype` can be typed by either an `ApplicationDataType` or else an `ImplementationDataType`. Therefore, the actual value of an `initValue` shall be interpreted according to the `AutosarDataType` that types the `DataPrototype`.

That is, if the `DataPrototype` is typed by an `ApplicationDataType` the value shall be interpreted as a physical value while if the `DataPrototype` is typed by an `ImplementationDataType` the value is to be interpreted as the direct numerical representation. ⌋*(RS_SWCT_3216, RS_SWCT_3217)*

[TPS_SWCT_1184] `ApplicationPrimitiveDataTypes` with category `VALUE` ⌈ In case of `ApplicationPrimitiveDataTypes` with category `VALUE` it is sufficient if the `initValues` are provided as physical values only because the RTE Generator should be able to evaluate the related `CompuMethod` appropriately. ⌋*(RS_SWCT_3216, RS_SWCT_3217)*

[TPS_SWCT_1185] `initValues` for compound primitive types ⌈ This is not applicable for "compound primitive types" (see 5.6). Here the `initValues` have to be provided as a `RecordValueSpecification` respectively an `ArrayValueSpecification` matching to the related `ImplementationDataType`. The additional representation can be provided and associated by means of a `ConstantSpecificationMapping`. ⌋*(RS_SWCT_3216)*

5.7.3 Constant Specification Mapping

[TPS_SWCT_1186] `ConstantSpecificationMapping` ⌈ The `ConstantSpecificationMapping` is used to associate `ValueSpecifications` defined in the implementation domain with corresponding `ValueSpecifications` defined in the application domain.

To make this possible the `ValueSpecification` actually needs to be a `ConstantReference`. The `ConstantSpecification` referenced by the `ConstantReference` is also the target of the references owned by `ConstantSpecificationMapping`. ⌋

[constr_1029] `ConstantSpecificationMapping` and `ConstantSpecification` ⌈ In this case it is required that one referenced `ConstantSpecification` needs to be defined in the application domain (`applConstant`) and the other needs to be defined in the implementation domain (`implConstant`). ⌋

[TPS_SWCT_1187] `ConstantSpecificationMappingSet` referenced by the `InternalBehavior` ⌈ In most cases the meta-class `ConstantSpecificationMappingSet` will be referenced by the `InternalBehavior`. This `ConstantSpecificationMappingSet` contains the applicable `ConstantSpecificationMappings`. ⌋

However, in some specializations the software-components will not have an `InternalBehavior`:

- **[constr_1030] `ParameterSwComponentType` references `ConstantSpecificationMappingSet`** ⌈ `ParameterSwComponentType`: here the `ConstantSpecificationMappingSet` is directly associated by the `ParameterSwComponentType`. ⌋
- **[constr_1031] `NvBlockSwComponentType` references `ConstantSpecificationMappingSet`** ⌈ `NvBlockSwComponentType`: in this case the `ConstantSpecificationMappingSet` is associated with the aggregated `NvBlockDescriptor`. ⌋

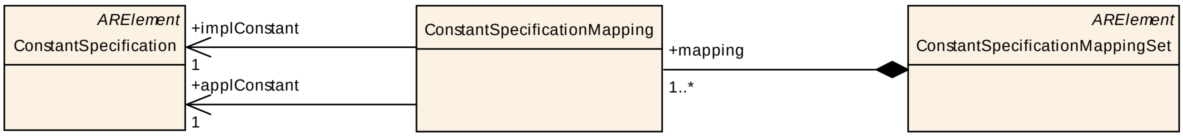

Figure 5.53: Constant Mapping

Class	**ConstantSpecificationMapping**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class is used to create an association of two ConstantSpecifications. One ConstantSpecification is supposed to be defined in the application domain while the other should be defined in the implementation domain. Hence the ConstantSpecificationMapping needs to be used where a ConstantSpecification defined in one domain needs to be associated to a ConstantSpecification in the other domain. This information is crucial for the RTE generator.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
applConstant	ConstantSpecification	1	ref	A ConstantSpecification defined in the application domain.
implConstant	ConstantSpecification	1	ref	A ConstantSpecification defined in the implementation domain.

Table 5.110: ConstantSpecificationMapping

Class	**ConstantSpecificationMappingSet**			
Package	M2::AUTOSARTemplates::CommonStructure::Constants			
Note	This meta-class represents the ability to map two ConstantSpecifications to each others. One ConstantSpecification is supposed to be described in the application domain and the other should be described in the implementation domain. **Tags:** atp.recommendedPackage=ConstantSpecificationMappingSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	Datatype	Mul.	Kind	Note
mapping	ConstantSpecificationMapping	1..*	aggr	ConstantSpecificationMappings owned by the ConstantSpecificationMappingSet.

Table 5.111: ConstantSpecificationMappingSet

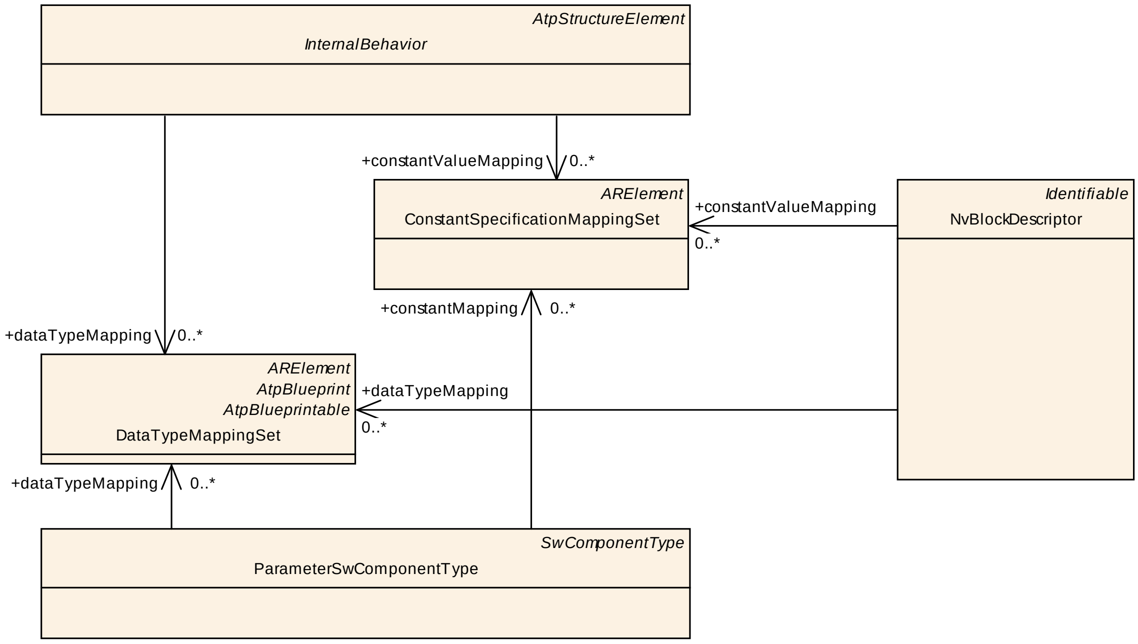

Figure 5.54: Aggregation of `ConstantSpecificationMappingSet`

5.7.4 Initial Values For CalibrationParameters

[TPS_SWCT_1188] Definition of calibration data sets through RTE-generator and compiler ⌈ It is possible to provide sets of initial values for calibration parameters which are instance specific, thus overriding any initial values predefined by a `ParameterDataPrototype`, `ParameterRequireComSpec` or a `ParameterProvideComSpec`.

This allows to create the calibration data sets through RTE-generator and compiler. These initial values are specified in `CalibrationParameterValueSet` and `CalibrationParameterValue`. The latter aggregates a `ValueSpecification` in two different roles:

- structured according to `ApplicationDataType`. In this case the values are defined in the physical domain.
- structured according to `ImplementationDataType`. In this case the values are defined in the numerical domain.

⌋*(RS_SWCT_3175)*

Anyhow, these initial values can be imported from e.g. an ASAM CDF file.

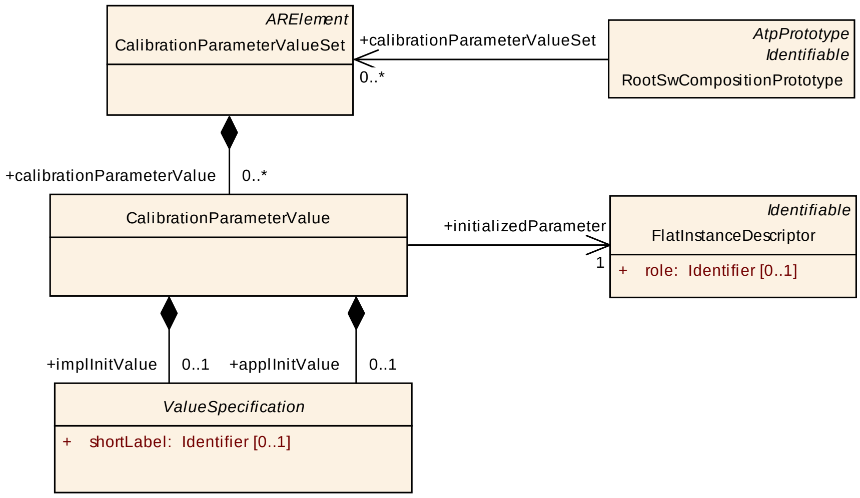

Figure 5.55: Calibration Parameter Values

Class	**CalibrationParameterValueSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::CalibrationParameterValues			
Note	Specification of a constant that can be part of a package, i.e. it can be defined stand-alone. **Tags:** atp.recommendedPackage=CalibrationParameterValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
calibration Parameter Value	CalibrationPara meterValue	*	aggr	This represents single CalibrationParameterValues in the CalibrationParameterValueSet.

Table 5.112: CalibrationParameterValueSet

Class	**CalibrationParameterValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::MeasurementAndCalibration::CalibrationParameterValues			
Note	Specifies instance specific calibration parameter values used to initialize the memory objects implementing calibration parameters in the generated RTE code. RTE generator will use the implInitValue to override the initial values specified for the DataPrototypes of a component type. The applInitValue is used to exchange init values with the component vendor not publishing the transformation algorithm between ApplicationDataTypes and ImplementationDataTypes or defining a instance specific initialization of components which are only defined with ApplicationDataTypes. Note: If both representations of init values are available these need to represent the same content. Note further that in this case an explicit mapping of ValueSpecification is not implemented because calibration parameters are delivered back after the calibration phase.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
applInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ApplicationDataType
implInitValue	ValueSpecification	0..1	aggr	This is the initial value specification structured according to the ImplementationDataType
initializedParameter	FlatInstanceDescriptor	1	ref	This represents the parameter that is initilaized by the CalibrationParameterValue.

Table 5.113: CalibrationParameterValue

6 Compatibility

6.1 Introduction

In order to connect PortPrototypes of SwComponentTypes, the compatibility of PortPrototypes needs to be verified. This section defines the basic rules for formal compatibility of PortPrototypes.

Compatibility will be defined bottom-up, i.e. first the rules for compatible AutosarDataTypes are set up, then the rules for the different types of PortInterfaces are derived.

6.2 Compatibility of Data Types

The AUTOSAR meta model defines a number of meta-classes (e.g. ApplicationPrimitiveDataType) that eventually refer to a set of attributes (e.g. a lower boundary for its values) relevant for compatibility checking. Instantiating a data-type related meta-class defines a data type on M1 level (e.g. *temperatureType*). In other words: ApplicationPrimitiveDataType is an M2 artifact; it is taken as the template for creating a corresponding M1 artifact *temperatureType*.

In this context, the issue of compatibility refers to the M1 objects, i.e. the instances of sub-classes of AutosarDataType need to be considered. For this purpose the relevant part of the AUTOSAR meta-model need to be fully explored with respect to compatibility.

6.2.1 ApplicationDataType

6.2.1.1 ApplicationPrimitiveDataType

[constr_1047] Compatibility of ApplicationPrimitiveDataTypes ⌈ Instances of ApplicationPrimitiveDataType are compatible if and only if

1. They have the same category (see table in figure 5.8).
2. The swDataDefProps attached to the M1 data types are compatible. The meaning of this statement is explained in section 6.2.3.

⌋

Please note that it is **not** required that the shortNames of two data types shall be identical in order to consider the two data types as compatible.

6.2.1.2 ApplicationCompositeDataType

An instance of an ApplicationRecordDataType is never compatible to an instance of an ApplicationArrayDataType.

[constr_1048] Compatibility of ApplicationRecordDataTypes ⌈ Instances of ApplicationRecordDataTypes are compatible if and only if

1. All elements *at the same record position* are of compatible AutosarDataTypes either ApplicationCompositeDataTypes or ApplicationPrimitiveDataTypes).

⌋

[constr_1049] Compatibility of ApplicationArrayDataTypes ⌈ Instances of ApplicationArrayDataType are compatible if and only if

1. Their elements are of a compatible AutosarDataTypes (either ApplicationCompositeDataTypes or ApplicationPrimitiveDataTypes).
2. The attributes maxNumberOfElements and arraySizeSemantics (given the existence) have identical values.

⌋

6.2.2 ImplementationDataType

[constr_1050] Compatibility of ImplementationDataTypes ⌈ Instances of ImplementationDataType are compatible if and only if

1. They have the same category (see table 5.17)
2. They have the identical structure (this refers to ImplementationDataTypeElement and their subElements).
3. The attributes arraySize and arraySizeSemantics have (given the existence) identical values.
4. The swDataDefProps attached to the M1 data types are compatible. The meaning of this statement is explained in section 6.2.3.

⌋

Please note that it is **not** required that the shortNames of two data types shall be identical in order to consider the two data types as compatible.

The following constraint applies for the case that mode manager and mode user are using different ImplementationDataTypes. From the point of view of the RTE there is only the necessity that all possible numbers used to represent ModeDeclarations of the mode manager has to fit into the range of the data type used for the mode user.

[constr_1168] Compatibility of `ImplementationDataTypes` used used in the `ModeRequestTypeMap` ⌈ Both `ImplementationDataType`s shall fulfill [constr_1167]. In addition to that, the possible numbers used for representing `ModeDeclaration`s on the side of the mode manager shall match the supported range of the `ImplementationDataType` used for representing `ModeDeclaration`s on the side of the mode user (see [constr_1075]). ⌋

6.2.3 Compatibility of SwDataDefProps

[constr_1051] Compatibility of `SwDataDefProps` ⌈ `SwDataDefProps` are compatible if and only if:

1. They refer to compatible `Unit` definitions, or neither of them has an associated `Unit`.
2. They refer to compatible conversion methods `compuPhysToInternal` from physical to internal values, or neither of them associates such a method.
3. They refer to compatible conversion methods `compuInternalToPhys` from internal to physical values, or neither of them associates such a method.
4. They contain (if applicable) the same `invalidValue`.
5. They refer to compatible data constraints `dataConstr`.
6. They refer to compatible `swRecordLayout`s

All other attributes (e.g. `calibrationAccess`) do not affect compatibility. ⌋

6.2.3.1 Compatibility of Units

[constr_1052] Compatibility of `Units` ⌈ Two `Unit` definitions are compatible if and only if:

1. They have identical attributes `factorSiToUnit` and `offsetSiToUnit`.
2. They either refer to identical definitions of `PhysicalDimension` or neither of them associates a `PhysicalDimension`.

⌋

Please note that it is **not** required that the `shortName`s of two `Unit`s shall be identical in order to consider the two units as compatible.

6.2.3.2 Compatibility of PhysicalDimensions

[constr_1053] Compatibility of `PhysicalDimensions` ⌈ Two `PhysicalDimension` definitions are compatible if and only if the values of

- shortName
- lengthExp
- massExp
- timeExp
- currentExp
- temperatureExp
- molarAmountExp
- luminousIntensityExp

are identical. ⌋

For clarification, there are some physical dimensions around that share the identical values for the exponents but still have a completely different meaning and shall therefor not be considered compatible. For precisely this reason [constr_1053] **requires** the shortNames of two PhysicalDimensions to be identical as a prerequisite for compatibility.

For example, there are at least two physical dimensions that share the values of

- lengthExp = 2
- massExp = 1
- timeExp = -2
- currentExp = 0
- temperatureExp = 0
- molarAmountExp = 0
- luminousIntensityExp = 0

The unit described by this set of exponents is usually referred to as "Nm" for *newton-meter* and it can be used for *torque* just as well as for *energy*. Obviously, two Units shall never be considered compatible if one refers to *torque* and the other one refers to *energy*.

6.2.3.3 Compatibility of Data Constraints

The compatibility of two DataConstrs depends on the context in which the owning data elements are connected:

[constr_1126] Compatibility of DataConstrs ⌈ The DataConstr (e.g. the limits) defined by the type of the providing data element shall be within the constraints defined by the type of the requiring data element. ⌋

In addition, it is always allowed if the requiring element defines no constraints.

[constr_1054] No DataConstr available at the provider ⌈ If the provider defines no constraints it is only compatible with a receiver which also defines no constraints at all. ⌋

In other words, this is not a compatibility rule for the types but for the data prototypes.

6.2.3.4 Compatibility in case of ImplementationDataType

In addition, if the SwDataDefProps are owned by an ImplementationDataType further conditions shall be met to ensure compatibility.

Note that depending on the category of the ImplementationDataType, at most one of these four constraints is actually relevant:

1. category **[constr_1055] ImplementationDataType has category VALUE** ⌈ **VALUE**: The attributes swBaseType shall refer to a compatible SwBaseType ⌋ (see explanation in the following rule).

2. category **TYPE_REFERENCE**: **[constr_1056] ImplementationDataType has category TYPE_REFERENCE** ⌈ The attributes implementationDataTypes shall refer to compatible ImplementationDataTypes ⌋

3. category **DATA_REFERENCE**: **[constr_1057] ImplementationDataType has category DATA_REFERENCE** ⌈ The attributes swPointerTargetProps shall have identical targetCategory and shall refer to SwDataDefProps where all attributes are identical ⌋ (in other words, the target types of the pointers shall be identical, not only compatible).

4. category **FUNCTION_REFERENCE**: **[constr_1058] ImplementationDataType has category FUNCTION_REFERENCE** ⌈ The attributes swFunctionPointerSignature shall refer to BswModuleEntry-s which each resolve to the same function signature (i.e. same number of arguments; return types and arguments shall have identical - not only compatible - types). ⌋

Two SwBaseTypes are compatible (in the sense of allowing a connection of ports via the RTE) if a simple conversion rule exists between the two types in the underlying programming language.

Admittedly, this is a rather weak condition. But because the definition of SwBaseTypes can contain a nativeDeclaration it is not possible to state this rule more specifically.

However, conversion between base types is considered as a less common use case than the simple case that the connected types just contain two identical SwBaseTypes (which is of course included in the rule).

Please note, that in addition the existence of ApplicationDataTypes also constraints the possible SwBaseTypes via the compatibility rules for the mapping between

ApplicationDataTypes and ImplementationDataType as will be explained in more detail in chapter 6.2.4.

6.2.3.5 Compatibility of CompuMethods

[constr_1163] Compatibility of CompuMethods ⌈ Two CompuMethod definitions are compatible if and only if all attributes **except**

- shortName
- desc
- introduction
- longName
- adminData
- annotation

are **identical and** the compuScales are compatible. ⌋

[constr_1153] Applicability of compatibility requirements for CompuScales ⌈ Compatibility requirements for CompuScales shall only apply for CompuScales where the category of the enclosing CompuMethod is one of the following:

- SCALE_LINEAR_AND_TEXTTABLE
- SCALE_RATIONAL_AND_TEXTTABLE
- TEXTTABLE
- TAB_NOINTP
- BITFIELD_TEXTTABLE
- LINEAR
- RAT_FUNC

⌋

[constr_1154] Compatibility of CompuScales for sender-receiver communication and similar use cases ⌈ For sender-receiver communication and similar use cases, it is required that the set of CompuScales defined in the CompuMethod of the provider of the communication (i.e. on the side of the PPortPrototype) shall be a subset of the set of CompuScales defined in the CompuMethod on the required side (i.e. on the side of the RPortPrototype). ⌋

[constr_1155] Compatibility of CompuScales for client-server communication ⌈ For client-server communication, the following rules apply:

For arguments of direction IN the CompuScales defined in the CompuMethod of the client (i.e. on the side of the RPortPrototype) shall be a subset of the set of

CompuScales defined in the CompuMethod supported at the server (i.e. on the side of the PPortPrototype).

For arguments of the direction OUT the set of CompuScales defined in the CompuMethod of the server (i.e. on the side of the PPortPrototype) shall be a subset of the set of CompuScales defined in the CompuMethod supported at the client (i.e. on the side of the RPortPrototype).

For arguments of direction INOUT the set of CompuScales defined in the CompuMethod of server and client shall be identical. ⌋

[constr_1156] Relevance of "names" of CompuScales ⌈ CompuScales which contribute to tabular conversion by having a compuConst are compatible **if and only if** the "names" of the compuScales, (namely shortLabel, compuConst and symbol) are equal. If the scale has no compuConst, "names" of CompuScales are not relevant for compatibility. ⌋

[constr_1157] Applicability of constraints of CompuScales ⌈ The constraints [constr_1154], [constr_1155], and [constr_1156] shall **only** apply in the absence of a TextTableMapping which shall take precedence regarding the compatibility if it exists. ⌋

[constr_1176] Compatibility of CompuScales of category LINEAR and RAT_FUNC ⌈ CompuScales of category LINEAR and RAT_FUNC are considered compatible if they yield the same conversion. ⌋

In other words, $\frac{n_0+n_1*phys}{d_0+d_1}$ is compatible to $\frac{N_0+N_1*phys}{D_0}$ if $n_0 \sim N_0$ && $n_1 \sim N_1$ && $d_0 \sim D_0$ && $d_1 \sim 0$.

6.2.3.6 Compatibility of Record Layouts

[constr_1162] Compatibility of SwRecordLayouts ⌈ Two SwRecordLayout definitions are compatible if and only if all attributes **except**

- shortName
- desc
- introduction
- longName
- adminData
- annotation

are **identical**. ⌋

6.2.4 Compatibility of ApplicationDataType and ImplementationDataType

Eventually, the usage of ApplicationDataTypes implies that also a corresponding ImplementationDataType exists. The latter is taken as the basis for configuring and generating the RTE and/or contract phase header files.

Therefore, it is necessary to define compatibility rules that unambiguously clarify the conformance of an ApplicationDataType with an ImplementationDataType and vice versa.

Please note that this kind of compatibility also supports situations where a dataElement typed by an ApplicationDataType without a corresponding ImplementationDataType in a PPortPrototype should be connected to a dataElement typed by an ImplementationDataType in an RPortPrototype.

In general, the compatibility rules for allowing a data type mapping are the same as the rules for connections. Exceptions are explicitly stated in the rules below.

Several rules depend on the category of the data types:

1. As a general rule, if an ImplementationDataType of category TYPE_REFERENCE is targeted by a type mapping or port connection all the rules given below apply to the ImplementationDataType which is finally valid after resolving all such references.

   This is not repeated in all rules. As an example, if we say that something can be mapped/connected to an ImplementationDataType of category VALUE this shall include the possibility of mapping/connecting to an ImplementationDataType of category TYPE_REFERENCE which refers to another ImplementationDataType of category VALUE.

2. **[constr_1059] Compatibility of data types with category VALUE** ⌈ An ApplicationDataType of category VALUE can only be mapped/connected to an ImplementationDataType which also has category VALUE. ⌋

   In this case, the ImplementationDataType.baseType shall be able to express all the numerical values required by the ApplicationDataType.

   This condition is fulfilled if the numerical range which can be expressed by the SwBaseType at least covers the range defined by the limits in ApplicationDataType.dataConstr (which are either internal limits or physical limits to be converted via the CompuMethod which also has to be provided by the ApplicationDataType).

   The condition is also fulfilled if the SwBaseType covers the range defined in the CompuMethod for an enumeration (see 5.5.1.1).

   Note that for sender-receiver communication of a data element via a network there is the possibility to reduce the numerical range against what has been defined via the corresponding data type. However, this is not achieved via mapping to another ImplementationDataType at the data element itself but via the networkRepresentation of the ComSpec (for further explanation of this aspect see 4.5.1).

3. **[constr_1060] Compatibility of data types with category ARRAY, VAL_BLK, or STRING** ⌈ An ApplicationDataType of category ARRAY, VAL_BLK, or STRING can only be mapped/connected to an ImplementationDataType of category ARRAY. ⌋

   In this case, the array size, the arraySizeSemantics (given that it exists) and the type of the array elements of the ImplementationDataType shall be such that they can be mapped resp. transferred 1:1 by order to the corresponding application data and vice versa.

   Note that in case of mapping between arrays it is not required that a DataTypeMap exists between the data types of the array elements or that the respective ShortNames are identical.

4. **[constr_1061] Compatibility of data types with category STRUCTURE** ⌈ An ApplicationDataType of category STRUCTURE can only be mapped/connected to an ImplementationDataType of category STRUCTURE. ⌋

   This means, that the corresponding pairs of elements shall also have compatible types. Note that it is not required that the types of the single elements have identical ShortNames or that a DataTypeMap exists for each pair of single element.

5. **[constr_1063] Compatibility of data types with category BOOLEAN** ⌈ An ApplicationDataType of category BOOLEAN can only be mapped/connected to an ImplementationDataType of category VALUE. ⌋

6. **[constr_1064] Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE or MAP** ⌈ An ApplicationDataType of category COM_AXIS, RES_AXIS, CURVE, or MAP can only be mapped/connected to an ImplementationDataType of category STRUCTURE or ARRAY. ⌋

   There are several possibilities how to express these types via plain or nested arrays and/or structures on implementation level.

   Some examples are given in 5.4.4. In any case, the primitive elements of the implementation type shall fit (by their order in memory) to the corresponding RecordLayout.

   It is not required, to define DataTypeMaps for the sub-elements or both representations.

7. **[constr_1066] ApplicationDataType is or is not compatible to specific ImplementationDataType** ⌈ An ApplicationDataType cannot be connected or mapped to an ImplementationDataType of category DATA_REFERENCE or FUNCTION_REFERENCE. ⌋

8. **[constr_1067] ApplicationDataType is or is not compatible to specific ImplementationDataType** ⌈ An ApplicationDataType cannot be connected or mapped to an ImplementationDataType of category UNION but

it is possible to define a type mapping (provided other rules allow it) between the elements of a UNION and individual ApplicationDataTypes. ⌋

Concerning the SwDataDefProps of an ApplicationDataType instance resp. an ImplementationDataType instance which shall be mapped/connected on M1, we refer to the table shown in figure 5.41. The following rules apply:

1. The cases where the ImplementationDataType is not allowed to set a property but only “inherits” it from the ApplicationDataType are not relevant for compatibility. These attributes are simply not allowed in the ImplementationDataType.

2. In case that only the ImplementationDataType may “define” the property this definition shall fit into the semantical requirements given by the ApplicationDataType in order to make the two types compatible.

   This is namely important for the attribute baseType and is explained above in the rule for types of category VALUE.

3. In case the ImplementationDataType may “add” a property it may only add but not change a property defined by the ApplicationDataType (namely note, displayFormat, and swImplPolicy) in order to be compatible.

   [constr_1158] Applicable categorys for attribute compuMethod ⌈ In case of attribute compuMethod the addition of this property is restricted to the computation method categories BITFIELD_TEXTTABLE, SCALE_RATIONAL_AND_TEXTTABLE, SCALE_LINEAR_AND_TEXTTABLE, and TEXTTABLE (these might be seen as implementation specific in certain cases). ⌋

   This means that the respective computation methods can be defined in only one of the types in order to be compatible. In all other cases, only the ApplicationDataType may define the computation method.

4. For the compatibility with respect to connectors there are some additional rules for the values of the attribute swImplPolicy which are considered general rules on the level of DataPrototypes and PortInterfaces.

   Therefore these additional rules are explained in chapter 6.3 and chapter 6.4.3.

5. The case that an ImplementationDataType may “redefine” a property which is already set by the ApplicationDataType is not considered as relevant for the compatibility with respect to mapping of the types in general but of course there may be project specific rules as to which redefinition is allowed (e.g. for swAddrMethod or dataConstr). See also 5.5.3 about data constraints.

6. For the compatibility with respect to connectors the attribute dataConstr shall be treated in the same way as for compatibility of data types in general, for more details please refer to 6.2.3.

6.3 Compatibility of Variable Data Prototypes and Parameter Data Prototypes

[constr_1068] Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by primitive data types ⌈ Two VariableDataPrototypes or ParameterDataPrototypes of ApplicationPrimitiveDataTypes or ImplementationDataTypes of category VALUE, BOOLEAN, or STRING are compatible if and only if

1. They are typed by (read "refer to") compatible AutosarDataTypes.
2. The two VariableDataPrototypes or ParameterDataPrototypes have identical shortNames. This is required to map VariableDataPrototypes in unordered SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces.
3. The attribute SwImplPolicy is either set to queued for both or none of the VariableDataPrototypes.

⌋

[constr_1187] Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types ⌈

DataPrototypes of ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY are compatible if one of the following conditions evaluates to true:

1. The underlying ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY are identical
2. The underlying ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY fulfill the following condition:
   - They consist of the same number of elements **and**
   - They are composed of compatible AutosarDataTypes (either ApplicationCompositeDataTypes or ImplementationDataTypes of category STRUCTURE or ARRAY **OR** ApplicationPrimitiveDataTypes or ImplementationDataTypes of category VALUE, BOOLEAN, or STRING) in *the same order* **and**
   - All attributes match exactly, with the exception of the shortName of the M1 AutosarDataType.
3. For each element of the DataPrototype on the required side a SubElementRef exists that references an element in the the required DataPrototype and a corresponding element in the provided DataPrototype.

⌋

6.4 Compatibility of Sender Receiver Interfaces, Parameter Interfaces and Non Volatile Data Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a receiver shall process a certain data value to correctly interpret the following values).

6.4.1 Connection of Required and Provided Port via AssemblySwConnector

The compatibility of SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces are considered for connecting of PortPrototypes with an AssemblySwConnector.

[constr_1069] Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors ⌈ PortPrototypes of different DataInterfaces are compatible if and only if

1. For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the provided PortPrototype.

   The table 6.1 defines which PortInterface elements are compatible dependent from the PortInterface type and the SwImplPolicy attributes of the PortInterface elements.

   Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

2. For each such pair, the values of their isService attributes are identical.

⌋

6.4.2 Connection of Inner and Outer Port via DelegationSwConnector

The compatibility of SenderReceiverInterfaces, NvDataInterfaces and ParameterInterfaces is considered for connecting of PortPrototypes with a DelegationSwConnector.

[constr_1070] Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors ⌈ PortPrototypes of different DataInterfaces are compatible if and only if

1. For each VariableDataPrototype or ParameterDataPrototype defined in the context of the DataInterface of the required inner PortPrototype a

compatible VariableDataPrototype or ParameterDataPrototype exists in the DataInterface of the required outer PortPrototype.

The table 6.1 defines which PortInterface elements are compatible dependent from the PortInterface type and the SwImplPolicy attributes of the PortInterface elements.

Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

2. For at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided inner PortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface or ParameterInterface of the provided outer PortPrototype.

   The table 6.1 defines which PortInterface elements are compatible dependent from the PortInterface type and the SwImplPolicy attributes of the PortInterface elements.

   Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

3. For each such pair, the values of their isService attributes are identical.

⌋

6.4.3 Compatibility of ParameterDataPrototype and VariableDataPrototype depending on PortInterface Type

[constr_1071] compatibility of ParameterDataPrototype and VariableDataPrototype ⌈⌋

The table 6.1 defines which PortInterface elements are compatible depending on the kind of PortInterface and the SwImplPolicy attributes of the PortInterface elements.

Additionally, VariableDataPrototypes defined in the context of the SenderReceiverInterface are only compatible if the invalidationPolicys have the same value.

For VariableDataPrototypes and ParameterDataPrototypes in the context of NvDataInterface respectively ParameterInterface the invalidationPolicy are treated like "Invalidation is switched off" (dontInvalidate).

Provided Port Require Outer Port Provided Inner Port			Required Port Required Inner Port Provided Outer Port					
Port Interface			Prm			S/R		NvD
Interface Element			PDP			VDP		VDP
		SwImplPolicy	fixed	const	standard	standard	queued	standard
Prm	PDP	fixed	yes	yes	yes	yes	no	yes
		const	no	yes	yes	yes	no	yes
		standard	no	no	yes	yes	no	yes
S/R	VDP	standard	no	no	no	yes	no	yes
		queued	no	no	no	no	yes	no
NvD	VDP	standard	no	no	no	yes	no	yes

Table 6.1: Overview of compatibility of ParameterDataPrototype and VariableDataPrototype

Caption of table 6.1:

Interface Element
PDP : `ParameterDataPrototype`
VDP : `VariableDataPrototype`

Port Interface
Prm : `ParameterInterface`
S/R : `SenderReceiverInterface`
NvD : `NvDataInterface`

6.5 Compatibility of Mode Switch Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a receiver shall process a certain data value to correctly interpret the following values).

Note that concerning the compatibility of `ModeSwitchInterface`s it is necessary to distinguish between the context of an `AssemblySwConnector` and the context of an `DelegationSwConnector`.

6.5.1 Connection of Required and Provided Port via AssemblySwConnector

Here, the compatibility of `ModeSwitchInterface`s is considered for the context of an `AssemblySwConnector`.

[constr_1072] Compatibility of `ModeSwitchInterfaces` in the context of an `AssemblySwConnector` ⌈ `PortPrototype`s of different `ModeSwitchInterface`s are compatible if and only if

1. For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the required PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the provided PortPrototype. The shortNames of the ModeDeclarationGroupPrototypes are used to identify the pair.
2. For each such pair, the values of their isService attributes are identical.

⌋

6.5.2 Connection of Inner and Outer Port via DelegationSwConnector

Here, the compatibility of ModeSwitchInterfaces is considered for the context of a DelegationSwConnector.

[constr_1073] Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector ⌈ PortPrototypes of different ModeSwitchInterfaces are compatible if and only if

1. For the ModeDeclarationGroupPrototype defined in the context of the ModeSwitchInterface of the inner PortPrototype a compatible ModeDeclarationGroupPrototype exists in the ModeSwitchInterface of the outer PortPrototype. The shortNames of the ModeDeclarationGroupPrototypes are used to identify the pair.
2. For each such pair, the values of their isService attributes are identical.

⌋

6.6 Compatibility of Mode Declaration Group Prototypes

[constr_1074] Compatibility of ModeDeclarationGroupPrototypes ⌈ ModeDeclarationGroupPrototypes are compatible if and only if

1. They are typed by (read "refer to") compatible ModeDeclarationGroups.
2. Each ModeDeclarationGroupPrototype on the required side corresponds to a ModeDeclarationGroupPrototypes on the provided side with an identical ShortName.

⌋

6.7 Compatibility of Mode Declaration Groups

[constr_1075] Compatibility of ModeDeclarationGroups ⌈ ModeDeclarationGroups are compatible if and only if

1. They define an identical number of ModeDeclarations.
2. Each ModeDeclaration on the required side corresponds to a ModeDeclaration on the provided side with an identical ShortName.
3. The initialModes on both sides refer to ModeDeclarations with identical ShortNames.

⌋

6.8 Compatibility of Argument Prototypes

[constr_1076] Compatibility of ArgumentDataPrototypes ⌈ Two ArgumentDataPrototypes are compatible if and only if

1. They are typed by compatible AutosarDataTypes.
2. They have the same direction (in, out or inout).

⌋

6.9 Compatibility of Application Errors

[constr_1077] Compatibility of ApplicationErrors ⌈ Two ApplicationErrors are compatible if and only if

1. They have the same shortName.
2. They have the same attributes. Especially the errorCode shall be identical in both ApplicationErrors.

⌋

6.10 Compatibility of Client/Server Operations

[constr_1078] Compatibility of ClientServerOperations ⌈ Two ClientServerOperations are compatible if their signatures match. In particular, they are compatible if and only if

1. They have the same number of ArgumentDataPrototypes.
2. The n-th arguments of both ClientServerOperations are compatible. This implies ordering of ArgumentDataPrototypes.
3. They have the same shortName (again allows for mapping in PortInterfaces).

4. The required ClientServerOperation specifies a compatible ApplicationError for each ApplicationError that is possibly raised by the provided ClientServerOperation, maybe more.

⌋

6.11 Compatibility of Client Server Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a client shall call a certain operation to allow the server to work correctly).

6.11.1 Connection of Required and Provided Port via AssemblySwConnector

[constr_1079] Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector ⌈ClientServerInterfaces are compatible if and only if

1. For each ClientServerOperation defined in the context of the ClientServerInterface of the required PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.
2. For each such pair, the values of their isService attributes are identical.

⌋

6.11.2 Connection of Inner and Outer Port via DelegationSwConnector

[constr_1080] Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector ⌈ClientServerInterfaces are compatible if and only if

1. For each ClientServerOperation defined in the context of the ClientServerInterface of the required inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the required outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.
2. For at least one ClientServerOperation defined in the context of the ClientServerInterface of the provided inner PortPrototype a compatible ClientServerOperation exists in the ClientServerInterface of the provided outer PortPrototype. The shortNames of ClientServerOperations are used to identify the pair.
3. For each such pair, the values of their isService attributes are identical.

⌋

6.12 Compatibility of Trigger Interfaces

Please note that this compatibility requirement only satisfies static correctness which means that logical consistency is not assured (e.g. that a client shall call a certain operation to allow the server to work correctly).

6.12.1 Connection of Required and Provided Port via AssemblySwConnector

[constr_1081] Compatibility of `TriggerInterfaces` in the context of an `AssemblySwConnector` ⌈ `TriggerInterface`s are compatible if and only if

1. For each `Trigger` defined in the context of the `TriggerInterface` of the required `PortPrototype` a compatible `Trigger` exists in the `TriggerInterface` of the provided `PortPrototype`. The `shortName`s of `Trigger` are used to identify the pair.
2. For each such pair, the values of their `isService` attributes are identical.

⌋

6.12.2 Connection of Inner and Outer Port via DelegationSwConnector

[constr_1082] Compatibility of `TriggerInterfaces` in the context of an `DelegationSwConnector` ⌈ `TriggerInterface`s are compatible if and only if

1. For each `Trigger` defined in the context of the `TriggerInterface` of the required inner `PortPrototype` a compatible `Trigger` exists in the `TriggerInterface` of the required outer `PortPrototype`. The `shortName`s of `Trigger` are used to identify the pair.
2. For at least one `Trigger` defined in the context of the `TriggerInterface` of the provided inner `PortPrototype` a compatible `Trigger` exists in the `ClientServerInterface` of the provided outer `PortPrototype`. The `shortName`s of `Trigger` are used to identify the pair.
3. For each such pair, the values of their `isService` attributes are identical.

⌋

6.13 Compatibility of Trigger

[constr_1083] Compatibility of `Triggers` ⌈ `Trigger`s are compatible if they have an identical `ShortName`. ⌋

6.14 Entire Delegation of a Provided Port Prototype

[constr_1084] delegation of an provided outer `PortPrototype` ⌈ The delegation of an provided outer `PortPrototype` is properly defined if the following criteria are fulfilled:

1. For each `VariableDataPrototype` or `ParameterDataPrototype` present in the `SenderReceiverInterface`, `NvDataInterface`, or `ParameterInterface` of the provided outer `PortPrototype` at least one connection via `DelegationSwConnector` to a provided inner `PortPrototype` with a compatible `VariableDataPrototype` or `ParameterDataPrototype` in the `SenderReceiverInterface` `NvDataInterface` or `ParameterInterface` of the provided inner `PortPrototype` exists.

   Either the `shortNames` of `VariableDataPrototype`s or `ParameterDataPrototype`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

   The table 6.1 defines which `PortInterface` elements are compatible dependent from the `PortInterface` type and the `SwImplPolicy` attributes of the `PortInterface` elements.

2. For the `ModeDeclarationGroupPrototype` present in the `ModeSwitchInterface` of the provided outer `PortPrototype` exactly one connection via `DelegationSwConnector` to a provided inner `PortPrototype` with a compatible `ModeDeclarationGroupPrototype` in the `ModeSwitchInterface` of the provided inner `PortPrototype` exists.

   Either the `shortNames` of `ModeDeclarationGroupPrototype`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

3. For each `ClientServerOperation` present in the `ClientServerInterface` of the provided outer `PortPrototype` exactly one connection via `DelegationSwConnector` to a provided inner `PortPrototype` with a compatible `ClientServerOperation` in the `ClientServerInterface` of the provided inner `PortPrototype` exists.

   Either the `shortNames` of `ClientServerOperation`s are used to identify the pair or a `PortInterfaceMapping` defines which differently named `PortInterface` elements correlate with each other.

4. For each Trigger present in the TriggerInterface of the provided outer PortPrototype exactly one connection via DelegationSwConnector to a provided inner PortPrototype with a compatible Trigger in the TriggerInterface of the provided inner PortPrototype exists.

   Either the shortNames of Triggers are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

⌋

6.14.1 Split and Merge of PortInterface Elements

With the definition of compatibility rules in chapter 6.4, 6.11, and 6.12 it is possible to split and distribute elements of a PortPrototype of type of a PortInterface containing a superset of PortInterface elements to PortPrototypes of type of PortInterfaces containing subsets of PortInterface elements.

Please find examples that explain the usage of splitting and merging in section 6.16.2.

6.15 Compatibility in Case of a Flat ECU Extract

Please note that in the case of a flat ECU extract of software-components specific compatibility rules apply. To some extent, these rules contradict the rules existing for the pure VFB approach (see chapter 6). That is, if the split-and-merge pattern has been applied on the creation of DelegationSwConnectors it might happen that compatibility rules defined in chapter 6 are violated.

However, given that the flattened ECU extract has been created out of a valid CompositionSwComponentType the flattened ECU extract does not become invalid in this case. In other words, the transformation does not create an invalid model out of a valid model.

However, to support this statement it is necessary to define additional compatibility rules that properly cover this case and allow for a successful validation of the flattened ECU extract.

For the flat ECU extract the compatibility of SenderReceiverInterfaces, NvDataInterfaces, and ParameterInterfaces is considered for connecting of PortPrototypes with a DelegationSwConnector.

[constr_1085] Compatibility in the case of a flat ECU extract ⌈ PortPrototypes of different SenderReceiverInterfaces, NvDataInterfaces, and ParameterInterfaces are compatible if and only if

1. For at least one VariableDataPrototype or ParameterDataPrototype defined in the context of the SenderReceiverInterface, NvDataInter-

face, or ParameterInterface of the RPortPrototype a compatible VariableDataPrototype or ParameterDataPrototype exists in the SenderReceiverInterface, NvDataInterface, or ParameterInterface of the provided PortPrototype.

The table 6.1 defines which PortInterface elements are compatible dependent from the PortInterface type and the SwImplPolicy attributes of the PortInterface elements.

Either the shortNames of VariableDataPrototypes and ParameterDataPrototypes are used to identify the pair or a PortInterfaceMapping defines which differently named PortInterface elements correlate with each other.

⌋

Please note that in case of the flat ECU extract it might happen that AssemblySwConnectors that connect to a specific RPortPrototype also connect to PPortPrototypes that do not fulfill the compatibility rule specified in 6.4.1.

In particular, the dataElements might correspond to dataElements defined in the scope of different PPortPrototypes. In other words, in the flat ECU extract it is possible to merge dataElements from different providers.

6.16 Compatibility Examples

This section provides some examples that may explain the compatibility of PortPrototypes.

6.16.1 Compatibility on Assembly Level

The rules for compatibility with respect to the connection of dataElements by means of AssemblySwConnectors are perhaps easier to digest than the delegation case but nonetheless it seems appropriate to provide a set of examples that illustrate the compatibility issue.

6.16.1.1 Legal Use

One of the less trivial examples of this kind is the case of sender/receiver n:1 communication. Figure 6.1 sketches a case where both sender software-components provide the dull set of dataElements that are required by the RPortPrototype of the receiving software-component.

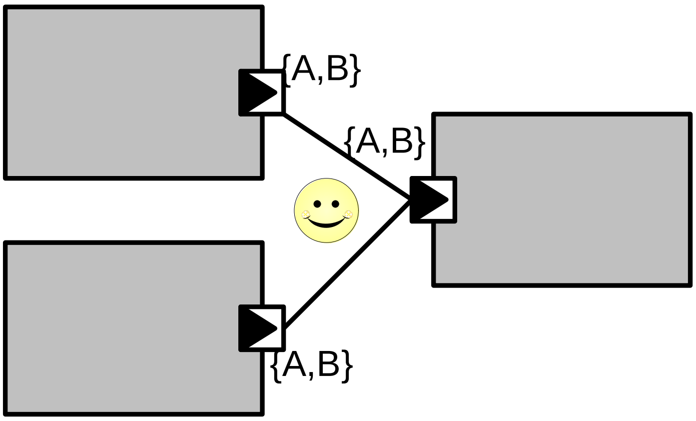

Figure 6.1: legal n:1 communication

The next case (exemplified by Figure 6.2) implements a situation where one sender provides two `dataElement`s {A,b} while the other sender provides only as subset of these, i.e. {B}.

As the `RPortPrototype` of the receiving software-component requires only the `dataElement` {B} compatibility issues will not occur because for every required `dataElement` a compatible `dataElement` is provided.

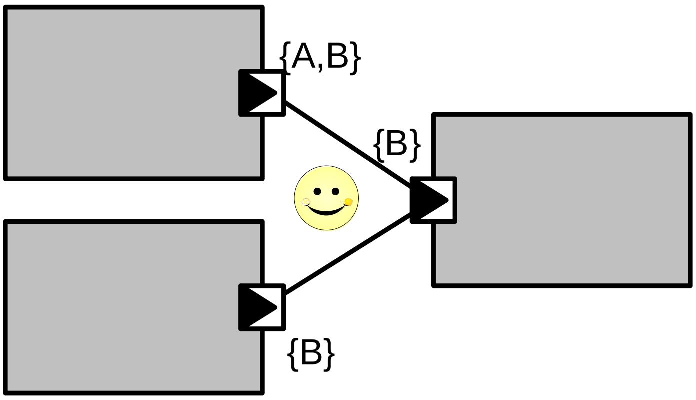

Figure 6.2: legal n:1 communication

6.16.1.2 Illegal Use

On possible example for an illegal configuration of a sender/receiver communication is the scenario sketched in Figure 6.3. Although the sender software-components in total provide the set of required `dataElement`s the *individual* `AssemblySwConnector`s create incompatible connections between sender and receiver.

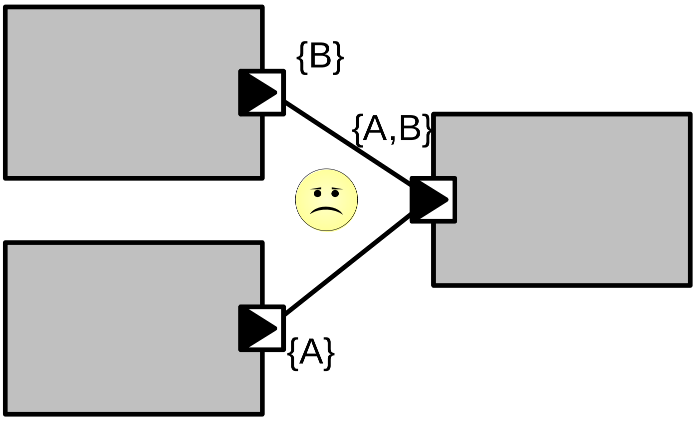

Figure 6.3: illegal n:1 communication

6.16.2 Compatibility on Delegation Level

The rules for compatibility with respect to the delegation of `dataElement`s perhaps require some explanation in terms of examples. The first example 6.4 describes a legal situation where two `DelegationSwConnector`s split the `dataElement`s contained in the `RPortPrototype` owned by a `CompositionSwComponentType`.

6.16.2.1 Legal Use

The examples explain the usage of `DelegationSwConnector`s in different configurations and different values of `DelegatedPortAnnotation`. Please note that the `DelegatedPortAnnotation` is usually defined before the internal structure of a `CompositionSwComponentType` is fully clarified.

At a later point in time it has to be consistent or can be removed. Decorating the example with applicable values of `DelegatedPortAnnotation` should facilitate the understanding of the meaning of the `DelegatedPortAnnotation`.

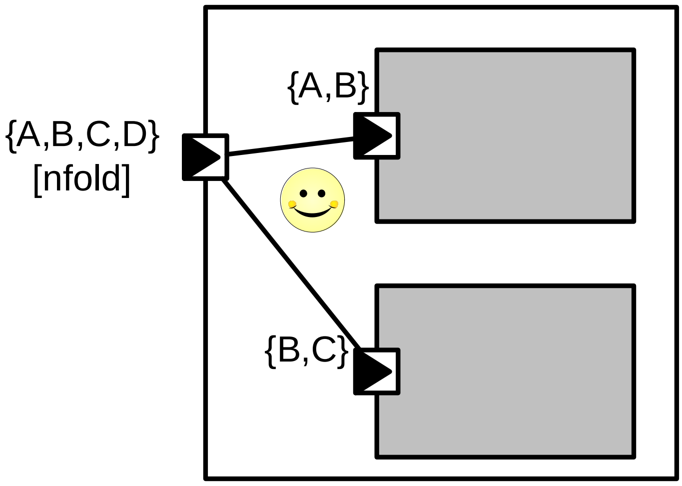

Figure 6.4: Legal split of delegation connector

All required dataElements are provided by the DelegationSwConnectors attached to the delegation RPortPrototype. The fact that dataElement D is not conveyed to any of the RPortPrototypes owned by the SwComponentPrototypes does not have any impact on the compatibility.

In other words: the RPortPrototype at the CompositionSwComponentType actually contains the superset of dataElements {A ,B, C, D}. The two required inner PortPrototypes of the SwComponentPrototypes contain the subsets of VariableDataPrototypes {A, B} and {B, C}. In this case the resulting communication pattern on the VFB for B would be 1:n.

This requires DelegatedPortAnnotation to be set to the value nfold.

In the next example the RPortPrototype of the CompositionSwComponentType contains the superset of dataElements {A ,B}. The two RPortPrototypes of the SwComponentPrototypes contain *different* subsets, i.e. {A} and {B}.

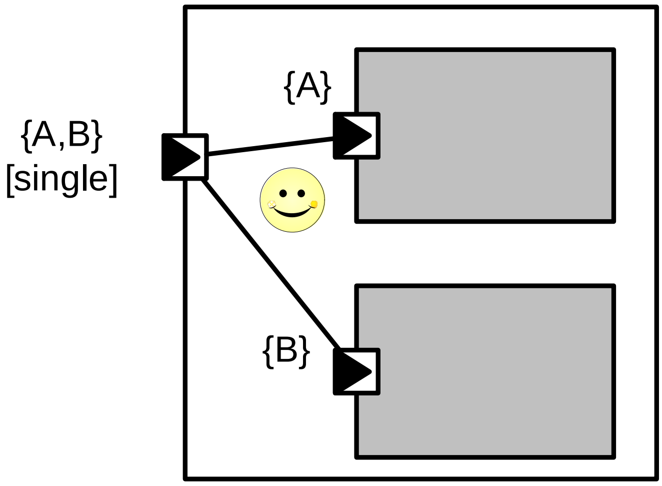

Figure 6.5: Legal split of delegation connector

In this case the resulting communication pattern on the VFB would be n:1. In this case the value of the DelegatedPortAnnotation should be set to single.

The next example is about the merge of DelegationSwConnectors. The PPortPrototype owned by the CompositionSwComponentType contains a superset of dataElements {A ,B}. The two PPortPrototypes of the SwComponentPrototypes contain a *disjoint* subset each, i.e. {A} and {B}.

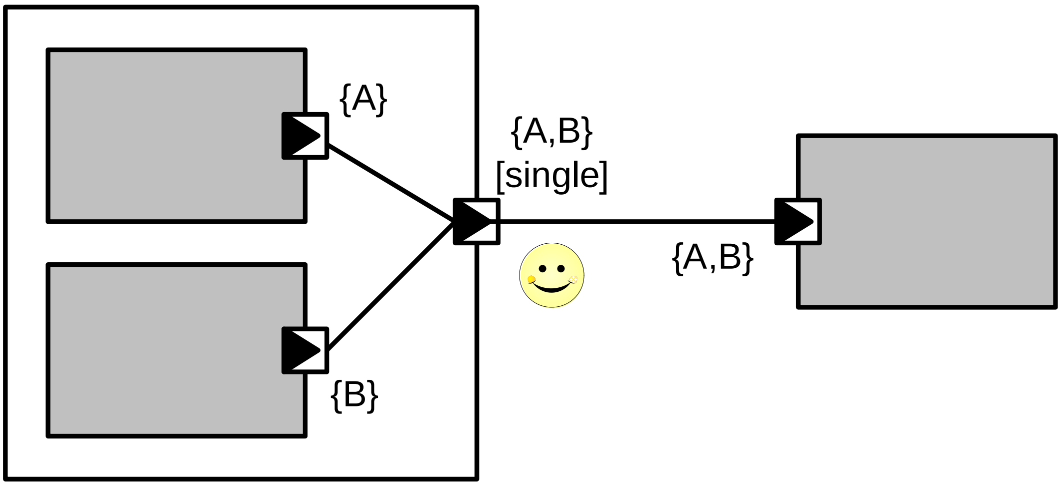

Figure 6.6: Legal merge of delegation connector

In this case the resulting communication pattern on the VFB would be 1:x, with x taking values between 0 and n. In this case the value of the DelegatedPortAnnotation should be set to single. All VariableDataPrototypes of the provided outer PortPrototypes are provided by exactly one provided inner PortPrototype.

As a variation of this theme, the next example features a PPortPrototype owned by a CompositionSwComponentType that contains the superset of dataElements {A ,B, C}.

The PPortPrototypes of the SwComponentPrototypes in turn contain subsets of dataElements, i.e. {A, B} and {B, C}. In this case the resulting communication pattern on the VFB for {B} would be n:1.

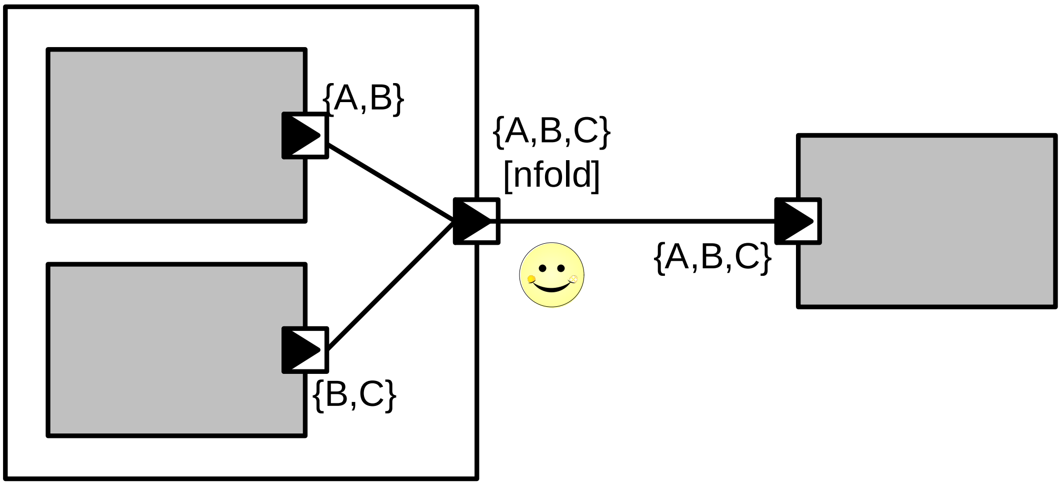

Figure 6.7: Legal merge of delegation connector

This would require the value of DelegatedPortAnnotation to be set to nfold. All dataElements of the delegation PPortPrototype are provided by at least one PPortPrototype of the SwComponentPrototypes. Therefore the criteria of entire delegation defined in chapter 6.14 are fulfilled.

The next example looks very similar. However, the subtle difference is that the second SwComponentPrototype provides dataElements {C,D} rather than {B,C}.

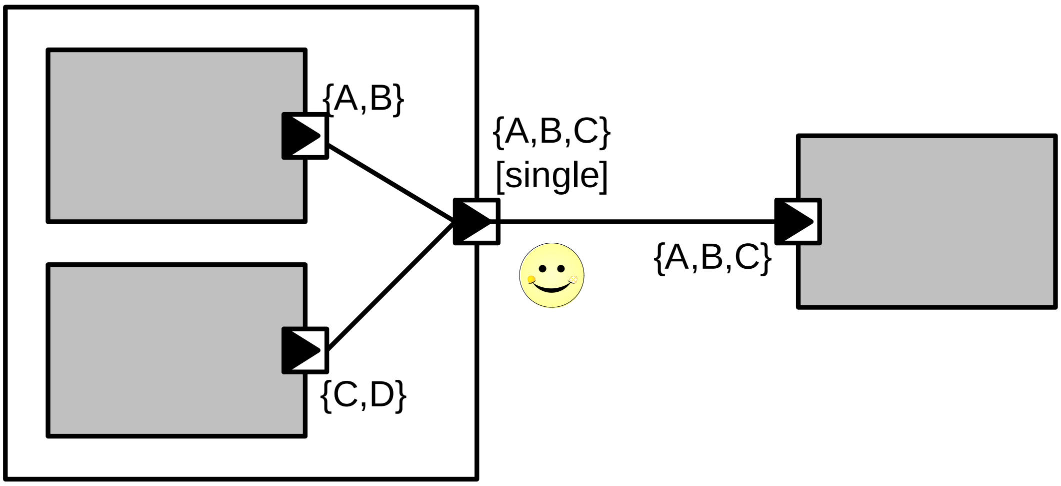

Figure 6.8: Legal merge of delegation connector

Although dataElement {D} does not appear in the delegation PPortPrototype the compatibility rules are fully satisfied with this scenario.

6.16.2.2 Illegal Use

The first example for an illegal use of splitting of dataElements suffers from the fact that not all dataElements owned by the RPortPrototypes of the SwComponentPrototypes are available from the connected RPortPrototypes owned by the CompositionSwComponentType.

Although dataElements the connections in total match ({A} and {B} are connected to a PortPrototype requiring {A,B}) the compatibility rules are not fulfilled because they apply separately for *each* SwConnector

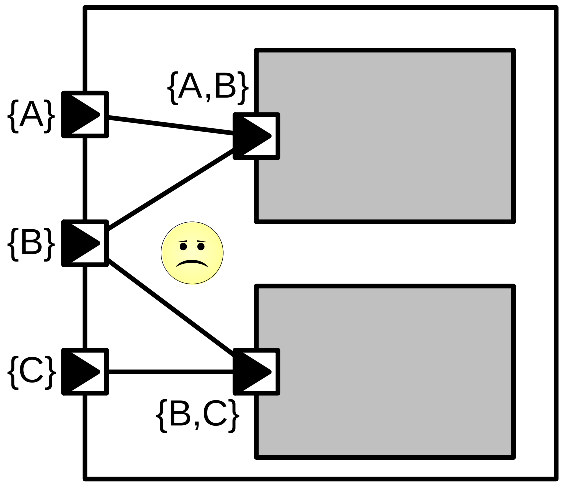

Figure 6.9: Illegal split of delegation connector

In the next example compatibility is also not fulfilled because the required dataElement {E} is not provided by the delegation RPortPrototype.

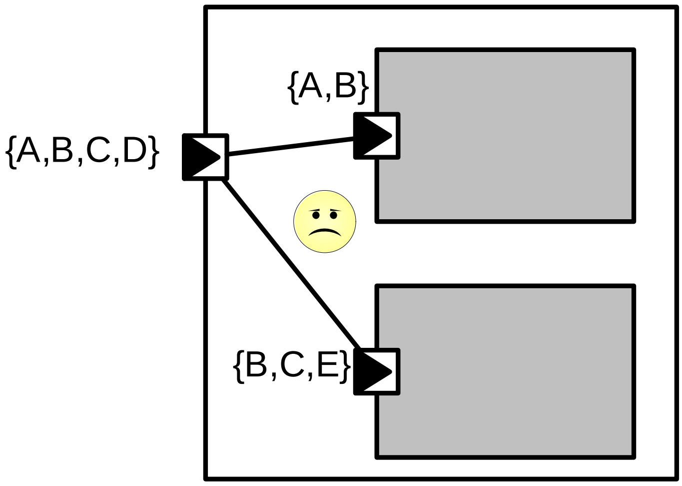

Figure 6.10: Illegal split of delegation connector

An incompatible merge of DelegationSwConnectors is sketched in Figure 6.11. In this case the dataElement {E} is *not* provided by one of the PPortPrototypes owned by the SwComponentPrototypes inside the CompositionSwComponentType.

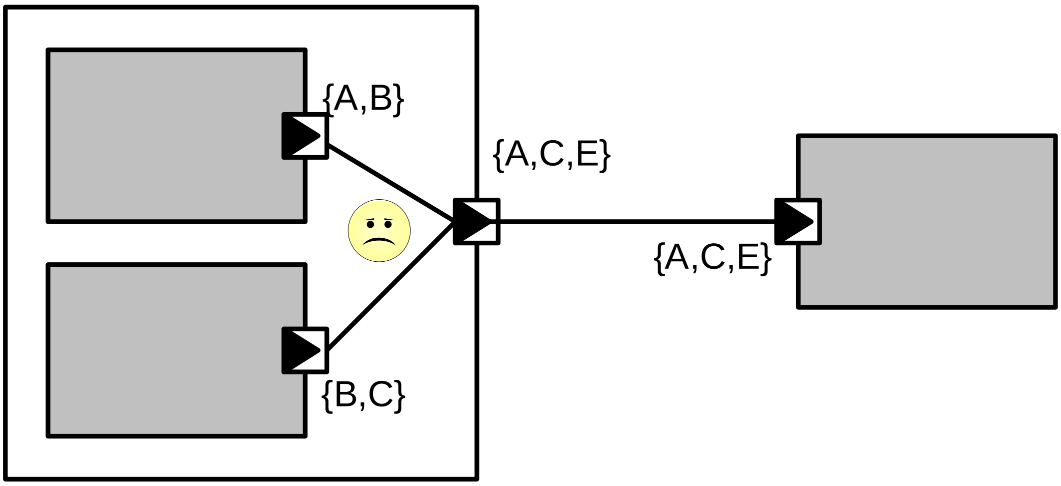

Figure 6.11: Illegal merge of delegation connector

7 Internal Behavior

7.1 Introduction

[TPS_SWCT_1075] **`SwcInternalBehavior`** ⌈ `SwcInternalBehavior` provids means for formally defining the behavior of an `AtomicSwComponentType`. ⌋*(RS_SWCT_3040)*

This chapter focuses on the description of the `SwcInternalBehavior` meta-class and the various meta-classes it aggregates. An overview of the meta-class is sketched in Figure 7.2. Please note that `SwcInternalBehavior` inherits from `InternalBehavior`.

The role of `SwcInternalBehavior` in the context of an AUTOSAR software-component is depicted in Figure 7.1. As mentioned in section 3.2, the reason to make the aggregation of `SwcInternalBehavior` to `AtomicSwComponentType` `≪atpSplitable≫` is to allow for the development of `SwcInternalBehavior` in a later process step (e.g. after the `VFB` view has been completed).

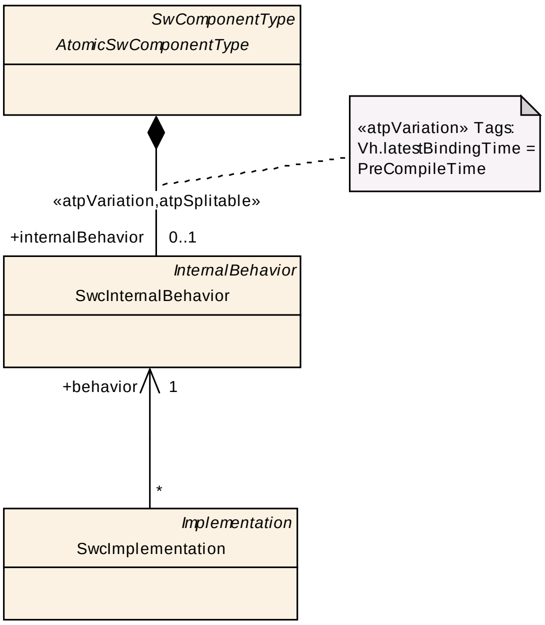

Figure 7.1: The "big picture" of `SwcInternalBehavior`

Class	**SwcInternalBehavior**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	The SwcInternalBehavior of an AtomicSwComponentType describes the relevant aspects of the software-component with respect to the RTE, i.e. the RunnableEntities and the RTEEvents they respond to.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Internal Behavior,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
arTypedPerInstanceMemory	VariableDataPrototype	*	aggr	Defines an AUTOSAR typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is TRUE or if the component defines NVRAM access via permanent blocks. The aggregation of arTypedPerInstanceMemory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
event	RTEEvent	*	aggr	This is a RTEEvent specified for the particular SwcInternalBehavior. The aggregation of RTEEvent is subject to variability with the purpose to support the conditional existence of RTE events. Note: the number of RTE events might vary due to the conditional existence of PortPrototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel
explicitInterRunnableVariable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of explicitInterRunnableVariable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
handleTerminationAndRestart	HandleTerminationAndRestartEnum	1	attr	This attribute controls the behavior with respect to stopping and restarting. The corresponding AtomicSwComponentType may either not support stop and restart, or support only stop, or support both stop and restart.

Attribute	Datatype	Mul.	Kind	Note
implicitInterRunnableVariable	VariableDataPrototype	*	aggr	Implement state message semantics for establishing communication among runnables of the same component. The aggregation of implicitInterRunnableVariable is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
includedDataTypeSet	IncludedDataTypeSet	*	aggr	The includedDataTypeSet is used by a software component for its implementation.
includedModeDeclarationGroupSet	IncludedModeDeclarationGroupSet	*	aggr	This aggregation represents the included ModeDeclarationGroups
instantiationDataDefProps	InstantiationDataDefProps	*	aggr	The purpose of this is that within the context of a given SwComponentType some data def properties of individual instantiations can be modified. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of PortPrototypes and component local memories like "perInstanceParameter" or "arTypedPerInstanceMemory". **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
perInstanceMemory	PerInstanceMemory	*	aggr	Defines a per-instance memory object needed by this software component. The aggregation of PerInstanceMemory is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
perInstanceParameter	ParameterDataPrototype	*	aggr	Defines parameter(s) or characteristic value(s) that needs to be available for each instance of the software-component. This is typically only useful if supportsMultipleInstantiation is TRUE. The aggregation of perInstanceParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel

Attribute	Datatype	Mul.	Kind	Note
portAPIOption	PortAPIOption	*	aggr	Options for generating the signature of port-related calls from a runnable to the RTE and vice versa. The aggregation of PortPrototypes is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
runnable	RunnableEntity	1..*	aggr	This is a RunnableEntity specified for the particular SwcInternalBehavior. The aggregation of RunnableEntity is subject to variability with the purpose to support the conditional existence of RunnableEntities. Note: the number of RunnableEntities might vary due to the conditional existence of PortPrototypes using DataReceivedEvents or due to different scheduling needs of algorithms. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel
serviceDependency	SwcServiceDependency	*	aggr	Defines the requirements on AUTOSAR Services for a particular item. The aggregation of SwcServiceDependency is subject to variability with the purpose to support the conditional existence of ports as well as the conditional existence of ServiceNeeds. The SwcServiceDependency owned by an SwcInternalBehavior can be located in a different physical file in order to support that SwcServiceDependency might be provided in later development steps or even by different expert domain (e.g OBD expert for Obd related Service Needs) tools. Therefore the aggregation is «atpSplitable». **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
sharedParameter	ParameterDataPrototype	*	aggr	Defines parameter(s) or characteristic value(s) shared between SwComponentPrototypes of the same SwComponentType The aggregation of sharedParameter is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel

Attribute	Datatype	Mul.	Kind	Note
supportsMultipleInstantiation	Boolean	1	attr	Indicate whether the corresponding software-component can be multiply instantiated on one ECU. In this case the attribute will result in an appropriate component API on programming language level (with or without instance handle).
variationPointProxy	VariationPointProxy	*	aggr	Proxy of a variation points in the C/C++ implementation.

Table 7.1: SwcInternalBehavior

Enumeration	HandleTerminationAndRestartEnum
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior
Note	Controls the behavior of an AtomicSwComponentType with respect to stop and restart.
Literal	**Description**
canBeTerminated	Supports termination.
canBeTerminatedAndRestarted	Supports termination and restarting.
noSupport	Stop and restart is not supported at all.

Table 7.2: HandleTerminationAndRestartEnum

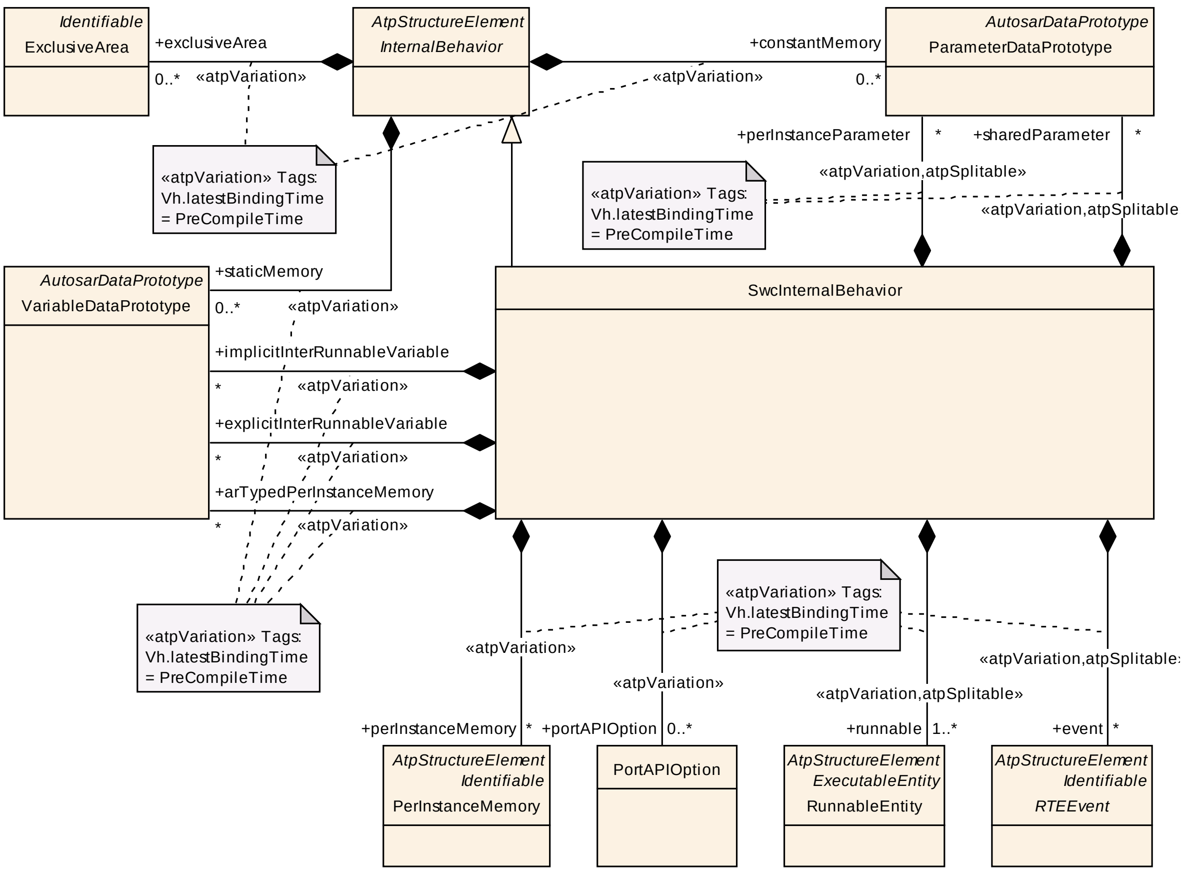

Figure 7.2: `SwcInternalBehavior`

7.2 Runnable Entity

The concept of RunnableEntity (more details can be found in Figure 7.3) is defined in the specification of the Virtual Function Bus [3].

[TPS_SWCT_1030] RunnableEntity ⌈ RunnableEntitys are the smallest code-fragments that are provided by a software-component and are (at least indirectly) a subject for scheduling by the underlying operating system. ⌋*(RS_SWCT_0070, RS_SWCT_0090, RS_SWCT_3050)*

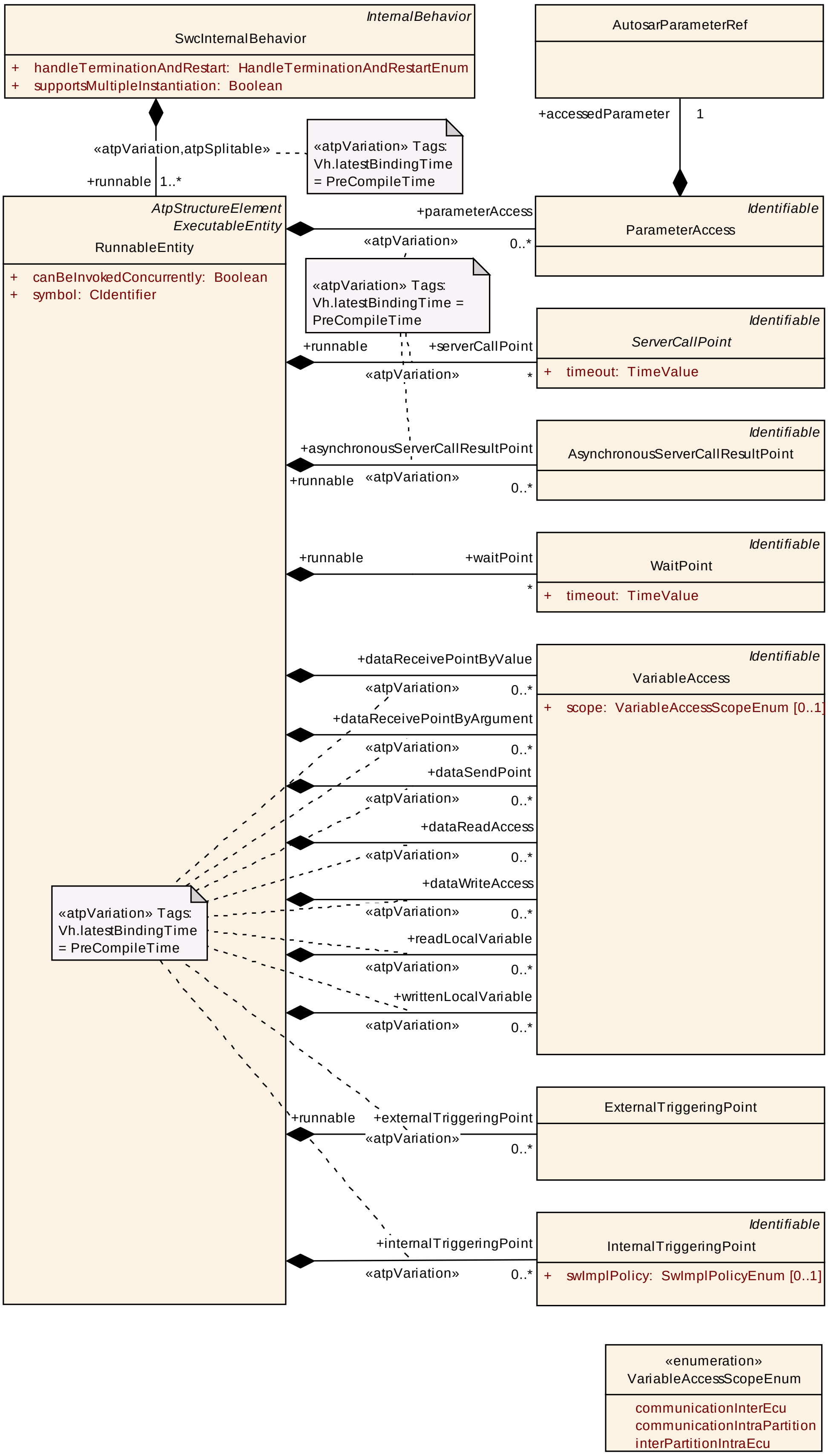

Figure 7.3: Details of `RunnableEntity`

[TPS_SWCT_1097] `CompositionSwComponentType` cannot have `RunnableEntitys` ⌈ It is intentionally not possible for `CompositionSwComponentType` to define a `SwcInternalBehavior`. Consequently, `CompositionSwComponentTypes` don't have `RunnableEntitys` by themselves. ⌋*(RS_SWCT_0070, RS_SWCT_0090, RS_SWCT_3050)*

[TPS_SWCT_1098] Only `AtomicSwComponentType` can have `RunnableEntitys` ⌈ Only the `AtomicSwComponentType` that are populating a `CompositionSwComponentType` in the role of `SwComponentPrototypes` may have `RunnableEntitys`. ⌋*(RS_SWCT_0070, RS_SWCT_0090, RS_SWCT_3050)*

This correlation is depicted in Figure 7.4.

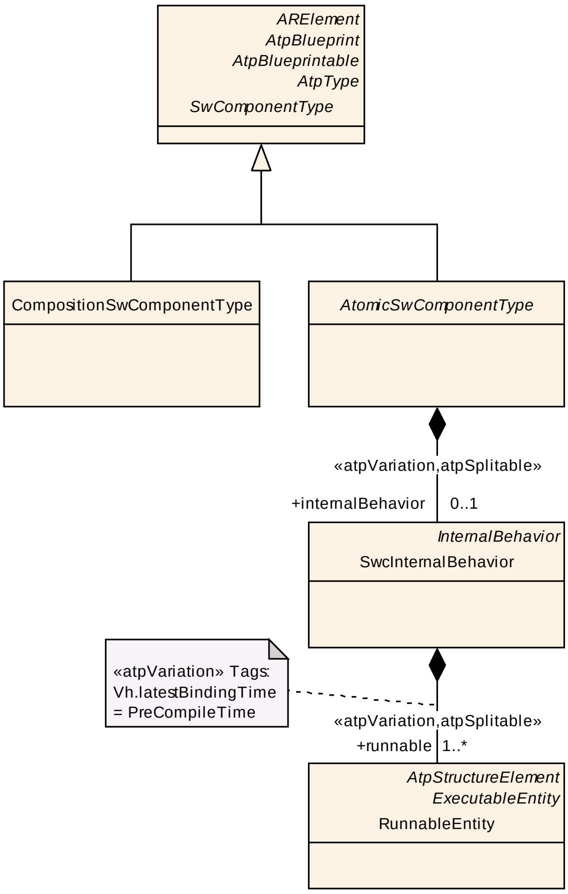

Figure 7.4: Only `AtomicSwComponentTypes` may have `RunnableEntitys`

Please note that `RunnableEntitys` exist in several categories that have different properties. Please find more explanation about categories of `RunnableEntitys` in section 7.2.4.4.

Class	RunnableEntity			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior			
Note	A RunnableEntity represents the smallest code-fragment that is provided by an AtomicSwComponentType and are executed under control of the RTE. RunnableEntities are for instance set up to respond to data reception or operation invocation on a server.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Executable Entity,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
argument (ordered)	RunnableEntity Argument	*	aggr	This represents the formal definition of a an argument to a RunnableEntity.
asynchronousServerCallResultPoint	AsynchronousServerCallResultPoint	*	aggr	The server call result point admits a runnable to fetch the result of an asynchronous server call. The aggregation of AsynchronousServerCallResultPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes and the variant existence of server call result points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
canBeInvokedConcurrently	Boolean	1	attr	If the value of this attribute is set to TRUE the enclosing RunnableEntity can be invoked concurrently (even for one instance of the corresponding AtomicSwComponentType). This implies that it is the responsibility of the implementation of the RunnableEntity to take care of this form of concurrency. Note that the default value of this attribute is FALSE.
dataReadAccess	VariableAccess	*	aggr	RunnableEntity has implicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataReadAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataReadAccess in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	Datatype	Mul.	Kind	Note
dataReceivePointByArgument	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The result is passed back to the application by means of an argument in the function signature. The aggregation of dataReceivePointByArgument is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
dataReceivePointByValue	VariableAccess	*	aggr	RunnableEntity has explicit read access to dataElement of a sender-receiver PortPrototyoe or nv data of a nv data PortPrototype. The result is passed back to the application by means of the return value. The aggregation of dataReceivePointByValue is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of data receive points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
dataSendPoint	VariableAccess	*	aggr	RunnableEntity has explicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataSendPoint is subject to variability with the purpose to support the conditional existence of sender receiver PortPrototype or the variant existence of data send points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
dataWriteAccess	VariableAccess	*	aggr	RunnableEntity has implicit write access to dataElement of a sender-receiver PortPrototype or nv data of a nv data PortPrototype. The aggregation of dataWriteAccess is subject to variability with the purpose to support the conditional existence of sender receiver ports or the variant existence of dataWriteAccess in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	Datatype	Mul.	Kind	Note
externalTriggeringPoint	ExternalTriggeringPoint	*	aggr	The aggregation of ExternalTriggeringPoint is subject to variability with the purpose to support the conditional existence of trigger ports or the variant existence of external triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
internalTriggeringPoint	InternalTriggeringPoint	*	aggr	The aggregation of InternalTriggeringPoint is subject to variability with the purpose to support the variant existence of internal triggering points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
modeAccessPoint	ModeAccessPoint	*	aggr	The runnable has a mode access point. The aggregation of ModeAccessPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode access points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
modeSwitchPoint	ModeSwitchPoint	*	aggr	The runnable has a mode switch point. The aggregation of ModeSwitchPoint is subject to variability with the purpose to support the conditional existence of mode ports or the variant existence of mode switch points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
parameterAccess	ParameterAccess	*	aggr	The presence of a ParameterAccess implies that a RunnableEntity needs read only access to a ParameterDataPrototype which may either be local or within a PortPrototype. The aggregation of ParameterAccess is subject to variability with the purpose to support the conditional existence of parameter ports and component local parameters as well as the variant existence of ParameterAccess (points) in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Attribute	Datatype	Mul.	Kind	Note
readLocal Variable	VariableAccess	*	aggr	The presence of a readLocalVariable implies that a RunnableEntity needs read access to a VariableDataPrototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of readLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicitInterRunnableVariable or the variant existence of readLocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
serverCall Point	ServerCallPoint	*	aggr	The RunnableEntity has a ServerCallPoint. The aggregation of ServerCallPoint is subject to variability with the purpose to support the conditional existence of client server PortPrototypes or the variant existence of server call points in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
symbol	CIdentifier	1	ref	The symbol describing this RunnableEntity's entry point. This is considered the API of the RunnableEntity and is required during the RTE contract phase.
waitPoint	WaitPoint	*	aggr	The WaitPoint associated with teh RunnableEntity.
writtenLoc alVariable	VariableAccess	*	aggr	The presence of a writtenLocalVariable implies that a RunnableEntity needs write access to a VariableDataPrototype in the role of implicitInterRunnableVariable or explicitInterRunnableVariable. The aggregation of writtenLocalVariable is subject to variability with the purpose to support the conditional existence of implicitInterRunnableVariable and explicitInterRunnableVariable or the variant existence of writtenLocalVariable (points) in the implementation. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 7.3: RunnableEntity

[TPS_SWCT_1302] Semantics of `minimumStartInterval` ⌈ The attribute `minimumStartInterval` defines the time interval that the RTE will guarantee to not go below between scheduling two consecutive executions of the corresponding `RunnableEntity`. ⌋

[TPS_SWCT_1303] symbol attribute describes the RunnableEntity's entry point ⌈ The symbol attribute is describing the RunnableEntity's entry point. ⌋

[constr_2025] Uniqueness of symbol attributes ⌈ In the context of a single ECU, the values of the symbol attribute of all deployed RunnableEntitys shall be unique. ⌋

Please note that the formal definition of the semantics of a RunnableEntity has strong relations to the specification of the AUTOSAR RTE [2]. The definition of the RTE semantics, however, is not in the scope of this document.

However, the formal definition requires some background discussion that can't be completely left out of this document. Otherwise the meaning of specific model elements could not be understood properly.

7.2.1 Concurrency and Reentrancy of a RunnableEntity that cannot be Invoked Concurrently

This section applies to the case that the value of the attribute canBeInvokedConcurrently is FALSE. During runtime, each RunnableEntity of each instance of an AtomicSwComponentType is in a specific run-time state.

The details of the definition and semantics of run-time states can be found in [2]. Nevertheless, this chapter contains a brief description of the fundamental concepts in order to properly being able to discuss the formal modeling of RunnableEntitys.

[TPS_SWCT_1313] Conditions for a transition from suspended to to be started ⌈ The SwcInternalBehavior describes for each RunnableEntity the conditions for a transition from suspended to to be started should occur. This is done using the concept of an RTEEvent. ⌋

When a RunnableEntity is in state to be started, the RTE can decide to start running the RunnableEntity. The delay between entering the state to be started (e.g. a message has been received in response to which the RunnableEntity should run) and moving into the state running (the first instruction of the RunnableEntity has been executed) depends on the scheduling strategy of the RTE, i.e. the mapping of RunnableEntitys on AUTOSAR OS tasks.

The transition from the state running into the state suspended is in the hands of the RunnableEntity: the transition occurs when the RunnableEntity returns (thereby handing over control to the AUTOSAR OS [25]). Some RunnableEntitys (like cat. 2 RunnableEntitys) might never return to the suspended state once they entered the running state.

They might enter the preempted state when being preempted. The same applies if a RunnableEntity needs to wait for a WaitPoint to be unblocked.

[TPS_SWCT_1304] Cat. 1A and 1B `RunnableEntity`s will eventually terminate ⌈ Cat. 1A and 1B `RunnableEntity`s will eventually return after having executed a specific finite algorithm (the execution time of which might be provided). ⌋

[TPS_SWCT_1305] `RunnableEntity` as one that cannot be invoked concurrently ⌈ In case the `SwcInternalBehavior` defines a `RunnableEntity` as one that cannot be invoked concurrently it is the responsibility of the RTE to make sure that the `RunnableEntity` is never started concurrently (for example, in two different AUTOSAR OS tasks). This implies that the implementation of the `AtomicSwComponentType` does not need to worry about concurrency issues. ⌋

For example: The internal behavior of an `AtomicSwComponentType` MyComponentType describes a `RunnableEntity` R1 which should be enabled when an operation on a client-server p-port of the `AtomicSwComponentType` is invoked. The `AtomicSwComponentType` specifies that the `RunnableEntity` R1 cannot be invoked concurrently.

The `AtomicSwComponentType` MyComponentType is instantiated on an ECU. When a call of the operation is received, the corresponding instance of the `RunnableEntity` R1 is enabled and the RTE will start executing the `RunnableEntity` (the `RunnableEntity` is in state `running`) in a task eventually managed by the AUTOSAR OS.

If another call of the operation is received while the `RunnableEntity` is in state `running` it is not allowed that the RTE runs the `RunnableEntity` again in a second task. Rather, the RTE has to wait (and maybe queue the second incoming request) until the `RunnableEntity` has returned and has moved to the `suspended` state.

7.2.2 Concurrency and Reentrancy of a RunnableEntity that can be Invoked Concurrently

This section applies to the case that the value of the attribute `canBeInvokedConcurrently` is set to `TRUE`.

In this case, it is allowed that the same `RunnableEntity` is running several times concurrently in different AUTOSAR OS tasks. This implies that the state machine defined in [2] is not the state of the `RunnableEntity` any more, but can be cloned an arbitrary number of times.

[TPS_SWCT_1306] Software-component description itself does not put any bounds on the number of concurrent invocations of a `RunnableEntity` ⌈ The software-component description itself does not put any bounds on the number of concurrent invocations of the `RunnableEntity` that are allowed. The software-component description only specifies whether the `RunnableEntity` can be invoked concurrently or not.

Allowing concurrent invocation of a RunnableEntity implies that the implementation of the AtomicSwComponentType needs to take care of this additional form of concurrency. ⌋

For example: The SwcInternalBehavior of a component-type MyComponentType describes a RunnableEntity R1 which should be enabled when a ClientServerOperation on a PPortPrototype typed by a ClientServerInterface of the AtomicSwComponentType is invoked.

The AtomicSwComponentType specifies that the RunnableEntity R1 can be invoked concurrently. The AtomicSwComponentType MyComponentType is instantiated on an ECU. When a call of the ClientServerOperation is received the corresponding instance of the RunnableEntity R1 is enabled and the RTE will start executing the RunnableEntity (the RunnableEntity is in state running) in a task eventually managed by the AUTOSAR OS.

If another call of the ClientServerOperation is received, it is allowed that the same RunnableEntity is started again in a different task.

A typical use-case of concurrent RunnableEntitys is the implementation of AUTOSAR services. The AUTOSAR services will typically take care of concurrency internally: several software-components can directly use the services in parallel. The ECU-integrator could then decide that the RunnableEntity implementing the AUTOSAR service runs directly in the context (in the task) of the AtomicSwComponentType invoking the service.

This is a very efficient and direct coupling between the client and the server: the connector between the client and the server is reduced to a local function-call.

7.2.3 Timed Activation of Runnable Entities

In many cases, RunnableEntitys need to be activated in response to timing events rather than related to communication (e.g. the reception of a response to an asynchronous operation invocation). Many RunnableEntitys will need to run cyclically with a fixed rate.

The approach taken in the software-component description is to define so-called TimingEvents (please find more details in Figure 7.5) as special kinds of RTEEvents. So far, only one kind of timing-related RTEEvent has been defined: a simple periodic TimingEvent.

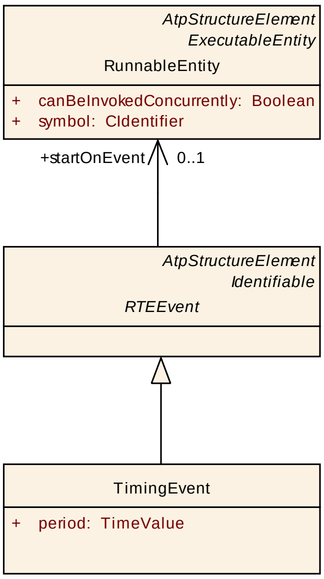

Figure 7.5: Periodic activation of RunnableEntities

Therefore, if the SwcInternalBehavior of an AtomicSwComponentType requires that the RTE executes certain RunnableEntitys periodically, the description needs to define a TimingEvent with the desired period. This TimingEvent then contains a reference to the Runnable that needs to be executed with this period.

7.2.4 Additional Remarks and Clarifications

7.2.4.1 Reentrancy and Multiple Instantiation

Note that it is useful to consider the combinations of the attributes InternalBehavior.supportsMultipleInstantiation and RunnableEntity.canBeInvokedConcurrently.

[TPS_SWCT_1307] supportsMultipleInstantiation vs. canBeInvokedConcurrently ⌈

supportsMultipleInstantiation	canBeInvokedConcurrently	Implication for an implementation of a RunnableEntity
FALSE	FALSE	This implies that the implementation of the RunnableEntity will never be invoked concurrently from several tasks. The implementation does not need to care about reentrancy issues and can typically use static variables to store state.
TRUE	FALSE	In case there are several instances of the same AtomicSwComponentType on the local ECU, the implementation of the RunnableEntity can still be invoked concurrently from several tasks. However, there will be no concurrent invocations of the implementation with the same instance handle. To ensure that this is safe, the implementation will typically use per-instance memory.

FALSE/TRUE	TRUE	In this case the RunnableEntity can be invoked concurrently from several tasks, even with the same instance handle.

Table 7.4: supportsMultipleInstantiation vs. canBeInvokedConcurrently

⌋

[TPS_SWCT_1308] Combination of supportsMultipleInstantiation=FALSE and canBeInvokedConcurrently=FALSE ⌈The combination of supportsMultipleInstantiation=FALSE and canBeInvokedConcurrently=FALSE is uncritical in case that each RunnableEntity is implemented by its own C-function. ⌋

In case the implementation of a AtomicSwComponentType decides to map several RunnableEntitys to the same symbol there are reentrancy problems to be sorted out. However, this scenario is not supported by RTE [2] anyway and shall therefore be avoided.

[constr_1094] Usage of symbol of RunnableEntity ⌈It is not allowed that several RunnableEntitys share the same value of the attribute symbol. ⌋

7.2.4.2 Reentrancy and "Library Functions"

Note that all code that is called by different RunnableEntitys (like e.g. library routines, etc.) shall obviously be reentrant. A filter algorithm implemented in C, for example, is not allowed to store values from previous runs by means of static variables or variables with external binding.

7.2.4.3 Compatibility of ClientServerOperations triggering the same RunnableEntity

[TPS_SWCT_1309] signature of a RunnableEntity depends on the connected RTEEvent ⌈The signature of a RunnableEntity depends on the connected RTEEvent. Multiple OperationInvokedEvents are only supported if all referred ClientServerOperations would result in the same RunnableEntity signature for the server RunnableEntity. ⌋

[constr_2000] Compatibility of ClientServerOperations triggering the same RunnableEntity ⌈The ClientServerOperations are considered compatible if the number of arguments (which can be ArgumentDataPrototypes or related PortDefinedArgumentValues) is equal and the corresponding arguments (i.e. first argument on both sides, second argument on both sides, etc.) are compatible.

In particular, this means that:

- for combinations of ArgumentDataPrototypes and ArgumentDataPrototypes where the ServerArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

  In case of data types of category STRUCTURE all by order matching ImplementationDataTypeElements shall be named equally.

- for combinations of PortDefinedArgumentValues and ArgumentDataPrototypes where the ServerArgumentImplPolicy is set to useArgumentType the referred ImplementationDataTypes shall be compatible.

- for combinations of ArgumentDataPrototypes and ArgumentDataPrototypes where the ServerArgumentImplPolicy is set to useArrayBaseType the referred ImplementationDataTypes of category ARRAY shall have compatible ImplementationDataTypeElements.

  In case of ImplementationDataTypeElements of category STRUCTURE all by order matching ImplementationDataTypeElements of the structure shall be named equally.

- for ArgumentDataPrototypes where the ServerArgumentImplPolicy is set to useVoid an arbitrary ImplementationDataType is referred to.

In addition, it is required that the return value defined on both sides shall match (in terms of Std_ReturnType vs. void) and also the possibleErrors are compatible.
⌋

7.2.4.4 Categories of Runnable Entities

[TPS_SWCT_1310] Categories of RunnableEntitys ⌈ RunnableEntitys are subdivided into the following categories:

Category 1
Category 1 RunnableEntitys do not have WaitPoints and are required to terminate in a finite amount of time. Category 1 is divided into two subcategories: Category 1A and Category 1B. Category 1A RunnableEntitys are only allowed to use implicit API's. Category 1B RunnableEntitys are additionally allowed to invoke a server and use explicit API's.

Category 2
In contrast to Category 1 RunnableEntitys, RunnableEntitys of category 2 always aggregate at least one WaitPoint, for more details see Figure 7.3[1]. Typically, such a RunnableEntity implements an internal loop where one iteration through the loop is triggered whenever a WaitPoint is resolved. ⌋

[1] Category 2 RunnableEntitys usually have to be mapped to *Extended Tasks*, because only extended tasks provide the task state WAITING.

7.2.4.5 Arguments of a Runnable Entity

In many cases an RTE generator will be able to figure out not only the number and data type of arguments to a `RunnableEntity` but also the name of the arguments. In some cases, however, formal support from the upstream templates is required to facilitate this task.

[TPS_SWCT_1311] Name of an operation argument ⌈ This support is available by means of the meta-class `RunnableEntityArgument` that contributes the name of the argument by means of the value of the attribute `symbol`. As a `RunnableEntity` might need to define many arguments the aggregation of `RunnableEntityArgument` at `RunnableEntity` has the multiplicity 0..* and as the order of these arguments is significant the meta-model defines the aggregation as ordered[2]. ⌋

[constr_1164] Number of `arguments` owned by a `RunnableEntity` ⌈ The number of owned `RunnableEntityArgument`s in the role `argument` of a given `RunnableEntity` shall be identical to the number of applicable `portArgValue`s of the `PortAPIOption` that references the `PortPrototype` that in turn is referenced by the `OperationInvokedEvent` that references the `RunnableEntity` **plus** the number of `ArgumentDataPrototype`s aggregated in the role `argument` by the `ClientServerOperation` referenced by said `OperationInvokedEvent`. ⌋

[constr_1165] Applicability of `RunnableEntityArgument` ⌈ The existence of a `RunnableEntityArgument` is limited to `RunnableEntity`s triggered by a `ClientServerOperation`. ⌋

[TPS_SWCT_1312] `RunnableEntity` has a mapping to `BswModuleEntry` ⌈ The existence of `RunnableEntityArgument`s in the role `argument` owned by a `RunnableEntity` shall be **ignored** by an RTE generator if a mapping to a `BswModuleEntry` exists. In this case the name of arguments to the `RunnableEntity` shall be derived from the applicable `SwServiceArg`s owned by the mapped `BswModuleEntry`. ⌋

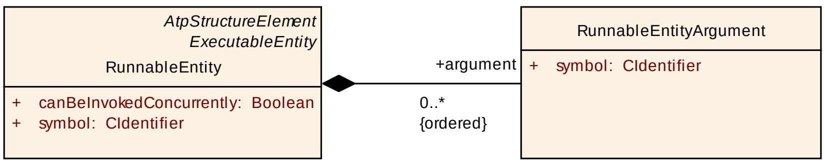

Figure 7.6: Arguments of a `RunnableEntity`

[2] as the arguments are **ordered** they do not need to be `Referrable` in order to be able to identify individual `argument`s

Class	**RunnableEntityArgument**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Runnable Entity			
Note	This meta-class represents the ability to provide specific information regarding the arguments to a RunnableEntity.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
symbol	CIdentifier	1	ref	This represents the symbol to be generated into the actual signature on the level of the C programming language.

Table 7.5: RunnableEntityArgument

7.3 RTEEvent

During execution, several `RTEEvent`s will occur, such as the reception of a remote invocation of a `ClientServerOperation` on a `PPortPrototype` or a timeout on an `RPortPrototype` that is not receiving the `VariableDataPrototype`s it expects to receive.

[TPS_SWCT_1314] **`RTEEvent`** ⌈ The description of an `RTEEvent` includes two aspects:

1. defining an `RTEEvent`
2. defining how the RTE should deal with the `RTEEvent` when it occurs.

⌋

Class	**RTEEvent (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	Abstract base class for all RTE-related events			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
disabledMode	ModeDeclaration	*	iref	Reference to the Modes that disable the Event.
startOnEvent	RunnableEntity	0..1	ref	RunnableEntity starts when the corresponding RTEEvent occurs.

Table 7.6: RTEEvent

Class	**AsynchronousServerCallReturnsEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when an asynchronous server call is finished.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	AsynchronousServerCallResultPoint	1	ref	The referenced AsynchronousServerCallResultPoint which is raises the RTEEvent in case of returning asynchronous server call.

Table 7.7: AsynchronousServerCallReturnsEvent

Class	**DataSendCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced data elements have been sent or an error occurs.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	VariableAccess	1	ref	The variable access that triggers the event.

Table 7.8: DataSendCompletedEvent

Class	**DataWriteCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised if an implicit write access was successful or an error occurred.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	VariableAccess	1	ref	The variable access that triggers the event.

Table 7.9: DataWriteCompletedEvent

Class	**DataReceivedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced data elements are received.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	1	iref	Data element referenced by event

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

Table 7.10: DataReceivedEvent

Class	**DataReceiveErrorEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised by the RTE when the Com layer detects and notifies an error concerning the reception of the referenced data element.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	1	iref	Data element referenced by event

Table 7.11: DataReceiveErrorEvent

Class	**OperationInvokedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The OperationInvokedEvent references the ClientServerOperation invoked by the client.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1	iref	The operation to be executed as the consequence of the event.

Table 7.12: OperationInvokedEvent

Class	**TimingEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	TimingEvent references the RunnableEntity that need to be started in response to the TimingEvent			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
period	TimeValue	1	attr	Period of timing event in seconds. The value of this attribute must be greater than zero.

Table 7.13: TimingEvent

[constr_2031] Period of `TimingEvent` shall be greater than 0 ⌈ The value of the attribute period of `TimingEvent` shall be greater than 0. ⌋

Class	**BackgroundEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is used to trigger RunnableEntities that are supposed to be executed in the background.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.14: BackgroundEvent

Class	**SwcModeSwitchEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised upon a received mode change.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
activation	ModeActivation Kind	1	attr	Specifies if the event is activated on entering or exiting the referenced Mode.
mode (ordered)	ModeDeclaration	1..2	iref	Reference to one or two Modes that initiate the Mode Switch Event.

Table 7.15: SwcModeSwitchEvent

Class	**ModeSwitchedAckEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced modes have been received or an error occurs.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
eventSource	ModeSwitchPoint	1	ref	Mode switch point that triggers the event.

Table 7.16: ModeSwitchedAckEvent

Class	**ExternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced trigger have been occurred.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
trigger	Trigger	1	iref	Reference to the applicable Trigger.

Table 7.17: ExternalTriggerOccurredEvent

Class	InternalTriggerOccurredEvent			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced internal trigger have been occurred.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	Datatype	Mul.	Kind	Note
eventSource	InternalTriggeringPoint	1	ref	Internal Triggering Point that triggers the event.

Table 7.18: InternalTriggerOccurredEvent

[TPS_SWCT_1315] Interaction of `RunnableEntity` with `RTEEvent` ⌈ As described in the Virtual Functional Bus specification [3], the `RunnableEntity`s of an `AtomicSwComponentType` can interact with the occurrence of such `RTEEvent`s in two ways:

- the RTE can be instructed to enable a specific `RunnableEntity` when the `RTEEvent` occurs
- the RTE can provide `WaitPoint`s, that allow a `RunnableEntity` to block until an `RTEEvent` in a set of `RTEEvent`s occurs.

⌋

7.3.1 Defining an Event

The description of the `SwcInternalBehavior` includes a description of all `RTEEvent`s that the `SwcInternalBehavior` of the `AtomicSwComponentType` relies on.

[TPS_SWCT_1316] Abstract base class `RTEEvent` ⌈ This `RTEEvent` shows up as an "abstract" base-class (see e.g. Figure 7.7) in the meta-model: the exact attributes of the `RTEEvent` depend on the specific sub-class of `RTEEvent` that is used for the purpose. ⌋

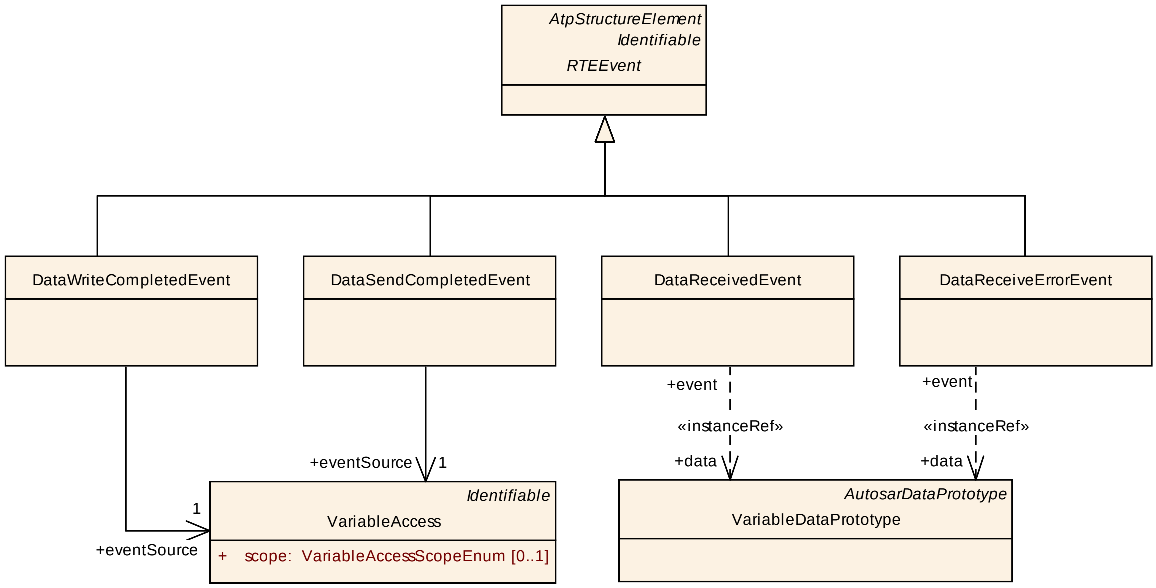

Figure 7.7: `RTEEvents` used in the context of sender/receiver communication

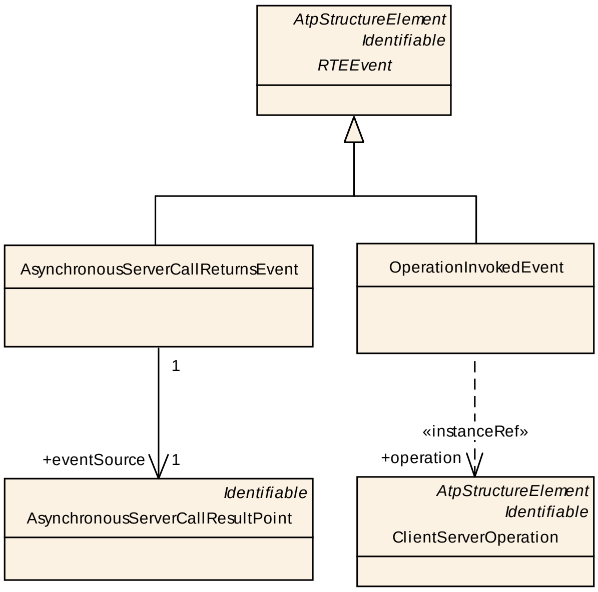

Figure 7.8: `RTEEvents` used in the context of client/server communication

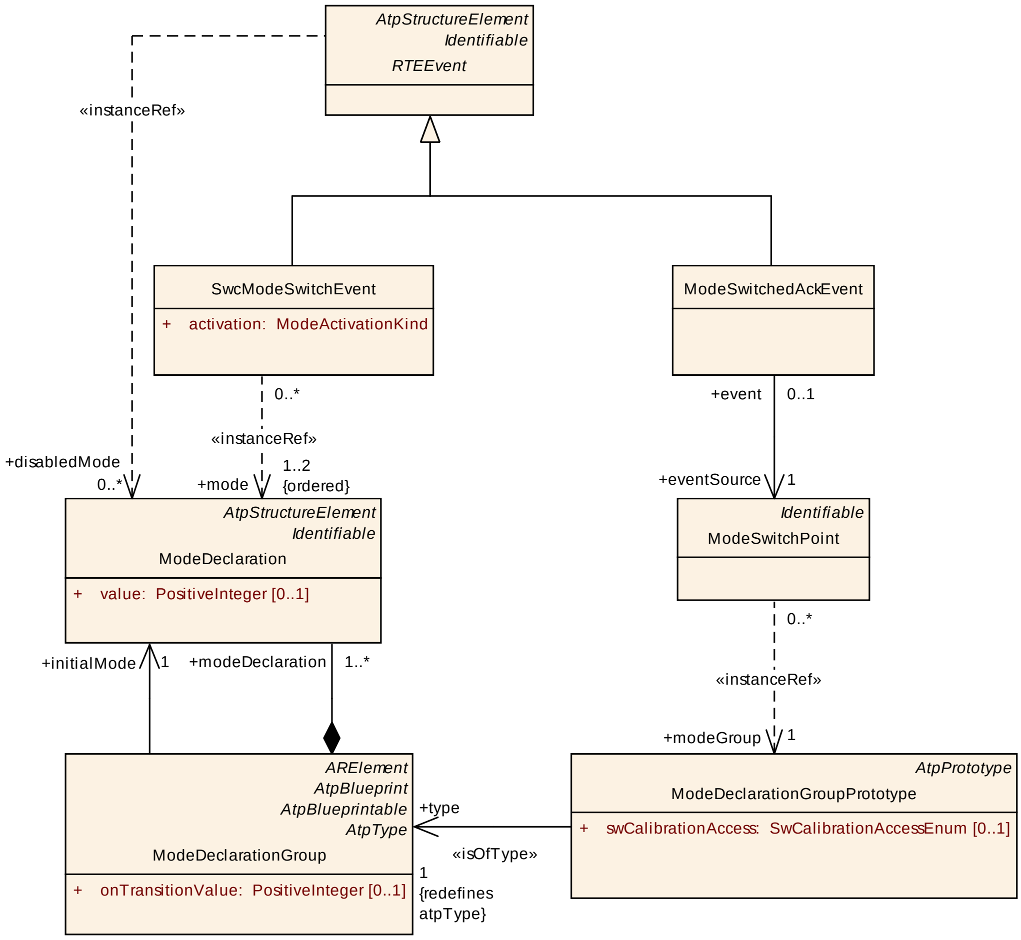

Figure 7.9: `RTEEvents` used in the context of mode communication

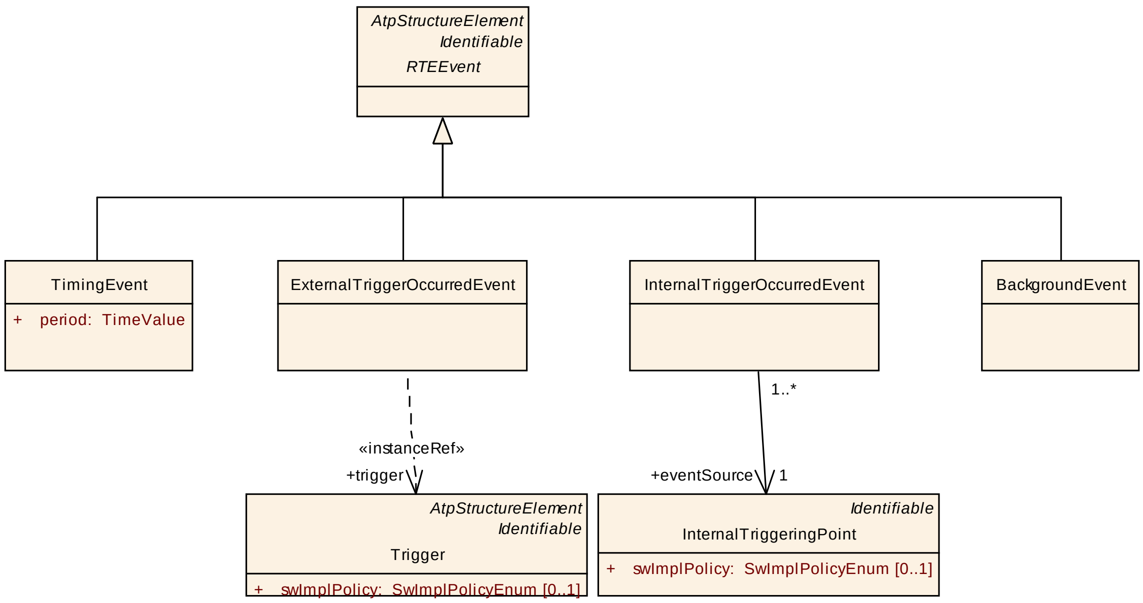

Figure 7.10: `RTEEvents` for purposes other than communication

The details of the various kinds of concrete RTEEvents (such as the TimingEvent, DataSendCompletedEvent, etc.), is described in chapters 7.5.1, 7.5.2 and 7.2.3.

7.3.2 Defining how to Respond to an Event

[TPS_SWCT_1317] RTE triggers RunnableEntity in response to occurring RTEEvent ⌈ If the software-component description contains a reference from an RTEEvent to a RunnableEntity it is the responsibility of the RTE to trigger the execution of the corresponding RunnableEntity when the RTEEvent occurs. ⌋

[TPS_SWCT_1318] RunnableEntity and WaitPoint ⌈ In case the RunnableEntity wants to block and wait for RTEEvents (which makes the RunnableEntity into a cat. 2 RunnableEntity), the description of the RunnableEntity may include the definition of a WaitPoint.

Such a WaitPoint (see Figure 7.11) contains a reference to an RTEEvent that can unblock the specific WaitPoint. In other words: the WaitPoint will block until the referenced RTEEvents occurs or the period specified in the attribute timeout expires. ⌋

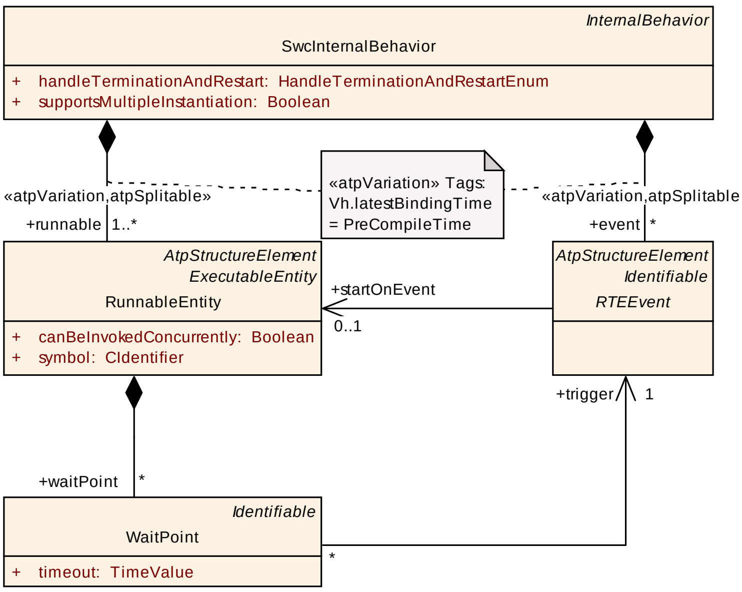

Figure 7.11: Description of the interaction between an RTEEvent and RunnableEntitys

[constr_1090] WaitPoint and RunnableEntity ⌈ A single RunnableEntity can actually wait only at a single WaitPoint provided that the RunnableEntity can only be scheduled a single time[3]. ⌋

[constr_1091] RTEEvents that can unblock a WaitPoint ⌈ As also explained in [2], the only RTEEvents that are qualified for unblocking a WaitPoint are:

[3]This constraint is valid at least in the OSEK standard where an extended task (that can have wait points) can only exist a single time in the context of the scheduler.

- DataReceivedEvent
- DataSendCompletedEvent
- ModeSwitchedAckEvent
- AsynchronousServerCallReturnsEvent

⌋

[TPS_SWCT_1319] RTEEvent can be used to trigger WaitPoints in different RunnableEntitys ⌈ It is in general possible that a single RTEEvent can be used to trigger WaitPoints in different RunnableEntitys. ⌋

Concerning DataReceivedEvents consider as well [constr_2021].

Class	**WaitPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This defines a wait-point for which the RunnableEntity can wait.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
timeout	TimeValue	1	attr	Time in seconds before the WaitPoint times out and the blocking wait call returns with an error indicating the timeout.
trigger	RTEEvent	1	ref	This is the RTEEvent this WaitPoint is waiting for.

Table 7.19: WaitPoint

[constr_1096] ModeSwitchEvent and WaitPoint ⌈ A RunnableEntity that has a WaitPoint shall not be referenced by a ModeSwitchEvent. ⌋

[TPS_SWCT_1320] RunnableEntitys of category 2 ⌈ RunnableEntitys that aggregate a WaitPoint are by definition of category 2 and therefore are not required to terminate ever. It is therefore difficult to let a RunnableEntity of category 2 implement a mode switch. ⌋

[constr_1097] RunnableEntity that has a WaitPoint ⌈ A RunnableEntity that has a WaitPoint shall not be referenced by a RTEEvent that has a reference in the role disabledInMode. ⌋

[TPS_SWCT_1324] Mode switches need to be completed in finite time ⌈ Mode switches need to be completed in finite time and a RunnableEntity that has a WaitPoint can never guarantee that the WaitPoint is resolved within finite time. ⌋

In addition to this, the RunnableEntity with a WaitPoint that would be affected by a mode disabling would typically already run when the mode disabling applies. It could not be terminated at this point in time.

7.4 Communication among Runnable Entities

[TPS_SWCT_1321] Communication among RunnableEntitys ⌈ It is taken for granted that particular RunnableEntitys within a specific AtomicSwComponentType will need to communicate among each other. This implies that the RTE needs to provide synchronization mechanisms to the RunnableEntitys such that safe (in the multi-threading sense) exchange of data is possible.

This also means that only the RunnableEntitys of the same "instance" of an AtomicSwComponentType can communicate among each others. A hidden (i.e. without involvement of PortPrototypes) communication among RunnableEntitys is not allowed. ⌋*(RS_SWCT_0120)*

Several concepts for implementing communication among RunnableEntitys can be identified. As an introduction, this section first describes the various techniques that the RTE might use to provide efficient interaction between RunnableEntitys within one AtomicSwComponentType.

Next, two possible approaches for formal specification of this kind of communication are described:

- Specifying that several RunnableEntitys belong in a specific ExclusiveArea
- Specifying the data exchanged between the RunnableEntitys

7.4.1 Description Possibility 1: Exclusive Area

This section describes how the concept of ExclusiveAreas can be used in the description of the SwcInternalBehavior of an AtomicSwComponentType. Please note that ExclusiveAreas are actually owned by the base class of SwcInternalBehavior, i.e. InternalBehavior. These ExclusiveAreas do not imply a specific implementation (e.g. with mutual-exclusion semaphores).

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 7.20: ExclusiveArea

[TPS_SWCT_1031] ExclusiveArea ⌈ An ExclusiveArea (please find details about the formal definition of this meta-class in Figure 7.12) merely specifies a constraint on the scheduling policy and configuration of the RTE: If two or more RunnableEntitys refer to the same ExclusiveArea only one of these

RunnableEntitys is allowed to be executed while being inside that ExclusiveArea. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

In other words: these RunnableEntitys shall not run concurrently (preempt each other) while executing inside the ExclusiveArea.

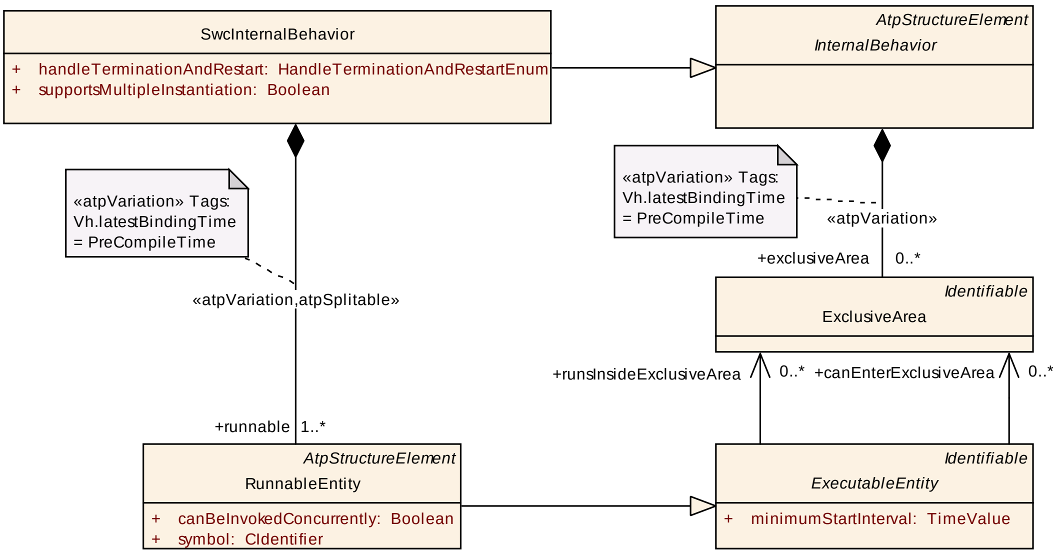

Figure 7.12: Description of logical exclusive areas

[TPS_SWCT_1049] Two ways to use the ExclusiveAreas ⌈ There are in general two ways to use the ExclusiveAreas. Note that it is even possible to use a specific ExclusiveArea in one RunnableEntity according to chapter 7.4.1.1 while another RunnableEntity might go for accessing the ExclusiveArea according to chapter 7.4.1.2. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

7.4.1.1 Entire Runnable Runs in the Exclusive Area

[TPS_SWCT_1050] RunnableEntity always runs inside an ExclusiveArea ⌈ In the first approach, the formal description specifies that certain RunnableEntitys always run inside an ExclusiveArea. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

For example, if the formal description specifies that both RunnableEntity 'r1' and RunnableEntity 'r2' run within ExclusiveArea 's1', the RTE shall make sure that RunnableEntitys 'r1' and 'r2' never run concurrently; the scheduler should never preempt 'r1' to run 'r2'.

Note that this pattern does not force the RTE to implement this by using semaphores or mutexes that are taken before the RunnableEntity starts and given when the RunnableEntity returns. It only obliges the RTE to make sure that both RunnableEntitys are never running concurrently.

This requirement could be implemented by several of the implementation strategies described above. For example:

1. Scheduling strategy: if, for example, RunnableEntitys 'r1' and 'r2' are mapped to the same task, the criterion is automatically satisfied. For this purpose it is necessary to make sure that the OS can only execute a single instance of the task into which the RunnableEntitys are put.
2. Mutual exclusion semaphores: in case 'r1' and 'r2' are mapped to different tasks ('T1', respectively 'T2'), the OS shall make sure that while 'T1' is executing 'r1', 'T2' running 'r2' can never preempt it and vice-versa. This could be implemented by taking a mutual-exclusion semaphore before executing 'r1' (resp. 'r2') in the context of 't1' (resp. 't2') and returning the semaphore on exiting the RunnableEntity.

7.4.1.2 Runnable would Dynamically Enter and Leave the Exclusive Area

[TPS_SWCT_1051] RunnableEntity explicitly enters and leaves a specific ExclusiveArea ⌈ In the second approach, the RunnableEntity would explicitly make API-calls to the RTE within the implementation of the RunnableEntity to enter and leave a specific ExclusiveArea. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

This could, for example, be implemented by means of the priority ceiling concept described in chapter 2.3.1.3.

Additionally it is possible to define the execution time the RunnableEntity will spend in this ExclusiveArea segment. Please note that although this aspect is described in [7] the concept can be applied to software-components as well.

7.4.2 Description Possibility 2: Inter-Runnable Variable

For certain important strategies (like the "variable copies" described above) the ExclusiveArea concept does not provide enough information to configure the RTE correctly.

The concept of copying concurrently accessed variables is very efficient and can even be used in ambitious automotive applications like, for example, engine management.

Please note however, that a certain amount of RAM has to be reserved for the copies. This is obviously a slight drawback of the concept.

Concerning the introduction in the AUTOSAR meta-model, data required for communication among RunnableEntitys needs to be explicitly identified.

[TPS_SWCT_1052] Inter-runnable variable ⌈ These so-called "inter-runnable variables" are described with the element VariableDataPrototype aggregated in the role explicitInterRunnableVariable or implicitInterRunnableVariable. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

[TPS_SWCT_1053] Relationship of interchanged data with RunnableEntitys ⌈Furthermore, the relationship of these data with RunnableEntitys shall be specified. For this purpose references with role writtenLocalVariable and readLocalVariable from RunnableEntity to VariableDataPrototype in the role of explicitInterRunnableVariable or implicitInterRunnableVariable are introduced. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

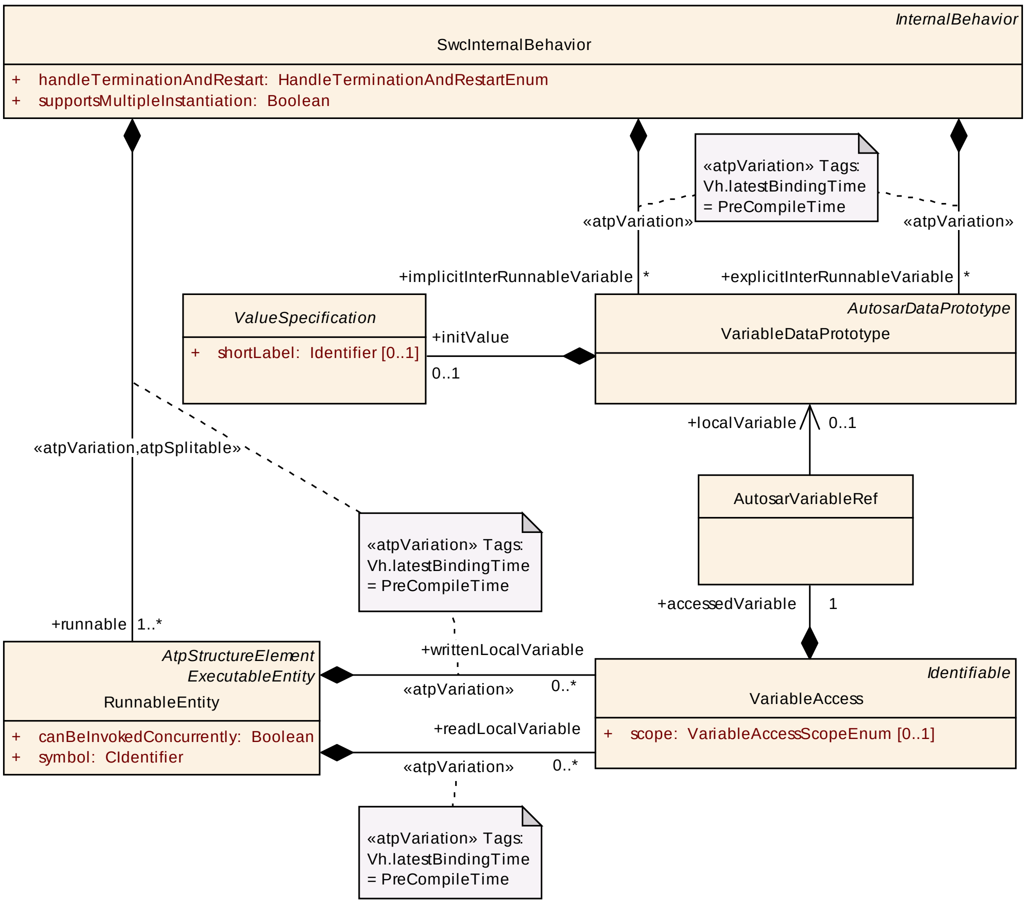

Figure 7.13: implicitInterRunnableVariable vs. explicitInterRunnableVariable

[constr_2026] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role writtenLocalVariable and readLocalVariable ⌈The RunnableEntity which defines a VariableAccess in role writtenLocalVariable and readLocalVariable shall use the localVariable reference of the AutosarVariableRef

The VariableDataPrototype in the localVariable reference needs to be owned by the same SwcInternalBehavior as this RunnableEntity belongs to, and the referenced VariableDataPrototype has to be defined in the role implicitInterRunnableVariable or explicitInterRunnableVariable. ⌋

The data type of an implicitInterRunnableVariable or explicitInterRunnableVariable is described by the data type of the VariableDataPrototype (which is derived from DataPrototype).

[constr_2001] Initial value for a specific `implicitInterRunnableVariable` or `explicitInterRunnableVariable` ⌈ It is possible but not mandatory to define an initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable. For this purpose the VariableDataPrototype in the role of explicitInterRunnableVariable or implicitInterRunnableVariable is able to aggregate a ValueSpecification in the role initValue. (see Figure 7.13). ⌋

Please note that the behavior is undefined if no initial value is specified and a RunnableEntity reads an implicitInterRunnableVariable or explicitInterRunnableVariable before it is actually written to by another RunnableEntity.

As already mentioned before, the concept of an "inter-runnable variable" can be used in *two different flavors* This is indicated by the two different roles explicitInterRunnableVariable or implicitInterRunnableVariable in which the VariableDataPrototype serving as the "inter-runnable variable" is aggregated.

These resemble the communication principles applied for the communication on the level of SwComponentTypes.

Please note that the two different kinds of inter-runnable variables are accessed via different RTE [2] API calls.

[TPS_SWCT_1054] Semantics of the `explicitInterRunnableVariable` ⌈ The semantics of the explicitInterRunnableVariable is that *explicit* implies the direct access to the value of an VariableDataPrototype used in the role explicitInterRunnableVariable or implicitInterRunnableVariable. By this means it is possible to get different values for a specific VariableDataPrototype each time the corresponding API call is executed. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

[TPS_SWCT_1055] Semantics of `implicitInterRunnableVariable` ⌈ The implicitInterRunnableVariable corresponds to an execution model where the value of an VariableDataPrototype does not change (for the reading RunnableEntity, obviously) during the runtime of a RunnableEntity. This approach is in detail described in chapter 2.3.1.4. ⌋*(RS_SWCT_0120, RS_SWCT_2090)*

7.4.3 Inter Runnable Triggering

The concept of Inter Runnable Triggering allows one RunnableEntity to trigger another one within a software-component. This supports the decoupling of calculation and processing sequences inside a Software Component.

By mappings of the InternalTriggerOccurredEvents to OS Tasks running at different priorities the triggered RunnableEntitys are in turn executed with a different priority as the triggering RunnableEntity.

For example, a cyclically triggered `RunnableEntity` which shall not exceed a certain worst case execution time (WCET) activates a second `RunnableEntity` an error occurred to process more time consuming exception handling with a lower priority.

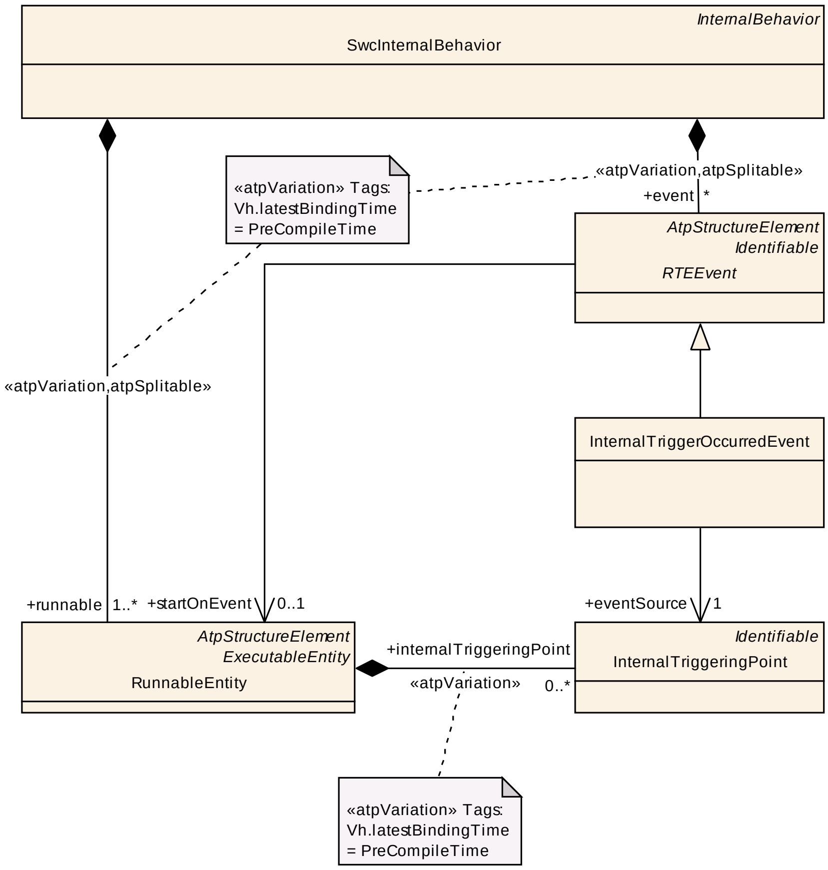

Figure 7.14: Model of software-component Inter Runnable Triggering

As illustrated in Figure 7.14 the triggering `RunnableEntity` needs an `InternalTriggeringPoint`.

The activation of `RunnableEntity`s in the same software-component instance is affected through the generic event handling mechanism.

A `RunnableEntity` that shall be activated at the occurrence of an internal trigger event is defined by means of an `InternalTriggerOccurredEvent` which references the particular `InternalTriggeringPoint` and additionally the to be activated `RunnableEntity`.

[TPS_SWCT_1022] Queued processing of internal trigger ⌈ Attribute `swImplPolicy` of `InternalTriggeringPoint` can be used to specify a requirement whether or not the internal triggering of the enclosing `RunnableEntity` using the given `InternalTriggeringPoint` shall be queued. ⌋

[constr_1182] Allowed values for `InternalTriggeringPoint.swImplPolicy` ⌈ The **only** allowed values for the attribute `swImplPolicy` of meta-class `InternalTriggeringPoint` are either `STANDARD` (in which case the processing of the internal triggering does not use a queue) or QUEUED (in which case the processing of internal triggering positively uses a queue). ⌋

Class	**InternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns a InternalTriggeringPoint it is entitled to trigger the execution of RunnableEntities of the corresponding software-component.			
Base	ARObject,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.

Table 7.21: InternalTriggeringPoint

Class	**InternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	The event is raised when the referenced internal trigger have been occurred.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
eventSource	InternalTriggeringPoint	1	ref	Internal Triggering Point that triggers the event.

Table 7.22: InternalTriggerOccurredEvent

7.5 Data Access of RunnableEntities

This section describes the communication properties of an AtomicSwComponentType. This is done mainly from the point of view of a RunnableEntity (the concept of a RunnableEntity is introduced in chapter 7.2). However, the usage of a PortPrototype in a specific role within an AtomicSwComponentType also has an impact on communication behavior.

7.5.1 RunnableEntities and Sender Receiver Communication

This section describes aspects relevant for the sender-receiver communication of a software-component. These mainly influence the behavior and API of the AUTOSAR RTE. T

[TPS_SWCT_1322] Interaction patterns for the application of the sender-receiver paradigm ⌈ The possible interaction patterns for the application of the sender-receiver paradigm are explained, namely:

1. Data-access in a cat. 1 RunnableEntity,

2. explicit sending,
3. the DataSendCompletedEvent: dealing with the success/failure of an explicit send, and
4. the DataReceivedEvent: responding to the reception of data
5. the DataReceiveErrorEvent: notifying an error concerning the reception of data.

⌋*(RS_SWCT_0200)*

7.5.1.1 Terminology

The AUTOSAR meta-model foresees two different approaches for sender-receiver communication. These are described in detail in chapters 7.5.1.2 and 7.5.1.3. However, it turned out that it is rather cumbersome to discuss issues of communication approaches directly on the basis of meta-classes and their attributes.

Therefore, it seems appropriate to introduce a dedicated terminology for this purpose. The approach eventually selected was originally introduced by the contributors to the RTE specification.

This terminology proposes to use the term "implicit" for communication based on Data-Access (for more information about details of this approach please consult chapter 7.5.1.2) and "explicit" for communication based on Data-Points (please refer to chapter 7.5.1.3).

The motivation for the differentiation between "implicit" and "explicit" was originally the characteristics of the RTE specification that foresaw an API for handling a DataSendPoint or DataReceivePoint in contrast to the Data-Access that was supposed to be part of the function signature (therefore, no API was required) of a specific RunnableEntity.

Although the specification of the RTE changed in the meantime (and the original motivation no longer applies) it turned out that the terminology based on "implicit" and "explicit" communication" was already widely used within AUTOSAR.

As no consensus could be reached over alternative proposals this terminology approach is taken over by this document as well.

7.5.1.2 Data Access

[TPS_SWCT_1323] Read and write access to a dataElement ⌈ The SwcInternalBehavior may specify that a RunnableEntity needs read-access (respectively write-access) to the VariableDataPrototypes in the role dataElement of an RPortPrototype (respectively PPortPrototype). ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1325] Read and write access is only applicable for RunnableEntitys of category 1 ⌈ The usage of this access mechanism to the VariableDataPrototypes is appropriate for cat. 1 RunnableEntitys only because it by concept guarantees finite response time (opposed to e.g. unlimited blocking wait for some data). ⌋ *(RS_SWCT_0200)*

Suppose a cat. 2 RunnableEntity would have a DataReadAccess and a DataWriteAccess. The received dataElement would be updated before the RunnableEntity actually starts being executed and even if the RunnableEntity runs for a very long time the value would remain as it is.

On the other hand, the RunnableEntity might use its DataWriteAccess to perform a write access on the dataElement but the actual value might never make it beyond the RunnableEntity because

1. the latter is not required to terminate ever and
2. the actual write access is executed *after* the RunnableEntity terminates.

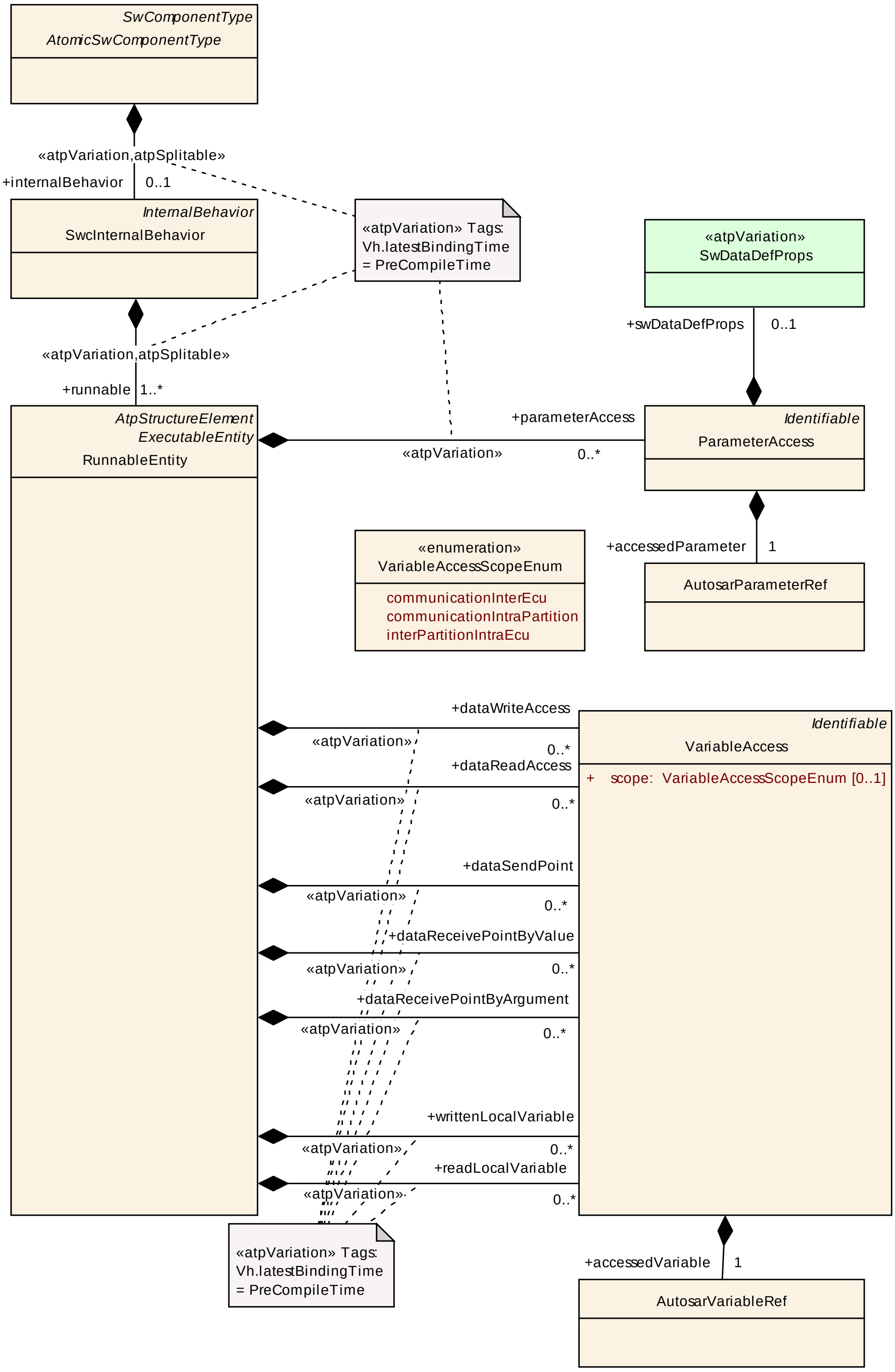

Figure 7.15: DataReadAccess and DataWriteAccess

Class	**VariableAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements			
Note	The presence of a VariableAccess implies that a RunnableEntity needs access to a VariableDataPrototype. The kind of access is specified by the role in which the class is used.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
accessedVariable	AutosarVariableRef	1	aggr	This denotes the accessed variable.
scope	VariableAccessScopeEnum	0..1	attr	This attribute allows for constraining the scope of the corresponding communication. For example, it possible to express whether the communication is intended to cross the boundary of an ECU or whether it is intended not to cross the boundary of a single partition.

Table 7.23: VariableAccess

Enumeration	**VariableAccessScopeEnum**
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::DataElements
Note	This enumeration defines scopes for communication.
Literal	*Description*
communicationInterEcu	This case is foreseen to express that the corresponding communication shall be considered inter-ECU, i.e. it will cross the ECU boundary. This is considered the default case.
communicationIntraPartition	This case is foreseen to express that the corresponding communication shall **not** cross the boundary of a partition.
interPartitionIntraEcu	In this case the communication shall cross the boundaries of partitions within one ECU but it shall not cross the boundaries of the ECU itself.

Table 7.24: VariableAccessScopeEnum

[TPS_SWCT_1326] Constrain the scope of a specific communication ⌈ The purpose of the attribute `scope` of meta-class `VariableAccess` is to constrain the scope of the corresponding communication. The main use-case for this ability is the development of a software-component where certain end-points of communication from or to the software-component are known to fulfill a certain constraint, e.g. execute within the same partition. ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1327] RTE generator can omit the creation of checks at run-time ⌈ Depending on the value of the constraint the RTE generator shall apply or else can omit checks executed during run-time. For example, within a single partition there are hardly any factors that lead to a failure of communication. This may generate a huge run-time benefit compared to an implementation that always applies full checks in all cases. ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1328] Default value of attribute scope ⌈ The default value of attribute scope is set to communicationInterEcu. ⌋*(RS_SWCT_0200)*

[constr_1141] Applicability of the scope attribute ⌈

The attribute scope of meta-class VariableAccess shall **only** be applied with respect to the aggregation of VariableAccess in the following roles:

- dataReadAccess
- dataWriteAccess
- dataSendPoint
- dataReceivePointByValue
- dataReceivePointByArgument

⌋

[TPS_SWCT_1329] Access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles ⌈ Please note that from the formal point of view access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles, i.e. dataReadAccess for a read-access while the write-access is defined by means of aggregating VariableAccess in the role dataWriteAccess. ⌋*(RS_SWCT_0200)*

This aspect is depicted in Figure 7.15.

The following constraints apply to the reference target of the AutosarVariableRef of VariableAccess in role dataReadAccess or dataWriteAccess.

[constr_2002] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReadAccess ⌈ A VariableAccess in the role dataReadAccess shall refer to an RPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface. ⌋

[constr_2003] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataWriteAccess ⌈ A VariableAccess in the role dataWriteAccess shall refer to an PPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface. ⌋

By access with VariableAccess in the dataReadAccess role always the last value of the VariableDataPrototype buffered before the RunnableEntity starts will be read during the execution of the RunnableEntity. There is no meaning to provide a queue of values for the dataReadAccess.

[constr_2020] dataReadAccess can not be used for queued communication ⌈ The swImplPolicy of the VariableDataPrototype referenced by a VariableAccess in role dataReadAccess shall not be set to queued. ⌋

7.5.1.3 Explicit Sending and Receiving

[TPS_SWCT_1330] RunnableEntity can also have dataSendPoints ⌈ A RunnableEntity can also have dataSendPoints (i.e. aggregate VariableAccess in the role dataSendPoint). Using an instanceRef association, these eventually reference a VariableDataPrototype in the context of a PPortPrototype, owned by the AtomicSwComponentType that is associated with the RunnableEntity that in turn owns the dataSendPoint. ⌋*(RS_SWCT_0200)*

[constr_2004] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataSendPoint ⌈ A VariableAccess in the role dataSendPoint shall refer to an PPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface. ⌋

[TPS_SWCT_1331] dataWriteAccess vs. dataSendPoint ⌈ As opposed to the dataWriteAccess:

- Using the dataSendPoint, the RunnableEntity needs to explicitly "send" through an API; when using a dataWriteAccess, the RunnableEntity only needs to modify the value of certain variables.
- Using dataSendPoint, the Runnable can decide to "send" an arbitrary number of times; when using dataWriteAccess the new value of the VariableDataPrototype is not made available before the RunnableEntity returns (exits the "Running" state).
- The presence of a dataSendPoint per definition lets the corresponding RunnableEntity attain cat. 1B.

⌋*(RS_SWCT_0200)*

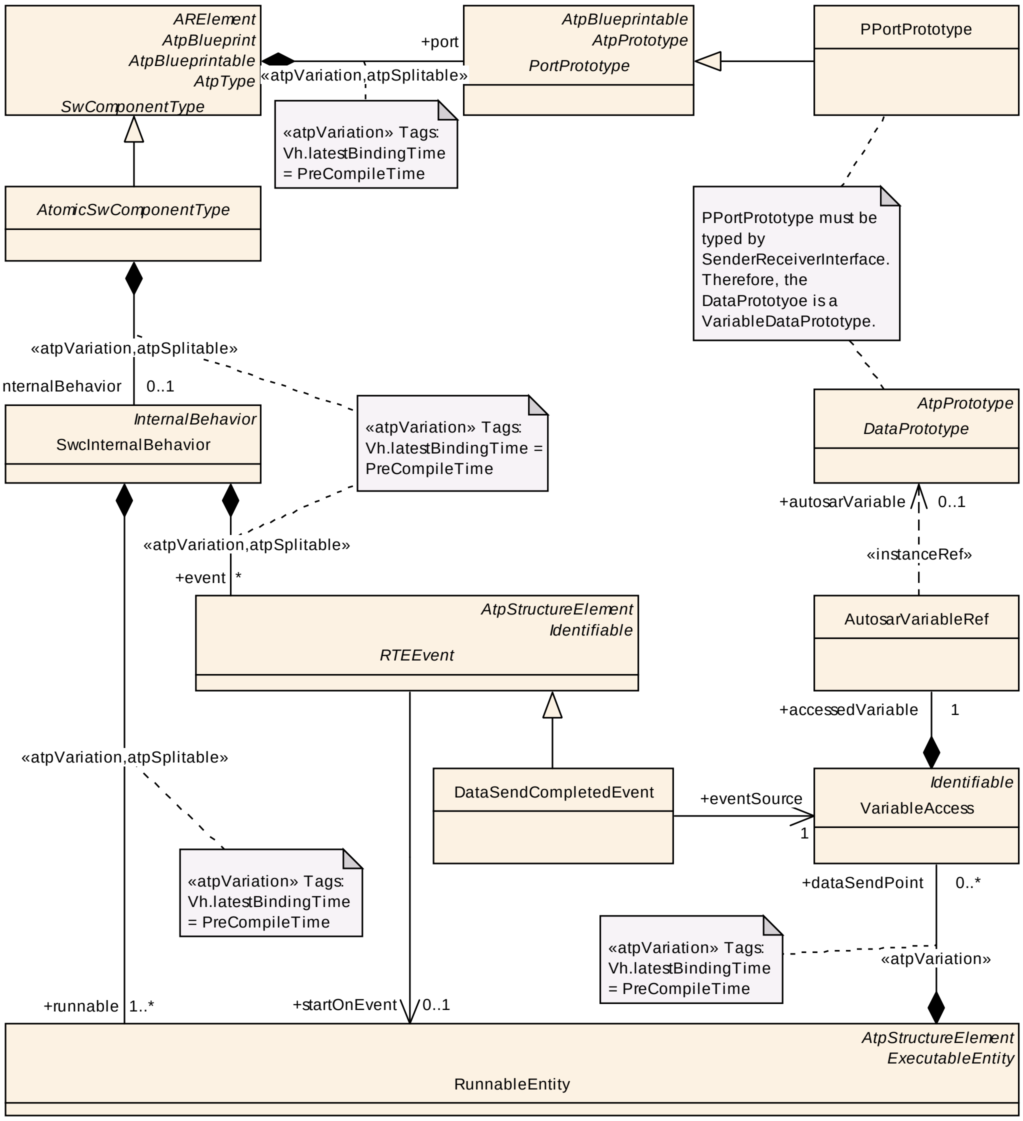

Figure 7.16: DataSendPoint

[TPS_SWCT_1332] **`dataReceivePointByValue` vs. `dataReceivePointByArgument`** ⌈ In analogy to explicitly sending data it is also possible to define explicit polling for new available data through a `dataReceivePointByValue` or `dataReceivePointByArgument` as shown in Figure 7.17. ⌋

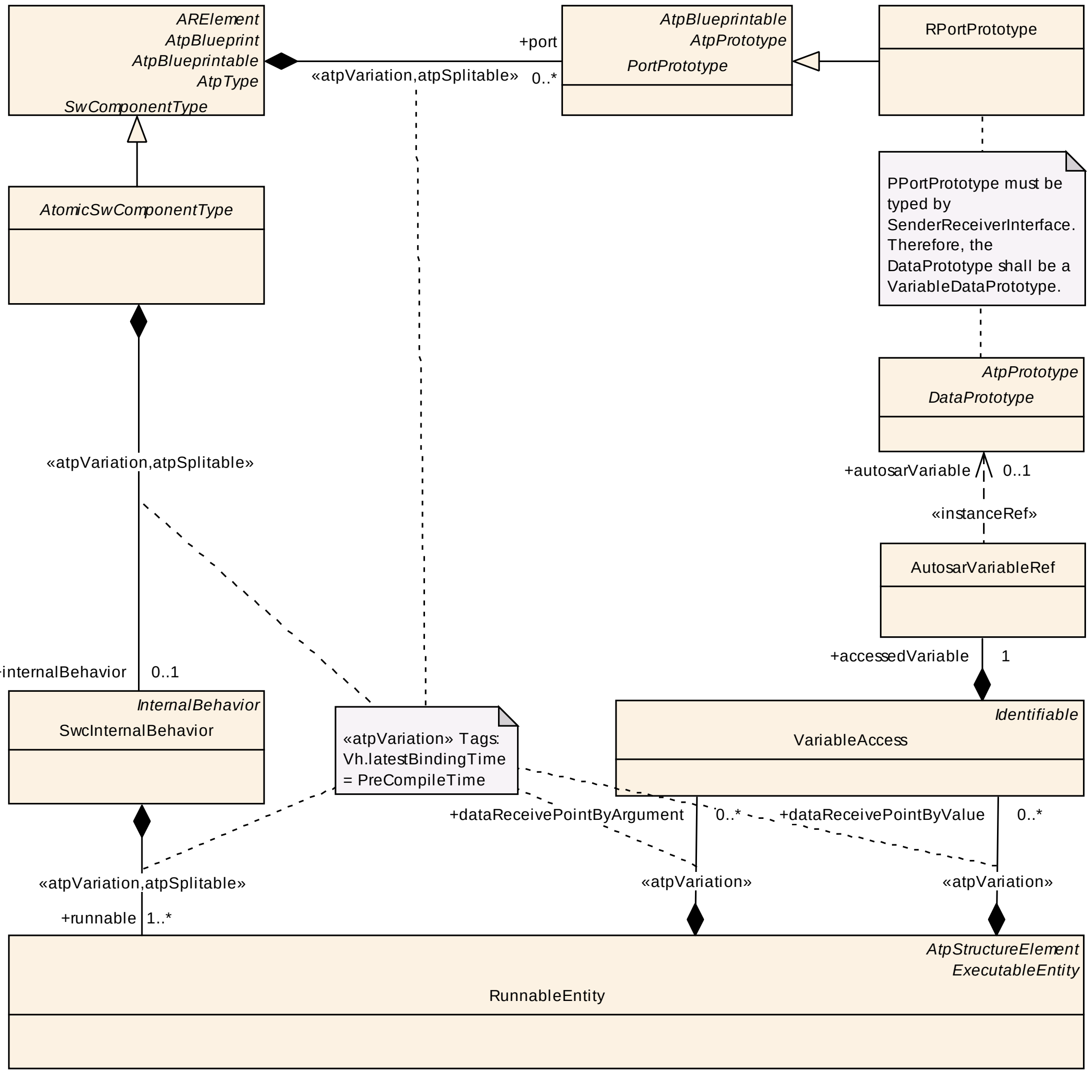

Figure 7.17: Definition of an explicit request to receive data

[TPS_SWCT_1333] `dataReceivePointByValue`/`dataReceivePointByArgument` vs. `dataReadAccess` ⌈ By using a `dataReceivePointByValue` or `dataReceivePointByArgument` instead of `dataReadAccess` the constraining access to the referenced `VariableDataPrototype` - other `RunnableEntity`s shall not change the `VariableDataPrototype` during the read execution - is limited to a short, well-defined amount of time. ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1334] `RunnableEntity`s of category 1 may have `dataReceivePoints` ⌈ Therefore, category 1 `RunnableEntity`s may also have `dataReceivePoints` and consequently become `RunnableEntity`s of category 1B, see section 7.2.4.4. ⌋*(RS_SWCT_0200)*

Similar to the `dataReadAccess` constraints apply to the reference target of the `AutosarVariableRef` of `VariableAccess` in role `dataReceivePointByValue` or `dataReceivePointByArgument`.

[constr_2005] Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument ⌈ A VariableAccess in the role dataReceivePointByValue or dataReceivePointByArgument shall refer to an RPortPrototype that is typed by either a SenderReceiverInterface or a NvDataInterface. ⌋

[TPS_SWCT_1335] Combine dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint ⌈ Please note that it would in general be possible to combine a dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint in the scope of a particular RunnableEntity. This would allow for a call to a blocking receive routine implemented by the RTE. The timeout attribute of meta-class WaitPoint can be used to specify the time until the blocking call expires.

But in case of non-queued communication it is not supported that a DataReceivedEvent is used with a WaitPoint. This would contradict the approach of the last is the best semantic. ⌋*(RS_SWCT_0200)*

[constr_2021] WaitPoint referencing a DataReceivedEvent can not be used for non-queued communication ⌈ A WaitPoint referencing a DataReceivedEvent is only permitted if the swImplPolicy of the VariableDataPrototype referenced by this DataReceivedEvent is set to queued. ⌋

Please note however, that in this case (in response to the presence of a WaitPoint) the RunnableEntity becomes category 2.

7.5.1.4 DataSendCompletedEvent

[TPS_SWCT_1336] dataSendPoint also allows for the definition of a DataSendCompletedEvent ⌈ The dataSendPoint also allows for the definition of a DataSendCompletedEvent, as shown in Figure 7.16. This RTEEvent occurs when the data has been successfully sent or when an error has occurred during sending. ⌋*(RS_SWCT_0200)*

Please note that this feature can only be used if the AtomicSwComponentType describes the meaning of success or failure of the send operation.

In particular, via a SenderComSpec class different acknowledgment requests (in this case: successful transmission) can be attached to a PPortPrototype, as is shown in Figure 4.31.

[constr_2032] transmissionAcknowledge requires a DataSendCompletedEvent ⌈ If a SenderComSpec does specify a transmissionAcknowledge there shall be also a DataSendCompletedEvent specified whose VariableAccess references the same VariableDataPrototype as the SenderComSpec. ⌋

This will configure the RTE that when data is sent it will try to obtain the specified acknowledgment; possibly by waiting a certain timeout period.

Class	**DataSendCompletedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced data elements have been sent or an error occurs.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
eventSource	VariableAccess	1	ref	The variable access that triggers the event.

Table 7.25: DataSendCompletedEvent

[constr_2033] Timeout of `DataSendCompletedEvent` ⌈ The `timeout` value of a `WaitPoint` associated with a `DataSendCompletedEvent` shall have the same value as the corresponding `TransmissionAcknowledgementRequest`'s `timeout` value ⌋

7.5.1.5 DataReceivedEvent

[TPS_SWCT_1337] `DataReceivedEvent` ⌈ A receiver is notified through the same event mechanism when a `VariableDataPrototype` is received. As shown in Figure 7.18, the `DataReceivedEvent` is directly associated with the corresponding `VariableDataPrototype`. ⌋*(RS_SWCT_0200)*

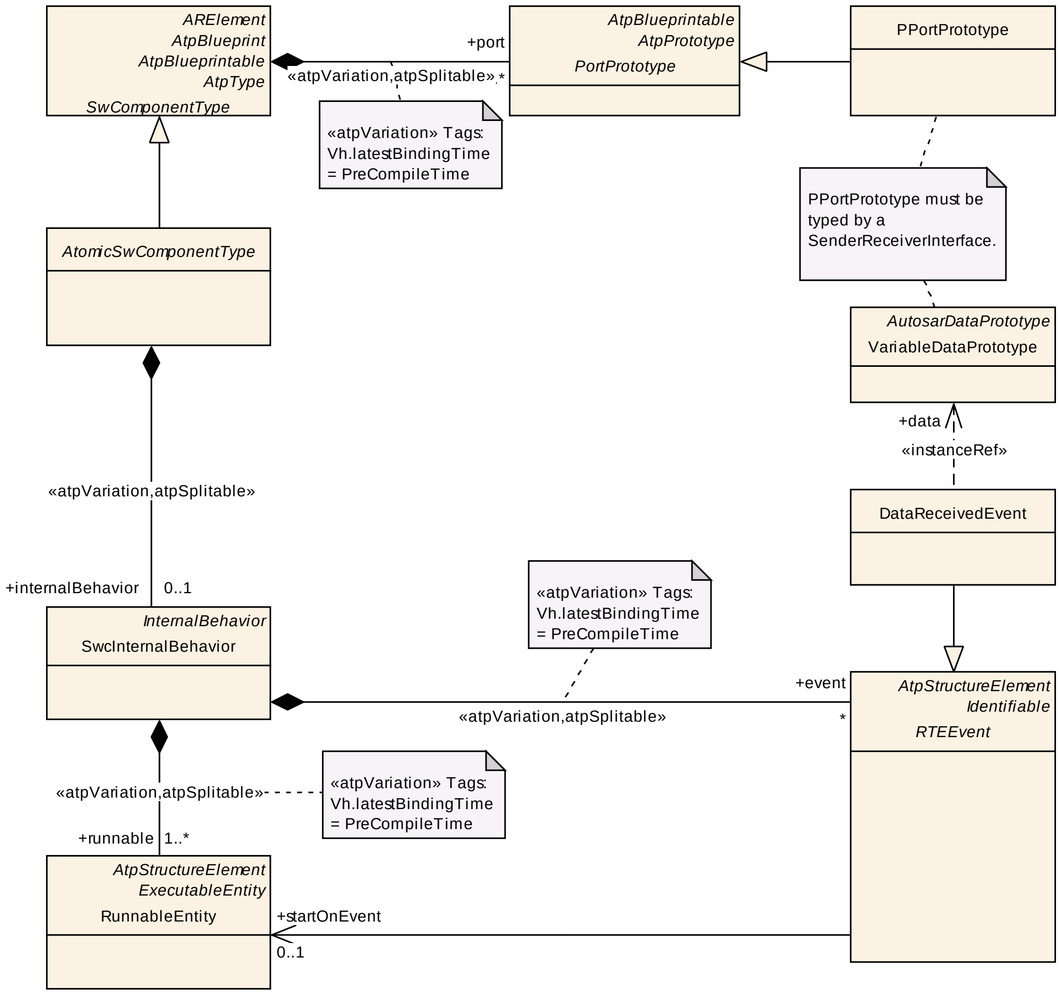

Figure 7.18: Receiver is notified by an event when new data has arrived

Class	**DataReceivedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced data elements are received.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	1	iref	Data element referenced by event

Table 7.26: DataReceivedEvent

7.5.1.6 DataReceiveErrorEvent

[TPS_SWCT_1338] **`DataReceiveErrorEvent`** ⌈ A receiver is notified of `DataReceiveErrorEvent` through the activation of its `RunnableEntity` which is referenced by this `RTEEvent`. A `DataReceiveErrorEvent` includes a reference to a `VariableDataPrototype` and is raised by the RTE when an error concerning the reception of the referenced data is detected by the COM [4] layer. The following cases present some situations which will cause the RTE to raise a `DataReceiveErrorEvent`:

- the RTE receives a signal-outdated notification from the COM layer when a monitored periodic signal is not received in time. The COM layer monitors the validity of the signal's value based on the value of the `aliveTimeout` attribute of `ReceiverComSpec` referencing the `VariableDataPrototype` associated with the signal. If the time elapsed since the last update of a signal's value exceeds its `aliveTimeout` then the COM layer notifies the RTE of a signal outdated error.

- The RTE receives a signal invalid notification from the COM layer when this latter detects that an incoming signal has the predefined 'invalid' value.

⌋*(RS_SWCT_0200)*

[TPS_SWCT_1339] RTE activates `RunnableEntity` in response to `DataReceiveErrorEvent` ⌈ This `RTEEvent` is used by the RTE to activate `RunnableEntity`s which handle the above-mentioned errors. The error code will be made available to the activated `RunnableEntity` through the appropriate RTE API function. ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1340] `DataReceiveErrorEvent` cannot be combined with a `WaitPoint` ⌈ Please note that this `RTEEvent` cannot be associated with a `WaitPoint`, see [constr_1091]. It can only be used for the receiver software-component in a sender-receiver communication and its data reference is restricted to `VariableDataPrototype`s with their `swImplPolicy` attribute not set to `queued`. ⌋*(RS_SWCT_0200)*

[4] In case of internal communication the RTE is not enforced to use the COM layer. It is also possible to implement the required behavior directly in the RTE [2].

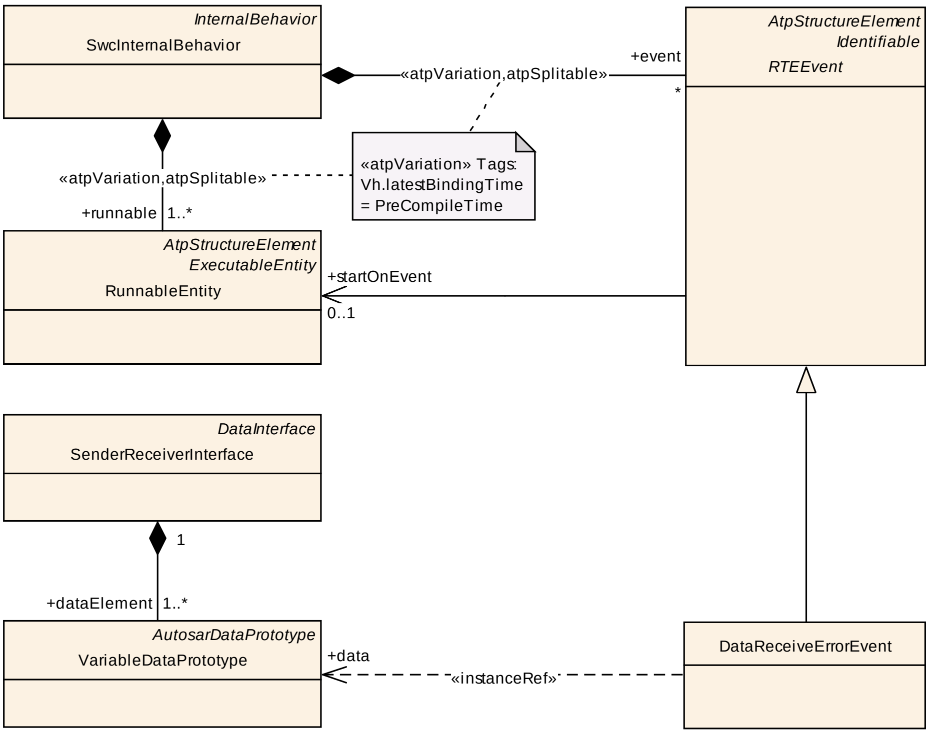

Figure 7.19: DataReceiveErrorEvent references a Runnable and a VariableDataPrototype

[TPS_SWCT_1341] `DataReceiveErrorEvent` is directly associated with the corresponding `VariableDataPrototype` ⌈ As shown in Figure 7.19, the `DataReceiveErrorEvent` is directly associated with the corresponding `VariableDataPrototype` and references the `RunnableEntity` that is activated due to the occurrence of this `RTEEvent`. ⌋ *(RS_SWCT_0200)*

Class	**DataReceiveErrorEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	This event is raised by the RTE when the Com layer detects and notifies an error concerning the reception of the referenced data element.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
data	VariableDataPrototype	1	iref	Data element referenced by event

Table 7.27: DataReceiveErrorEvent

7.5.2 RunnableEntities and Client Server Communication

7.5.2.1 Invoking an Operation

[TPS_SWCT_1342] Invocation of a server operation ⌈ A `RunnableEntity` invokes a server operation formally defined as a `ClientServerOperation` via an `RPort-`

Prototype of the enclosing SwComponentPrototype typed by a particular AtomicSwComponentType. ⌋(RS_SWCT_0200)

[TPS_SWCT_1343] Synchronous vs. asynchronous invocation ⌈ A ClientServerOperation itself can be invoked either "synchronously" or "asynchronously". ⌋(RS_SWCT_0200)

In the majority of cases the ClientServerOperation will be invoked at a different SwComponentPrototype but in general it would be possible to invoke a ClientServerOperation on the same SwComponentPrototype as well.

The decision whether a specific ClientServerOperation is called synchronously or asynchronously needs to be specified in the formal description of the corresponding AtomicSwComponentType, namely in the context of an SwcInternalBehavior (see Figure 7.20 for more details).

But it is not supported to invoke the same instance of a ClientServerOperation synchronously and asynchronously together.

[constr_2022] Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints ⌈ A ClientServerOperation of a particular RPortPrototype a shall mutually exclusive be referenced by either SynchronousServerCallPoints or AsynchronousServerCallPoints. ⌋

[TPS_SWCT_1344] Consistency of values of timeout ⌈ The timeout values need to be consistent in case of multiple ServerCallPoints referencing the same instance of ClientServerOperation. ⌋(RS_SWCT_0200)

[constr_2023] Consistency of timeout values ⌈ The timeout values of all ServerCallPoints referencing the same instance of ClientServerOperation in a RPortPrototype shall be identical. ⌋

[TPS_SWCT_1345] Synchronous operation invocation ⌈ In case of a synchronous operation invocation the particular RunnableEntity merely needs a SynchronousServerCallPoint (see Figure 7.20). ⌋(RS_SWCT_0200)

[TPS_SWCT_1346] Asynchronous operation invocation ⌈ Asynchronous invocation is a bit more complex because it is necessary to specify how to respond to a notification about the completion of the corresponding operation.

This is done using the generic RTEEvent mechanism: the notification about an asynchronously executed operation having completed is implemented as an AsynchronousServerCallReturnsEvent.

Therefore, if an AsynchronousServerCallReturnsEvent is raised the RTE can either trigger the execution of a specific RunnableEntity or the AtomicSwComponentType can implement a WaitPoint that blocks the execution of the calling RunnableEntity until the AsynchronousServerCallReturnsEvent is recognized. ⌋(RS_SWCT_0200)

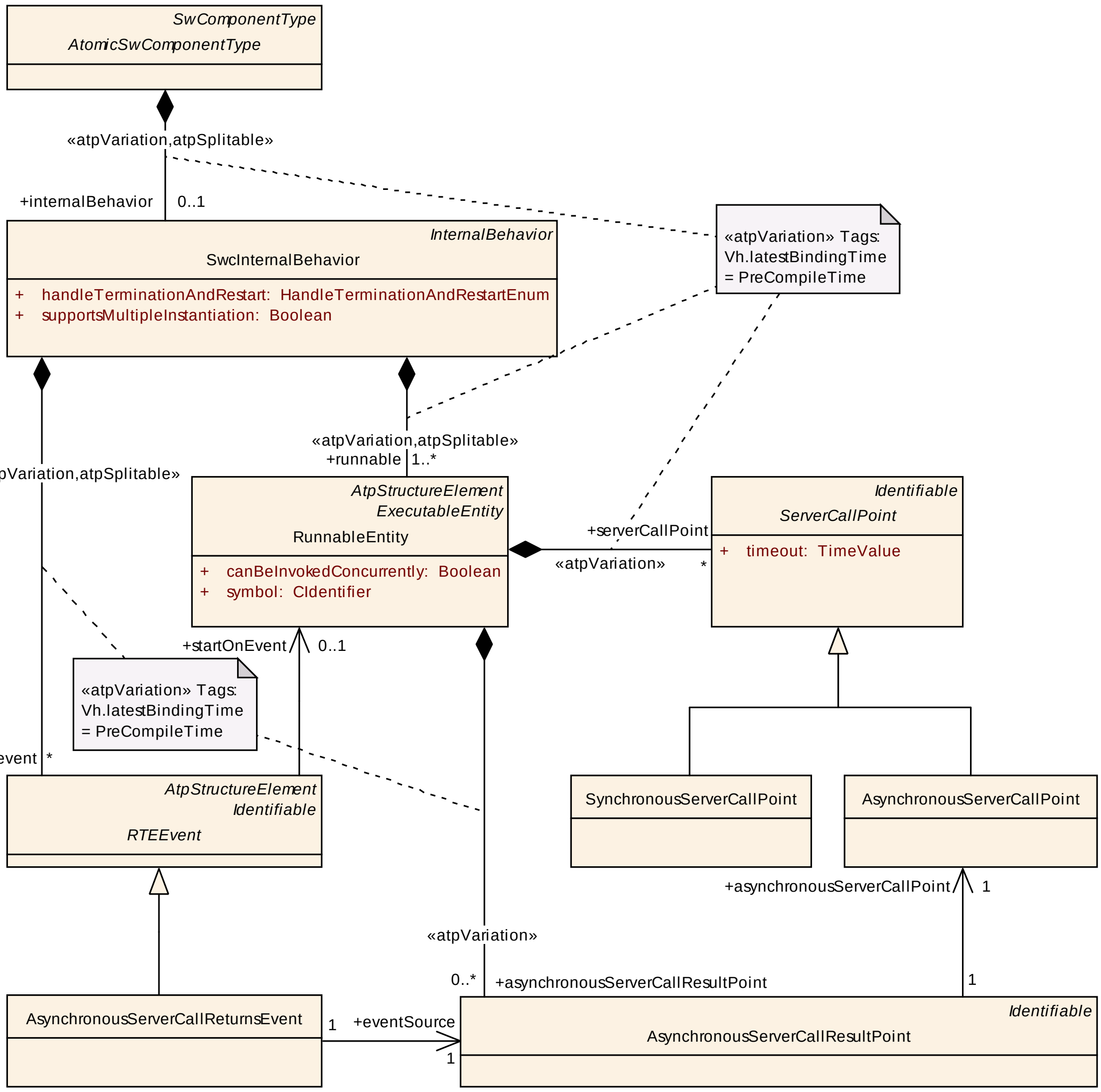

Figure 7.20: Model of a server call point.

For example, let's consider the case of an asynchronous call to a remote operation where the RTE is supposed to trigger a specific `RunnableEntity` when the operation completes. The description of the corresponding AtomicSwComponentType would typically contain the following elements:

1. The `AtomicSwComponentType` contains a `RPortPrototype` 'myPort' typed by a `PortInterface` that in turn contains the definition of an `ClientServerOperation` 'remoteOperation'.
2. The `AtomicSwComponentType`'s `SwcInternalBehavior` contains at least two `RunnableEntity`s: the `RunnableEntity` 'main' is supposed to invoke the operation; the `RunnableEntity` 'callback' is the one that should be called when the operation completes.
3. The description of the `RunnableEntity` 'main' contains an `AsynchronousServerCallPoint` 'invokeMyOperation' referencing the respective `ClientServerOperation` in the `PortInterface` used to type the `PortPro-`

totype 'myPort'. This implies that the RunnableEntity is allowed to invoke this operation asynchronously.

4. The description of the RunnableEntity 'callback' contains an AsynchronousServerCallResultPoint 'fetchMyOperationResults' referencing the respective AsynchronousServerCallPoint 'invokeMyOperation' This implies that the RunnableEntity is allowed to fetch the results of the asynchronously invoked operation.
5. The description of the SwcInternalBehavior includes an AsynchronousServerCallReturnsEvent 'myOperationReturns' which references the previously defined AsynchronousServerCallResultPoint 'fetchMyOperationResults'
6. The description of the AsynchronousServerCallReturnsEvent 'myOperationReturns' references the RunnableEntity 'callback', indicating that the RTE should trigger the execution of this Runnable when 'myOperationReturns' is raised.

Class	**ServerCallPoint (abstract)**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a ServerCallPoint it is entitled to invoke a particular ClientServerOperation of a specific RPortPrototype of the corresponding AtomicSwComponentType			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
operation	ClientServerOperation	1	iref	The operation that is called by this runnable.
timeout	TimeValue	1	attr	Time in seconds before the server call times out and returns with an error message. It depends on the call type (synchronous or asynchronous) how this is reported.

Table 7.28: ServerCallPoint

.tex

Class	**SynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	This means that the RunnableEntity is supposed to perform a blocking wait for a response from the server.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServerCallPoint			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table 7.29: SynchronousServerCallPoint

Class	**AsynchronousServerCallPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	An AsynchronousServerCallPoint is used for asynchronous invocation of a ClientServerOperation. IMPORTANT: a ServerCallPoint cannot be used concurrently. Once the client RunnableEntity has made the invocation, the ServerCallPoint cannot be used until the call returns (or an error occurs!) at which point the ServerCallPoint becomes available again.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServerCallPoint			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.30: AsynchronousServerCallPoint

Class	**AsynchronousServerCallResultPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServerCall			
Note	If a RunnableEntity owns a AsynchronousServerCallResultPoint it is entitled to get the result of the referenced AsynchronousServerCallPoint. If it is associated with AsynchronousServerCallReturnsEvent, this RTEEvent notifies the completion of the required ClientServerOperation or a timeout. The occurrence of this event can either unblock a WaitPoint or can lead to the invocation of a RunnableEntity.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
asynchronousServerCallPoint	AsynchronousServerCallPoint	1	ref	The referenced Asynchronous Server Call Point defines the asynchronous server call from which the results are returned.

Table 7.31: AsynchronousServerCallResultPoint

[constr_2006] Number of `AsynchronousServerCallResultPoint` referencing to one `AsynchronousServerCallPoint` ⌈ The `AsynchronousServerCallPoint` has to be referenced by exactly one `AsynchronousServerCallResultPoint`. This means that only the `RunnableEntity` with this `AsynchronousServerCallResultPoint` can fetch the result of the asynchronous server invocation of this particular `AsynchronousServerCallPoint`. ⌋

This information might be used by the RTE generator to optimize the data consistency mechanisms.

Class	**AsynchronousServerCallReturnsEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTEEvents			
Note	This event is raised when an asynchronous server call is finished.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,RTEEvent,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
eventSource	AsynchronousServerCallResultPoint	1	ref	The referenced AsynchronousServerCallResultPoint which is raises the RTEEvent in case of returning asynchronous server call.

Attribute	Datatype	Mul.	Kind	Note

Table 7.32: AsynchronousServerCallReturnsEvent

[TPS_SWCT_1347] Blocking access to operation result in an asynchronous operation invocation ⌈ If the call of the RTE fetching the operations results shall block until the server returns the `RunnableEntity` with the `AsynchronousServerCallResultPoint` needs additional a `WaitPoint` referencing the `AsynchronousServerCallReturnsEvent` which is associated with the `AsynchronousServerCallResultPoint` representing the operations results access.

In this case the `AsynchronousServerCallReturnsEvent` shall not define a `startOnEvent` reference to a `RunnableEntity`. ⌋*(RS_SWCT_0200)*

[constr_2030] `AsynchronousServerCallResultPoint` combined with `WaitPoint` shall belong to the same `RunnableEntity` ⌈ The `WaitPoint` which references a `AsynchronousServerCallReturnsEvent` and the `AsynchronousServerCallResultPoint` which is referenced by this `AsynchronousServerCallReturnsEvent` shall be aggregated by the same `RunnableEntity`. ⌋

7.5.2.2 Providing an Implementation of an Operation

A software-component can define an `OperationInvokedEvent` for each operation inside one of the server `PPortPrototype`s. This way a `RunnableEntity` may respond to such an invocation through the generic event handling mechanisms described above (as formally expressed in Figure 7.21).

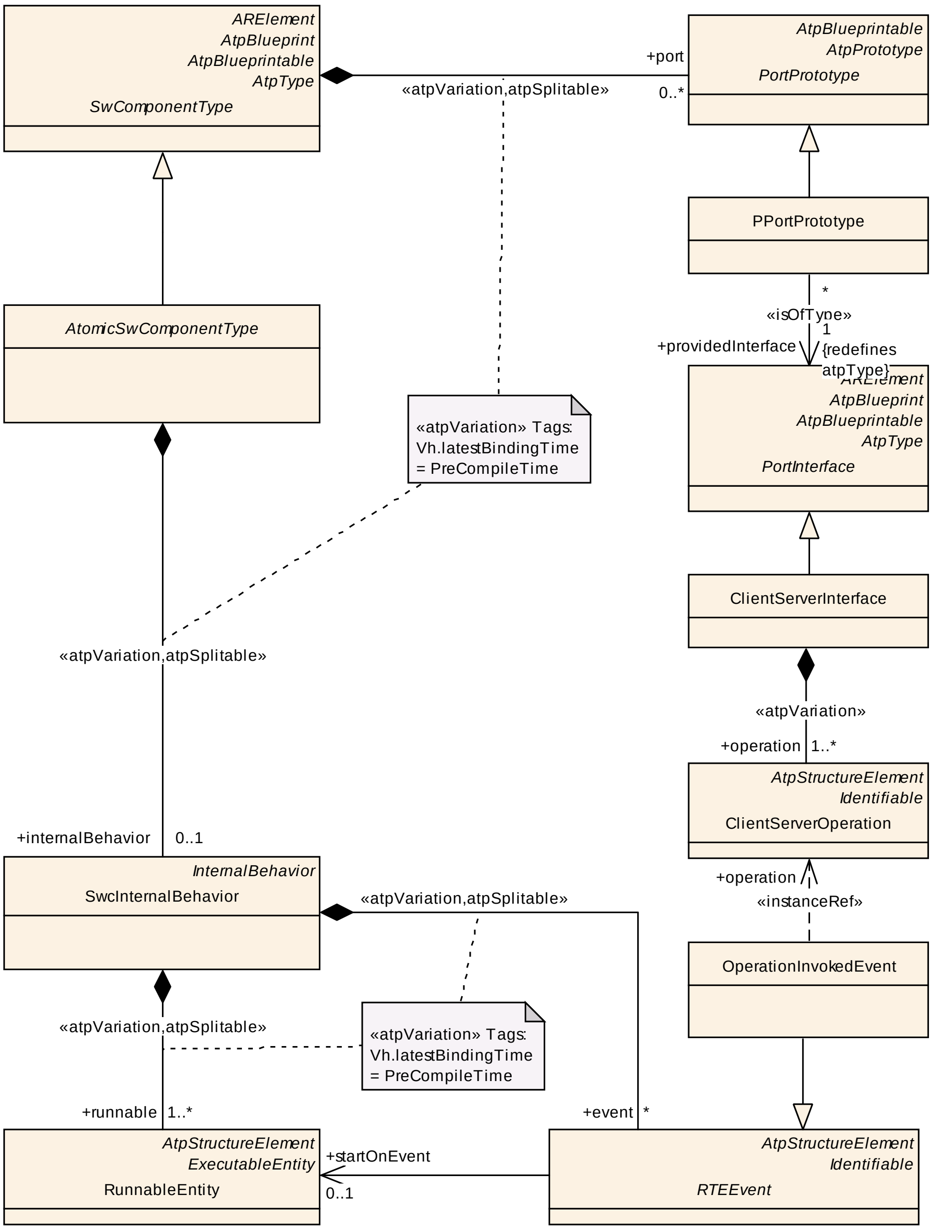

Figure 7.21: The `OperationInvokedEvent` references the operation that was called by a client.

Class	**OperationInvokedEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The OperationInvokedEvent references the ClientServerOperation invoked by the client.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
operation	ClientServerOperation	1	iref	The operation to be executed as the consequence of the event.

Table 7.33: OperationInvokedEvent

7.5.3 RunnableEntities and External Trigger Event Communication

7.5.3.1 Trigger Source

[TPS_SWCT_1348] Trigger source ⌈ A RunnableEntity of the triggering software-component raises an external trigger event via an PPortPrototype of the enclosing SwComponentPrototype typed by a particular AtomicSwComponentType.

For this purpose the particular RunnableEntity needs an ExternalTriggeringPoint that references the particular instance of the trigger in a PPortPrototype. ⌋*(RS_SWCT_0200)*

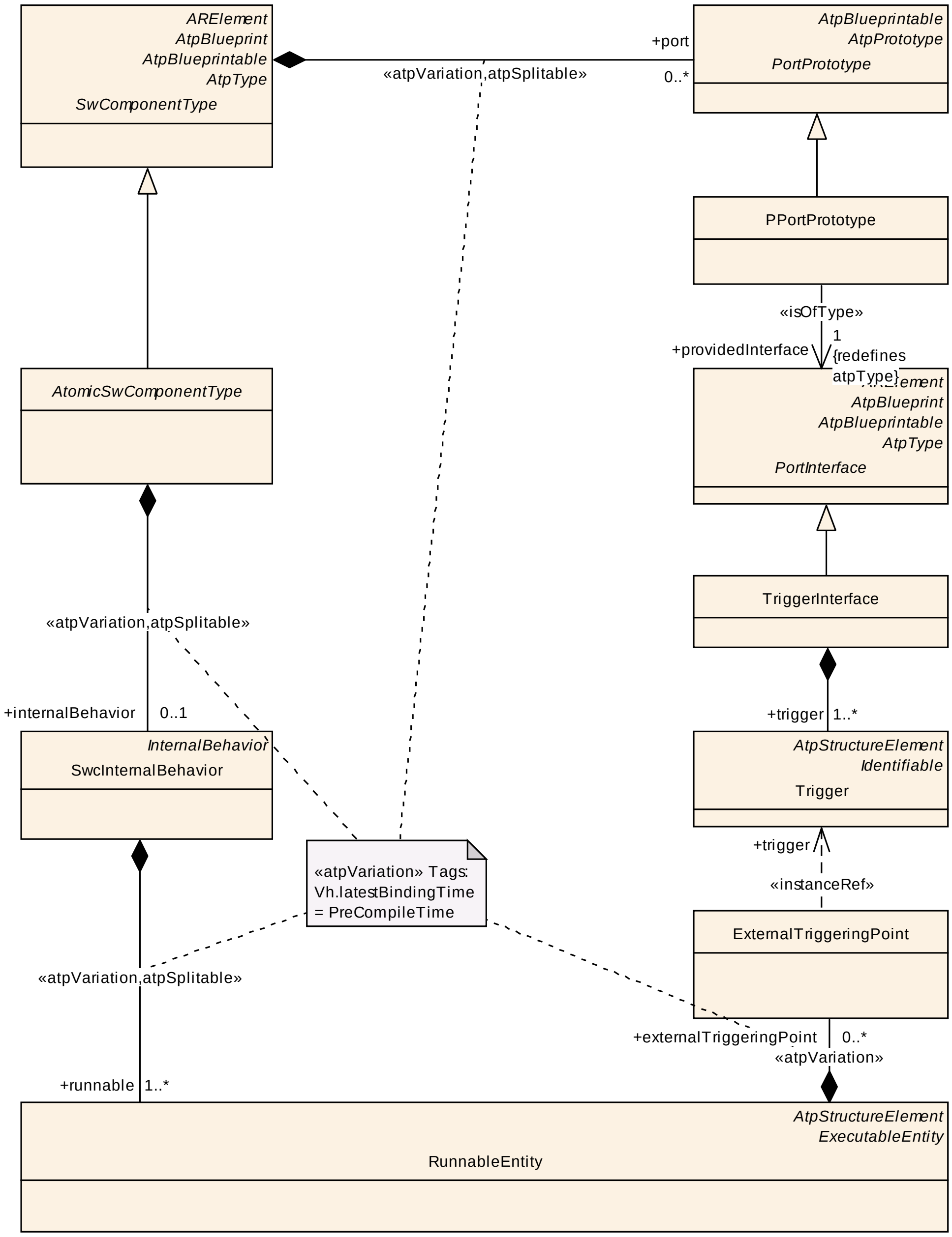

Figure 7.22: Model structure of a trigger source.

Class	**ExternalTriggeringPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Trigger			
Note	If a RunnableEntity owns an ExternalTriggeringPoint it is entitled to raise an ExternalTriggerOccurredEvent.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
trigger	Trigger	1	iref	The trigger taken for the ExternalTriggeringPoint. **Tags:** xml.namePlural=TRIGGER-IREF; xml.roleElement=false; xml.roleWrapperElement=true; xml.typeElement=true; xml.typeWrapperElement=false

Table 7.34: ExternalTriggeringPoint

7.5.3.2 Trigger Sink

The activation of `RunnableEntity`s in the trigger sink is effected through the generic event handling mechanism.

[TPS_SWCT_1349] Trigger sink ⌈ The fact that a `RunnableEntity` shall be activated on occurrence of an external trigger event is formally defined by means of `ExternalTriggerOccurredEvent` that references a particular instance of the trigger in a `RPortPrototype` and additionally the `RunnableEntity` to be executed in response to the event. ⌋*(RS_SWCT_0200)*

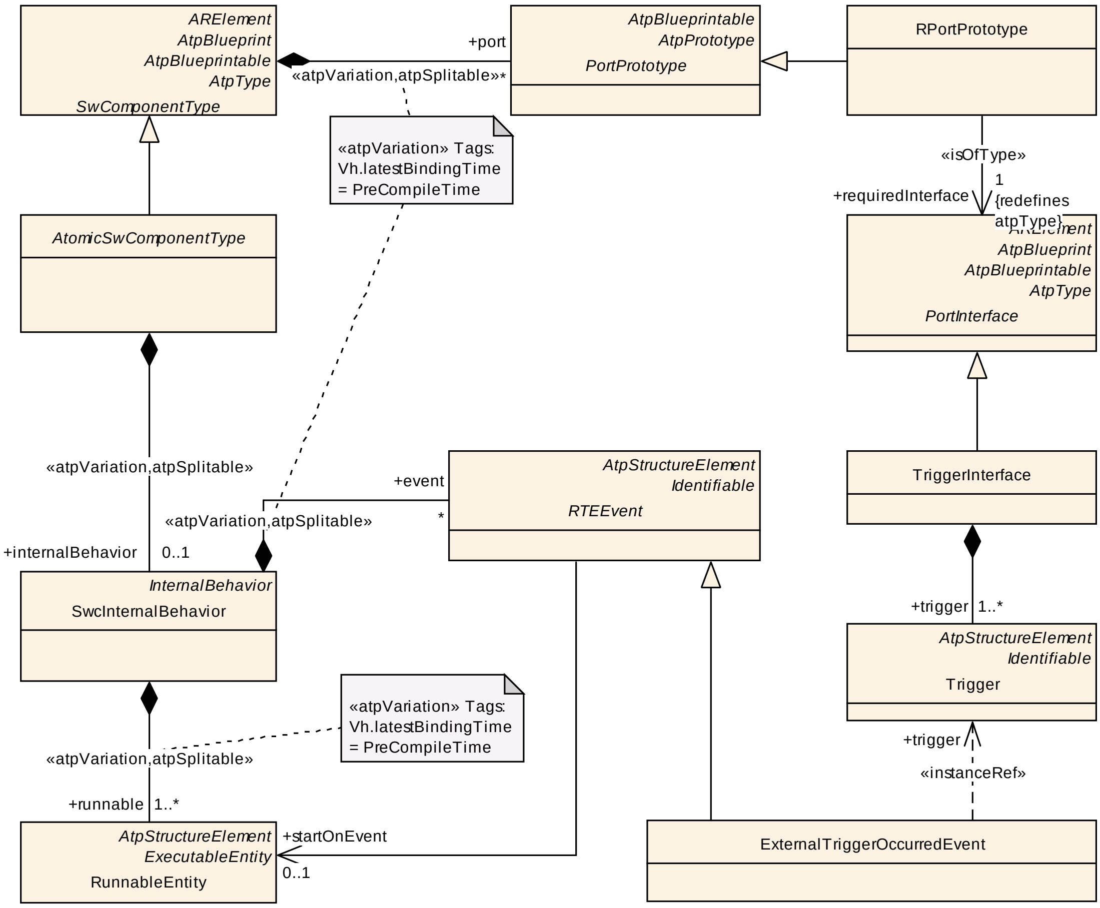

Figure 7.23: Model structure of a trigger sink

Class	**ExternalTriggerOccurredEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced trigger have been occurred.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
trigger	Trigger	1	iref	Reference to the applicable Trigger.

Table 7.35: ExternalTriggerOccurredEvent

7.5.4 RunnableEntities and Parameter Access

There are several ways a Calibration Parameter is provided within a software component.

[TPS_SWCT_1350] Calibration Parameters shared among several `SwComponentTypes` ⌈ As mentioned above, if Calibration Parameters are shared among several `SwComponentType`s a dedicated `PortInterface` in a `PortPrototype` will be used. ⌋ *(RS_SWCT_0200)*

The designer of a software-component can use this access mechanism when designing a RunnableEntity using, as input value, a DataPrototype

- from an arbitrary RPortPrototype associated either with a ClientServerInterface, SenderReceiverInterface or a NvDataInterface,
- VariableDataPrototype in the context of an SwcInternalBehavior

This input value will be fed to an interpolation routine whose result can be used internally or transferred to a adjacent SwComponentPrototype via dedicated PortPrototypes. Typically, there will be a dedicated RunnableEntity (with "ReceiveMode" set to "activation_of_runnable_entity") that itself calls the interpolation routine with the appropriate input value and the appropriate ParameterDataPrototype.

Note that the ParameterAccess also allows to set input values or shared axis through SwDataDefProps which are specific to the access point.

The result of this interpolation routine call is provided as an ArgumentDataPrototype with Direction being either set to out or inout in a ClientServerInterface.

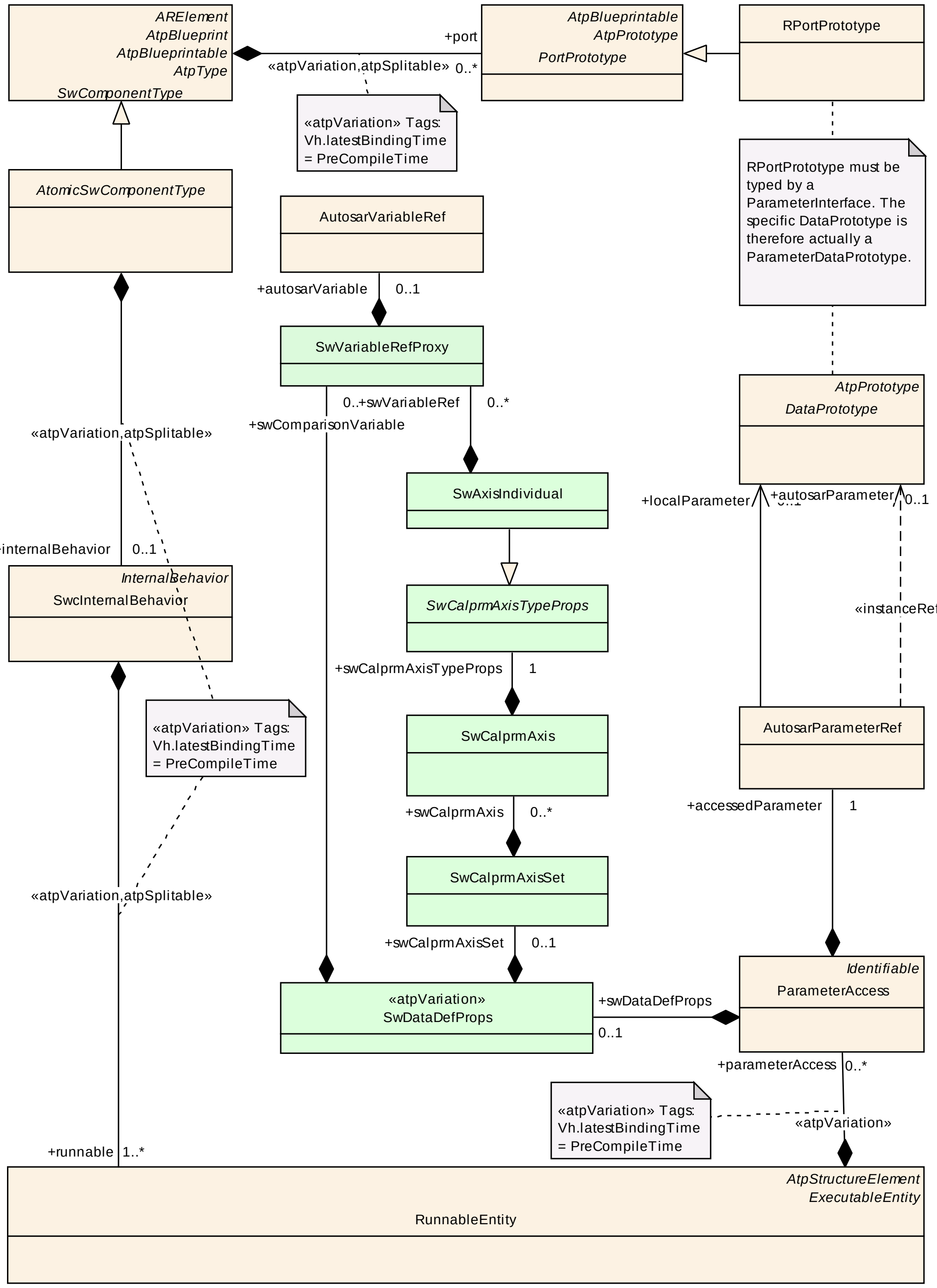

Figure 7.24: Runnable Access to a Calibration Port

Class	**ParameterAccess**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Data Elements			
Note	The presence of a ParameterAccess implies that a RunnableEntity needs access to a ParameterDataPrototype.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
accessedParameter	AutosarParameterRef	1	aggr	Refernce to the accessed calibration parameter.
swDataDefProps	SwDataDefProps	0..1	aggr	This allows denote instance and access specific properties, mainly input values and common axis.

Table 7.36: ParameterAccess

[TPS_SWCT_1351] Access to a ParameterDataPrototype ⌈The access to a ParameterDataPrototype will be indicated

- by the ParameterAccess entity if the RunnableEntity wants to access it from a RPortPrototype. This is shown in Figure 7.24
- by defining the ParameterAccess association from a RunnableEntity to the ParameterDataPrototype in the roles sharedParameter or perInstanceParameter. This is shown in Figure 2.3 in the lower association from RunnableEntity to ParameterDataPrototype

⌋*(RS_SWCT_0200)*

Note: A ParameterDataPrototype in the roles constantMemory is not provided by the RTE and therefore the ParameterAccess association is not required to control the RTE API generation.

Typically the accessibility and further information like alias names for a particular data is modeled on the level of DataPrototypes (especially VariableDataPrototypes, ParameterDataPrototypes). But due to the recursive structure of the meta-model concerning data types (a composite (data) type consists of data prototypes) a part of the MCD information is described in the data type (in case of composite data type).

This is a strong restriction in the reuse of data typed because the data type should be re-used for different VariableDataPrototypes and ParameterDataPrototypes to guarantee type compatibility on C-implementation level (e.g. data of a Port is stored in PIM or NvRom Block shall be from same data type as NvRAM Block).

This restriction is overcome by InstantiationDataDefProps as shown in figure 7.25

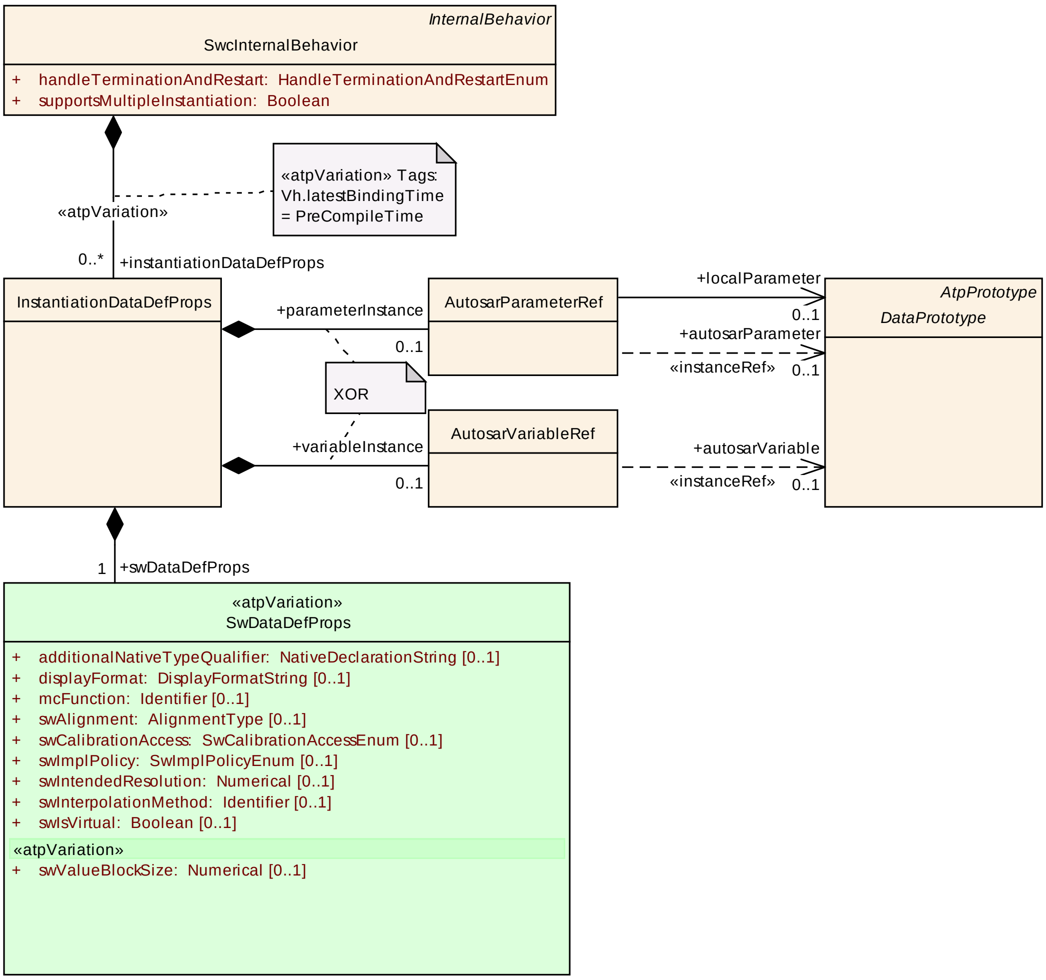

Figure 7.25: applying instantiation specific data definition properties

.

Class	InstantiationDataDefProps			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::InstantiationDataDefProps			
Note	This is a general class allowing to apply additional SwDataDefProps to particular instantiations of a DataPrototype. Typically the accessibility and further information like alias names for a particular data is modeled on the level of DataPrototypes (especially VariableDataPrototypes, ParameterDataPrototypes). But due to the recursive structure of the meta-model concerning data types (a composite (data) type consists out of data prototypes) a part of the MCD information is described in the data type (in case of ApplicationCompositeDataType). This is a strong restriction in the reuse of data typed because the data type should be re-used for different VariableDataPrototypes and ParameterDataPrototypes to guarantee type compatibility on C-implementation level (e.g. data of a Port is stored in PIM or NvRom Block shall be from same data type as NvRAM Block). This class overcomes such a restriction if applied properly.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
parameterInstance	AutosarParameterRef	0..1	aggr	This is the particular ParameterDataPrototypes on which the swDataDefProps shall be applied.
swDataDefProps	SwDataDefProps	1	aggr	These are the particular data definition properties which shall be applied
variableInstance	AutosarVariableRef	0..1	aggr	This is the particular VariableDataPrototypes on which the swDataDefProps shall be applied.

Table 7.37: InstantiationDataDefProps

7.5.5 RunnableEntities and Mode Communication

For the communication of modes between `RunnableEntity`s we have to distinguish between two use cases.

[TPS_SWCT_1352] Requested mode is just sent and received as an ordinary data value ⌈ In the first case, a requested mode is just sent and received as an ordinary data value without specifying the details of mode switching in the corresponding port interface.

This mechanism is used if the receiving `RunnableEntity` is not directly implementing a mode switch but does further processing of the mode request. This is especially needed to transfer mode requests between ECUs.

In this case, the mode is transferred via sender-receiver communication so that the involved `RunnableEntity`s just need the same type of APIs against the RTE as for sender-receiver communication. This is possible, because `ModeDeclarationGroupPrototype`s can be mapped to an `ImplementationDataType`s. This concept and the meta-classes needed for the mapping are further explained in chapter 4.2.5. ⌋*(RS_SWCT_0200)*

[TPS_SWCT_1353] `RunnableEntitys` react on a mode request via a corresponding `RTEEvent` ⌈ In the second case, one `RunnableEntity` "sends" a mode request and one or more other `RunnableEntity`s react on the request via a corresponding `RTEEvent` or by being suppressed from being triggered any longer by other `RTEEvent`s.

In this case, special APIs against the RTE are required and the RTE has to implement the actual mode switch. This kind of communication is only possible between software-components on the same ECU. For further explanation of the general concept refer to chapter 4.2.5 and for the details of the meta-model for mode switches refer to chapter 9. ⌋*(RS_SWCT_0200)*

7.6 Port API Options

[TPS_SWCT_1354] `PortAPIOption` ⌈ The RTE Generator needs additional options per `PortPrototype` to choose the proper generation schema. These are subsumed in the `PortAPIOption` element which is shown in Figure 7.26. ⌋

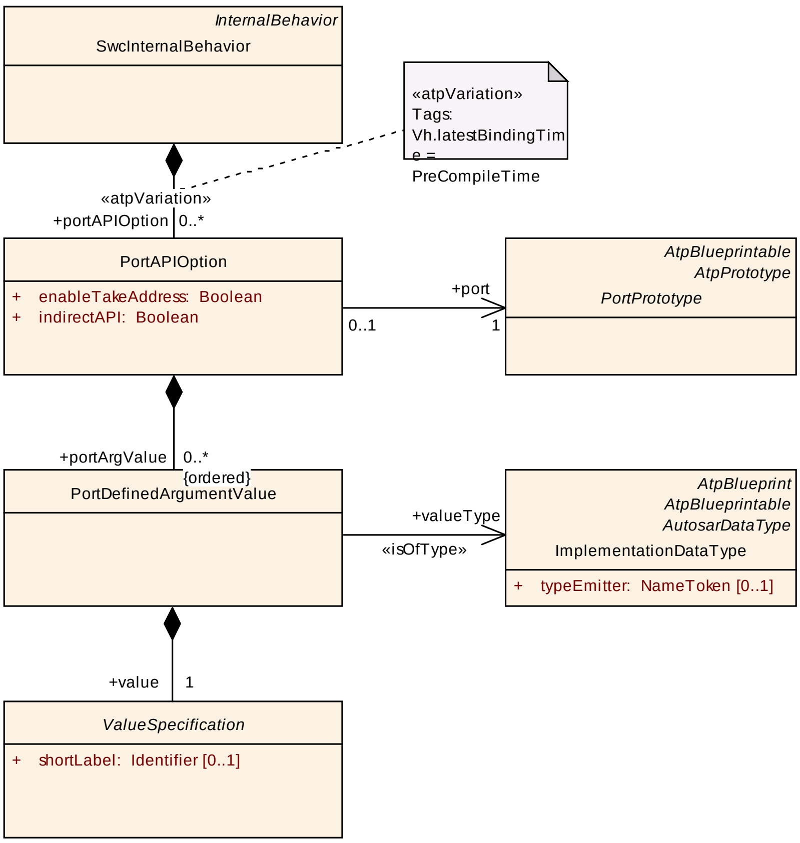

Figure 7.26: Port API Options.

Class	PortAPIOption			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	Options how to generate the signatures of calls for an AtomicSwComponentType in order to communicate over a PortPrototype (for calls into a RunnableEntity as well as for calls from a RunnableEntity to the PortPrototype).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
enableTakeAddress	Boolean	1	attr	If set to true, the software-component is able to use the API reference for deriving a pointer to an object.
indirectAPI	Boolean	1	attr	If set to true this attribute specifies an "indirect API" to be generated for the associated port which means that the SWC is able to access the actions on a port via a pointer to an object representing a port. This allows e.g. iterating over ports in a loop. This option has no effect for PPortPrototypes of client/server interfaces.
port	PortPrototype	1	ref	The option is valid for generated functions related to communication over this port
portArgValue (ordered)	PortDefinedArgumentValue	*	aggr	An argument value defined by this port.

Table 7.38: PortAPIOption

7.6.1 Enable to TakeAddress

[TPS_SWCT_1355] `enableTakeAddress = TRUE` ⌈ If the attribute `enableTakeAddress = TRUE` the generated API related to this `PortPrototype` is provided in a way that the software-component is able to use the API reference for deriving an pointer to an object. ⌋

The main focus of the feature is support for configuration of AUTOSAR Services which are limited to single instances.

[constr_2024] `enableTakeAddress` is restricted to single instantiation ⌈ `PortAPIOption` with `enableTakeAddress` set to `true` is only permitted for software components where the `supportsMultipleInstantiation` attribute of the `SwcInternalBehavior` is set to false. ⌋

7.6.2 Indirect API Generation

[TPS_SWCT_1356] `indirectAPI` option switches the generation of the RTE's indirect API functionality ⌈ The `indirectAPI` option switches the generation of the RTE's indirect API functionality for a certain `PortPrototype`. The generated indirect API does allow to iterate over ports within the SW-Component. ⌋

7.6.3 Port Defined Argument Value

[TPS_SWCT_1357] Definition of implicit values that are passed by the RTE to the server's entry point ⌈ In addition to the formal parameters of a client/server invocation that are defined as part of the server's PortInterface, it is possible to specify a number of implicit values that are passed by the RTE to the server's entry point. ⌋

The initial need for this feature arises in the context of basic software services - although it is not limited to those.

For a service like the NVRAM manager every accessing port is in addition to its logical identity - as a sequence of ShortNames - uniquely identified through a NVRAM specific memory block id. This block id shall be defined in the context of ECU integration and not by the client components.

Instead of exposing this mechanism on the logical ClientServerInterface level in form of a formal Argument, one or more PortDefinedArgumentValues can be specified.

[TPS_SWCT_1358] Values are hidden from the client components ⌈ Because these values are specified in the context of the provide-port only they are hidden from the client components keeping their design and code independent from the server component details. ⌋

In the example of the NVRAM manager, this allows to define the block id in the context of ECU integration and not by the client components.

Figure 7.26 shows the meta-model of Port API Options and the portArgValue.

[constr_1150] Usage of valueType for PortDefinedArgumentValue ⌈ The valueType (typically this boils down to integer values used to specify an "id") associated with PortDefinedArgumentValue shall be of category VALUE or TYPE_REFERENCE. The latter case is only supported if the value of category of the target data type is set to VALUE. ⌋

In case of a PPortPrototype of the NVRAM example this list would have just one value of type int8 or int16 holding the memory block id.

Class	**PortDefinedArgumentValue**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PortAPIOptions			
Note	A PortDefinedArgumentValue is passed to a RunnableEntity dealing with the ClientServerOperations provided by a given PortPrototype. Note that this is restricted to PPortPrototypes of a ClientServerInterface.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	ValueSpecification	1	aggr	Specifies the actual value.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
valueType	Implementation DataType	1	tref	The implementation type of this argument value. It should not be composite type or a pointer. **Stereotypes:** isOfType

Table 7.39: PortDefinedArgumentValue

7.7 PerInstanceMemory

[TPS_SWCT_1359] Private memory per instance ⌈ AtomicSwComponentTypes that support multiple instantiation (attribute supportsMultipleInstantiation == TRUE) will typically need a given amount of private memory per instance. It is the responsibility of the RTE to provide a mechanisms with which each instance of an AtomicSwComponentType can access its own instance-specific memory. ⌋

[TPS_SWCT_1360] Arbitrary number of per-instance memory blocks ⌈ An AtomicSwComponentType can define an arbitrary number of per-instance memory blocks. ⌋

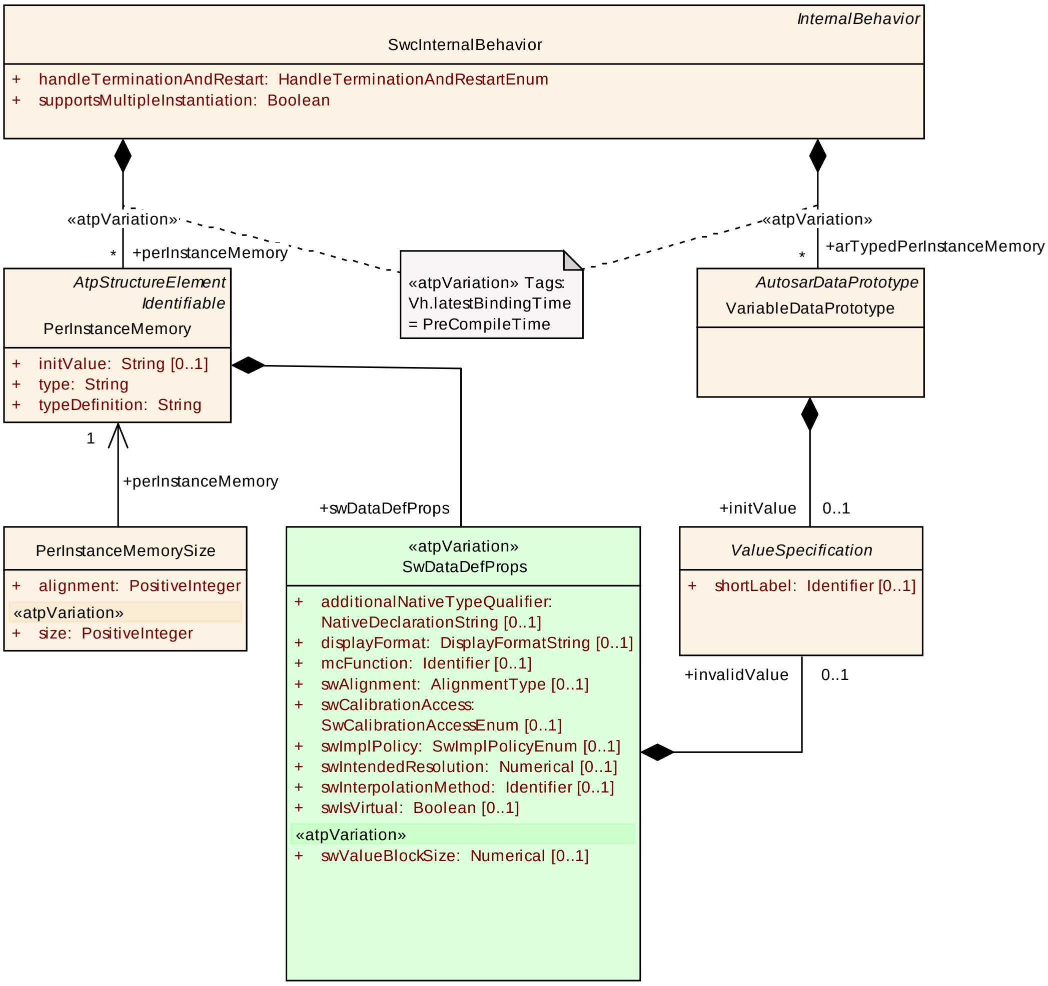

Figure 7.27: PerInstanceMemory

[TPS_SWCT_1361] attribute supportsMultipleInstantiation == FALSE ⌈ AtomicSwComponentTypes that do *not* support multiple instantiation (attribute supportsMultipleInstantiation == FALSE) do not necessarily need to use the PerInstanceMemory: because there will only be a single instance of the AtomicSwComponentType on an ECU, the AtomicSwComponentType can use static variables to store the AtomicSwComponentType's internal state. However, the usage of PerInstanceMemory is also allowed in this case. ⌋

[TPS_SWCT_1362] Initialization of `PerInstanceMemory` ⌈ Note that the PerInstanceMemory is not initialized by the RTE if no initValue is defined. In this case, it is the responsibility of the AtomicSwComponentType to initialize the PerInstanceMemory. ⌋

7.7.1 PerInstanceMemory typed by 'C' Data Types

[TPS_SWCT_1363] `PerInstanceMemory` typed by 'C' Data Types ⌈ For each such memory block, the software-component description shall provide the name of the data type (the "C"-type) it needs to store in the memory block in the attribute type. This attribute allows for the RTE to generate an API function that provides a convenient and type-safe access to the data item.

In addition, the software-component description shall define the data type in the attribute typeDefinition. This attribute is supposed to contain a *C* typedef of the data type in valid C-syntax. ⌋

In other words, this typeDefinition shall be formulated such that it can be included verbatim in a C header file.

[constr_2007] Consistency of `typeDefinition` attribute ⌈ Please note that all PerInstanceMemorys of the same SwcInternalBehavior with identical type attribute shall defined the identical typeDefinition attribute as well. ⌋

[TPS_SWCT_1364] Initial value of a `PerInstanceMemory` typed by 'C' Data Types ⌈ The initValue is a comma separated list which can be used verbatim by the RTE generator as constant initializer. ⌋

More details on the use of these attributes in the generation of software-component header-files can be found in the RTE specification [2].

Class	**PerInstanceMemory**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::PerInstanceMemory			
Note	Defines a 'C' typed memory-block that needs to be available for each instance of the SW-component. This is typically only useful if supportsMultipleInstantiation is TRUE of if the component defines NVRAM access via permanent blocks.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
initValue	String	0..1	attr	Specifies initial value(s) of the PerInstanceMemory
swDataDefProps	SwDataDefProps	1	aggr	This represents the ability to to allocate RAM at specific memory sections, for example, to support the RAM block recovery strategy by mapping to uninitialized RAM.
type	String	1	attr	The name of the "C"-type
typeDefinition	String	1	attr	A definition of the type with the syntax of a 'C' typedef.

Table 7.40: PerInstanceMemory

7.7.2 PerInstanceMemory typed by AUTOSAR Data Types

[TPS_SWCT_1365] `PerInstanceMemory` typed by AUTOSAR Data Types ⌈ A `PerInstanceMemory` typed with AUTOSAR data types is defined by a `VariableDataPrototype` in the role `arTypedPerInstanceMemory`. `VariableDataPrototype` is derived from `DataPrototype` which has an association to an AUTOSAR `Datatype`. ⌋

This defines the data type of the AUTOSAR-typed `PerInstanceMemory`.

[TPS_SWCT_1366] Initial value of a `PerInstanceMemory` typed by AUTOSAR Data Types ⌈ The `initValue` is described with a `ValueSpecification` ⌋

[TPS_SWCT_1367] Typed by AUTOSAR data type vs. typed by C data type ⌈ In difference to the 'C' typed `PerInstanceMemory` the AUTOSAR-typed `PerInstanceMemory` is able to define information controlling the visibility in a MCD system via a `SwDataDefProps` for the purpose of measurement (see chapter 5.4.3) or defining an input value of an axis (see chapter 5.4.5). ⌋

Note: Due to the use of `AutosarDataType` the AUTOSAR-typed `PerInstanceMemory` can not support C++ specific types or pointer types directly.

7.8 Static Memory and Constant Memory

The `Software Component Template` provides the means to describe static and constant memory in the `InternalBehavior`.

[TPS_SWCT_1368] Describe static and constant memory ⌈ These are described via a `VariableDataPrototype` in `staticMemory` role or a `ParameterDataPrototype` in `constantMemory` role. ⌋

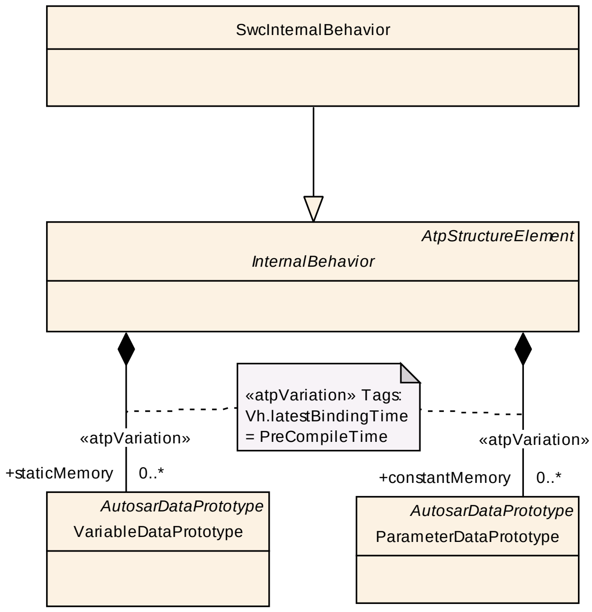

Figure 7.28: Static Memory and Constant Memory

The information about these characteristic values and variables is given with the purpose to support Measurement and Calibration (see chapter 2.2) and has to be taken into account for the A2L file generation.

[TPS_SWCT_1369] Static and constant memory is not instantiated by the RTE ⌈ In contrast to the other kinds of memory like implicitInterRunnableVariable, implicitInterRunnableVariable, PerInstanceMemory, sharedParameter or perInstanceParameter the staticMemory and constantMemory are **not** instantiated by the RTE. ⌋

This allows for more efficient implementations (especially for software-components provided as object code) by avoidance of the additional indirection caused by the RTE's component data structure.

Further on, this kind of memory reduces the dependencies of the software-component implementation to generated RTE code which is appreciated for safety related functionalities.

Due to the instantiation of the memory by the software-component's implementation the constantMemory behaves like a sharedParameter (see chapter 2.2.3.2)

[constr_2028] staticMemory is restricted to single instantiation ⌈ The staticMemory is only supported if the attribute supportsMultipleInstantiation of the owning SwcInternalBehavior is set to FALSE ⌋

This constraint prevents hidden communication between SwComponentPrototypes of the same SwComponentType.

[constr_2029] shortName of constantMemory and staticMemory ⌈ The shortName of a VariableDataPrototype in role staticMemory or a ParameterDataPrototype in role constantMemory has to be equal with the 'C' identifier of the described variable resp. constant. ⌋

7.9 Included AUTOSAR Data Types

[TPS_SWCT_1155] IncludedDataTypeSet ⌈ An IncludedDataTypeSet declares that a set of AutosarDataTypes are used for the C / C++ implementation of the software component. The AutosarDataTypes become part of the contract. ⌋

[TPS_SWCT_1156] Required if the AutosarDataType is not used for any DataPrototype ⌈ This information is required if the AutosarDataType is not used for any DataPrototype owned by this software component or if a prefix for C language identifiers belonging to AutosarDataTypes shall be defined. ⌋

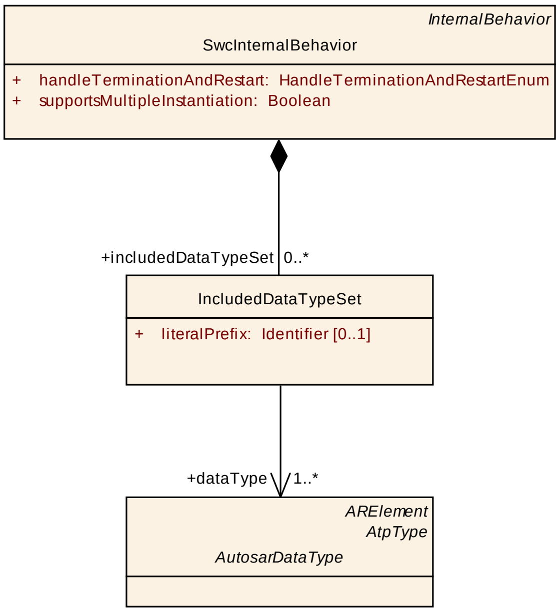

Figure 7.29: Included AUTOSAR Data Types

Class	**IncludedDataTypeSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::IncludedDataTypes			
Note	An includedDataTypeSet declares that a set of AutosarDataType is used by the software component for its implementation and the AutosarDataType becomes part of the contract. This information is required if the AutosarDataType is not used for any DataPrototype owned by this software component or if the enumeration literals, lowerLimit and upperLimit constants shall be generated with a literalPrefix. The optional literalPrefix is used to add a common prefix on enumeration literals, lowerLimit and upperLimit constants created by the RTE.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
dataType	AutosarDataType	1..*	ref	AutosarDataType belonging to the includedDataTypeSet
literalPrefix	Identifier	0..1	ref	LiteralPrefix defines a common prefix for all AutosarDataTypes of the includedDataTypeSet to be added on enumeration literals, lowerLimit and upperLimit constants created by the RTE.

Table 7.41: IncludedDataTypeSet

This supports the common usage of the AUTOSAR data type system for RTE provided memory objects and memory objects declared by the software component implementation.

Further on, this enables the generation of the RTE Application Types Header File for AUTOSAR services containing the required data types for the C-API before the data type usage in dedicated ports for an ECU is known.

[TPS_SWCT_1157] Attribute `literalPrefix` of `IncludedDataTypeSet` ⌈ In addition the `literalPrefix` might be used to separate the namespace of C language identifiers belonging to equally named `AutosarDataType`s used for the same software component C implementation. ⌋

7.10 Included Mode Declaration Groups

[TPS_SWCT_1153] `IncludedModeDeclarationGroupSet` ⌈ Similar to the consideration of data types using `IncludedDataTypeSet`, `SwcInternalBehavior` aggregates `IncludedModeDeclarationGroupSet` that in turn allows for referencing `ModeDeclarationGroup`s with the intent to express that the referenced `ModeDeclarationGroup`s are used in the context of the enclosing `AtomicSwComponentType`. ⌋

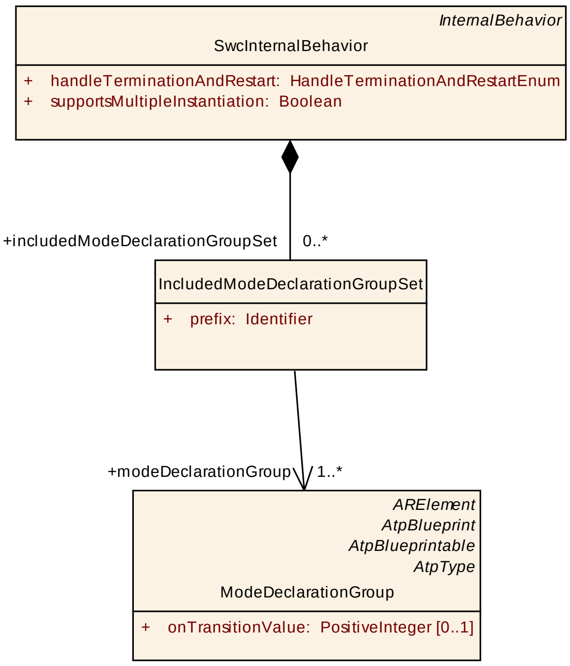

Figure 7.30: Included `ModeDeclarationGroups`

Class	**IncludedModeDeclarationGroupSet**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	An IncludedModeDeclarationGroupSet declares that a set of ModeDeclarationGroups used by the software component for its implementation and consequently these ModeDeclarationGroups become part of the contract.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
modeDeclarationGroup	ModeDeclarationGroup	1..*	ref	This represents the referenced ModeDeclarationGroup.

Attribute	Datatype	Mul.	Kind	Note
prefix	Identifier	1	ref	The prefix shall be used by the RTE generator as a prefix for the creation of symbols related to the referenced ModeDeclarationGroups, e.g RTE_TRANSITION_<ModeDeclarationGroup>.

Table 7.42: IncludedModeDeclarationGroupSet

[TPS_SWCT_1154] Attribute `prefix` of `IncludedModeDeclarationGroupSet` ⌈ The attribute `prefix` of `IncludedModeDeclarationGroupSet` can be used to define a prefix that the RTE generator shall use to define symbols related to the included `ModeDeclarationGroup`s with the intent to avoid potential name clashes. ⌋

7.11 Service Needs

7.11.1 Overview

[TPS_SWCT_1043] `ApplicationSwComponentType`s are independent from actual ECU Hardware ⌈ `ApplicationSwComponentType`s are designed to be independent of their mapping to actual ECU Hardware. ⌋*(RS_SWCT_2060)*

However, each software-component might need services which are provided by the ECU Basic Software through AUTOSAR Services.

[TPS_SWCT_1044] `ServiceNeeds` ⌈ The `ServiceNeeds` (see Figures 7.31, 7.32, and 7.33) are used to provide detailed information what the software-component expects from the AUTOSAR Services when integrated on an actual ECU. Note that only `AtomicSwComponentType`s and `NvBlockSwComponentType`s can be connected to AUTOSAR Services. ⌋*(RS_SWCT_2060)*

[TPS_SWCT_1045] Actual values of ECU configuration parameters fulfill the requirements given by the `ServiceNeeds` ⌈ When integrating application software-components on an ECU, the actual values of ECU configuration parameters shall be chosen so that they fulfill the requirements given by the `ServiceNeeds` of all the integrated `AtomicSwComponentType`s. ⌋*(RS_SWCT_2060)*

Note that the actual values of configuration parameters will in addition depend on the properties of the basic software and the hardware of that specific ECU, see also chapter 11. For further information about the relation between the `ServiceNeeds` and the ECU configuration parameters see [26].

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 7.43: ServiceNeeds

The meta-class `ServiceNeeds` and the sub-classes for several Services are located in the `CommonStructure` package of the meta-model because they are also used in the `Basic Software Module Description Template` [7].

The meta-classes derived from `ServiceNeeds` is shown in the next three figures.

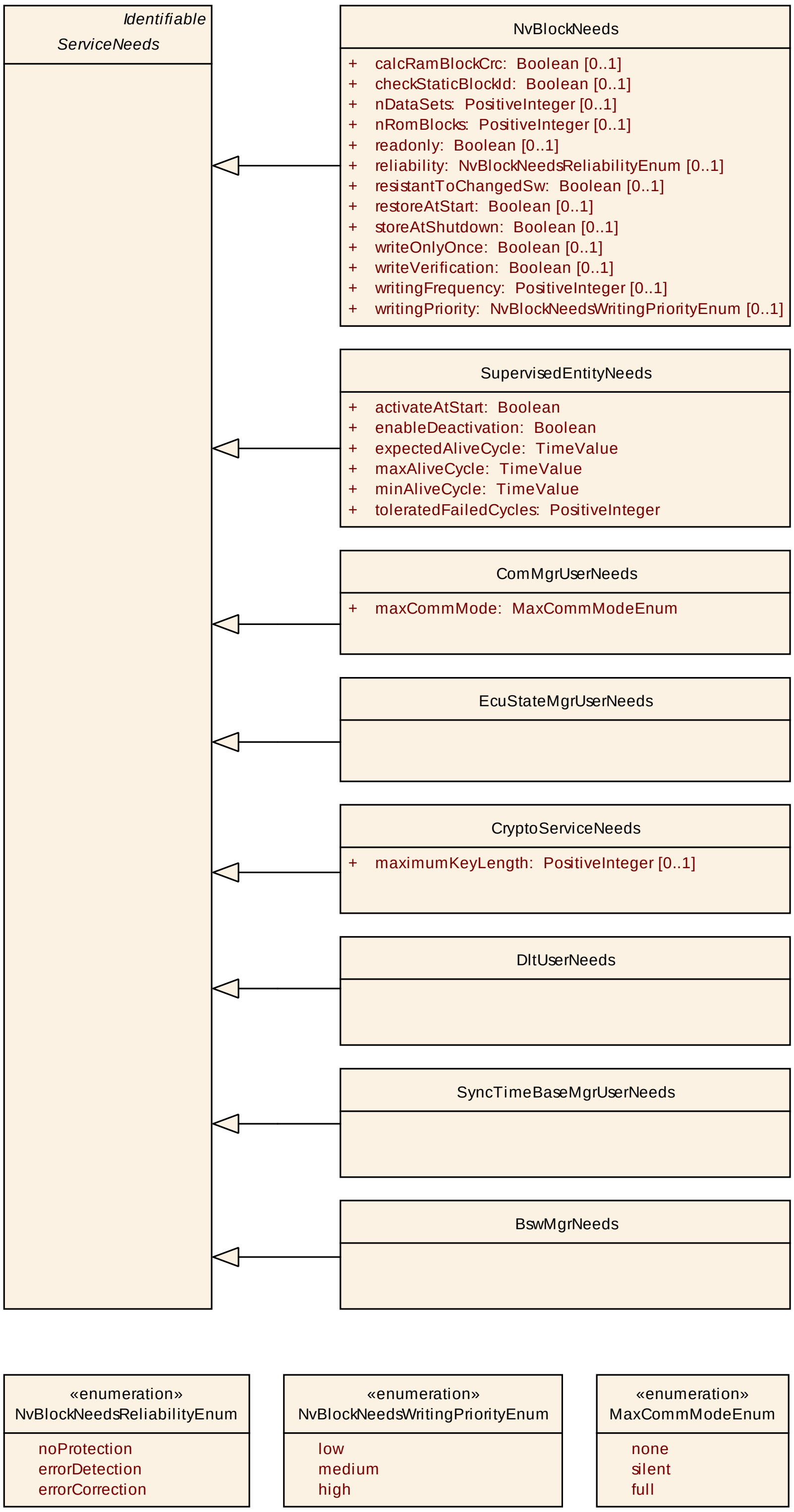

Figure 7.31: `ServiceNeeds`: General `ServiceNeeds`

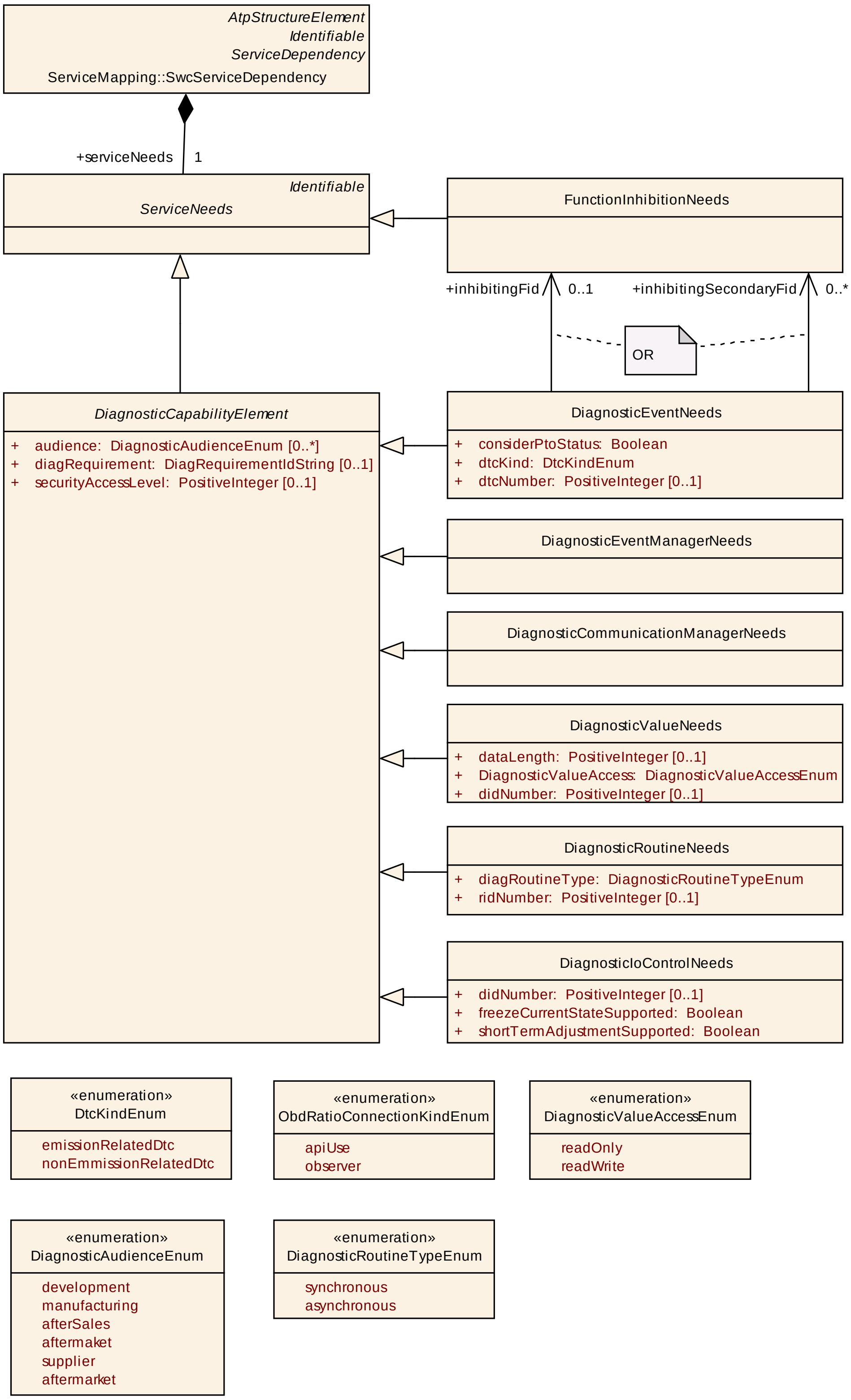

Figure 7.32: `ServiceNeeds`: General diagnostic-related `ServiceNeeds`

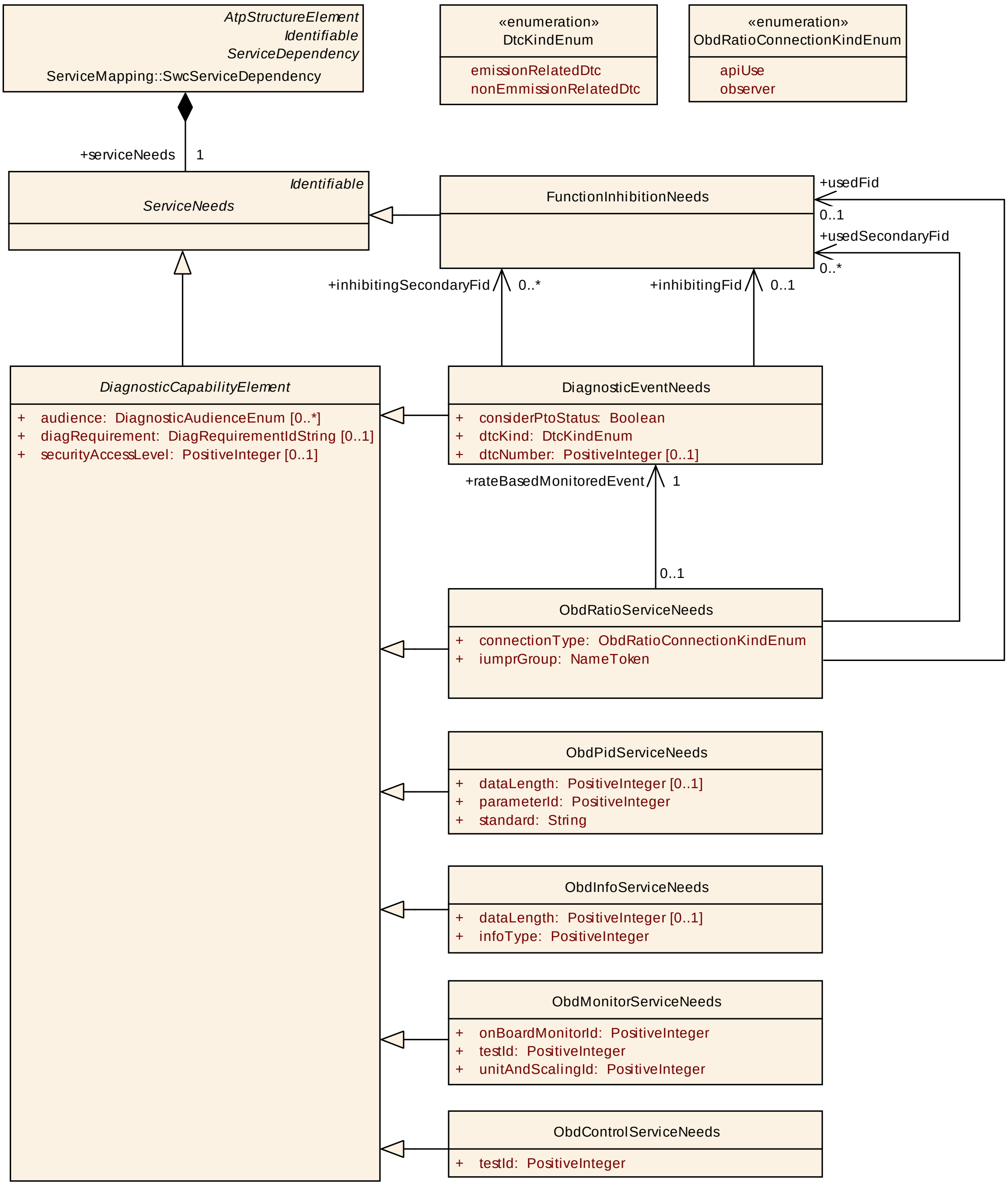

Figure 7.33: `ServiceNeeds`: Diagnostic-related `ServiceNeeds` with emphasis on OBD

7.11.2 Assignment of Service Needs to Ports and Data

[TPS_SWCT_1046] `ServiceNeeds` are defined in the scope of the `SwcInternalBehavior` ⌈ `ServiceNeeds` specified by `AtomicSwComponentType`s are defined in the scope of the `SwcInternalBehavior` because in several cases they need associations to other parts of the `SwcInternalBehavior`. In most cases they are related to certain ports belonging to the `AtomicSwComponentType`s because `Atom-`

icSwComponentTypes communicate with AUTOSAR Services via these PortPrototypes. ⌋*(RS_SWCT_2060)*

In addition, a ServiceNeeds element can also have relations to some data declared within the same SwcInternalBehavior, namely some use cases of the NVRAM Service require statically defined RAM mirror data and/or ROM default data declared in the context of the single software component.

A further use case requires that a ServiceNeeds element is linked to a PortGroup. Especially, a ServiceNeeds can represent a group of ports as input to configure the communication manager in order to handle the communication state of those ports.

These relationships to ports, data and port groups are required as input for tools in order to generate the XML descriptions and configurations of the basic software which implements the Service according to the needs of several atomic software components are integrated on an ECU, see chapter 11.

The relationship to ports is defined via the meta-class RoleBasedPortAssignment and the relationship to data is defined via the meta-class RoleBasedDataAssignment. Both are aggregating an attribute role which allows to defined the role of the ports or data in the specific context.

[constr_2027] SwcServiceDependency shall be defined for service ports only ⌈ A PortPrototype that is referenced by a SwcServiceDependency via assignedPort shall be typed by a PortInterface that has isService set to TRUE. This rule does **not** apply to PortPrototypes used in the context of NV data management, i.e. for connections between an ApplicationSwComponentType and an NvBlockSwComponentType. ⌋

Please consider: it is permitted that a SwcServiceDependency containing a DiagnosticValueNeeds may reference via assignedData a dataElement instance in a PPortPrototype typed by a SenderReceiverInterface that has its attribute isService set to FALSE.

The actual mapping between the ServiceNeeds element and its various relationships is provided by the meta-class SwcServiceDependency as shown in figure 7.34. Note the difference between the associations to PortPrototypes and to PortGroups: While the RoleBasedPortAssignment is part of the SwcInternalBehavior a PortGroup is defined for the SwComponentType (thus belongs to the VFB level) and it is linked to the PortGroups of other SwComponentTypes.

This means a PortGroup represents a system feature, whereas the RoleBasedPortAssignment is a local feature for the purpose of communication with the AUTOSAR Service.

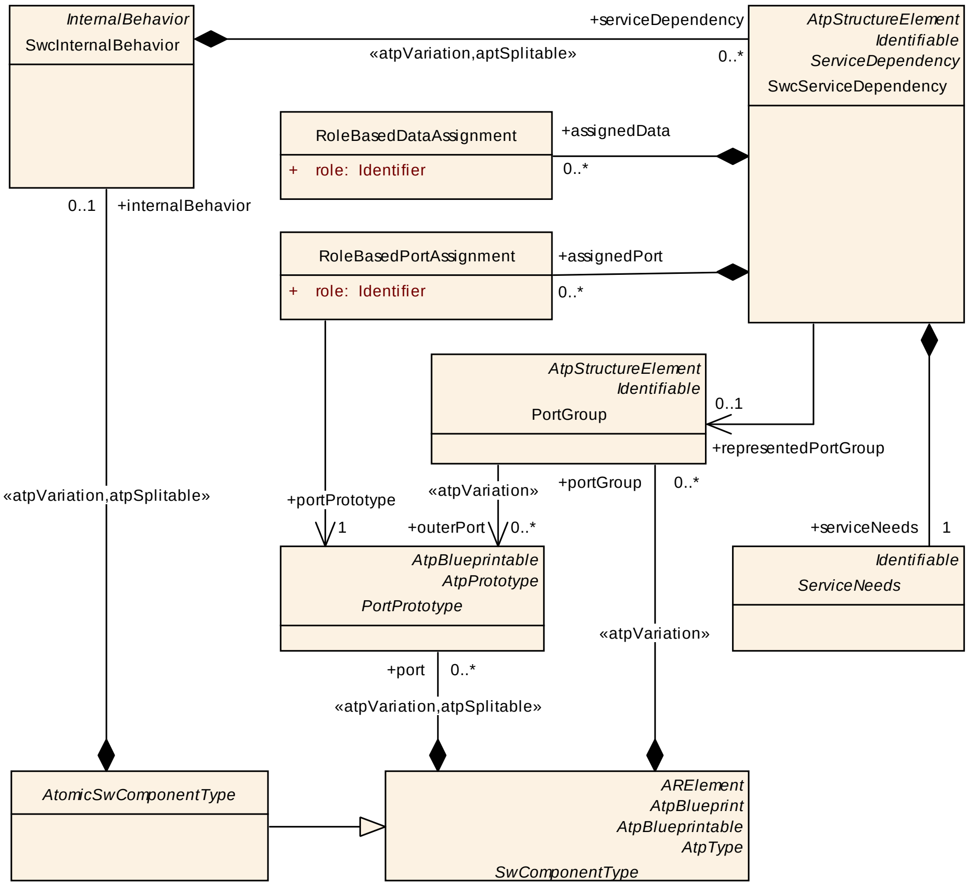

Figure 7.34: `SwcServiceDependency` in the `SwcInternalBehavior`

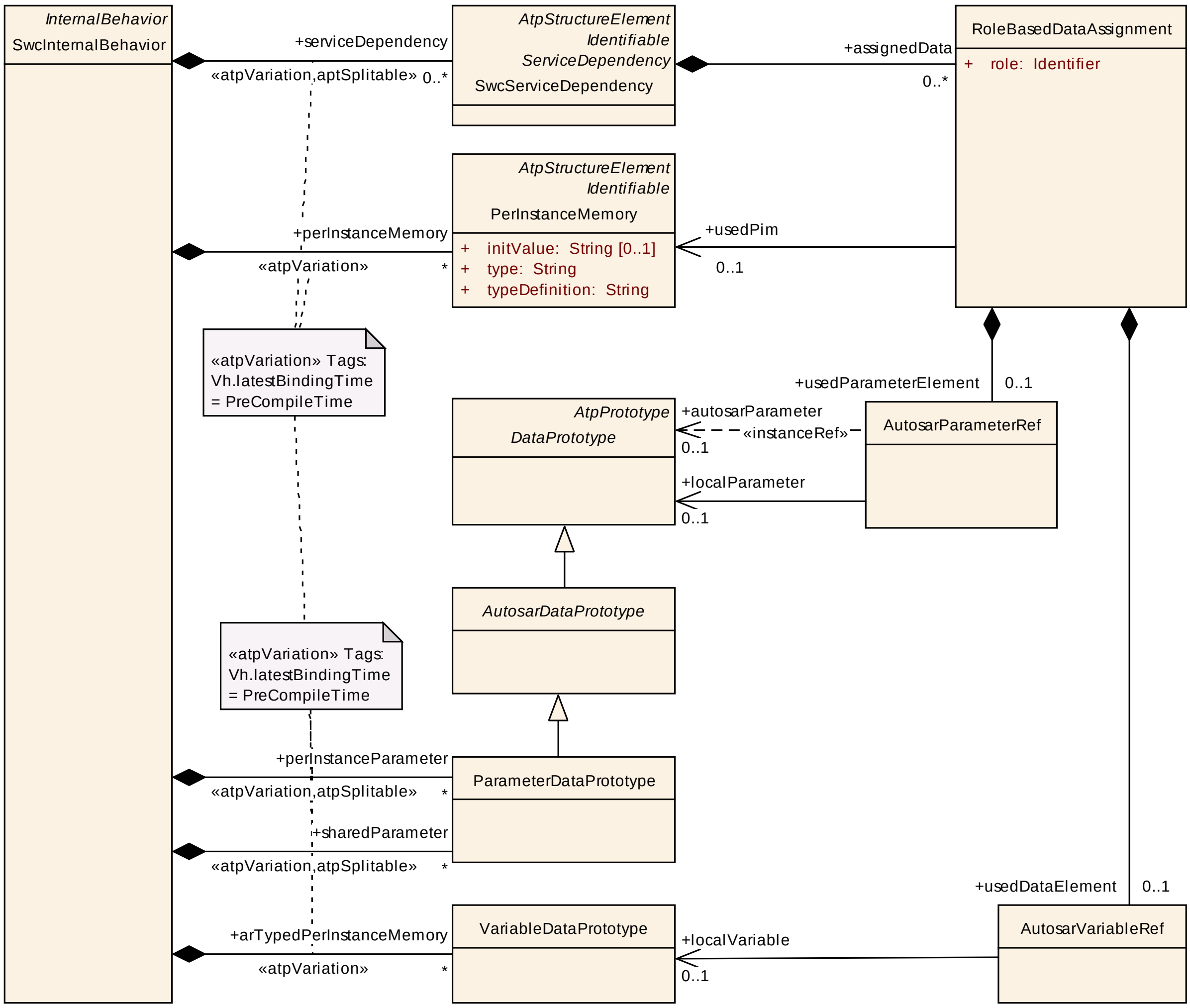

Figure 7.35: Details of `RoleBasedDataAssignment` for local data

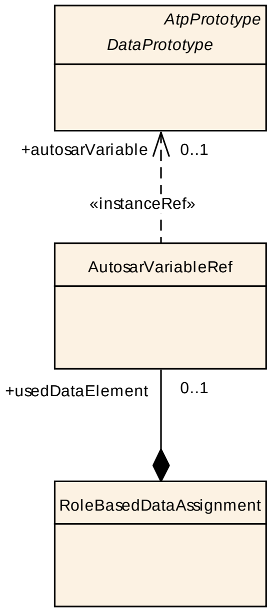

Figure 7.36: Details of `RoleBasedDataAssignment` for access into `PortPrototypes`

Class	SwcServiceDependency			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	Specialization of ServiceDependency in the context of an SwcInternalBehavior. It allows to associate ports, port groups and (in special cases) data defined for an atomic software component to a given ServiceNeeds element.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,MultilanguageReferrable,Referrable,ServiceDependency			
Attribute	Datatype	Mul.	Kind	Note
assignedData	RoleBasedDataAssignment	*	aggr	Defines the role of an associated data object of the same component.
assignedPort	RoleBasedPortAssignment	*	aggr	Defines the role of an associated port of the same component.
representedPortGroup	PortGroup	0..1	ref	This reference specifies an association between the ServiceNeeeds and a PortGroup, for example to request a communication mode which applies for communication via these ports. The referred PortGroup must be local to this atomic SWC, but via the links between the PortGroups, a tool can evaluate this information such that all the ports linked via this port group on the same ECU can be found.
serviceNeeds	ServiceNeeds	1	aggr	The associated ServiceNeeds.

Table 7.44: SwcServiceDependency

Class	RoleBasedPortAssignment			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ServiceMapping			
Note	This class specifies an assignment of a role to a particular service port (RPortPrototype or PPortPrototype) of an AtomicSwComponentType. With this assignment, the role of the service port can be mapped to a specific ServiceNeeds element, so that a tool is able to create the correct connector.			
Base	ARObject			
Attribute	Datatype	Mul.	Kind	Note
portPrototype	PortPrototype	1	ref	Service port used in the assigned role. This port must either belong to the same AtomicSoftwareComponent as the SwcInternalBehavior which owns the ServiceDependency or to the same NvBlockComponentType as the NvBlockDescriptor.
role	Identifier	1	ref	This is the role of the assigned Port in the given context. The value must be a name of a PortInterface as standardized in the Software Specification of the related AUTOSAR Service.

Table 7.45: RoleBasedPortAssignment

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	PerInstanceMemory	0..1	ref	The (untyped) PerInstanceMemory used in this role (e.g. as a RAM mirror for an Nv block).

Table 7.46: RoleBasedDataAssignment

7.11.3 Specific Service Dependencies

7.11.3.1 NvM Service Dependencies

This chapter describes the usage of the specific meta-classes derived from `ServiceNeeds` within an `AtomicSwComponentType`.

The meta-class `NvBlockNeeds` is used to define requirements to configure the NVRAM Manager Service. An `SwcInternalBehavior` may provide several `SwcServiceDependency`s that in turn aggregate an `NvBlockNeeds` element where each defines the requirements from one NV block (for more information on the AUTOSAR NVRAM Manager see [27]). There are several use cases how a software-component can interact with the NVRAM Manager service. Each use case is discussed in a separate sub-chapter.

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.

Attribute	Datatype	Mul.	Kind	Note
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.

Table 7.47: NvBlockNeeds

7.11.3.1.1 `Nvm` Use Case: RAM Mirror

Scenario: a `AtomicSwComponentType` is using an an *NvBlock* with a permanent mirror implemented by a `PerInstanceMemory` section or a `VariableDataPrototype` in the role `arTypedPerInstanceMemory`. In either case, the required memory for the mirror is allocated by the RTE during ECU Configuration.

In this case the following rules apply:

[TPS_SWCT_2501] Setup for `Nvm` Use Case: RAM Mirror ⌈

RoleBasedPortAssignment
For every used `ClientServerInterface` provided by the `NvM` it is necessary to create a `RoleBasedPortAssignment` and set the value of the attribute `role` of the `RoleBasedPortAssignment` to the name of the used standardized `ClientServerInterface`. The following `ClientServerInterface`s shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- `NvmService` [0 .. 1]

- NvMNotifyJobFinished [0 .. 1]
- NvMNotifyInitBlock [0 .. 1]
- NvMAdmin [0 .. 1]

RoleBasedDataAssignment

RoleBasedDataAssignment shall be created that refers to either the PerInstanceMemory in the role usedPim or to the VariableDataPrototype in the role usedDataElement. The value of the attribute role of the RoleBasedDataAssignment shall be set to ramBlock.
Optionally, it is possible to create an additional RoleBasedDataAssignment to a ParameterDataPrototype in the role usedParameterElement. The value of the ParameterDataPrototype is then taken as the initial or default value for the NvBlock. In this case the value of the attribute role of the RoleBasedDataAssignment shall be set to defaultValue.
Therefore, the following roles are applicable:

- ramBlock [0 .. 1]
- defaultValue [0 .. 1]

RepresentedPortGroup

N/A

⌋*(NVM734, NVM735, NVM736, NVM737)*

The same mechanisms applies also for an NvBlockSwComponentType. For each *NvBlock* the NVRAM Manager can be configured (with the help of SwcServiceDependency.assignedData) to use the same RAM mirror.

It is the responsibility of the NVRAM Manager to provide the content of the NV block in this RAM mirror during startup or on explicit request and to write back the content to the storage medium during shut-down or on explicit request.

7.11.3.1.2 Nvm Use Case: Non RAM Mirror

Scenario: an AtomicSwComponentType is using some NV blocks without a permanent RAM mirror. in this case the AtomicSwComponentType is responsible for allocating the allocation of sufficient memory. In other words, the AtomicSwComponentType shall provide a memory area that is available to the API call to the NVRAM Manager for storage of the NV data.

[TPS_SWCT_2502] Setup for Nvm Use Case: Non RAM Mirror ⌈

RoleBasedPortAssignment

This is mandatory for the described scenario. For every used ClientServerInterface provided by the Nvm it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following

ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvmService [1]
- NvMNotifyJobFinished [0 .. 1]
- NvMNotifyInitBlock [0 .. 1]
- NvMAdmin [0 .. 1]

RoleBasedDataAssignment
The usage of a RoleBasedDataAssignment with attribute role set to defaultValue is optional and depends on whether or not an initial value is required.

- defaultValue [0..1]

RepresentedPortGroup
N/A

⌋*(NVM734, NVM735, NVM736, NVM737)*

7.11.3.1.3 Nvm Use Case: RAM Block synchronized using Mirror Interfaces

Scenario: an AtomicSwComponentType is using an NV block where the RAM block is synchronized by means of mirror interfaces. In this case the RAM block does not necessarily have to be formally described by means of a PerInstanceMemory or a VariableDataPrototype in the role arTypedPerInstanceMemory.

Consequently, the software-component itself is responsible for the allocation of memory. On the other hand, this can also mean that the software-component can use several RAM blocks instead of just one RAM block.

[TPS_SWCT_2504] Setup for Nvm Use Case: RAM Block synchronized using Mirror Interfaces ⌈

RoleBasedPortAssignment
This is mandatory for the described scenario. For every used ClientServerInterface provided by the Nvm it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvMService [0..1]
- NvMNotifyJobFinished [0..1]
- NvMNotifyInitBlock [0..1]
- NvMAdmin [0..1]

- NvMMirror [1]

RoleBasedPortAssignment
In this scenario the existence of a RoleBasedDataAssignment is optional. The RoleBasedDataAssignment needs to reference a ParameterDataPrototype aggregated by the enclosing SwcInternalBehavior in the role perInstanceParameter or sharedParameter.

- defaultValue [0..1]

RepresentedPortGroup
N/A

⌋*(NVM734, NVM735, NVM736, NVM737, NVM738)*

7.11.3.1.4 NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType (not ServiceSwComponent of NvM)

Scenario: an AtomicSwComponentType is using an NV block provided by an NvBlockSwComponentType (see section 11.5.2, as opposed to an NV block provided by a ServiceSwComponentType). Constraints [constr_1148], [constr_1149], and [constr_2011] apply.

[TPS_SWCT_2503] Setup for NVM Use Case: Software-Components using Nv Data provided by NvBlockSwComponentType ⌈

RoleBasedPortAssignment
This is mandatory for the described scenario. For every used ClientServerInterface provided by the NvM it is necessary to create a RoleBasedPortAssignment and set the value of the attribute role of the RoleBasedPortAssignment to the name of the used ClientServerInterface. The following ClientServerInterfaces shall (i.e. lower multiplicity > 0) or can (lower multiplicity = 0) be used in this context:

- NvDataPort [1..*]
- NvMService [0..1]
- NvMNotifyJobFinished [0..1]
- NvMNotifyInitBlock [0..1]
- NvMAdmin [0..1]

RoleBasedPortAssignment
N/A

RepresentedPortGroup
N/A

⌋*(NVM734, NVM735, NVM736, NVM737)*

Note that `NvBlockNeeds` described in Chapter 11.5.4) is not in the scope of this use case.

7.11.3.2 Watchdog Service Dependencies

The meta-class `SupervisedEntityNeeds` is used to define requirements to configure the Watchdog Service. For the terms related to the AUTOSAR Watchdog Manager see [28].

7.11.3.2.1 Watchdog Service use Case: Supervision

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 7.48: SupervisedEntityNeeds

Scenario: an `AtomicSwComponentType` contains a *Supervised Entity*. In this case it is required that the *Supervised Entity* indicates to the *Watchdog Manager* that a Checkpoint within the *Supervised Entity* has been reached. Further on the the *Local Supervision Status* of a single *Supervised Entity* may be signaled to the software component. In this case the following setup applies:

[TPS_SWCT_2018] Setup for `AtomicSwComponentType` which contains a Supervised Entity ⌈

RoleBasedPortAssignment valid roles:

- `WdgM_AliveSupervision` [1]
- `WdgM_IndividualMode` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_WDGM_0333, SWS_WDGM_0335)*

Please note that an `SwcInternalBehavior` may provide several `SupervisedEntityNeeds` elements where each defines the requirements in relation to one supervised entity.

7.11.3.2.2 Watchdog Service use Case: *Global Supervision Status* notification

Scenario: an `AtomicSwComponentType` requires to receive the *Global Supervision Status* that is combined from all individual *Supervised Entities*. In this case the following setup applies:

[TPS_SWCT_2019] Setup for `AtomicSwComponentType` which requires *Global Supervision Status* notification ⌈

RoleBasedPortAssignment valid roles:

- `WdgM_GlobalMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_WDGM_0336)*

7.11.3.3 COM Manager Service Needs

The meta-class `ComMgrUserNeeds` is used to define requirements to configure the `ComM` Service. An `SwcInternalBehavior` may provide several `ComMgrUserNeeds` elements where each defines the requirements from one "user" of the ComM Service (for the terms related to the AUTOSAR Communication Manager see [18]). Especially, it defines which `PortGroup` is associated with this "user".

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 7.49: ComMgrUserNeeds

7.11.3.3.1 ComM Use Case: read current ComM Mode

Scenario: a `AtomicSwComponentType` reads the current ComM mode.

In this case the following rules apply:

[TPS_SWCT_1019] `AtomicSwComponentType` reads the current ComM mode ⌈

RoleBasedPortAssignment valid roles:

- `ComM_CurrentMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(SWS_ComM_0847)*

7.11.3.3.2 ComM Use Case: request ComM Mode

Scenario: a `AtomicSwComponentType` requests a ComM mode. It may also check later whether the requested ComM mode has become effective.

In this case the following rules apply:

[TPS_SWCT_1020] `AtomicSwComponentType` requests a ComM mode. It may also check later whether the requested ComM mode has become effective ⌈

RoleBasedPortAssignment valid roles:

- `ComM_CurrentMode` [1]
- `ComM_UserRequest` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the applicable PortGroup [0..1]

⌋(SWS_ComM_0848)

7.11.3.3.3 ComM Use Case: Software-Component acts as a Mode Manager that influences the ECU State

Scenario: a AtomicSwComponentType acts as a mode manager that influences the ECU state.

In this case the following rules apply:

[TPS_SWCT_1021] AtomicSwComponentType acts as a mode manager that influences the ECU state ⌈

RoleBasedPortAssignment valid roles:

- ComM_CurrentMode [0..1]
- ComM_UserRequest [0..1]
- ComM_ECUModeLimitation [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋(SWS_ComM_0741)

7.11.3.4 ECU State Manager Service Needs

The meta-class EcuStateMgrUserNeeds is used to define the requirements to configure the ECU State Manager Service. There are actually two variants of AUTOSAR ECU management: flexible and fixed. An SwcInternalBehavior may provide several EcuStateMgrUserNeeds elements where each defines the requirements from one "user" of the EcuM Service (for the terms related to the AUTOSAR ECU State Manager see [29]).

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 7.50: EcuStateMgrUserNeeds

7.11.3.4.1 EcuM Fixed Use Case: read current ECU Mode

Scenario: a `AtomicSwComponentType` reads the current ECU mode.

In this case the following rules apply:

[TPS_SWCT_1012] `AtomicSwComponentType` reads the current ECU mode (fixed variant) ⌈

RoleBasedPortAssignment valid roles:

- `EcuM_CurrentMode` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(SWS_EcuM_2762, SWS_EcuM_2749)*

7.11.3.4.2 EcuM Fixed Use Case: request a certain ECU state

Scenario: a `AtomicSwComponentType` needs to keep the ECU alive or needs to execute operations before the ECU is shut down. For this purpose the `AtomicSwComponentType` may request either the state `RUN` or `POST_RUN`.

In this case the following rules apply:

[TPS_SWCT_1013] `AtomicSwComponentType` shall keep the ECU alive (fixed variant) ⌈

`AtomicSwComponentType` needs to keep the ECU alive or needs to execute operations before the ECU is shut down.

RoleBasedPortAssignment valid roles:

- `EcuM_StateRequest` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋*(SWS_EcuM_2762)*

7.11.3.4.3 EcuM Fixed Use Case: select Shutdown Target

Scenario: a `AtomicSwComponentType` wants to select a shutdown target. This corresponds to the "select shutdown target" use case of the flex EcuM.

In this case the following rules apply:

[TPS_SWCT_1014] `AtomicSwComponentType` wants to select a shutdown target (fixed variant) ⌈

RoleBasedPortAssignment valid roles:

- `EcuM_ShutdownTarget` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋

7.11.3.4.4 EcuM Fixed Use Case: select Boot Target

Scenario: a `AtomicSwComponentType` wants to select a boot target.

In this case the following rules apply:

[TPS_SWCT_1015] `AtomicSwComponentType` wants to select a boot target (fixed variant) ⌈

RoleBasedPortAssignment valid roles:

- `EcuM_BootTarget` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋

7.11.3.4.5 EcuM Flex Use Case: select Shutdown Target

Scenario: a `AtomicSwComponentType` wants to select a shutdown target. This corresponds to the "select shutdown target" use case of the fix EcuM.

In this case the following rules apply:

[TPS_SWCT_1016] AtomicSwComponentType wants to select a shutdown target (flexible variant) ⌈

RoleBasedPortAssignment valid roles:

- EcuM_ShutdownTarget [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋

7.11.3.4.6 EcuM Flex Use Case: select Boot Target

Scenario: a AtomicSwComponentType wants to select a boot target.

In this case the following rules apply:

[TPS_SWCT_1017] AtomicSwComponentType wants to select a boot target (flexible variant) ⌈

RoleBasedPortAssignment valid roles:

- EcuM_BootTarget [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋

7.11.3.4.7 EcuM Flex Use Case: use Alarm Clock

Scenario: a AtomicSwComponentType wants to use an alarm clock.

In this case the following rules apply:

[TPS_SWCT_1018] AtomicSwComponentType wants to use an alarm clock (flexible variant) ⌈

RoleBasedPortAssignment valid roles:

- EcuM_AlarmClock [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
N/A

⌋

7.11.3.5 BswM

The BswM does not have an associated dedicated subclass of ServiceNeeds. The only use case for the existence of a SwcServiceDepedency with respect to the interaction with the BswM is the support for partial networking, in particular the association of a PortGroup and the associated PortPrototypes that act as *VFC control ports* and *VFC status ports*. For more details please refer to section 4.8.

In this case the following rules apply:

[TPS_SWCT_1126] Access to partial networking via BswM ⌈

RoleBasedPortAssignment valid roles:

- control [0 .. 1]
- status [0 .. 1]

RoleBasedDataAssignment
N/A

RepresentedPortGroup
Reference to the applicable PortGroup associated with the particular partial network.

⌋*(RS_SWCT_3201)*

The mulitplicities of the RoleBasedPortAssignments for this case have been defined under the assumption that a given software-component may or may not have a *VFC control port*. Also, it may have a *VFC status port*. Technically, there could be several *VFC status ports* per software-component but most likely there is only one *VFC status port*.

7.11.3.6 Crypto Service Dependencies

The meta-class CryptoServiceNeeds is used to define the requirements to configure the CryptoServiceManager.

An SwcInternalBehavior may provide several CryptoServiceNeeds elements where each relates to one ConfigID (see [30]) for details). In tis context it is of special importance to note which PortPrototypes belong to this ConfigID in order to be able to properly generate the callbacks.

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 7.51: CryptoServiceNeeds

Please note that for all described use cases of the Crypto Service following rule applies:
For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`. Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`. The possible role attribute values and the multiplicity of the related `PortPrototype`s are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

7.11.3.6.1 Crypto Service Service Use Case: Hash calculation

Scenario: a `AtomicSwComponentType` uses the hash calculation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2020] `AtomicSwComponentType` uses the hash calculation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmHash` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0775, SWS_CSM_0801)*

7.11.3.6.2 Crypto Service Service Use Case: MAC calculation

Scenario: a `AtomicSwComponentType` uses the message authentication code (MAC) calculation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2021] `AtomicSwComponentType` uses the message authentication code (MAC) calculation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmMacGenerate` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0776, SWS_CSM_0801)*

7.11.3.6.3 Crypto Service Service Use Case: MAC verification

Scenario: a `AtomicSwComponentType` uses the message authentication code (MAC) verification of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2022] `AtomicSwComponentType` uses the message authentication code (MAC) verification of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmMacVerify` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0777, SWS_CSM_0801)*

7.11.3.6.4 Crypto Service Service Use Case: seeding of random generator

Scenario: a `AtomicSwComponentType` uses the generation of random numbers of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2023] `AtomicSwComponentType` uses the generation of random seed of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmRandomSeed` [1]

- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0778, SWS_CSM_0801)*

7.11.3.6.5 Crypto Service Service Use Case: generation of random numbers

Scenario: a AtomicSwComponentType uses the generation of random numbers of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2024] AtomicSwComponentType uses the generation of random numbers of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmRandomGenerate [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0779, SWS_CSM_0801)*

7.11.3.6.6 Crypto Service Service Use Case: symmetrical block encryption

Scenario: a AtomicSwComponentType uses the symmetrical block encryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2025] AtomicSwComponentType uses the symmetrical block encryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmSymBlockEncrypt [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(SWS_CSM_0780, SWS_CSM_0801)

7.11.3.6.7 Crypto Service Service Use Case: symmetrical block decryption

Scenario: a AtomicSwComponentType uses the symmetrical block decryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2026] AtomicSwComponentType uses the symmetrical block decryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmSymBlockDecrypt [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(SWS_CSM_0781, SWS_CSM_0801)

7.11.3.6.8 Crypto Service Service Use Case: symmetrical encryption

Scenario: a AtomicSwComponentType uses the symmetrical encryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2027] AtomicSwComponentType uses the symmetrical encryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmSymEncrypt [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(SWS_CSM_0782, SWS_CSM_0801)

7.11.3.6.9 Crypto Service Service Use Case: symmetrical decryption

Scenario: a `AtomicSwComponentType` uses the symmetrical decryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2028] `AtomicSwComponentType` uses the symmetrical decryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmSymDecrypt` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0783, SWS_CSM_0801)*

7.11.3.6.10 Crypto Service Service Use Case: asymmetrical encryption

Scenario: a `AtomicSwComponentType` uses the asymmetrical encryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2029] `AtomicSwComponentType` uses the asymmetrical encryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmAsymEncrypt` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0784, SWS_CSM_0801)*

7.11.3.6.11 Crypto Service Service Use Case: asymmetrical decryption

Scenario: a `AtomicSwComponentType` uses the asymmetrical decryption of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2030] `AtomicSwComponentType` uses the asymmetrical decryption of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmAsymDecrypt [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0785, SWS_CSM_0801)*

7.11.3.6.12 Crypto Service Service Use Case: signature generation

Scenario: a AtomicSwComponentType uses the signature generation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2031] AtomicSwComponentType uses the signature generation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmSignatureGenerate [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0786, SWS_CSM_0801)*

7.11.3.6.13 Crypto Service Service Use Case: signature verification

Scenario: a AtomicSwComponentType uses the signature verification of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2032] AtomicSwComponentType uses the signature verification of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmSignatureVerify [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0787, SWS_CSM_0801)*

7.11.3.6.14 Crypto Service Service Use Case: checksum calculation

Scenario: a `AtomicSwComponentType` uses the checksum calculation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2033] `AtomicSwComponentType` uses the checksum calculation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmChecksum` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0788, SWS_CSM_0801)*

7.11.3.6.15 Crypto Service Service Use Case: key derivation

Scenario: a `AtomicSwComponentType` uses the key derivation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2034] `AtomicSwComponentType` uses the key derivation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmKeyDerive` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0789, SWS_CSM_0801)*

7.11.3.6.16 Crypto Service Service Use Case: symmetric key derivation

Scenario: a `AtomicSwComponentType` uses the symmetric key derivation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2035] `AtomicSwComponentType` uses the symmetric key derivation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmKeyDeriveSymKey` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0790, SWS_CSM_0801)*

7.11.3.6.17 Crypto Service Service Use Case: key exchange protocol, public value calculation

Scenario: a `AtomicSwComponentType` uses the key exchange interface for public value calculation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2036] `AtomicSwComponentType` uses the key exchange interface for public value calculation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmKeyExchangeCalcPubVal` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0791, SWS_CSM_0801)*

7.11.3.6.18 Crypto Service Service Use Case: key exchange protocol, secret value calculation

Scenario: a `AtomicSwComponentType` uses the key exchange interface for secret value calculation of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2037] `AtomicSwComponentType` uses the key exchange interface for secret value calculation of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmKeyExchangeCalcSecret` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0792, SWS_CSM_0801)*

7.11.3.6.19 Crypto Service Service Use Case: key exchange protocol, calculate symmetric key

Scenario: a `AtomicSwComponentType` uses the key exchange interface to calculate symmetric key with the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2038] `AtomicSwComponentType` uses the key exchange interface to calculate symmetric key with the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmKeyExchangeCalcSymKey` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0793, SWS_CSM_0801)*

7.11.3.6.20 Crypto Service Service Use Case: symmetrical key extraction

Scenario: a `AtomicSwComponentType` uses the symmetrical key extraction of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2039] `AtomicSwComponentType` uses the symmetrical key extraction of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- `CsmSymKeyExtract` [1]

- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0794, SWS_CSM_0801)*

7.11.3.6.21 Crypto Service Service Use Case: symmetrical key wrapping with symmetrical wrapping key

Scenario: a `AtomicSwComponentType` uses the symmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a symmetric key. In this case the following setup apply:

[TPS_SWCT_2040] `AtomicSwComponentType` uses the symmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a symmetric key ⌈

RoleBasedPortAssignment valid roles:

- `CsmSymKeyWrapSym` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0795, SWS_CSM_0801)*

7.11.3.6.22 Crypto Service Service Use Case: symmetrical key wrapping with asymmetrical wrapping key

Scenario: a `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a asymmetric key. In this case the following setup apply:

[TPS_SWCT_2041] `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a asymmetric key ⌈

RoleBasedPortAssignment valid roles:

- `CsmSymKeyWrapAsym` [1]

- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0796, SWS_CSM_0801)*

7.11.3.6.23 Crypto Service Service Use Case: asymmetrical public key extraction

Scenario: a AtomicSwComponentType uses the asymmetrical public key extraction of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2042] AtomicSwComponentType uses the asymmetrical public key extraction of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmAsymPublicKeyExtract [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0797, SWS_CSM_0801)*

7.11.3.6.24 Crypto Service Service Use Case: asymmetrical private key extraction

Scenario: a AtomicSwComponentType uses the asymmetrical private key extraction of the Crypto Service. In this case the following setup apply:

[TPS_SWCT_2043] AtomicSwComponentType uses the asymmetrical private key extraction of the Crypto Service ⌈

RoleBasedPortAssignment valid roles:

- CsmAsymPrivateKeyExtract [1]
- CsmCallback [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0798, SWS_CSM_0801)*

7.11.3.6.25 Crypto Service Service Use Case: asymmetrical key wrapping with symmetrical wrapping key

Scenario: a `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a symmetrical wrapping key. In this case the following setup apply:

[TPS_SWCT_2044] `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a symmetrical wrapping key ⌈

RoleBasedPortAssignment valid roles:

- `CsmAsymPrivateKeyWrapSym` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(SWS_CSM_0799, SWS_CSM_0801)*

7.11.3.6.26 Crypto Service Service Use Case: asymmetrical key wrapping with asymmetrical wrapping key

Scenario: a `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a asymmetrical wrapping key. In this case the following setup apply:

[TPS_SWCT_2045] `AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a asymmetrical wrapping key ⌈

RoleBasedPortAssignment valid roles:

- `CsmAsymPrivateKeyWrapAsym` [1]
- `CsmCallback` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋(SWS_CSM_0800, SWS_CSM_0801)*

7.11.3.7 Diagnostic Service Dependency

This chapter describes the usage of the specific diagnostic meta-classes derived from `ServiceNeeds` within an atomic software-component. An overview of common diagnostic service needs were already introduced in figure 7.31 and can be divided into four main parts: Function Inhibition Needs 7.11.3.7.1, Diagnostic Event Needs 7.11.3.7.2, Diagnostic Communication Needs 7.11.3.7.3, and needs to fulfill the OBD related requirements 7.11.3.7.4.

Please note that for the described use cases of the Diagnostic Servicees following rule applies:

[TPS_SWCT_1129] Express diagnostic capabilities ⌈ For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`. Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`.

The possible role attribute values and the multiplicity of the related `PortPrototype`s are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.
⌋*(RS_SWCT_3190)*

7.11.3.7.1 Function Inhibition Needs

The meta-class `FunctionInhibitionNeeds` is used to define requirements in order to configure the Diagnostic Event Manager Service.

An `SwcInternalBehavior` may provide several `FunctionInhibitionNeeds` elements, each defines the requirements related to one function inhibition ID (for the terms related to the AUTOSAR Function Inhibition Manager see [31]).

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 7.52: FunctionInhibitionNeeds

7.11.3.7.1.1 Function Inhibition Manager Service use Case: read function permission

[TPS_SWCT_2505] Setup for Function Inhibition Manager Service use Case: read function permission ⌈ Scenario: a AtomicSwComponentType read the function permission from FiM in order to enable or disable a functionality. In this case the following setup apply:

RoleBasedPortAssignment valid roles:

- FunctionInhibition [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(FIM090)*

7.11.3.7.2 Diagnostic Event Needs

The meta-classes DiagnosticEventManagerNeeds is used to define requirements in order to configure the Diagnostic Event Manager Service.

An SwcInternalBehavior may provide several DiagnosticEventManagerNeeds elements which defines the mappings for the general diagnostic event manager behavior (for the terms related to the AUTOSAR Diagnostic Event Manager see [32]).

Class	**DiagnosticEventManagerNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Event Manager (DEM) which are not related to a particular item.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 7.53: DiagnosticEventManagerNeeds

The meta-classes DiagnosticCapabilityElement is used to provide generic information about diagnostic capabilities. Further on this indicates that all ServiceNeeds which are inherit from DiagnosticCapabilityElement are on one hand the need to interact with AUTOSAR Service Dem or Dcm but on the other hand are indicating capabilities to provide services for the on-board diagnostics.

Class	DiagnosticCapabilityElement (abstract)			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class identifies the capability to provide generic information about diagnostic capabilities			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
audience	DiagnosticAudienceEnum	*	attr	This specifies the intended audience for the diagnostic object. Note that this is not only for the documentation but also subsequent audience specific implementation.
diagRequirement	DiagRequirementIdString	0..1	attr	This denotes the requirement identifier to which the object can be linked to. Note that with a generic tracing concept in AUTOSAR this might superseded.
securityAccessLevel	PositiveInteger	0..1	attr	This attribute denotes the level of security which is touched by the diagnostic object. The higher the level the more relevance for the security exists. This level must be mapped to the security level in the ECU.

Table 7.54: DiagnosticCapabilityElement

Enumeration	DiagnosticAudienceEnum
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	The possible values of the intended audience for a diagnostic object.
Literal	***Description***
afterSales	The object is relevant for the OEM after-sales organization.
aftermaket	The object is for free aftermarket service organizations. **Tags:** atp.Status=obsolete
aftermarket	The object is for free aftermarket service organizations.
development	The object is relevant for engineering only.
manufacturing	The object is relevant for manufacturing.
supplier	The object is relevant for the ECU-supplier aftermarket organization.

Table 7.55: DiagnosticAudienceEnum

The meta-classes `DiagnosticEventNeeds` is used to define requirements to configure the Diagnostic Event Manager Service. An `SwcInternalBehavior` may provide several `DiagnosticEventNeeds` elements where each defines all the requirements related to one diagnostic event (for the terms related to the AUTOSAR Diagnostic Event Manager see [32]).

In addition, `ObdPidServiceNeeds` and `ObdRatioServiceNeeds` are required in order to specify the needs for OBD diagnostic service calls.

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 7.56: DiagnosticEventNeeds

Enumeration	**DtcKindEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumerator defines the possible kinds of diagnostic monitors regarding the OBD relevance.
Literal	**Description**
emissionRelatedDtc	This indicates that the monitor reports a OBD relevant malfunction.
nonEmmissionRelatedDtc	This indicates that the monitor reports a non OBD relevant malfunction.

Table 7.57: DtcKindEnum

The diagEventDebounceAlgorithm attribute defines the kind of expected debouncing by the Diagnostic Event Manager or defines that the debouncing is implemented by the software component.

The class DiagEventDebounceAlgorithm inherits from Identifiable in order to allow further documentation of the debouncing algorithm as well as non formalized description or non standardized description by the means of Sdg on expected configuration of the DiagEventDebounceAlgorithm in the Diagnostic Event Manager.

[constr_1138] assignedPort and DiagEventDebounceMonitorInternal ⌈The existence of an assignedPort in combination with a DiagEventDebounceAlgorithm shall only be respected for the concrete subclass DiagEventDebounceMonitorInternal.⌋

[constr_1139] assignedPort of DiagEventDebounceMonitorInternal shall refer to an RPortPrototype ⌈Concerning the debouncing, the software-component acts as a client and thus the assignedPort defined with respect to a DiagEventDebounceMonitorInternal may only refer to an RPortPrototype. The standardized value of the role identifier of the assignedPort shall be DiagFaultDetectionCounterPort.⌋

Class	**DiagEventDebounceAlgorithm (abstract)**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This class represents the ability to specify the pre-debounce algorithm which is selected and/or required by the particular monitor. This class inherits from Identifiable in order to allow further documentation of the expected or implemented debouncing and to use the category for the identification of the expected / implemented debouncing.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 7.58: DiagEventDebounceAlgorithm

Class	**DiagEventDebounceCounterBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the counter-based pre-debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemPrdebounceAlgorithmClass to DemPreDebounceCounterBased.			
Base	ARObject,DiagEventDebounceAlgorithm,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
counterDecrementStepSize	Integer	1	attr	This value shall be taken to decrement the internal debounce counter.

Attribute	Datatype	Mul.	Kind	Note
counterFailedThreshold	Integer	1	attr	This value defines the event-specific limit that indicates the "failed" counter status.
counterIncrementStepSize	Integer	1	attr	This value shall be taken to increment the internal debounce counter.
counterJumpDown	Boolean	1	attr	This value activates or deactivates the counter jump-down behavior.
counterJumpDownValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from incrementing to decrementing.
counterJumpUp	Boolean	1	attr	This value activates or deactivates the counter jump-up behavior.
counterJumpUpValue	Integer	1	attr	This value represents the initial value of the internal debounce counter if the counting direction changes from decrementing to incrementing.
counterPassedThreshold	Integer	1	attr	This value defines the event-specific limit that indicates the "passed" counter status.

Table 7.59: DiagEventDebounceCounterBased

Class	**DiagEventDebounceTimeBased**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the time-based pre-debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to set the ECUC choice container DemPredebounceAlgorithmClass to DemPreDebounceTimeBase.			
Base	ARObject,DiagEventDebounceAlgorithm,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
timeFailedThreshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "failed" status.
timePassedThreshold	TimeValue	1	attr	This value represents the event-specific delay indicating the "passed" status.

Table 7.60: DiagEventDebounceTimeBased

Class	**DiagEventDebounceMonitorInternal**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	This meta-class represents the ability to indicate that the pre-debounce algorithm shall be used by the DEM for this diagnostic monitor. This is related to setting the ECUC choice container DemPredebounceAlgorithmClass to DemPredebounceMonitorInternal. If the FaultDetectionAlogrithm is already known to be implemented by a specific BswModuleEntry the reference bswModuleEntry points to the function specification. If the FaultDetectionCounter value is accessible at a PortPrototype this PortPrototype shall be referenced by an assignedPort.			
Base	ARObject,DiagEventDebounceAlgorithm,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.61: DiagEventDebounceMonitorInternal

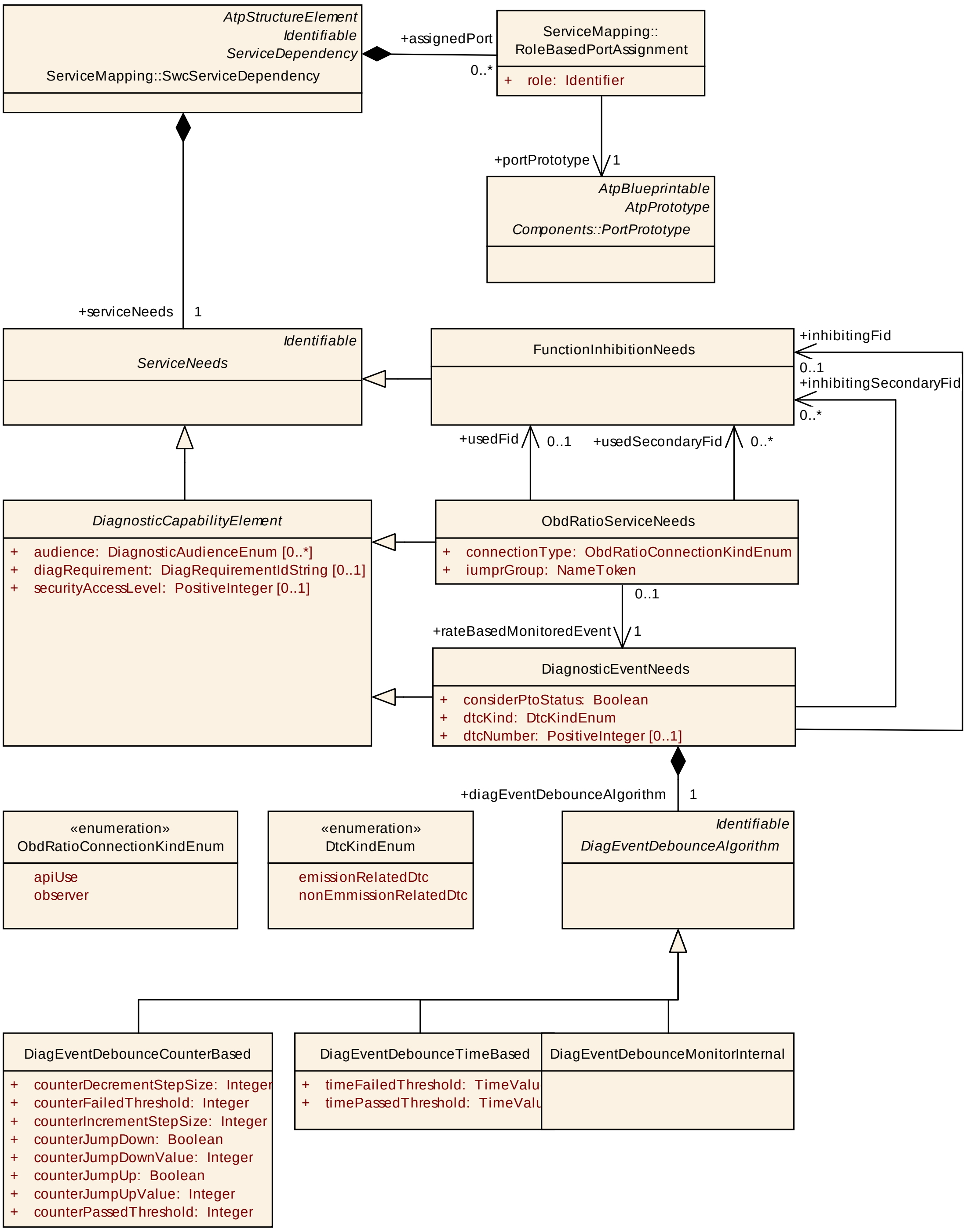

Figure 7.37: Relationship of `DiagnosticEventNeeds` and `FunctionInhibitionNeeds`

The figure 7.37 shows the relationship of the class `DiagnosticEventNeeds`. The given M2 structure support to express following properties of a diagnostic monitor in addition to the basic set of attributes provided by `DiagnosticCapabilityElement`:

With the `inhibitingFid` reference to an `FunctionInhibitionNeeds` instance on M1 it is declared that either the monitoring of a symptom or the reporting of detected faults can be inhibited by the usage of the Function Inhibition Managers.

The used PortPrototype which has to be connected to the Function Inhibition Managers is determined by the RoleBasedPortAssignment of the related FunctionInhibitionNeeds instance on M1.

The reference from a M1 instance of an ObdRatioServiceNeeds to an M1 instance of a DiagnosticEventNeeds specifies that the related Diagnostic Monitor supports Rate Based Monitoring. For further details see 7.11.3.7.4

7.11.3.7.2.1 Dem Service Use Case: diagnostic monitor, debouncing by Dem

Scenario: an AtomicSwComponentType implements a Diagnostic Monitor. The debouncing of the failure condition shall be configured and processed by the Dem. In this case the following setup apply:

[TPS_SWCT_1028] AtomicSwComponentType implements a Diagnostic Monitor
⌈

RoleBasedPortAssignment valid roles:

- DiagnosticMonitor [1]
- DiagnosticInfo [0 .. 1]
- CallbackInitMonitorForEvent [0 .. 1]
- CallbackEventStatusChange [0 .. 1]
- CallbackClearEventAllowed [0 .. 1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170)*

7.11.3.7.2.2 Dem Service Use Case: diagnostic monitor, debouncing by SWC

Scenario: an AtomicSwComponentType implements a Diagnostic Monitor. The debouncing of the failure condition shall be processed by the software component. In this case the following setup apply:

[TPS_SWCT_1029] AtomicSwComponentType implements a Diagnostic Monitor
⌈

RoleBasedPortAssignment valid roles:

- DiagnosticMonitor [1]
- DiagnosticInfo [0 .. 1]

- `CallbackInitMonitorForEvent` [0 .. 1]
- `CallbackEventStatusChange` [0 .. 1]
- `CallbackClearEventAllowed` [0 .. 1]
- `CallbackGetFaultDetectCounter` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170)*

7.11.3.7.2.3 Dem Service Use Case: restart entire Function rather than a single Diagnostic Event

Scenario: a `AtomicSwComponentType` accepts a request to restart an entire function (as opposed to restarting a single monitor/event)

[TPS_SWCT_1131] `AtomicSwComponentType` accepts a request to restart an entire function ⌈

RoleBasedPortAssignment valid roles:

- `CallbackInitMonitorForFunction` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0614)*

7.11.3.7.2.4 Dem Service Use Case: software-component provides information about operation cycles

Scenario: an `AtomicSwComponentType` provides information about operating cycles, e.g. ignition cycle or driving cycle.

[TPS_SWCT_1132] `AtomicSwComponentType` provides information about operating cycles ⌈

RoleBasedPortAssignment valid roles:

- `OperationCycle` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0601)*

7.11.3.7.2.5 Dem Service Use Case: software-component provides information about aging cycles

Scenario: an `AtomicSwComponentType` provides information about aging cycles.

[TPS_SWCT_1133] `AtomicSwComponentType` provides information about aging cycles ⌈

RoleBasedPortAssignment valid roles:

- `AgingCycle` [0 .. 1]
- `ExternalAgingCycle` [0 .. 1]

Please note that **either** `AgingCycle` **or** `ExternalAgingCycle` can be utilized for this use case (xor-relation).

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0602, SWS_DEM_0603)*

7.11.3.7.2.6 Dem Service Use Case: software-component enables storage of DTCs in general

Scenario: a `AtomicSwComponentType` enables the storage of DTCs in general.

[TPS_SWCT_1134] `AtomicSwComponentType` enables storage of DTCs in general ⌈

RoleBasedPortAssignment valid roles:

- `EnableCondition` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0604)*

7.11.3.7.2.7 Dem Service Use Case: software-component enables storage of subsequent DTCs

Scenario: an `AtomicSwComponentType` enables the storage of subsequent DTCs.

[TPS_SWCT_1135] `AtomicSwComponentType` enables storage of subsequent DTCs ⌈

RoleBasedPortAssignment valid roles:

- `StorageCondition` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0605)*

The relevant DTCs shall be configured in ECUC because at the time the `AtomicSwComponentType` is designed the information about which DTCs are relevant is not fully available.

7.11.3.7.2.8 Dem Service Use Case: retrieve information from the fault storage

Scenario: an `AtomicSwComponentType` retrieves information from the fault storage.

[TPS_SWCT_1136] `AtomicSwComponentType` retrieves information from the fault storage ⌈

RoleBasedPortAssignment valid roles:

- `IndicatorStatus` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0606)*

7.11.3.7.2.9 Dem Service Use Case: DEM provides information that the fault storage overflows

Scenario: the `Dem` provides information that the fault storage overflows.

[TPS_SWCT_1137] `Dem` provides information that the fault storage overflows ⌈

RoleBasedPortAssignment valid roles:

- `EvMemOverflowIndication` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0607)*

7.11.3.7.2.10 Dem Service Use Case: software-component suppresses the storage of DTCs

Scenario: an `AtomicSwComponentType` suppresses the storage of DTCs within the `Dem`.

[TPS_SWCT_1138] `AtomicSwComponentType` suppresses the storage of DTCs within the `Dem` ⌈

RoleBasedPortAssignment valid roles:

- `DTCSuppression` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0608)*

7.11.3.7.2.11 Dem Service Use Case: software-component informs that the PTO is active

Scenario: an `AtomicSwComponentType` informs the `Dem` that the PTO is active.

[TPS_SWCT_1139] `AtomicSwComponentType` informs the `Dem` that the PTO is active ⌈

RoleBasedPortAssignment
The following roles are applicable:

- `PowerTakeOff` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0612)*

7.11.3.7.2.12 Dem Service Use Case: software-component needs information about specific DTC without being a diagnostic monitor

Scenario: an `AtomicSwComponentType` needs information about specific DTC without being a diagnostic monitor.

[TPS_SWCT_1140] `AtomicSwComponentType` needs information about specific DTC without being a diagnostic monitor ⌈

RoleBasedPortAssignment valid roles:

- `CallbackDTCStatusChange` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0617)*

7.11.3.7.2.13 Dem Service Use Case: call operation if the data of a given diagnostic event changes (I)

Scenario: an `AtomicSwComponentType` provides a `PPortPrototype` typed by the `ClientServerInterface` `CallbackEventDataChanged`. The service component calls the `ClientServerOperation` `EventDataChanged` if the corresponding diagnostic event changes in terms of the underlying data.

For each diagnostic events to which the `AtomicSwComponentType` is conceptually connected it needs to provide one `PPortPrototype` towards the service component.

[TPS_SWCT_1425] `AtomicSwComponentType` provides one callback per event if diagnostic event data change ⌈

RoleBasedPortAssignment valid roles:

- `CallbackEventDataChanged` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0618)*

7.11.3.7.2.14 Dem Service Use Case: call operation if the data or status of any diagnostic event changes

Scenario: an `AtomicSwComponentType` shall react on any diagnostic event status change and/or any diagnostic event data change. For instance this may be used to write a time stamp when any event status changes regardless of the event id. Please note that the `Dem` only supports exactly one `AtomicSwComponentType` using `GeneralCallbackEventDataChanged` / `GeneralCallbackEventStatusChange` on a ECU. In contrast to the scenario described in chapter 7.11.3.7.2.13 or 7.11.3.7.2.12 this case foresees the existence of a single `PPortPrototype` that covers all relevant diagnostic events.

[TPS_SWCT_1426] `AtomicSwComponentType` provides callback if any diagnostic event data and/or status changed ⌈

RoleBasedPortAssignment valid roles:

- `GeneralCallbackEventDataChanged` [0..1]
- `GeneralCallbackEventStatusChange` [0..1]
- `GeneralDiagnosticInfo` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

In order to react on diagnostic event status changes the software component shall provide a single `PPortPrototype` typed as a client server interface compatible to `GeneralCallbackEventDataChanged`.

In order to react on diagnostic event data changes the software component shall provide a single `PPortPrototype` typed as a client server interface compatible to `GeneralCallbackEventDataChanged`.

If the software component additionally has to read further information of the specific diagnostic event from `Dem` it shall provide a `RPortPrototype` typed as a client server interface compatible to `GeneralDiagnosticInfo`. ⌋*(RS_SWCT_0170, SWS_DEM_0616, SWS_DEM_0619, SWS_DEM_0600)*

7.11.3.7.2.15 Dem Service Use Case: software-component provides data for diagnostic purposes

Scenario: an AtomicSwComponentType provides data to be used for diagnostic purposes. The provision of data can be done by means of PPortPrototypes typed by either ClientServerInterfaces or SenderReceiverInterfaces. The usage of the latter, however, is not further detailed in the applicable SWS [32] and therefore no more details are to be provided in this document.

[TPS_SWCT_1427] AtomicSwComponentType provides data for diagnostic purposes via ClientServerInterface ⌈

RoleBasedPortAssignment valid roles:

- CSDataServices [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0621)*

7.11.3.7.2.16 Dem Service Use Case: interface to DCM

Scenario: a ServiceSwComponentType representing the Dem provides a PPortPrototype for the Dcm. Although this scenario does not apply to ApplicationSwComponentTypes it is included for the sake of completeness.

[TPS_SWCT_1428] ServiceSwComponentType representing the Dem provides a PPortPrototype for the Dcm ⌈

RoleBasedPortAssignment valid roles:

- DcmIf [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0609)*

7.11.3.7.3 Diagnostic Communication Needs

The meta-class DiagnosticCommunicationManagerNeeds is used to define requirements in order to configure the Diagnostic Communication Manager Service.

An SwcInternalBehavior may provide a DiagnosticCommunicationManagerNeeds element which defines the mappings for the general diagnostic communication (for the terms related to the AUTOSAR Diagnostic Communication Manager see [33]).

Class	**DiagnosticCommunicationManagerNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID or DiagnosticRoutineNeeds). The main use case is the mapping of service ports to the DCM which are not related to a particular item.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
–	–	–	–	–

Table 7.62: DiagnosticCommunicationManagerNeeds

The meta-class DiagnosticRoutineNeeds is used to define requirements to configure the Diagnostic Communication Manager Service. A PPortPrototype typed by a ClientServerInterface[5] may provide ClientServerOperations (for example, “start”, “stop”, and “RequestResults”).

The PPortPrototype corresponds to the diagnostic service RoutineControl. Within the SwcInternalBehavior up to three RunnableEntitys are defined for implementing the ClientServerOperations mentioned before.

The enumeration parameter DiagnosticRoutineTypeEnum is used to define whether the diagnostic server or client is responsible for stopping the routine.

Class	**DiagnosticRoutineNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the DCM which are not related to a particular item.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
diagRoutineType	DiagnosticRoutineTypeEnum	1	attr	This denotes the type of diagnostic routine which is implemented by the referenced server port.
ridNumber	PositiveInteger	0..1	attr	This represents a routine identifier for the diagnostic routine. This allows to predefine the RID number if the a function developer has received a particular requirement from the OEM or from a standardization body.

Table 7.63: DiagnosticRoutineNeeds

[5] where isService shall be set to TRUE

Enumeration	**DiagnosticRoutineTypeEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	This enumerator specifies the different types of diagnostic routines.
Literal	***Description***
asynchronous	This indicates that the diagnostic server is not blocked while the diagnostic routine is running.
synchronous	This indicates that the diagnostic routine blocks the diagnostic server in the ECU while the routine is running.

Table 7.64: DiagnosticRoutineTypeEnum

The meta-class `DiagnosticIoControlNeeds` is used to define requirements to configure the Diagnostic Communication Manager Service. The `PPortPrototype` corresponds to the diagnostic service `InputOutputControlByIdentifier`. Within the `SwcInternalBehavior` up to three `RunnableEntity`s are defined for implementing the `ClientServerOperation`s mentioned before.

Class	**DiagnosticIoControlNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the DCM which are not related to a particular item.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
didNumber	PositiveInteger	0..1	attr	This represents a Data identifier for the diagnostic value. This allows to predefine the DID number if the a function developer has received a particular requirement from the OEM or from a standardization body.
freezeCurrentStateSupported	Boolean	1	attr	This attribute determines, if the referenced port supports temporary freezing of I/O value.
shortTermAdjustmentSupported	Boolean	1	attr	This attribute determines, if the referenced port supports temporarily setting of I/O value to a specific value provided by the diagnostic tester.

Table 7.65: DiagnosticIoControlNeeds

The meta-class `DiagnosticValueNeeds` is used to define requirements in order to configure the Diagnostic Communication Manager Service as well as the Diagnostic Event Manager Service.

The DCM can access either local values via a `ClientServerInterface` or it may access `dataElement`s in a `PPortPrototype` typed by a `SenderReceiverInterface`. For this purpose the `DiagnosticValueNeeds` require associations to local values (i.e. inside `InternalBehavior`) or respectively `dataElement`s.

The enumeration parameter `DiagnosticValueAccessEnum` distinguish between current values to read diagnostic information (readOnly) and data elements which are additionally classified as configurable (readWrite).

Class	**DiagnosticValueNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the general needs on the configuration of the Diagnostic Communication Manager (DCM) which are not related to a particular item (e.g. a PID). The main use case is the mapping of service ports to the DCM which are not related to a particular item. In the case of using a sender receiver communicated value, the related value shall be taken via assignedData in the role "signalBasedDiagnostics". In case of using a client/server communicated value, the related value shall be communicated via the port referenced by asssignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the related software specifications (SWS).			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
DiagnosticValueAccess	DiagnosticValueAccessEnum	1	attr	This attribute controls whether the data can be read and written or whether it is to be handled read-only.
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeed is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this particular PID signal.
didNumber	PositiveInteger	0..1	attr	This represents a Data identifier for the diagnostic value. This allows to predefine the DID number if the a function developer has received a particular requirement from the OEM or from a standardization body.

Table 7.66: DiagnosticValueNeeds

Enumeration	**DiagnosticValueAccessEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Defines the access of the configured diagnostic current values which will be used by the DEM or DCM module.
Literal	**Description**
readOnly	The access to the data element is limited to read-only. This is typically used to read-out diagnostic information (e.g. current values).
readWrite	The value of the diagnostic data element is classified as configurable (read and write access is possible).

Table 7.67: DiagnosticValueAccessEnum

7.11.3.7.3.1 Dcm Service Use Case: read/write current values by Client Server Interface

Scenario: an `AtomicSwComponentType` offers a server port to read/write current value via diagnostic services (e.g. measurements, variant coding)

[TPS_SWCT_2002] `AtomicSwComponentType` offers a server port to read/write current value via diagnostic services ⌈

RoleBasedPortAssignment valid roles:

- `DataServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0686)*

7.11.3.7.3.2 Dcm Service Use Case: read/write current values by Sender Receiver Interface

Scenario: an `AtomicSwComponentType` offers sender receiver ports to read/write current values via diagnostic services (e.g. measurements, variant coding) This is mainly used for data which are available at ports anyhow used for other communication purpose.

[TPS_SWCT_2003] `AtomicSwComponentType` offers sender receiver ports to read/write current values via diagnostic services ⌈

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- `signalBasedDiagnostics` [1..2]

RepresentedPortGroups
N/A

To read the signal the `AtomicSwComponentType` shall offer a provide port, to write the signal the `AtomicSwComponentType` shall offer a require port. ⌋*(RS_SWCT_0170, SWS_DCM_0687)*

7.11.3.7.3.3 Dcm Service Use Case: start/stop or request routine results

Scenario: an `AtomicSwComponentType` offers a server ports to start/stop or request routine results of diagnostic routines.

[TPS_SWCT_2004] `AtomicSwComponentType` offers a server port to start/stop or request routine results of diagnostic routines ⌈

RoleBasedPortAssignment valid roles:

- `RoutineServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0690)*

7.11.3.7.3.4 Dcm Service Use Case: IO control by Client Server Interface

Scenario: an `AtomicSwComponentType` offers a server port to adjust the IO signal via diagnostic services.

[TPS_SWCT_2005] `AtomicSwComponentType` offers a server ports to adjust the IO signal via diagnostic services ⌈

RoleBasedPortAssignment valid roles:

- `DataServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0686)*

7.11.3.7.3.5 Dcm Service Use Case: IO control by Sender Receiver Interface

Scenario: an `AtomicSwComponentType` offers sender receiver ports to adjust the IO signal via diagnostic services.

[TPS_SWCT_2006] `AtomicSwComponentType` offers sender receiver ports to adjust the IO signal via diagnostic services ⌈

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment valid roles:

- `signalBasedDiagnostics` [1..2]

RepresentedPortGroups
N/A

To read the signal the `AtomicSwComponentType` shall offer a provide port, to write the signal the `AtomicSwComponentType` shall offer a require port. ⌋*(RS_SWCT_0170, SWS_DCM_0687)*

7.11.3.7.4 OBD related Needs

The `ObdRatioServiceNeeds` describes further properties of the implementation of the Rate Based Monitoring (e.g. `connectionType`) as well as the logical dependencies relevant for the ECU configuration (e.g. `iumprGroup`)

Class	**ObdRatioServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular "ratio monitoring" which is supported by this component or module.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
connection Type	ObdRatioConne ctionKindEnum	1	attr	Defines how the DEM is connected to the component or module to perform the IUMPR service.
iumprGrou p	NameToken	1	attr	Defines the IUMPR Group of the SAE standard. Note that possible values are not predefined by an enumeration meta-type in order to make the meta-model independent of the details of the SAE standard. Possible values are currently (AUTOSAR R3.1): CAT1 CAT2 OXS1 OXS2 EGR SAIR EVAP SECOXS1 SECOXS2 NMHCCAT NOXCAT NOXADSORB PMFILTER EGSENSOR BOOSTPRS NOGROUP NONE.
rateBased Monitored Event	DiagnosticEvent Needs	1	ref	The rate based monitored Diagnostic Event.
usedFid	FunctionInhibitio nNeeds	0..1	ref	This represents the primary Function Inhibition Identifier used for the rate based monitor. This is an optional attribute.
usedSecon daryFid	FunctionInhibitio nNeeds	*	ref	This represents the secondary Function Inhibition Identifier used for the rate based monitor. This is an optional attribute. The "primary" and all "secondary" FID inhibitions are combined by "OR".

Table 7.68: ObdRatioServiceNeeds

Enumeration	**ObdRatioConnectionKindEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Defines the way how the IUMPR service connection between the DEM and the client component or module is handled (for details see the DEM Specification).
Literal	*Description*
apiUse	The IUMPR service (of the DEM) uses an explicit API to connect to the component or module.
observer	The IUMPR service (of the DEM) uses no API but "observes" the associated diagnostic event.

Table 7.69: ObdRatioConnectionKindEnum

In addition, `ObdPidServiceNeeds`, `ObdInfoServiceNeeds`, `ObdMonitorServiceNeeds` and `ObdControlServiceNeeds` are required in order to specify the specific needs for OBD diagnostic service calls. Note that `ObdPidServiceNeeds` is used for the Diagnostic Event Manager as well.

Class	**ObdControlServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Service 08 (request control of on-board system) in relation to a particular test-Identifier (TID) supported by this component or module.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
testId	PositiveInteger	1	attr	Test Identifier (TID) according to ISO 15031-5.

Table 7.70: ObdControlServiceNeeds

Class	**ObdPidServiceNeeds**			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular PID (parameter identifier) which is supported by this component or module. In case of using a client/server communicated value, the related value shall be communicated via the port referenced by asssignedPort. The details of this communication (e.g. appropriate naming conventions) are specified in the relate software specifications (SWS).			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeed is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this particular PID signal.

Attribute	Datatype	Mul.	Kind	Note
parameterId	PositiveInteger	1	attr	Standardized parameter identifier (PID) according to the OBD standard specified in attribute "standard".
standard	String	1	attr	Annotates the standard according to which the PID is given, e.g. "ISO15031-5" or "SAE J1979 Rev May 2007".

Table 7.71: ObdPidServiceNeeds

Class	ObdInfoServiceNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a given InfoType (OBD Service 09) which is supported by this component or module.			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
dataLength	PositiveInteger	0..1	attr	This attribute is applicable only if the ServiceNeed is aggregated within BswModuleDependency. This attribute represents the length of data (in bytes) provided for this InfoType.
infoType	PositiveInteger	1	attr	The InfoType according to ISO 15031-5

Table 7.72: ObdInfoServiceNeeds

Class	ObdMonitorServiceNeeds			
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds			
Note	Specifies the abstract needs of a component or module on the configuration of OBD Services in relation to a particular on-board monitoring test supported by this component or module. (OBD Service 06).			
Base	ARObject,DiagnosticCapabilityElement,Identifiable,Multilanguage Referrable,Referrable,ServiceNeeds			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
onBoardMonitorId	PositiveInteger	1	attr	On-board monitor ID according to ISO 15031-5.
testId	PositiveInteger	1	attr	Test Identifier (TID) according to ISO 15031-5.
unitAndScalingId	PositiveInteger	1	attr	Unit and scaling ID according to ISO 15031-5.

Table 7.73: ObdMonitorServiceNeeds

7.11.3.7.4.1 Dem Service Use Case: In-Use-Monitor Performance Ratio calculation

Scenario: an `AtomicSwComponentType` implements a OBD system monitor with In-Use-Monitor Performance Ratio (IUMPR) and offers client ports to provide the capability to define the number of times a fault could have been found.

[TPS_SWCT_2007] `AtomicSwComponentType` implements a OBD system monitor with In-Use-Monitor Performance Ratio ⌈

RoleBasedPortAssignment valid roles:

- `IUMPRNumerator` [0..1]
- `IUMPRDenominator` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DEM_0610, SWS_DEM_0611)*

[constr_2053] Consistency between `role IUMPRNumerator`and `ConnectionType` ⌈ If a `SwcServiceDependency` with a `ObdRatioServiceNeeds` is defined and the attribute `ConnectionType` of the contained `ObdRatioServiceNeeds` is set to `apiUse` a `RoleBasedPortAssignment` with the `role IUMPRNumerator` shall be defined. If the the attribute `ConnectionType` of the contained `ObdRatioServiceNeeds` is set to `observer` the `role IUMPRNumerator` is not applicable. ⌋

7.11.3.7.4.2 Dcm Service Use Case: read parameter identifier via diagnostic services by Client Server Interface

Scenario: an `AtomicSwComponentType` offers a server ports to read/write current value via OBD services.

[TPS_SWCT_2008] `AtomicSwComponentType` offers a server port to read/write current value via OBD services ⌈

RoleBasedPortAssignment
The following roles are applicable:

- `DataServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0686)*

7.11.3.7.4.3 Dcm Service Use Case: read parameter identifier via diagnostic services by Sender Receiver Interface

Scenario: an `AtomicSwComponentType` offers sender receiver ports to read/write current values via OBD services.

[TPS_SWCT_2009] `AtomicSwComponentType` offers sender receiver ports to read/write current values via OBD services ⌈

RoleBasedPortAssignment
N/A

RoleBasedDataAssignment
The following roles are applicable:

- `signalBasedDiagnostics` [1..2]

RepresentedPortGroups
N/A

To read the signal the `AtomicSwComponentType` shall offer a provide port, to write the signal the `AtomicSwComponentType` shall offer a require port. ⌋*(RS_SWCT_0170, SWS_DCM_0687)*

7.11.3.7.4.4 Dcm Service Use Case: Request vehicle information

Scenario: an `AtomicSwComponentType` offers a server port to read vehicle information values via OBD services.

[TPS_SWCT_2010] `AtomicSwComponentType` offers a server port to read vehicle information values via OBD services ⌈

RoleBasedPortAssignment valid roles:

- `InfotypeServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0688)*

7.11.3.7.4.5 Dcm Service Use Case: Read DTR data from SW-C for OBD Service $06

Scenario: an `AtomicSwComponentType` offers a server ports to read DTR value via OBD services.

[TPS_SWCT_2011] `AtomicSwComponentType` offers a server port to read DTR value via OBD services ⌈

RoleBasedPortAssignment valid roles:

- `DTRServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0689)*

7.11.3.7.4.6 Dcm Service Use Case: request control of on-board system, test or component

Scenario: an `AtomicSwComponentType` offers a server port for request control of on-board system, test or component via OBD services.

[TPS_SWCT_2012] `AtomicSwComponentType` offers a server port for request control of on-board system, test or component via OBD services ⌈

RoleBasedPortAssignment
The following roles are applicable:

- `RequestControlServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0691)*

7.11.3.7.4.7 Dcm Service Use Case: Access to protocol, session and security information

Scenario: an `AtomicSwComponentType` offers a server port to get protocol, session and security information or to request a Reset to Default Session.

[TPS_SWCT_2013] `AtomicSwComponentType` offers a server port to get protocol, session and security information or to request a Reset to Default Session ⌈

RoleBasedPortAssignment valid roles:

- `DCMServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0698)*

7.11.3.7.4.8 Dcm Service Use Case: Response On Event via diagnostic services

Scenario: an `AtomicSwComponentType` offers client server ports to support Response On Event (ROE) via diagnostic services.

[TPS_SWCT_2014] `AtomicSwComponentType` supports Response On Event (ROE) via diagnostic services ⌈

RoleBasedPortAssignment valid roles:

- `ROEServices` [1]
- `DCM_Roe` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

The role `ROEServices` is applicable for a server port of the `AtomicSwComponentType` and the role `DCM_Roe` is applicable for a client port of the `AtomicSwComponentType`s ⌋*(RS_SWCT_0170, SWS_DCM_0695, SWS_DCM_0699)*

7.11.3.7.4.9 Dcm Service Use Case: Verify the access to security level

Scenario: an `AtomicSwComponentType` provides a server port to verify the access to security level via diagnostic services.

[TPS_SWCT_2015] `AtomicSwComponentType` verifies the access to security level via diagnostic services ⌈

RoleBasedPortAssignment valid roles:

- `SecurityAccess` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0685)*

7.11.3.7.4.10 Dcm Service Use Case: get status of the protocol communication and disallow protocols

Scenario: an `AtomicSwComponentType` provides a server port to get information on the status of the protocol communication. Further on the `AtomicSwComponentType` may disallow a protocol.

[TPS_SWCT_2016] `AtomicSwComponentType` requires information on the status of the protocol communication and may disallow a protocol ⌈

RoleBasedPortAssignment valid roles:

- `CallbackDCMRequestServices` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0692)*

7.11.3.7.4.11 Dcm Service Use Case: Service Request Notification

Scenario: an `AtomicSwComponentType` provides a server port to get notified about a Service Request via diagnostic services. This indicates the successful reception of a new request to application. Within this Service Request Notification this function application can examine the permission of the diagnostic service / environment.

[TPS_SWCT_2017] `AtomicSwComponentType` requires the notification about a Service Request via diagnostic services ⌈

RoleBasedPortAssignment valid roles:

- `ServiceRequestNotification` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(RS_SWCT_0170, SWS_DCM_0694)*

7.11.3.8 Diagnostic Log and Trace Dependency

The meta-class `DltUserNeeds` is used together with the `SwcServiceDependency` to define requirements in order to configure the Diagnostic Log and Trace module (for the terms related to the AUTOSAR Specification of Module DLT see [34]).

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 7.74: DltUserNeeds

Please note that for the described use case of the Dlt Service the following rule applies: For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`. Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`. The possible role attribute values and the multiplicity of the related `PortPrototype`s are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

7.11.3.8.1 Dlt use Case: Application software component accesses the Synchronized Time-Base Manager

Scenario: `AtomicSwComponentType` sends log messages. In this case the following setup applies:

[TPS_SWCT_2506] Setup for Dlt use Case: Application software component accesses the Synchronized Time-Base Manager ⌈

RoleBasedPortAssignment valid roles:

- `DLTService` [1]
- `LogTraceSessionControl` [1]
- `VerboseModeControl` [0..1]
- `InjectionCallback` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

⌋*(Dlt495, Dlt496, Dlt497, Dlt498)*

In this case the software-component has to provide one Client Port (DLTService) in order to register the context and to send log or trace messages. Further on the component has to provide a Server Port (LogTraceSessionControl) to receive the current log level and trace status. Server Ports for `VerboseModeControl` and `InjectionCallback` are optional.

7.11.3.9 Synchronized Time-Base Manager Dependency

The meta-class `SyncTimeBaseMgrUserNeeds` is used together with the `SwcServiceDependency` to define requirements in order to configure the Synchronized Time-Base Manager module (for the terms related to the AUTOSAR Specification of Module StbM see [35]).

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 7.75: SyncTimeBaseMgrUserNeeds

Please note that for the described use cases of the StbM Service following rule applies: For every used `ClientServerInterface` it is necessary to create a `RoleBasedPortAssignment`. Thereby the value of the attribute `role` of the `RoleBasedPortAssignment` has to be set to the name of the used standardized `ClientServerInterface`. The possible role attribute values and the multiplicity of the related `PortPrototype`s are listed at the use case descriptions in the paragraph **RoleBasedPortAssignment**.

7.11.3.9.1 StbM use Case: Application software component accesses the Synchronized Time-Base Manager

Scenario: an `AtomicSwComponentType` autonomously calls the Synchronized Time-Base Manager, getting knowledge about the definition of time and the state of the module. In this case the following setup applies:

RoleBasedPortAssignment valid roles:

- `StbM_TimeBaseValue` [1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

In this case the software component has to provide one Client Port (StbM_TimeBaseValue) to

access the current status of the synchronized time-base

access the current definition of time represented by the notion of ticks

access the current definition of tickDuration

7.11.3.9.2 StbM use Case: Synchronized Time-Base Manager notifies application software component

Scenario: an `AtomicSwComponentType` shall be informed by the Synchronized Time-Base Manager about state changes and/or error occurrences (e.g. the synchronisation state of a time-base has changed). In this case the following setup applies:

RoleBasedPortAssignment valid roles:

- `StbM_TimeBase_TriggerCustomer` [0..1]
- `StbM_TimeBase_StateNotification` [0..1]

RoleBasedDataAssignment
N/A

RepresentedPortGroups
N/A

In this case the software-component has to provide one Receiver Port (StbM_TimeBase_TriggerCustomer) to receive the current value of the synchronized time-base and / or one Receiver Port (StbM_TimeBase_StateNotification) to receive the current state of the synchronized time-base.

Please note that at least one of the two possible `RoleBasedPortAssignment`s shall be exist for this use case.

7.12 Variation Point Proxy

[TPS_SWCT_1370] `VariationPointProxy` ⌈ The `VariationPointProxy` represents a variation point in the software components implementation. In other words, this enables the developer of a software-component to implement variability in the software-component's implementation which is resolved either by a code generator (`bindingTime = CodeGenerationTime`) or the Preprocessor (`bindingTime = PreCompileTime`). ⌋*(RS_SWCT_3100)*

Please note that when evaluating `conditionAccess` the formula shall be replaced by the result.

[TPS_SWCT_1371] `VariationPointProxy` vs. `VariationPoint` ⌈ The difference to `VariationPoint` is that if during the binding the formula evaluates to 0 the `VariationPointProxy` remains in the model while the `VariationPoint` is removed together with it's container from the model. ⌋*(RS_SWCT_3100)*

Nevertheless, the binding of the variability is described by the means of `SwSystemconstantValueSet`s.

[TPS_SWCT_1372] `bindingTime = PreCompileTime` ⌈ In case of `bindingTime = PreCompileTime` the RTE provides macro definitions that can be used for Preprocessor directives to implement `PreCompileTime` variability in C/C++ code. ⌋*(RS_SWCT_3100)*

[TPS_SWCT_1373] RTE generator shall evaluate the `SwSystemconstDependantFormula` ⌈ It is in the scope of the RTE generator to evaluate the `SwSystemconstDependantFormula` which has a higher precedence than the standard C Preprocessor and to provide the resulting values to the software components implementation. ⌋*(RS_SWCT_3100)*

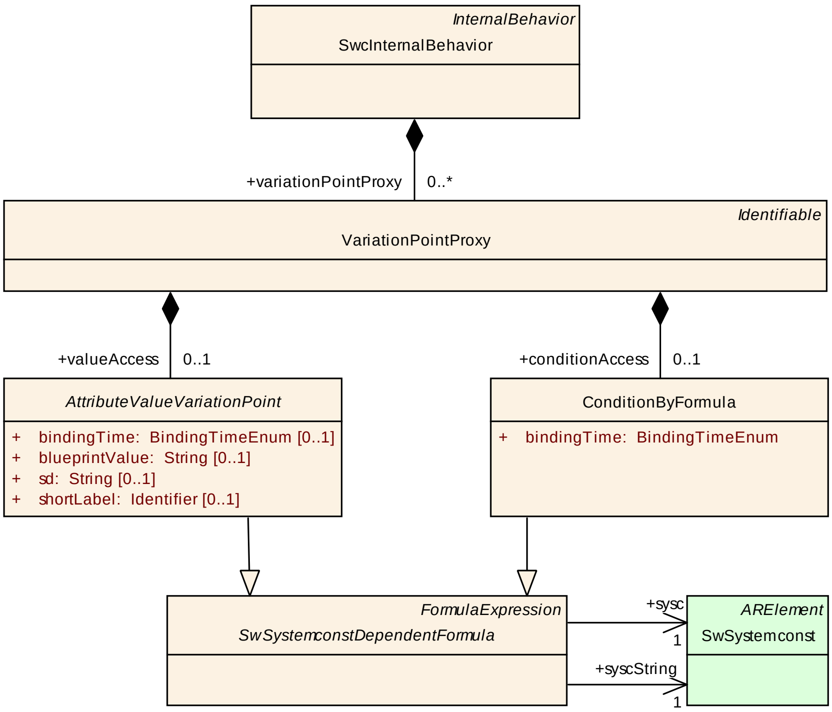

Figure 7.38: `VariationPointProxy`

Class	**VariationPointProxy**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::Variant Handling			
Note	The VariationPointProxy represents variation points of the C/C++ implementation. In case of bindingTime = compileTime the RTE provides defines which can be used for Pre Processor directives to implement compileTime variability.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
conditionAccess	ConditionByFormula	0..1	aggr	This condition acts as Binding Function for the VariationPoint.
valueAccess	AttributeValueVariationPoint	0..1	aggr	This value acts as Binding Function for the VariationPoint.

Table 7.76: VariationPointProxy

8 Implementation

Previous versions of this document contained a comprehensive description of the meta-class `Implementation`. This meta-class still exists but the description of most of its content has been moved to another document, in particular the specification of the `Basic Software Module Description Template` [7].

Please note that the `Software Component Template` and the `Basic Software Module Description Template` share the content of `Implementation`. However, the semantics of `Implementation` is closer to the `Basic Software Module Description Template`.

Nevertheless, there is still content strictly related to the `Software Component Template`. This part of `Implementation` consisting of `SwcImplementation` (see Figure 8.1) remains in this document.

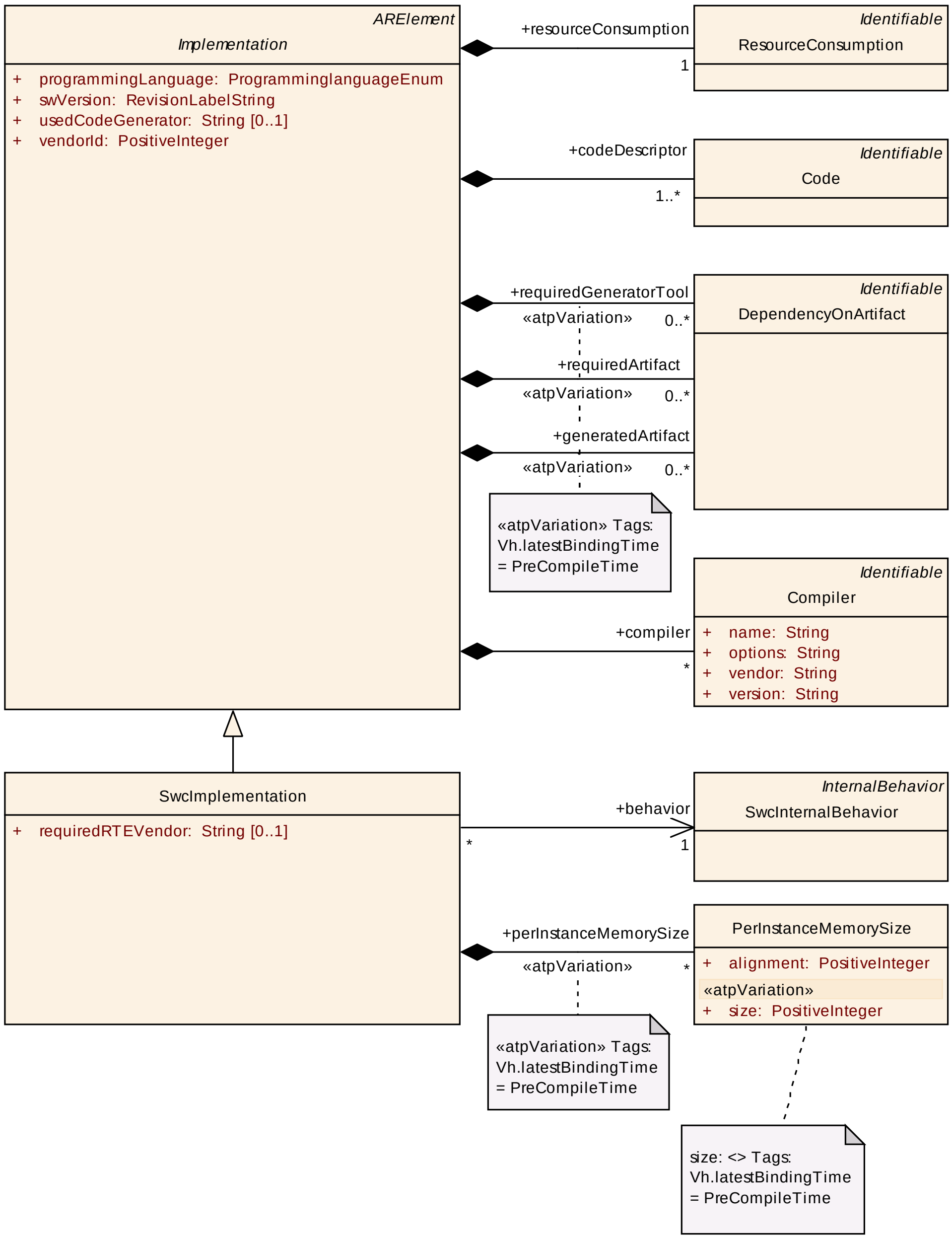

Figure 8.1: Implementation part specific to the Software Component Template

Class	**SwcImplementation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	This meta-class represents a specialization of the general Implementation meta-class with respect to the usage in application software. **Tags:** atp.recommendedPackage=SwcImplementations			
Base	ARElement,ARObject,CollectableElement,Identifiable,Implementation,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
behavior	SwcInternalBeh avior	1	ref	The internal behavior implemented by this Implementation.

Attribute	Datatype	Mul.	Kind	Note
perInstanceMemorySize	PerInstanceMemorySize	*	aggr	Allows a definition of the size of the per-instance memory for this implementation. The aggregation of PerInstanceMemorySize is subject to variability with the purpose to support variability in the software components implementations. Typically different algorithms in the implementation are requiring different number of memory objects, in this case PerInstanceMemory. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
requiredRTEVendor	String	0..1	attr	Identify a specific RTE vendor. This information is potentially important at the time of integrating (in particular: linking) the application code with the RTE. The semantics is that (if the association exists) the corresponding code has been created to fit to the vendor-mode RTE provided by this specific vendor. Attempting to integrate the code with another RTE generated in vendor mode is in general not possible.

Table 8.1: SwcImplementation

Class	**PerInstanceMemorySize**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcImplementation			
Note	Resources needed by the allocation of PerInstanceMemory for each SWC instance. Note that these resources are not covered by an ObjectFileSection, because they are supposed to be allocated by the RTE.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
alignment	PositiveInteger	1	attr	Required alignment (1,2,4,...) of the referenced PerInstanceMemory
perInstanceMemory	PerInstanceMemory	1	ref	This represents the referenced PerInstanceMemory.
size	PositiveInteger	1	attr	Size (in bytes) of the reference perInstanceMemory. The aggregation of PerInstanceMemorySize is subject to variability with the purpose to support variability in the software components implementations. Different algorithms in the implementation might require a different PerInstanceMemorySize. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 8.2: PerInstanceMemorySize

9 Mode Management

In general, the Software Component Template doesn't define the kind of modes that shall be supported by State Managers or software-components explicitly. However the Software Component Template provides generic mechanisms for describing modes.

In this section the general relationship between modes, interfaces, and software-components is discussed.

The assumption from the software-component point of view is that State Managers are using a Standardized AUTOSAR PortInterface[1] to influence the SwComponentType and also provide a PortInterface to get requests and confirmations from the SwComponentType.

They will be implemented as AUTOSAR services and be part of the Basic Software on each ECU. The actual modes a State Manager provides will have to be standardized as well to allow compatibility between software-components.

[constr_1101] Mode-related communication ⌈ Mode-related communication shall implement a 1:1 or 1:n scenario but n:1 shall be considered invalid. Formally speaking, an RPortPrototype typed by ModeSwitchInterface shall not be referenced by more than one SwConnector. ⌋

9.1 Declaration of Modes

The SW-Component Template provides some simple means to define collections of modes.

[TPS_SWCT_1071] ModeDeclaration ⌈ The name of the mode is the most important attribute that has to be provided for each ModeDeclaration. The ModeDeclarations are grouped together within the ModeDeclarationGroup. ⌋*(RS_SWCT_3200, RS_SWCT_3110)*

[TPS_SWCT_1067] Initial mode ⌈ The initialMode is active before any mode switches occurred. ⌋*(RS_SWCT_3200)*

This is shown in Figure 9.1

[1] See also AUTOSAR Glossary for "Standardized AUTOSAR Interface".

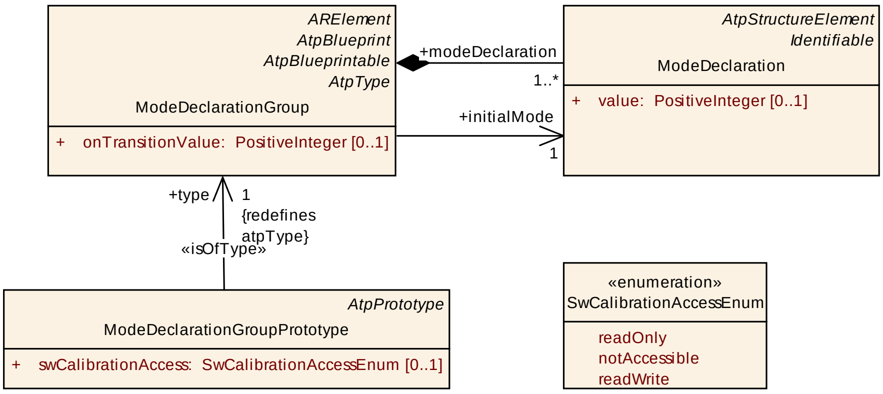

Figure 9.1: ModeDeclaration

The class ModeDeclarationGroup has been introduced to support the grouping of modes and (on M1 level) to provide predefined sets of modes that could be standardized and re-used. The set of modes eventually defines a flat (i.e. no hierarchical states) state-machine where only one mode can be active at a given point in time.

Please note that the actual definition of modes and their relationship is not in the responsibility of this document. In other words: the definition of modes represents M1 artifacts whereas this document is limited to describing M2 model elements.

Both ModeDeclaration and ModeDeclarationGroup own attributes that facilitate the generation of C source code from the formal definition.

[TPS_SWCT_1008] Definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration ⌈ The attributes ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue allow for the definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration and ModeDeclarationGroup in the source code. ⌋*(RS_SWCT_3200)*

As the attributes ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue are optional the following rule applies:

[constr_1179] Existence of ModeDeclaration.value within a ModeDeclarationGroup ⌈ Either all or no ModeDeclarations owned by a ModeDeclarationGroup shall define the value attribute. ⌋

[constr_1180] Existence of ModeDeclarationGroup.onTransitionValue ⌈ If ModeDeclarations define the value attribute the ModeDeclarationGroup shall also define the attribute onTransitionValue. ⌋

[constr_1181] Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue ⌈ The numerical values used to define the value attributes and the onTransitionValue attribute of a ModeDeclarationGroup shall not overlap. ⌋

In other words, it is not allowed that the values of two `value` attributes within one `ModeDeclarationGroup` have the same numerical value. Neither is it allowed that the numerical value of the `onTransitionValue` attribute and the numerical value of one of the corresponding `value` attributes are identical.

[TPS_SWCT_1009] The numerical values used to define the values of `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue` can be arbitrarily defined ⌈ As long as [constr_1182] is fulfilled, the numerical values used to define the values of `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue` can be arbitrarily defined. The numerical values are not required to be consecutive. Gaps are positively allowed. ⌋*(RS_SWCT_3200)*

Example: the following example of a set of numerical values fulfills all requirements on the definition of `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue`: {1,2, 5, 100}.

Please note that the ability to define `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue` introduces a second heuristics for "ordering" `ModeDeclaration`s. If `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue` are not defined the assignment of numerical values to the representations of individual `ModeDeclaration`s it is up to the RTE generator to come up with the applicable numerical values.

[TPS_SWCT_1010] `category`s for the definition of a `ModeDeclarationGroup` ⌈ In order to support a clear separation between the two possible ways to influence the definition of the programatic representation of `ModeDeclaration`s two `category`s shall be defined for the definition of a `ModeDeclarationGroup`.

- The value of `category` of a `ModeDeclarationGroup` shall be set to `EXPLICIT_ORDER` if it is intended to control the source code generation by means of the values of the attributes `ModeDeclaration.value` and `ModeDeclarationGroup.onTransitionValue`.
- The value of `category` of a `ModeDeclarationGroup` shall be set to `ALPHABETIC_ORDER` if it is intended to let the RTE generator control the the source code generation according to the alphabetical sorting.

⌋*(SWS_RTE_2568, RS_SWCT_3200)*

[TPS_SWCT_1011] Default `category` of a `ModeDeclarationGroup` ⌈ For reasons of backwards-compatibility with previous releases of AUTOSAR the default value of the `category` of a `ModeDeclarationGroup` shall be `ALPHABETIC_ORDER`. ⌋*(RS_SWCT_3200)*

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 9.1: ModeDeclaration

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,Atp Type,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
initialMode	ModeDeclaratio n	1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.
modeDecl aration	ModeDeclaratio n	1..*	aggr	The ModeDeclarations collected in this ModeDeclarationGroup.
onTransitio nValue	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 9.2: ModeDeclarationGroup

9.2 Modes and Events

[TPS_SWCT_1376] Software-components need to be capable of reacting to state changes ⌈ Software-components need to be capable of reacting to state changes issued by some `Mode Manager` and adopt their behavior to the new situation. ⌋*(RS_SWCT_3110)*

Such a mode dependent software-component is shown in Figure 9.2.

[TPS_SWCT_1077] Configure the response to mode changes ⌈ Since the behavior of `AtomicSwComponentType`s is mainly determined by the `RunnableEntity`s contained in the `SwcInternalBehavior` it is necessary to configure the response to mode changes on the level of `RunnableEntity`s. ⌋*(RS_SWCT_3120)*

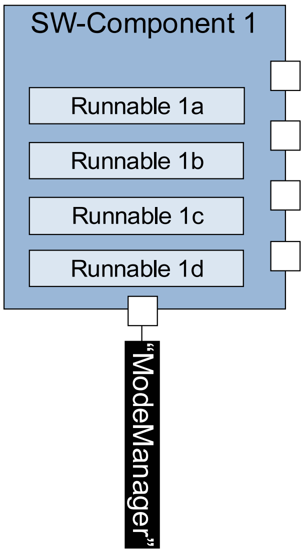

Figure 9.2: State Managers and software-components

Figure 9.3 shows an excerpt of the meta-model illustrating how the relationship between the current mode and the `SwcInternalBehavior` of the `AtomicSwComponentType` can be described.

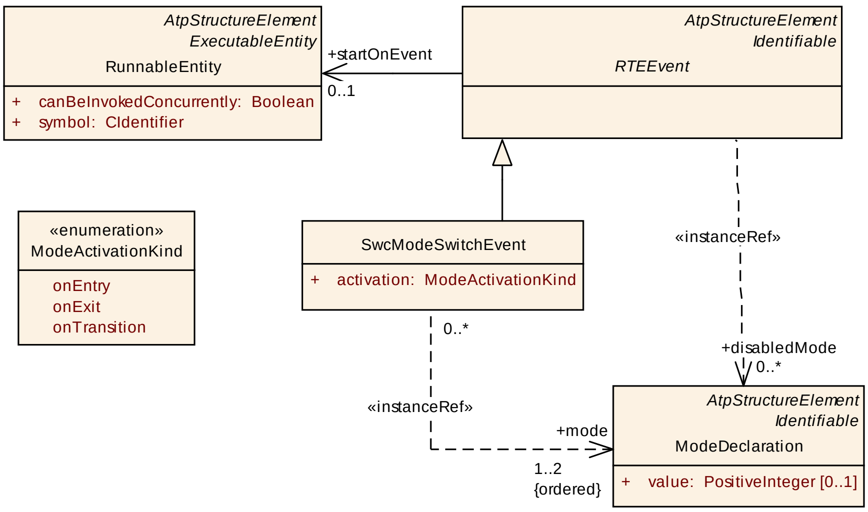

Figure 9.3: Modes and events

[TPS_SWCT_1377] Two mechanisms to define how SwcInternalBehavior should interact with the mode management ⌈ A AtomicSwComponentType can use two mechanisms to define how its SwcInternalBehavior should interact with the mode management. Both mechanisms are visible in Figure 9.3. ⌋*(RS_SWCT_3110)*

[TPS_SWCT_1378] AtomicSwComponentType can define an SwcModeSwitchEvent to execute RunnableEntity ⌈ Using the first mechanism, an AtomicSwComponentType can define an SwcModeSwitchEvent to specify that a particular RunnableEntity shall be started whenever a mode is entered, exited, or a transition between two specified modes occurs. ⌋*(RS_SWCT_3110)*

[constr_4003] Semantics of ModeSwitchEvent ⌈ If the value of SwcModeSwitchEvent.activation is onTransition then SwcModeSwitchEvent shall refer to two different ModeDeclarations belonging to the same instance of ModeDeclarationGroup.

Their order defines the direction of the transition from one mode into another. In all other cases SwcModeSwitchEvent shall refer to exactly one ModeDeclaration. ⌋

[TPS_SWCT_1379] AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode ⌈ Using the second mechanism, the AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode.

That is, RTEEvents without an association in the role disabledInMode are processed regularly according to their definition.

RTEEvents with the optional association disabledInMode have the additional limitation that the associated RunnableEntity is *not* started when the ModeDeclaration referenced as disabledInMode is active. ⌋*(RS_SWCT_3110)*

The mechanisms discussed so far have to be applied for the SwcInternalBehavior on the receiver side of mode switches. Since mode switches are received via PortPrototypes the following constraints apply:

[TPS_SWCT_1380] Mode management behavior on the sender side ⌈ On the sender side, a RunnableEntity shall have ModeSwitchPoints that eventually associate a RunnableEntity with the specific ModeDeclarationGroups which it manages, see Figure 9.4. ⌋*(RS_SWCT_3110)*

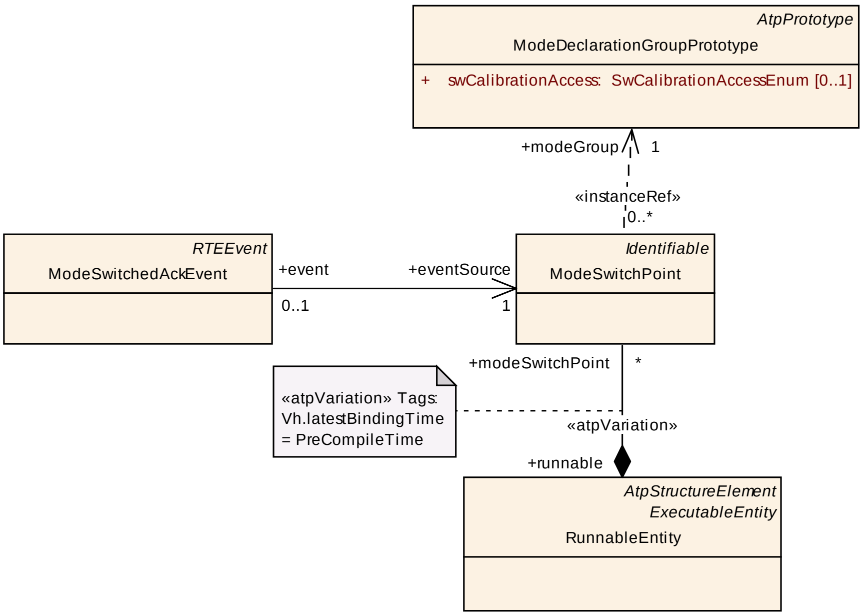

Figure 9.4: `ModeSwitchPoint`

Class	**ModeSwitchPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeSwitchPoint is required by a RunnableEntity owned a Mode Manager. Its semantics implies the ability to initiate a mode switch.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
modeGroup	ModeDeclarationGroupPrototype	1	iref	The mode declaration group that is switched by this runnable.

Table 9.3: ModeSwitchPoint

[TPS_SWCT_1383] `ModeSwitchPoint` ⌈ The `ModeSwitchPoint` also allows for the definition of a `ModeSwitchedAckEvent` if this is requested by the definition of the `PPortPrototype` (see also 4.5.3). This `RTEEvent` is eventually owned by a mode manager to allow for getting confirmation of a mode change. ⌋ *(RS_SWCT_3110)*

The definition of such an `RTEEvent` depends on the definition of other elements. The following constraints apply:

[constr_4011] `ComSpec` and `ModeSwitchedAckEvent` ⌈ If a `ModeSwitchSenderComSpec` specifies a `ModeSwitchedAckRequest` there shall be also a `ModeSwitchedAckEvent` specified whose `eventSource` references the same `ModeDeclarationGroupPrototype` as the `ModeSwitchSenderComSpec`. ⌋

[constr_4012] Timeout of `ModeSwitchedAckEvent` ⌈ The timeout value of a `WaitPoint` associated with a `ModeSwitchedAckEvent` shall be equal to the corresponding `ModeSwitchedAckRequest.timeout`. ⌋

Class	**ModeSwitchedAckRequest**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Communication			
Note	Requests acknowledgements that a mode switch has been proceeded successfully			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
timeout	TimeValue	1	attr	Number of seconds before an error is reported or in case of allowed redundancy, the value is sent again.

Table 9.4: ModeSwitchedAckRequest

Class	**ModeSwitchedAckEvent**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::RTE Events			
Note	The event is raised when the referenced modes have been received or an error occurs.			
Base	ARObject,AtpClassifier,AtpFeature,AtpStructureElement,Identifiable,Multilanguage Referrable,RTEEvent,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
eventSource	ModeSwitchPoint	1	ref	Mode switch point that triggers the event.

Table 9.5: ModeSwitchedAckEvent

[TPS_SWCT_1381] Read the currently active mode ⌈ For *Mode Manager* and *Mode User* it might additionally be required to read the currently active mode. For that purpose the a `RunnableEntity` that requires read access to the `ModeDeclarationGroupPrototype`'s current mode has to define a `ModeAccessPoint`. ⌋*(RS_SWCT_3110)*

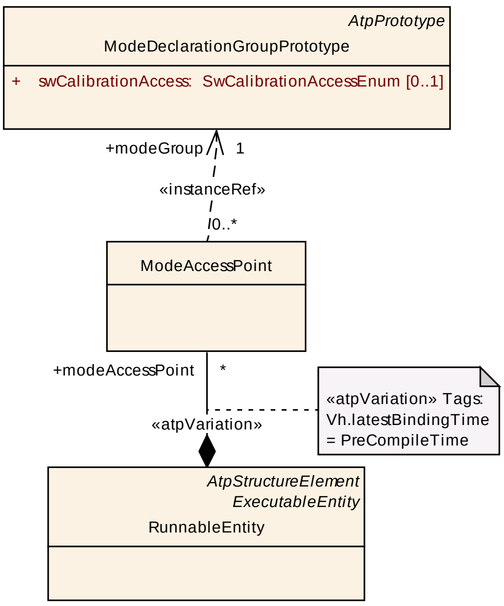

Figure 9.5: `ModeAccessPoint`

Class	**ModeAccessPoint**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SwcInternalBehavior::ModeDeclarationGroup			
Note	A ModeAccessPoint is required by a RunnableEntity owned by a Mode Manager or Mode User. Its semantics implies the ability to access the current mode (provided by the RTE) of a ModeDeclarationGroupPrototype's ModeDeclarationGroup.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
modeGroup	ModeDeclarationGroupPrototype	1	iref	The mode declaration group that is accessed by this runnable.

Table 9.6: ModeAccessPoint

[TPS_SWCT_1382] Mode switch requests are handled asynchronously by the RTE ⌈ Mode switch requests are handled asynchronously by the RTE. Therefore, *Mode Manager*s implementation might require to read back the current active mode to synchronize internally to the RTE. A ModeSwitchPoint does **not** automatically provide read access to the ModeDeclarationGroupPrototype's current mode. ⌋*(RS_SWCT_3110)*

[constr_1098] Mode switch and mode disabling ⌈ A SwcModeSwitchEvent shall not simultaneously reference to the same ModeDeclaration in both the roles mode and disabledInMode. ⌋

If [constr_1098] would not apply it might happen that a RunnableEntity would be triggered by a SwcModeSwitchEvent and on the same time it would be suppressed by the mode disabling.

9.3 Initialization / Finalization

The AUTOSAR standard shall support the execution of initialization code for every AtomicSwComponentType.

[TPS_SWCT_1384] Execution of initialization code for software-components ⌈ Most AtomicSwComponentTypes will need to initialize by executing specific code; this code shall complete before any other code in the component is executed. Data will be initializing to specific values before the "normal" application software is running. ⌋*(RS_SWCT_3110)*

The AUTOSAR standard shall also support the execution of finalization code for every AtomicSwComponentType.

[TPS_SWCT_1385] Execution of initialization code for software-components ⌈ Most AtomicSwComponentTypes will need to finalize by calling specific code; this code shall complete before the functionality of the application software shut down (e.g. a motor drive in a start or end position). ⌋*(RS_SWCT_3110)*

With the mechanisms provided by the mode manager and the activation of RunnableEntitys driven by SwcModeSwitchEvents it is easily possible to define a mode "Initialization".

[TPS_SWCT_1386] Initialization by mode management ⌈ When "Entering" this mode initialization RunnableEntitys can be activated. When all initialization RunnableEntitys have finished the mode manager can change to further modes. ⌋*(RS_SWCT_3110)*

[TPS_SWCT_1387] Finalization by mode management ⌈ Also the equivalent can be realized for the finalization of AtomicSwComponentTypes. ⌋*(RS_SWCT_3110)*

[TPS_SWCT_1388] Initial modes of AtomicSwComponentTypes are defined by the initialMode ⌈ The initial modes of AtomicSwComponentTypes are defined by the initialMode references of the required ModeDeclarationGroups. These modes are activated before any other mode activation has occurred. It is the responsibility of the RTE to activate all initial modes on a certain ECU. ⌋*(RS_SWCT_3110)*

9.4 Summary Meta-Model Excerpt Related to Modes

Figure 9.6 provides an overview of all meta-model elements that have a direct relationship to the meta-classes involved in the modelling of mode switches.

To get the complete picture, it should be noted that also the concepts of PortGroups (see 4.6) and ServiceProxySwComponentType (see 11.4) have a semantical relationship to mode management, though this is not expressed via relations in the meta-model.

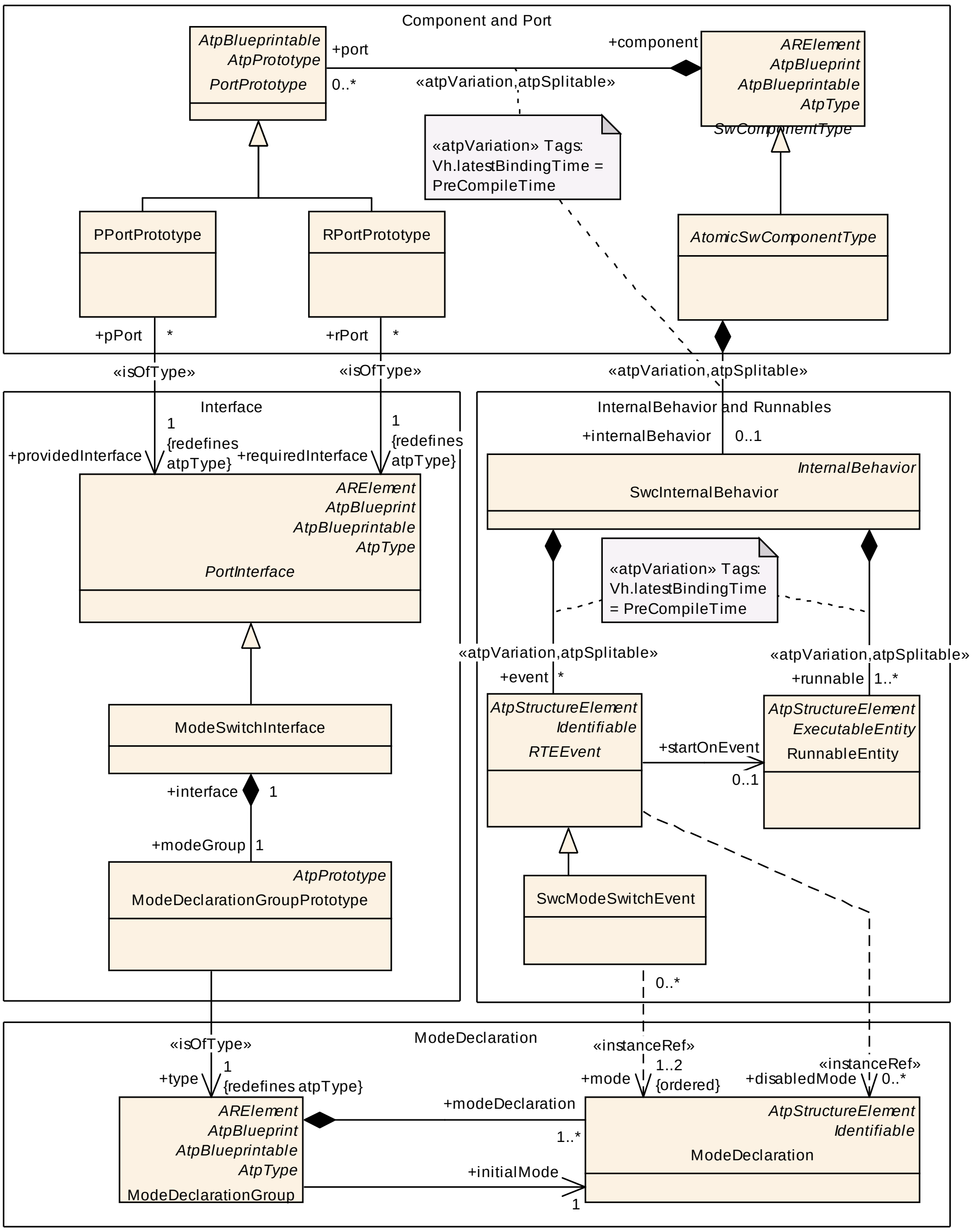

Figure 9.6: Summary meta-model excerpt related to modes

10 ECU Abstraction and Complex Drivers

10.1 Introduction

During the design of embedded systems there is one crucial point where the hardware and software have to be related. In AUTOSAR the `ECU Resource Template` describes the provided hardware resources.

On the other hand, the `Software Component Template` describes software generally without specific hardware in mind. But there are some places where both have to meet and fit.

One interface between hardware and software is discussed in the memory and execution time section of [7]. In this chapter the overall system view of the interface between sensors/actuators and software is described and the consequences for the `Software Component Template` are derived.

10.2 High Level Hardware and Software Architecture

The AUTOSAR concept defines a software architecture (see Figure 10.1) and within this layered architecture the interfaces between the hardware and the software are explicitly modeled.

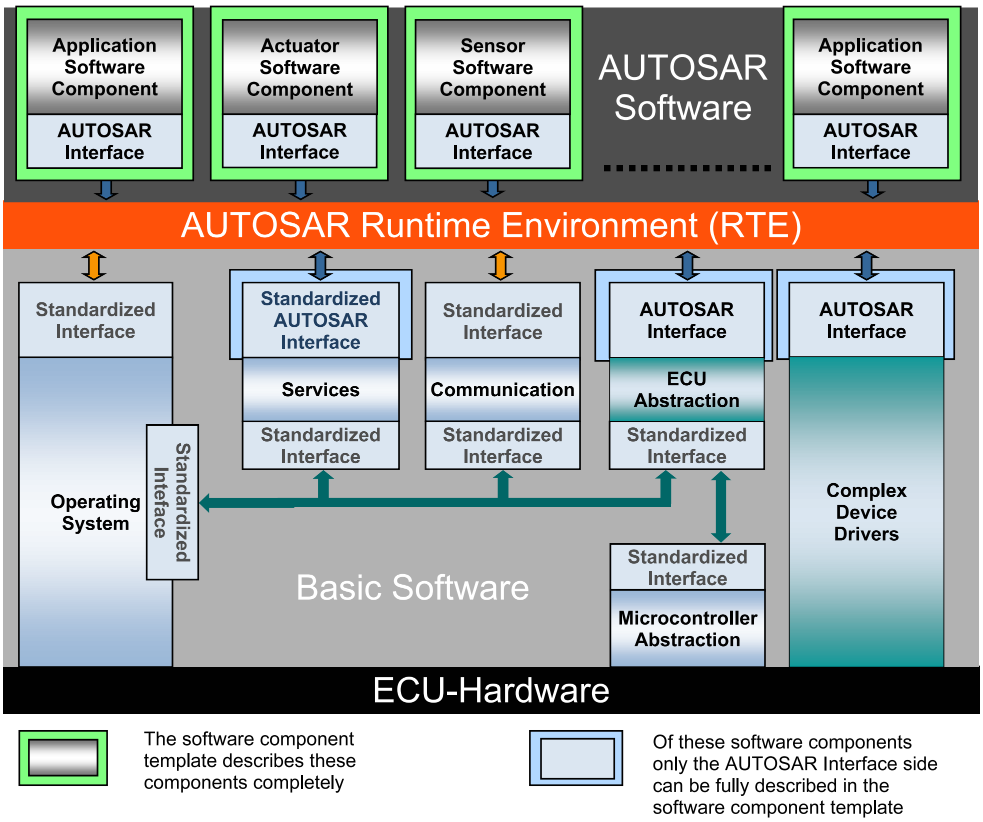

Figure 10.1: AUTOSAR ECU Software Architecture

The signal[1] flow from a hardware to software and vice versa will be described in the following sections.

A sensor[2] is converting a physical value (1) in Figure 10.2 (e.g. temperature, force, light intensity) into an electrical signal (2) which can be either a current or a voltage.

Inside the ECU generally there will be some electronics to enhance the electrical signal provided by the sensor. In AUTOSAR this is called ECU Electronics. This electronics is also responsible for the conversion of the electrical signal into a microcontroller compatible form (3), usually a voltage.

After the electrical signal has been enhanced and converted it will be captured by the microcontroller. This can either be done by a simple digital input, an analogue to digital converter or maybe a pulse-width demodulation module. Now the electrical signal is available as a software data value (4).

This signal flow is sketched in the top part of Figure 10.2.

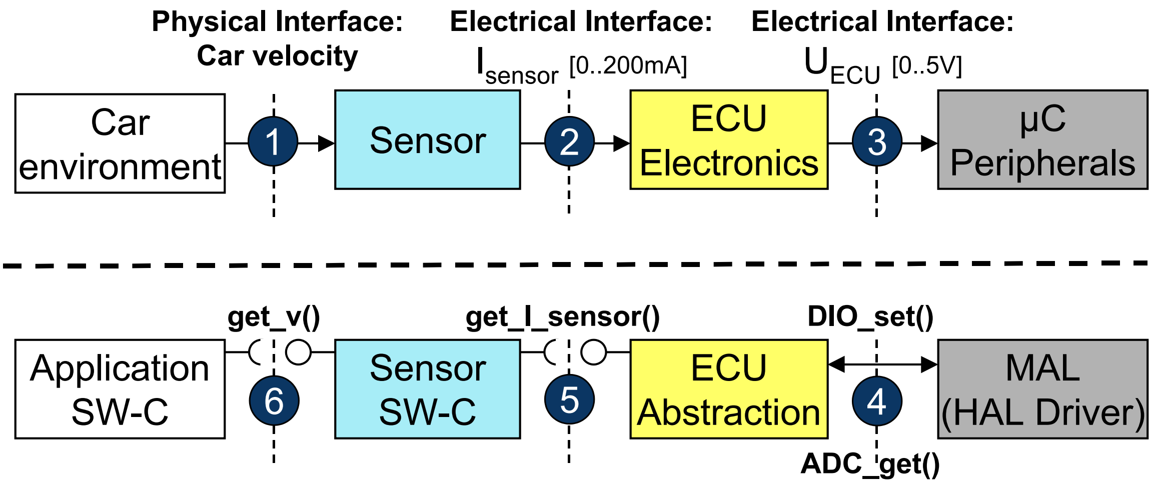

Figure 10.2: Interfaces between hardware and software

This signal chain is represented one to one in the AUTOSAR software architecture and depicted in the lower part of Figure 10.2.

In an implementation of AUTOSAR only the Microcontroller Abstraction (MCAL) has direct access to the peripheral hardware. This layer is going to be standardized and all hardware access should go through this layer. The idea of the AUTOSAR signal flow is to map the hardware to the corresponding software modules.

So if an electrical current is the input to the microcontroller peripheral, the MCAL will deliver a data value that represents this current. As the ECU Electronics has enhanced

[1]The term "signal" is not going to be used here at its own but more specific terms will be used for the different abstractions of signals at the different stages of the signal flow.

[2]For the sake of simplicity this discussion is limited to the sensor aspects. Nevertheless, the same applies also for actuators.

and converted the electrical signal prior to the microcontroller, the corresponding software entity is reversing this conversion. This is performed in the ECU Abstraction layer.

So if the input to the ECU is an electrical current and the ECU Electronics has converted this current into a voltage (from 2 to 3), the ECU Abstraction will convert the data value voltage into an AUTOSAR signal representing a current (from 4 to 5). This AUTOSAR signal represents the actual current that was provided by the sensor (2).

Now the first step in the conversion has to be reversed: the sensor has converted a physical value into an electrical signal. And so the Sensor Software Component has to reverse this again. The Sensor Software Component will read the AUTOSAR signal representing the electrical value and transform it into an AUTOSAR signal representation of the physical value (from 5 to 6).

Now this physical value is available on the RTE and can be consumed or read by other SW-Components. Although the interface between the ECU Abstraction and the Sensor Software Component is also an AUTOSAR interface and could be routed through some communication bus, it will not be practical to separate the ECU Abstraction and the corresponding `SensorActuatorSwComponentType` due to potentially high communication effort.

In Figure 10.3 a complete signal flow from a sensor input to an actuator output is shown.

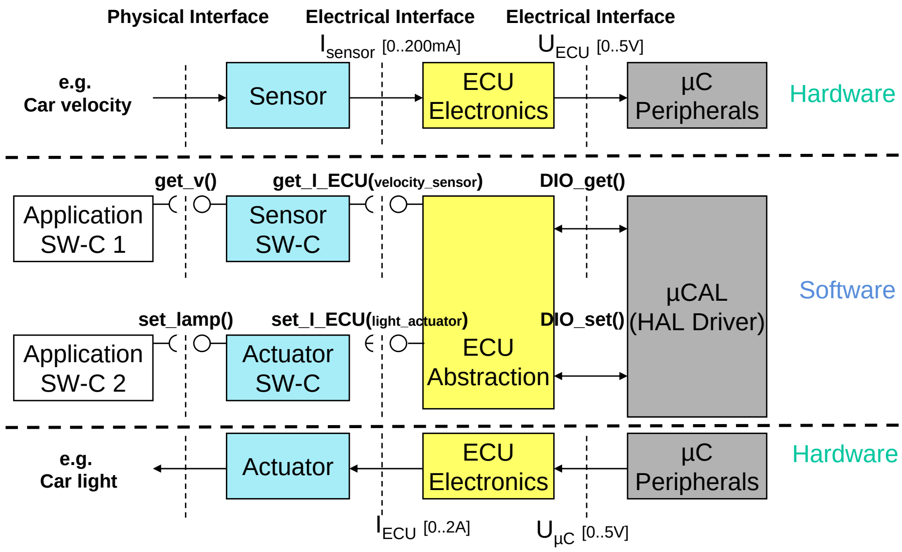

Figure 10.3: Sensor and Actuator Signal Flow

In the next section the interfaces between the involved software modules are discussed.

10.3 Interfaces and APIs

Two fundamentally different interfaces are involved when converting from sensors/actuators to software components, see markers "4" and "5" in Figure 10.2.

The interface between the Microcontroller Abstraction and the ECU Abstraction is a Standardized Interface (see AUTOSAR Glossary [36]). This interface is not visible on the Virtual Function Bus and therefore the MCAL and ECU Abstraction have to be present on the same ECU.

For further description of this interface please refer to the `ECU Resource Template` documentation.

The interface to the `SensorActuatorSwComponentType`s is visible on the `Virtual Function Bus`. In general the `SensorActuatorSwComponentType` should be on the same ECU as the ECU hardware abstraction.

Also the interface between the `SensorActuatorSwComponentType`s and the actual `AtomicSwComponentType`s representing the application is visible on the VFB. To describe the data that is going to be exchanged via this interface the standard AUTOSAR Interface description mechanisms are used (see chapter 3.4).

10.3.1 ECU Abstraction and its AUTOSAR Interfaces

Since the AUTOSAR standard is designed with the focus on the integration of software-components coming from different contractors, the interfaces between the different software-components obviously have to be compatible.

In the case of the sensors and actuators the interface is gathered in the ECU Abstraction. For each sensor and actuator there is one AUTOSAR `PortPrototype` that represents the AUTOSAR Signal that is delivered by the sensor or the AUTOSAR Signal that is consumed by the actuator. This relationship is depicted in Figure 10.4

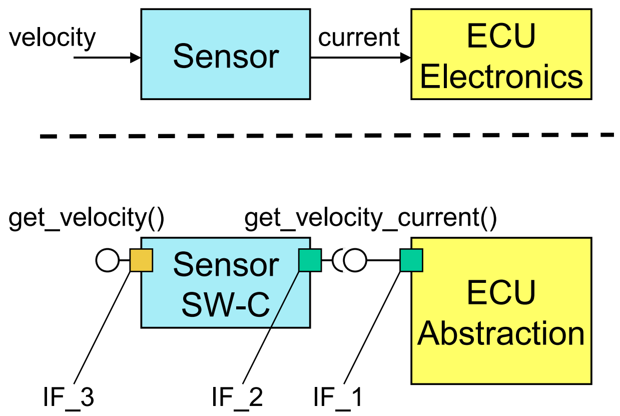

Figure 10.4: Interfaces of signals in software

Each sensor and actuator has an AUTOSAR PortPrototype at the ECU Abstraction. Connected to this port is the SensorActuatorSwComponentType. The SensorActuatorSwComponentType has one PortPrototype (i.e. IF_2) to the ECU Abstraction (which provides the values via IF_1) where it gets the AUTOSAR signals from the hardware, and one PortPrototype (i.e. IF_3) to AtomicSwComponentTypes where it provides the actual physical value to the rest of AUTOSAR on the RTE.

In addition, the Interfaces between the ECU Abstraction and the SensorActuatorSwComponentType have to be compatible like defined in chapter 6.

10.4 Sensors/Actuators

In the layered software architecture described in [6] each hardware sensor/actuator is coupled to a SensorActuatorSwComponentType (see Figure 10.5).

[TPS_SWCT_1047] Reference from the software representation of a sensor/actuator to the actual hardware element ⌈ Since the Software Component Template is going to be used to describe the SensorActuatorSwComponentType as well, there is also a reference needed from the software representation of a sensor/actuator to the actual hardware element described in the ECU Resource description. ⌋*(RS_SWCT_2080, RS_SWCT_3090)*

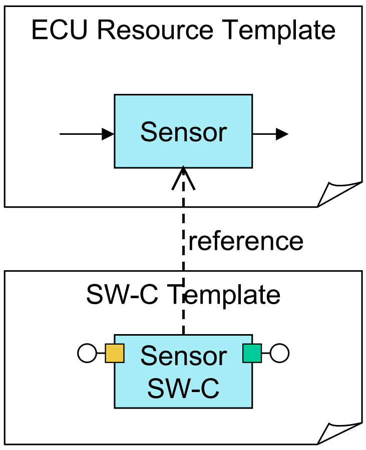

Figure 10.5: Shipment of a sensor

So each time a sensor/actuator is selected to be connected to an ECU also the corresponding SensorActuatorSwComponentType is available.

[constr_1144] SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType ⌈ The attribute sensorActuator of SensorActuatorSwComponentType, the attribute hardwareElement of EcuAbstractionSwComponentType, and the attribute hardwareElement of ComplexDeviceDriverSwComponentType may

only reference a HwType. References to other subclasses of HwDescriptionEntity are not allowed. ⌋

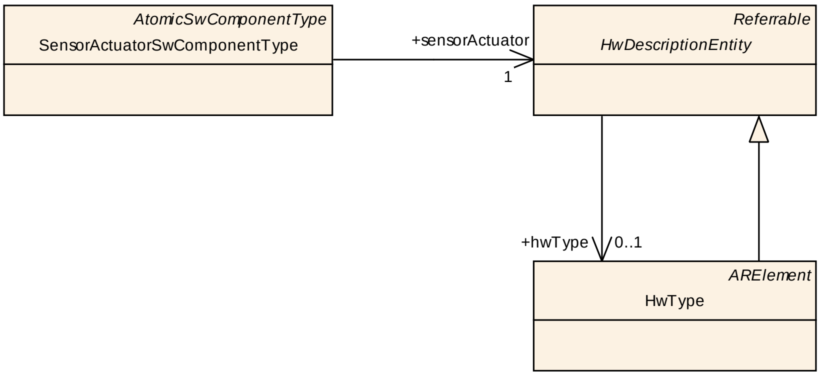

Figure 10.6: Sensor/actuator to Hardware Relationship

Figure 10.6 depicts the reference of SensorActuatorSwComponentType designed as a specialization of an AtomicSwComponentType with an additional reference to a HwType.

[constr_1109] Mapping of SwComponentPrototypes typed by a SensorActuatorSwComponentType ⌈ A SwComponentPrototype typed by a SensorActuatorSwComponentType needs to be mapped and run on exactly that ECU that contains the HwElement corresponding to the HwType that its SensorActuatorSwComponentType refers to in case it accesses the hardware via the I/O hardware abstraction layer. ⌋

[TPS_SWCT_1048] SensorActuatorSwComponentType may use the I/O hardware abstraction directly ⌈ In contrast to an ApplicationSwComponentType, an SensorActuatorSwComponentType may use the I/O hardware abstraction directly (via ports/connectors). ⌋*(RS_SWCT_2080, RS_SWCT_3090)*

In case the sensor/actuator hardware is accessed via bus communication, e.g. is located on a LIN slave, no such mapping constraints apply (note that this is not handled via the IO hardware abstraction layer).

Class	**SensorActuatorSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The SensorActuatorSwComponentType introduces the possibility to link from the software representation of a sensor/actuator to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
sensorActuator	HwDescriptionEntity	1	ref	Reference from the Sensor Actuator Software Component Type to the description of the actual hardware.

Attribute	Datatype	Mul.	Kind	Note

Table 10.1: SensorActuatorSwComponentType

10.5 I/O Hardware Abstraction

[TPS_SWCT_1389] `I/O Hardware Abstraction` interfaces MCAL drivers ⌈ The `I/O Hardware Abstraction` interfaces on one side the MCAL drivers via `Standardized Interfaces` and on the other side the Sensor Actuator Software Component via `AUTOSAR Interfaces`. On the `VFB`[3] the `I/O Hardware Abstraction` is represented by the `EcuAbstractionSwComponentType`. ⌋

[TPS_SWCT_1390] `I/O Hardware Abstraction` might have sub-structures ⌈ Depending on the complexity of an ECU, the `I/O Hardware Abstraction` might have sub-structures. In this case the `I/O Hardware Abstraction` Layer is described by several different `EcuAbstractionSwComponentType`s on M1. ⌋

Class	**EcuAbstractionSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ECUAbstraction is a special AtomicSwComponentType that resides between a software-component that wants to access ECU periphery and the Microcontroller Abstraction. The EcuAbstractionSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
hardwareElement	HwDescriptionEntity	*	ref	Reference from the EcuAbstractionComponentType to the description of the used HwElements.

Table 10.2: EcuAbstractionSwComponentType

[TPS_SWCT_1391] `I/O Hardware Abstraction` abstracts from the location of peripheral I/O devices ⌈ The `I/O Hardware Abstraction` abstracts from the location of peripheral I/O devices (on-chip or on- board) and the ECU hardware layout and has therefore dependencies to ECU Hardware described by `HWElements`. In addition, the `EcuAbstractionSwComponentType` is a hybrid concept sharing features of both software-components and basic software modules. ⌋

[TPS_SWCT_1392] Mapping between the `EcuAbstractionSwComponentType` and the corresponding `BswModuleDescription` ⌈ The `BSW` part is described by the means of the `Basic Software Module Template`. The mapping between the `EcuAbstractionSwComponentType` and the corresponding `BswModuleDescrip-`

tion is provided by the class SwcBswMapping which in addition also maps the two corresponding InternalBehaviors. This mechanism is further explained in [7]. ⌋

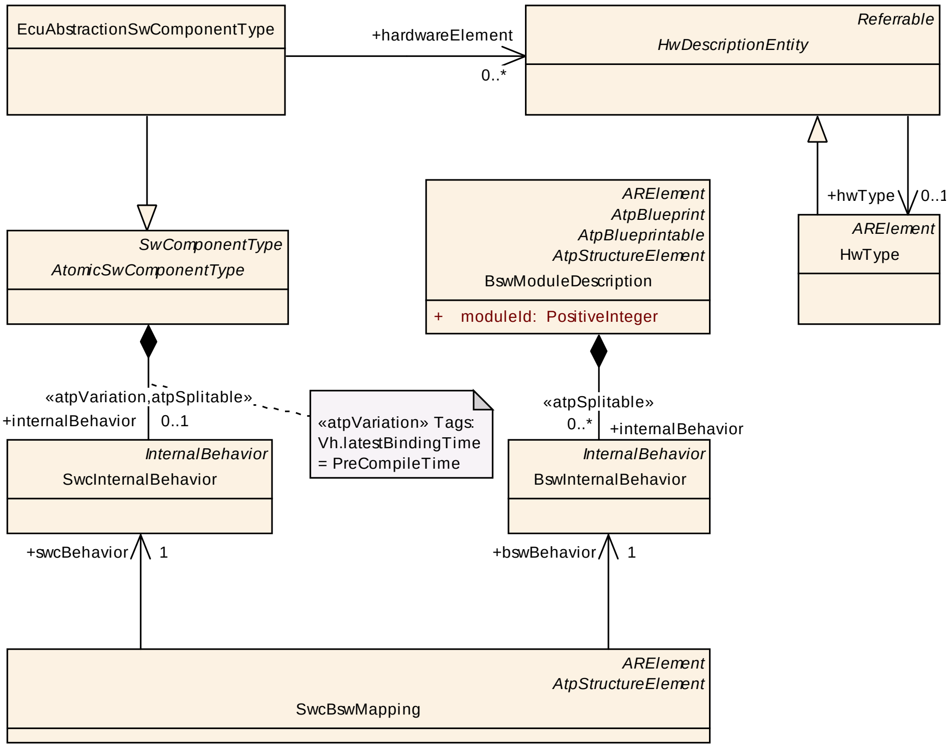

Figure 10.7: EcuAbstractionSwComponentType

10.6 Complex Driver

[TPS_SWCT_1393] Complex Driver ⌈ A Complex Driver implements complex sensor evaluation and actuator control with direct access to the Microcontroller using specific interrupts and/or complex Microcontroller peripherals to fulfill the special functional and timing requirements.

In addition it might be used to implement enhanced services / protocols or encapsulates legacy functionality of a non-AUTOSAR system. ⌋

See also document [3].

[TPS_SWCT_1394] Complex Driver is represented by the ComplexDeviceDriverSwComponentType ⌈ On the VFB the Complex Driver is represented by the ComplexDeviceDriverSwComponentType. An ECU might have zero to many different ComplexDeviceDriverSwComponentTypes. ⌋

Class	**ComplexDeviceDriverSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The ComplexDeviceDriverSwComponentType is a special AtomicSwComponentType that has direct access to hardware on an ECU and which is therefore linked to a specific ECU or specific hardware. The ComplexDeviceDriverSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,Atp Classifier,AtpType,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
hardwareElement	HwDescriptionEntity	*	ref	Reference from the ComplexDeviceDriverSwComponentType to the description of the used HwElements.

Table 10.3: ComplexDeviceDriverSwComponentType

[TPS_SWCT_1395] `ComplexDeviceDriverSwComponentType` has dependencies to ECU Hardware ⌈ Similar to `EcuAbstractionSwComponentType` the `ComplexDeviceDriverSwComponentType` has dependencies to ECU Hardware described by `HWElement`s and is a hybrid between `Software Component` and `Basic Software Module`. ⌋

[TPS_SWCT_1396] Mapping between the `ComplexDeviceDriverSwComponentType` and the corresponding `BswModuleDescription` ⌈ The `BSW` part is described by the means of the `Basic Software Module Template`. The mapping between the `ComplexDeviceDriverSwComponentType` and the corresponding `BswModuleDescription` is provided by the class `SwcBswMapping` which in addition also maps the two correponding `InternalBehavior`s. This mechnism is further explained in [7]. ⌋

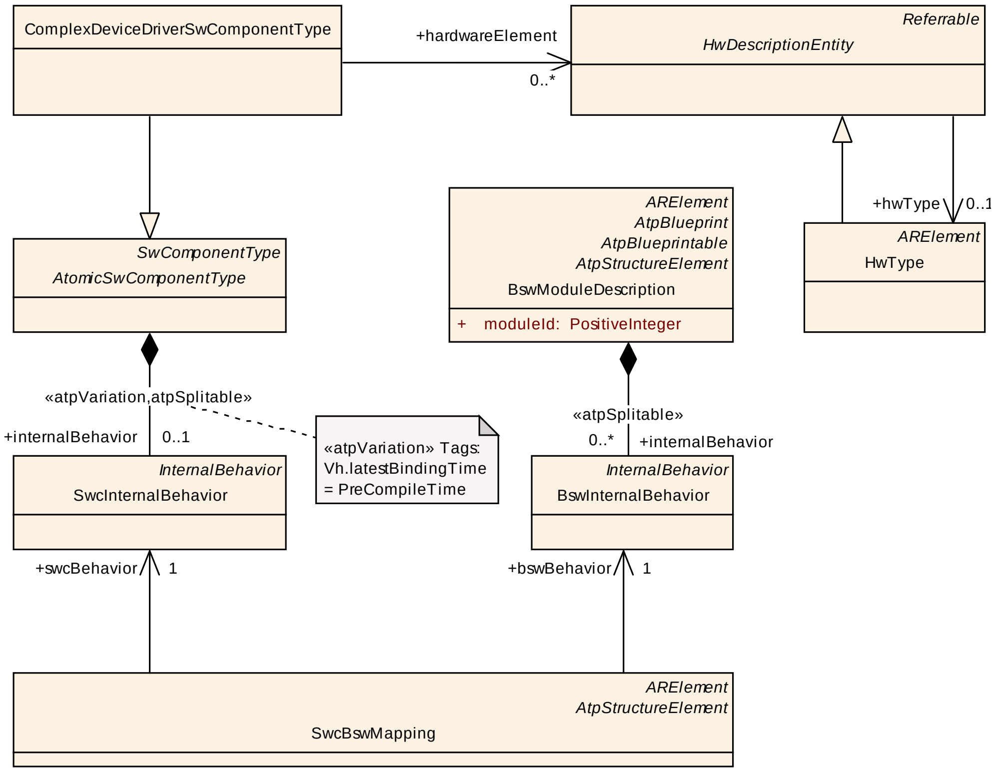

Figure 10.8: `ComplexDeviceDriverSwComponentType`

11 Services

11.1 Overview: Generation of Service-related Model Elements

This chapter covers the description and handling of AUTOSAR Service configuration.

[TPS_SWCT_1397] Hybrid concept between Basic Software Modules and a SwComponentType ⌈ AUTOSAR Services can be seen as a hybrid concept between Basic Software Modules and a SwComponentType. AUTOSAR Services actually provide access to low-level and ECU-wide "standard functionalities" commonly referred to as "service".

AtomicSwComponentTypes that require AUTOSAR Services use Standardized AUTOSAR Interfaces to communicate with these. The connection of PortPrototypes of the ServiceSwComponentTypes and PortPrototypes of the AtomicSwComponentTypes implement several communication patterns. ⌋

[TPS_SWCT_1398] Communication patterns for AUTOSAR services ⌈ The following patterns are defined and used in further chapters.

Pattern Name	Com. pattern Client:Server Sender:Receiver	Kind of PortPrototype at Service : SW-C	Description / use case
A	1:n	PPort : RPort	distribution of data or modes to n SW-Cs, e.g. used for ECU mode
A*	1:n	RPort : PPort	currently not used, not supported for client-server communication
B	1:1	PPort : RPort	SW-C acts as Server, used for so called "call-backs",
B	1:1	RPort : PPort	Service acts as Server, typical Service usage
C*	n:1	PPort : RPort	conceptually not used to support index abstraction via PortDefinedArgumentValues
C	n:1	RPort:PPort	SW-C acts as Server, used for so called "call-backs" invoked by more than one Service

Table 11.1: ServiceConnectorPattern

⌋

[TPS_SWCT_1403] Impact of AUTOSAR services on the methodology ⌈ Due to this special nature, such AUTOSAR Services need to be handled with particular attention in the methodology [4]. That is, a number of elements need to be generated during ECU integration. ⌋

The following list of paragraphs presents a short overview over the steps required for the configuration of AUTOSAR Services.

Note that most of these steps are performed by tools and the model elements being created in these steps are rather specific to Service configuration and are not to be modeled manually within AUTOSAR authoring tools.

In particular, the following requirements apply:

- **[TPS_SWCT_1399] Dependency is modeled by aggregating required and provided PortPrototypes** ⌈ The dependency of an AtomicSwComponentType (or more precisely, one of its non-abstract derived meta-classes) from an AUTOSAR Service is modeled by aggregating required and provided PortPrototypes. ⌋

  [TPS_SWCT_1400] PortInterface selected from the set of standardized Service Interfaces ⌈ The PortInterface being implemented by the PortPrototypes needs to be one of a number of standardized Service Interfaces which is indicated by having its isService attribute set to TRUE and is (via several levels of indirection) finally referenced by ServiceNeeds. ⌋

  Additionally, the software components and Basic Software Modules shall specify ServiceNeeds containing further input information for the later Service configuration step.

- **[TPS_SWCT_1401] Form a top-level RootSwCompositionPrototype** ⌈ When defining a software system, the AtomicSwComponentType is used in the form of SwComponentPrototypes within a CompositionSwComponentType. In this step, the non-service ports of all required interfaces are being connected using AssemblySwConnectors and DelegationSwConnectors in order to eventually form a top-level RootSwCompositionPrototype which can be referenced in an AUTOSAR System. ⌋

- **[TPS_SWCT_1402] Mapping of all AtomicSwComponentType instances to ECUInstances** ⌈ In System Configuration Phase, the mapping of all AtomicSwComponentType instances to ECUInstances is done (for the specification of ECUInstance see [11]). The ServiceNeeds may be used by tools to check for available resources on the targeted ECUs. ⌋

- **[TPS_SWCT_1404] Creation of the EcuExtract** ⌈ The ECU Extract is extracted from the System Configuration for each ECU. As explained in the AUTOSAR System Template [11], this contains an ECU-centric view onto the system description.

  This includes a reduced version of the system's RootSwCompositionPrototype where SwComponentPrototypes not being mapped to the ECU are being left out and all Compositions are stripped off, so that in the ECU Extract only one instance of CompositionSwComponentType remains which aggregates all SwComponentPrototypes on the ECU in a flat manner. ⌋

- **[TPS_SWCT_1405] Creation of the ServiceSwComponentTypes** ⌈ In ECU Configuration, for each Service required on the ECU exactly one ServiceSwComponentType is created based on the needs from the Atomic-

SwComponentTypes: An adequate number of PortPrototypes are created on this ServiceSwComponentType for each needed port at the AtomicSwComponentType.

Thereby the specified communication pattern A, B or C for a specific kind of ServicePort has to be considered. See also chapter 11.3 and table 11.1. ⌋

- **[TPS_SWCT_1406] Creation of SwComponentPrototype typed by a ServiceSwComponentType** ⌈ Per Service exactly one SwComponentPrototype typed by a ServiceSwComponentType is created based on the ServiceSwComponentType. Additionally, the connectors are constructed that connect the pairs of PortPrototypes belonging to the SwComponentPrototypes requiring services and those belonging to the actual services. ⌋
- **[TPS_SWCT_1407] Creation of InternalBehavior typed by a ServiceSwComponentType** ⌈ For each ServiceSwComponentType an SwcInternalBehavior is created or extended providing the information about Port Defined Argument Values, SwcRunnableEntitys and RTEEvents necessary for RTE generation. ⌋

  Further detailing of the service ports by filling in these Port Defined Argument Values is also done in ECU Configuration phase. See also chapter 7.6.3.
- **[TPS_SWCT_1408] Creation of SwcBswMapping** ⌈ For the RTE module configuration an implementation of the AUTOSAR Service described by a Basic Software Module Description needs to be selected. The SwcBswMapping to the corresponding SwComponentPrototype needs to be created accordingly.

  For each SwcInternalBehavior one SwcImplementation is being created. The information for SWCImplementation should be generated based on the available information of BswImplementation. ⌋
- **[TPS_SWCT_1409] Update of Port Defined Argument Values** ⌈ Depending of the configuration of the Service BSW it might be necessary to update the ValueSpecifications belonging to the Port Defined Argument Values generated in a previous step. ⌋

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 11.2: ServiceNeeds

11.2 Extending the ECU Software Composition

As explained in chapter 11.1, Service Configuration takes place in ECU Configuration phase. In the ECU extract of the System, the Software Components and their ECU-internal connectors are represented as a flat set aggregated by RootSwCompositionPrototype as indicated in Figure 11.1.

ECU Configuration extends this aggregation by adding SwComponentPrototypes (each typed by a specific ServiceSwComponentType) and the required AssemblySwConnectors to the RootSwCompositionPrototype. This is possible without changing the initial artifacts of the ECU extract, because these aggregations are stereotyped as ≪atpSplitable≫ in the meta-model.

After this step, the RootSwCompositionPrototype (denoted by EcucValueCollection.ecuExtract.rootSoftwareComposition) represents the whole Software Composition on the given ECU. This collection includes both the software components mapped to the ECU **and** the necessary service components represented as one SwComponentPrototype for each AUTOSAR Service utilized on the given ECU.

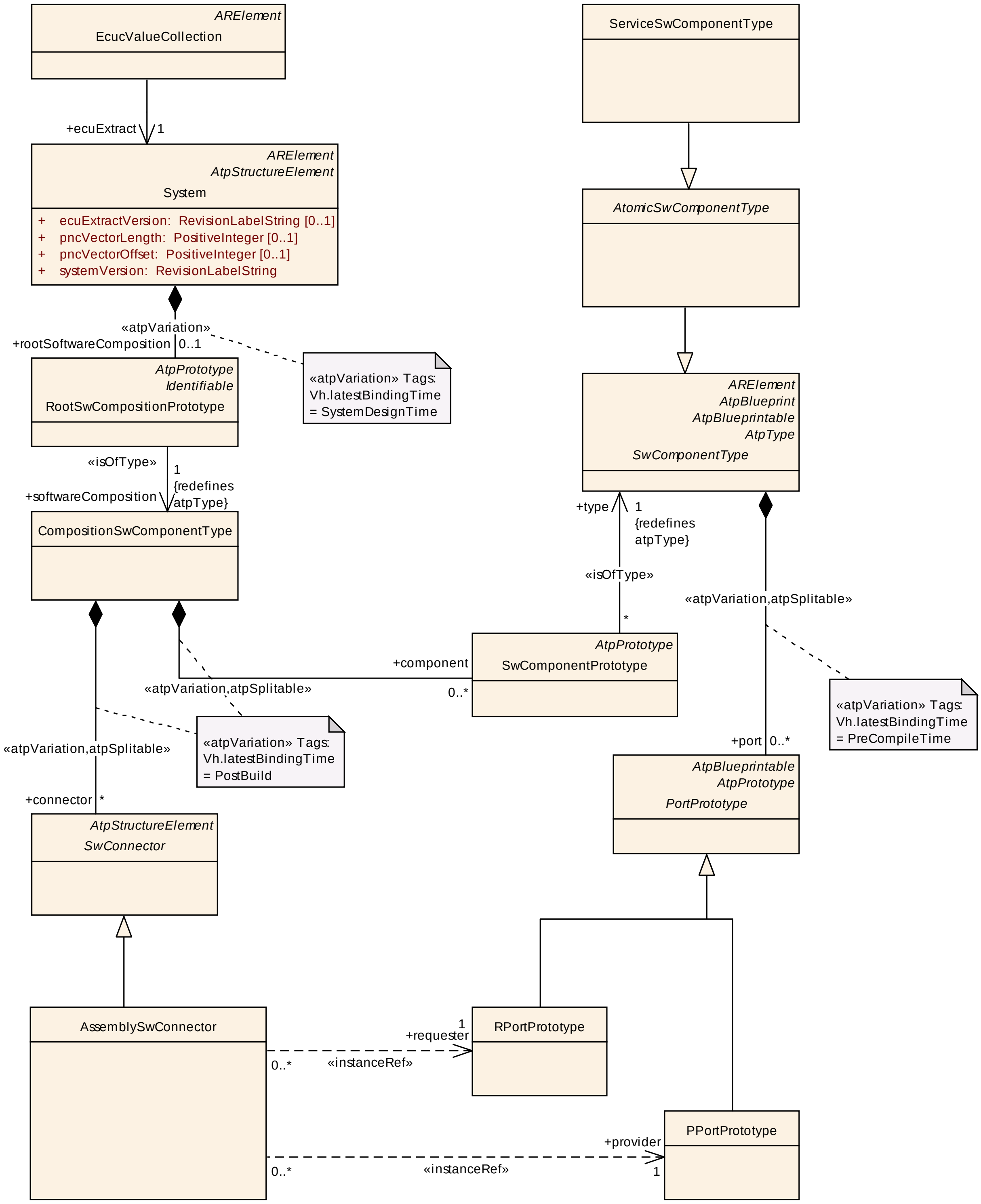

Figure 11.1: Usage of `RootSwCompositionPrototype` on an ECU

11.3 Service Software Component Type

As mentioned in [TPS_SWCT_1405], AUTOSAR Services are represented by a meta model class of their own, the ServiceSwComponentType. As can be seen in Figure 11.2 ServiceSwComponentType is a specialization of AtomicSwComponentType.

Like any other SwComponentType they can aggregate PortPrototypes.

[constr_2019] ServiceSwComponentType shall have service ports only ⌈ In the case of ServiceSwComponentType, all aggregated PortPrototypes need to have an isOfType relationship to a PortInterface which has its isService attribute set to TRUE. One exception as described in [TPS_SWCT_1410] applies. ⌋

[TPS_SWCT_1410] Dcm and Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface ⌈ One exception from the rule described in [constr_2019] applies: the Dcm and Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface. For this purpose the ServiceSwComponentType that represents the Dcm or Dem functionality can have RPortPrototypes typed by a compatible SenderReceiverInterface that may set isService to false. ⌋

[TPS_SWCT_1411] Use cases for a ServiceSwComponentType to express ServiceNeeds ⌈ There are valid use cases for a ServiceSwComponentType to express ServiceNeeds[1]. This leads to a situation where ServiceSwComponentTypes are iteratively created in response to ServiceNeeds expressed by other ServiceSwComponentTypes. Please refer to the AUTOSAR methodology [4] for more details about how this shall be implemented into the workflow. ⌋

Similar to an EcuAbstractionSwComponentType and a ComplexDeviceDriverSwComponentType, the ServiceSwComponentType represents a hybrid concept between Software Component and Basic Software Module. The BSW part is described by the means of the BSW Module Description Template [7].

The mapping between the ServiceSwComponentType and the corresponding BswModuleDescription is provided by the class SwcBswMapping which in addition also maps the two corresponding InternalBehaviors (see [TPS_SWCT_1408]. This mechanism is further explained in [7].

[1] Thereby the previously existing constraint 1127 becomes invalid.

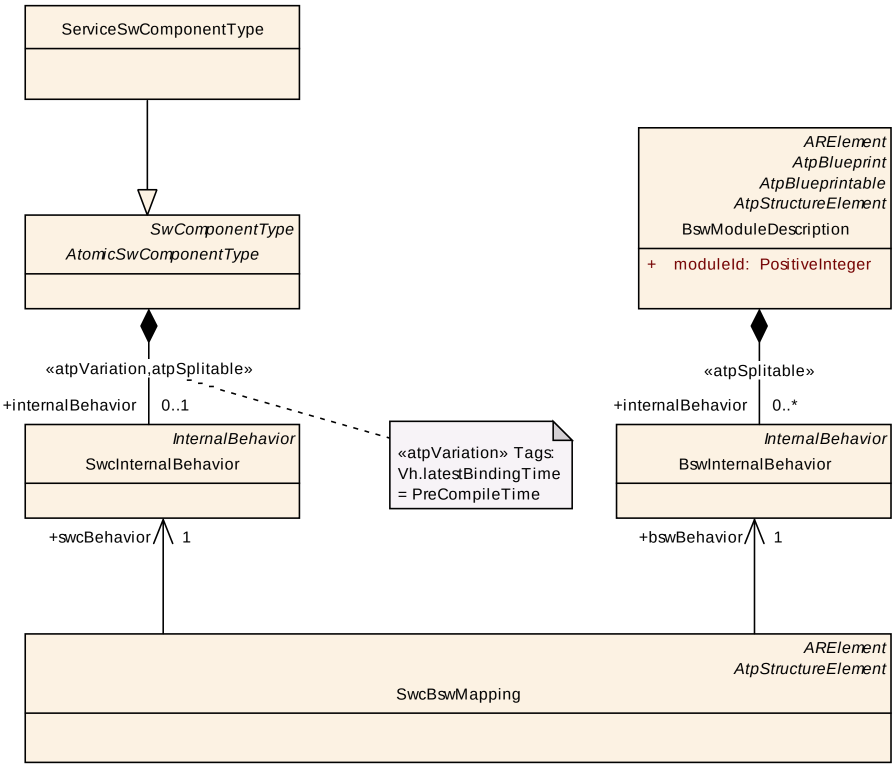

Figure 11.2: `ServiceSwComponentType`

Class	**ServiceSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	ServiceSwComponentType is used for configuring services for a given ECU. Instances of this class are only to be created in ECU Configuration phase for the specific purpose of the service configuration. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 11.3: ServiceSwComponentType

[TPS_SWCT_01412] `ServiceSwComponentType` shall be added in ECU Configuration phase ⌈ `ServiceSwComponentType` shall not be used when modeling application software using `CompositionSwComponentType`; they are only added in ECU Configuration phase where exactly one `SwComponentPrototype` per `ServiceSwComponentType` per ECU is added to the ECU Description model.

The `Base ECU Config Generator` tool needs to take care that for all service ports of `SwComponentPrototype`s mapped to the ECU service ports at the appropriate `ServiceSwComponentType`s are created. In the process the specified communication pattern A, B, or C for a specific kind of service port has to be considered, see table 11.1.

In case of pattern A for each different type of service port one port on the ServiceSwComponentType is created.

In case of pattern B and C for each service port of a SwComponentPrototype one port on the ServiceSwComponentType is created.

More explicitly, all instances of AtomicSwComponentType need to be checked for PortPrototypes of PortInterfaces with isService attribute set to TRUE and referenced by ServiceNeeds and for each of these PortInterface instances belonging to the AUTOSAR Service to be configured one PortPrototype implementing the same or a compatible PortInterface needs to be created on the ServiceSwComponentType. ⌋

[TPS_SWCT_2500] Roles on Application/Service Components need to Match ⌈ The roles of the PortPrototypes (required/provided) on the Application Component and the Service Component side obviously need to match. For example an RPortPrototype attached to an application AtomicSwComponentType matches a PPortPrototype attached to a ServiceSwComponentType. ⌋

11.4 Service Proxy Component Type

[TPS_SWCT_1413] Local communication with services ⌈ Application software components may communicate with an instance of a ServiceSwComponentType only locally on an ECU. ⌋

[TPS_SWCT_1414] Mode manager needs to communicate with application software components located on other ECUs ⌈ There are however use cases for the application and vehicle mode management, where a mode manager (namely the Basic Software Mode Manager, see [19]) is part of the basic software but conceptually still needs to communicate with application software components located on other ECUs (as exemplified by Figure 11.3).

In order to make this communication possible, the ServiceProxySwComponentType is used.

For the application software and the RTE it behaves like a "normal" AtomicSwComponentType, but it is actually a proxy for an AUTOSAR Service. ⌋

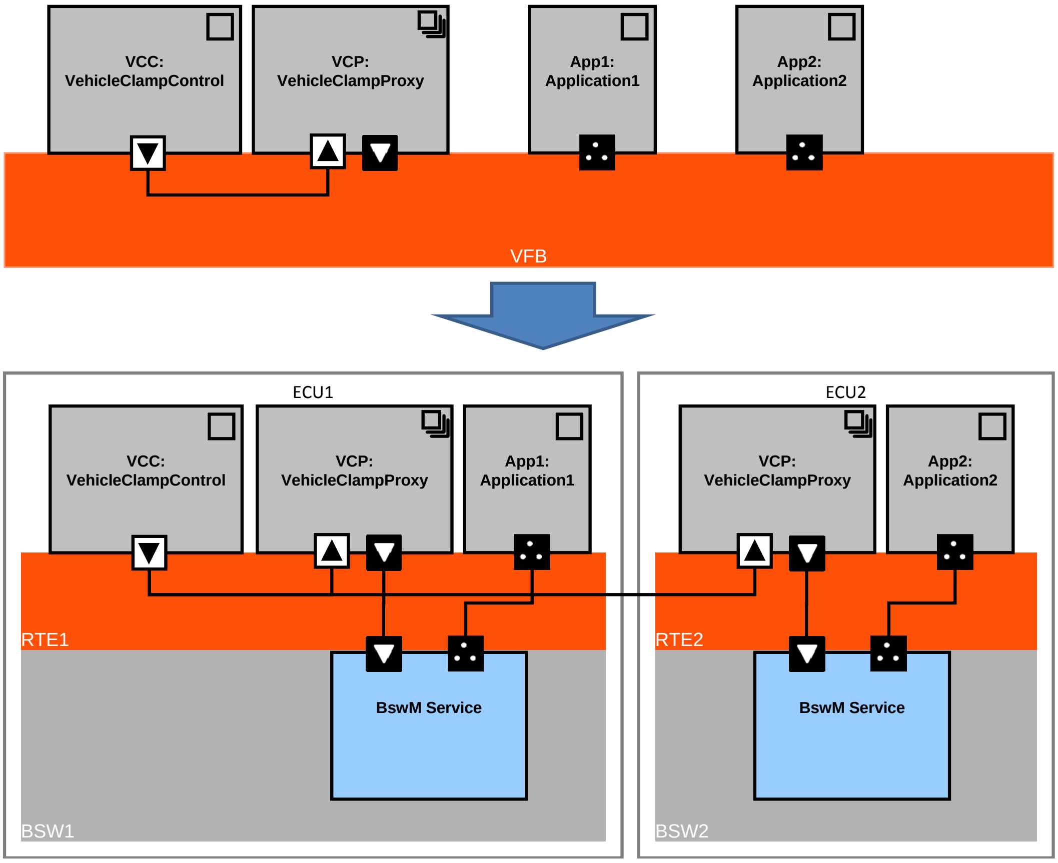

Figure 11.3: Mode request over the network [3]

[TPS_SWCT_1415] Interfaces of `ServiceProxySwComponentType` ⌈ This means that on the one side it has to communicate over service ports with the ECU-local `ServiceSwComponentType` it represents. On the other side it has to offer the corresponding `PortPrototype`s to the `ApplicationSwComponentType`s. ⌋

In the meta-model, the `ServiceProxySwComponentType` does not differ from an `ApplicationSwComponentType` except by its class. It is up to the implementer to meet the restrictions imposed by the semantics as a proxy.

[TPS_SWCT_1416] Difference between a `ServiceProxySwComponentType` and an `ApplicationSwComponentType` ⌈ The main difference between a `ServiceProxySwComponentType` and an `ApplicationSwComponentType` is on system level:

A prototype of a `ServiceProxySwComponentType` can be mapped to several ECUs even if it appears only once in the VFB system, because such a prototype is required on each ECU, where it has to address a local `ServiceSwComponentType`.

As a result of this, a `ServiceProxySwComponentType` can only receive but not send signals over the network. More details are explained in the class table below. ⌋

<table>
<tr><td>Class</td><td colspan="4">ServiceProxySwComponentType</td></tr>
<tr><td>Package</td><td colspan="4">M2::AUTOSARTemplates::SWComponentTemplate::Components</td></tr>
<tr><td>Note</td><td colspan="4">This class provides the ability to express a software-component which provides access to an internal service for remote ECUs. It acts as a proxy for the service providing access to the service.

An important use case is the request of vehicle mode switches: Such requests can be communicated via sender-receiver interfaces across ECU boundaries, but the mode manager being responsible to perform the mode switches is an AUTOSAR Service which is located in the Basic Software and is not visible in the VFB view. To handle this situation, a ServiceProxySwComponentType will act as proxy for the mode manager. It will have R-Ports to be connected with the mode requestors on VFB level and Service-Ports to be connected with the local mode manager at ECU integration time.

Apart from the semantics, a ServiceProxySwComponentType has these specific properties:
A prototype of it can be mapped to more than one ECUs in the system description.
Exactly one additional instance of it will be created in the ECU-Extract per ECU to which the prototype has been mapped.
For remote communication, it can have only R-Ports with sender-receiver interfaces and 1:n semantics.
There shall be no connectors between two prototypes of any ServiceProxySwComponentType.
Tags: atp.recommendedPackage=SwComponentTypes</td></tr>
<tr><td>Base</td><td colspan="4">ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType</td></tr>
<tr><td>Attribute</td><td>Datatype</td><td>Mul.</td><td>Kind</td><td>Note</td></tr>
<tr><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
</table>

Table 11.4: ServiceProxySwComponentType

[constr_2016] Connections between `SwComponentPrototypes` of type `ServiceProxySwComponentType` ⌈ A connection between `PortPrototype`s belonging to `SwComponentPrototype`s where both are typed by `ServiceProxySwComponentType` is not permitted. ⌋

[constr_2017] Ports of `ServiceProxySwComponentTypes` ⌈ `ServiceProxySwComponentType` is only permitted to define

- `RPortPrototype`s that are typed by `SenderReceiverInterface` or
- `PortPrototype`s that are typed by a `PortInterface` where the `isService` attribute is set to true.

⌋

[constr_2018] Supported remote communication of a ServiceProxySwComponentType ⌈ For remote communication, ServiceProxySwComponentType can have only RPortPrototypes typed by SenderReceiverInterfaces in a 1:n communication scenario. ⌋

11.5 Non Volatile Memory

11.5.1 Introduction

The AUTOSAR Architecture defines two alternatives how a software component can access non volatile memory. The first option is that the software component defines in its InternalBehavior a PerInstanceMemory and an NvBlockNeeds referring to the PerInstanceMemory via an RoleBasedDataAssignment.

In this case the *nv block* is exclusively accessed by this software component and the NvM [27]. Therefore the *nv data* is encapsulated inside the software component and can not be accessed directly by other software components.

The PerInstanceMemory can be typed with AUTOSAR Data Types in the case of arTypedPerInstanceMemory or with C data types in the case of perInstanceMemory. For further information see 7.7 and 7.11.3.

The second option is that the software component uses port based communication to access *nv data* provided by a *NvBlockComponent*.

In this case it is possible that *nv data* used by different SWC is packed in one larger *nv block* to reduce the *nv block* management overhead or that the same *nv data* used by several software components with a reduced RAM overhead. The *nv data* of a *NvBlockComponent* is typed with AUTOSARDataTypes.

More details regarding particular scenarios of interacting with the NvM [27] can be found in section 7.11.3.1.

11.5.2 NvBlockComponent

[TPS_SWCT_1142] non-volatile data are provided by a specialized AtomicSwComponentType ⌈ On the VFB [3], the non-volatile data are provided by a specialized AtomicSwComponentType, the NvBlockSwComponentType. An *NvBlockComponent* can represent one or more *NvBlocks* managed by the *NVRAM Manager*. The *nv data* ports of the NvBlockSwComponentType are exclusively typed by NvDataInterfaces. ⌋*(RS_SWCT_3225)*

[TPS_SWCT_1143] Non-volatile data represented by an *NvBlockComponent* can be read and written ⌈ The non-volatile data represented by an *NvBlockComponent* can be read and written. For this purpose the NvBlockSwComponentType is allowed to have PPortPrototypes and RPortPrototypes. ⌋*(RS_SWCT_3225)*

Additional the `NvBlockSwComponentType` might have client server ports to offer the block-related services, administrative services or notifications.

[constr_2009] Supported kinds of ports of a `NvBlockSwComponentType` ⌈ `NvBlockSwComponentType` is only permitted to defined `PortPrototype`s which are either typed by `NvDataInterface` or `ClientServerInterface`. ⌋

A connection of `PortPrototype`s between "*NvBlockSwComponentPrototypes*" is not supported.

[constr_2010] Connections between `SwComponentPrototypes` of type `NvBlockSwComponentType` ⌈ A connection between `PortPrototype`s belonging to `SwComponentPrototype`s where both are typed by `NvBlockSwComponentType` is not permitted. ⌋

Class	**NvBlockSwComponentType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	The NvBlockSwComponentType defines non volatile data which data can be shared between SwComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports. **Tags:** atp.recommendedPackage=SwComponentTypes			
Base	ARElement,ARObject,AtomicSwComponentType,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable,SwComponentType			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
nvBlockDescriptor	NvBlockDescriptor	*	aggr	Specification of the properties of exactly on NvBlock. **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=shortName, variationPoint.shortLabel

Table 11.5: NvBlockSwComponentType

Class	**NvDataInterface**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	A non volatile data interface declares a number of VariableDataPrototypes to be exchanged between non volatile block components and atomic software components. **Tags:** atp.recommendedPackage=PortInterfaces			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpClassifier,AtpType,CollectableElement,DataInterface,Identifiable,MultilanguageReferrable,PackageableElement,PortInterface,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
nvData	VariableDataPrototype	1..*	aggr	The VariableDataPrototype of this nv data interface.

Table 11.6: NvDataInterface

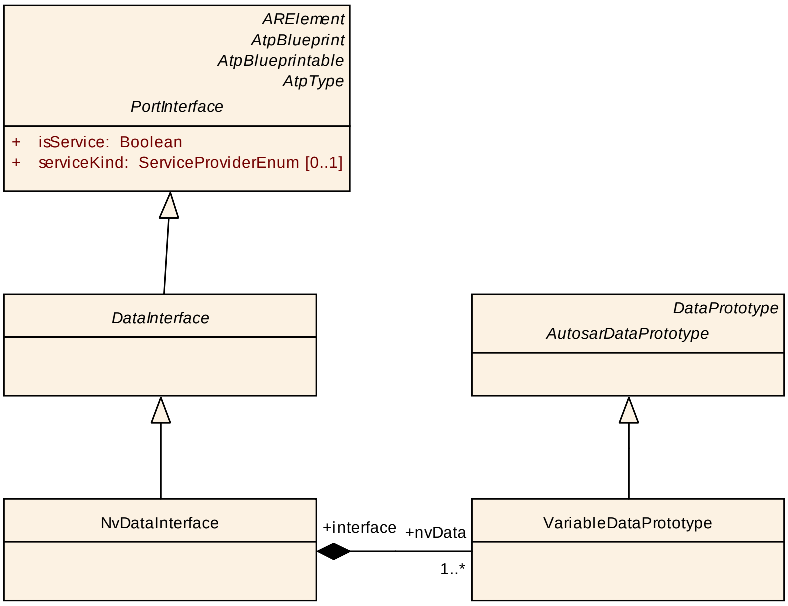

Figure 11.4: NvDataInterface

11.5.3 Software-Components using *nv data* of NvBlockComponents

[constr_2011] Connections between SwComponentPrototypes typed by NvBlockSwComponentType and SwComponentPrototypes typed by other AtomicSwComponentTypes ⌈ The *nv data* ports of the *NvBlockSwComponentPrototype* (SwComponentPrototype which is typed by NvBlockSwComponentType) are either connected with *nv data* ports or with sender/receiver ports of other atomic software components. ⌋

[constr_1148] PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes ⌈ PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes as well as the PortInterfaces used in the context of NvBlockSwComponentTypes shall **always** set the value of the attribute isService set to FALSE. ⌋

[constr_1149] PortPrototypes used for NV data management ⌈ A PortPrototype typed by a ClientServerInterface used for NV data management, i.e. the interaction of ApplicationSwComponentTypes with NvBlockSwComponentTypes, shall be typed by ClientServerInterfaces that are compatible to the particular ClientServerInterfaces standardized by the SWS NvM [27]. [constr_1148] applies. ⌋

For details see chapter 6.4.3.

Note: In case of *nv data* which is read and written and shared between several SwComponentPrototypes the NvBlockSwComponentType establishes a not directly obvious kind of communication. Nevertheless this is intentionally supported and it is under responsibility of the VFB designer to take care that only *nv data* is shared where the functionality of the software components is not impaired.

To determine for an VFB designer which *nv data* can be potentially by mapped into the same NvBlock a software-component can specify further attributes for its *nv data* ports by the definition of `SwcServiceDependency`(s) with `NvBlockNeeds`. In this case the role attribute of the `assignedPort` has no be set to `NvDataPort`. This aspect is also explained in section 7.11.3.1.4.

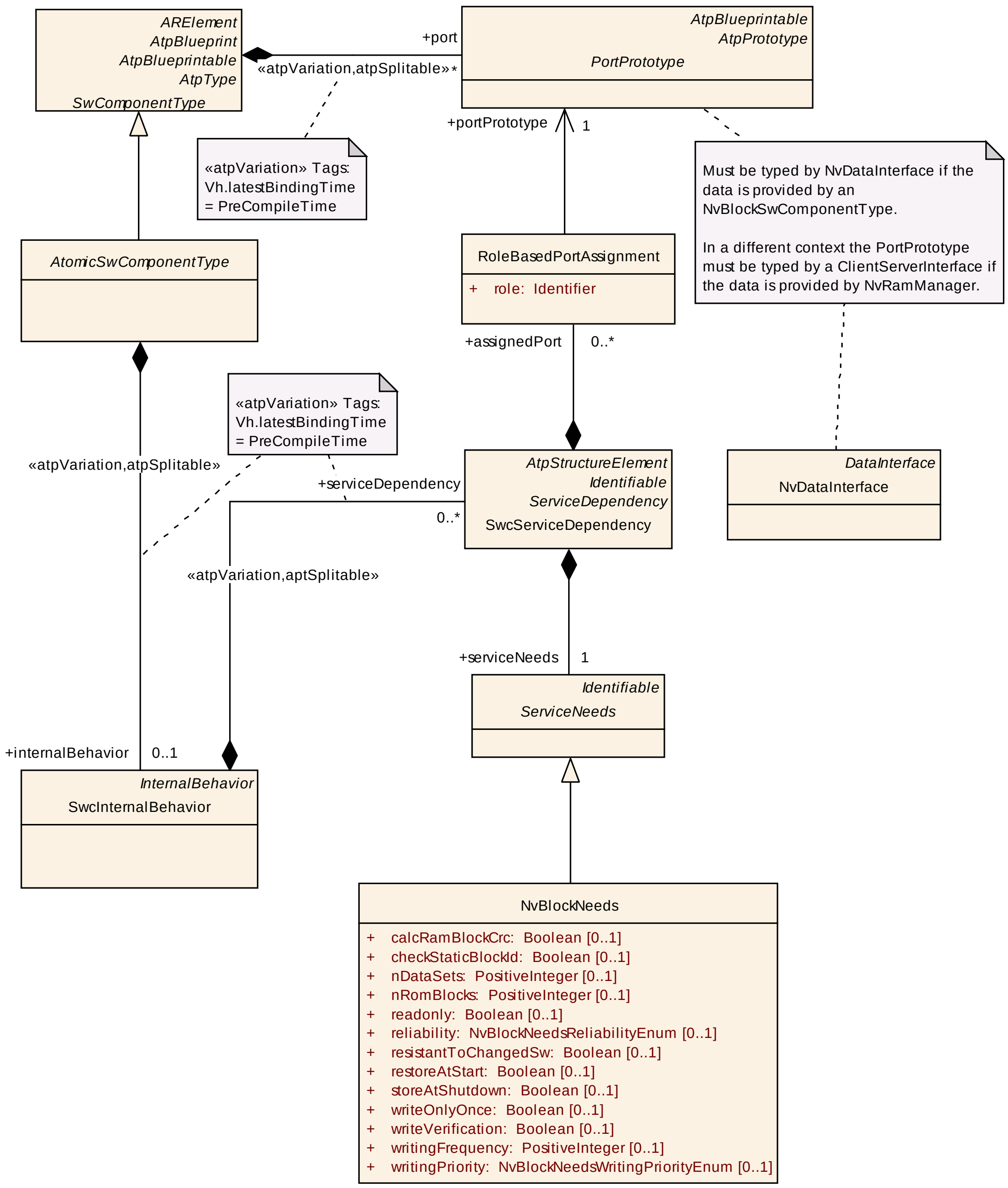

Figure 11.5: `NvBlockNeeds` for *nv data* ports

In contrast to the `NvBlockNeeds` that describe the expected configuration of a whole NvBlock the `NvBlockNeeds` for *nv data* ports defines only the attributes which are required from the point of view of a software-component to ensure its functionality. This means an empty attribute has the semantic of "don't care".

Further on the VFB designer has freedom in its design how the requested NvBlock attributes are fulfilled by the created NvBlockDescriptor.

For instance, *nv data* with different writingFrequency might be mapped to one NvBlock. In this case the NvBlockNeeds of the NvBlockDescriptor has to indicate the worst case which is the higher frequency. The recommended relationship is shown in table 11.7. But please note that this table does not represent a binding constraint.

attribute	NvBlockNeeds of different *nv data* ports of software-components	NvBlockNeeds of NvBlockDescriptor
readonly	recommended to match for all connected *nv data* ports if specified	recommended to be identical as requested by *nv data* ports
reliability	can be different	recommended to be set to the highest reliability class request by any mapped *nv data* ports
resistantToChangedSw	recommended to match for all connected *nv data* ports if specified	recommended to be identical as requested by *nv data* ports
restoreAtStart	recommended to match for all connected *nv data* ports if specified	recommended to be identical as requested by *nv data* ports
writeOnlyOnce	recommended to match for all connected *nv data* ports if specified	recommended to be identical as requested by *nv data* ports
writingFrequency	can be different	recommended to be set to the highest requested frequency of the mapped *nv data* ports
writingPriority	can be different	recommended to be set to the highest requested frequency of the mapped *nv data* ports

Table 11.7: NvBlockNeeds dependencies

11.5.4 NvBlockDescriptor

[TPS_SWCT_1144] NvBlockDescriptor specifies the properties of exactly one *NvBlock* ⌈ A NvBlockDescriptor specifies the properties of exactly one *NvBlock* of a NvBlockSwComponentType. It contains information about the requested *NvBlock* configuration of the *NVRAM Manager*, RAM Block and ROM Block, the mapping between the ports of the NvBlockSwComponentType and the data inside a RAM Block as well as the role of the client/server ports. ⌋

Class	**NvBlockDescriptor**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Specifies the properties of exactly on NvBlock.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	Datatype	Mul.	Kind	Note
clientServerPort	RoleBasedPortAssignment	*	aggr	The RoleBasedPortAssignement defines which client server port of the NvBlockSwComponentType serves for which kind of service or notification. In case of notifications one common callback function is provided by the RTE for each individual kind of notification defined by the "role". The aggregation of RoleBasedPortAssignment is subject to variability with the purpose to support the conditional existence of ports. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstanSpecificationMapping to be applied for the particular NvBlock
dataTypeMapping	DataTypeMappingSet	*	ref	Reference to the DataTypeMapping to be applied for the particular NvBlock
instantiationDataDefProps	InstantiationDataDefProps	*	aggr	The purpose of InstantiationDataDefProps are the refinement of some data def properties of individual instantiations within the context of a NvBlockSwComponentType. The aggregation of InstantiationDataDefProps is subject to variability with the purpose to support the conditional existence of ports, component internal memory objects and those attributes. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
nvBlockDataMapping	NvBlockDataMapping	1..*	aggr	Defines the mapping between the VariableDataPrototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The aggregation of NvBlockDataMapping is subject to variability with the purpose to support the conditional existence of nv data ports. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime
nvBlockNeeds	NvBlockNeeds	1	aggr	Specifies the abstract needs on the configuration of the NvRam Manager for the single NvRam Block described by this NvBlockDescriptor.
ramBlock	VariableDataPrototype	1	aggr	Defines the RAM Block of the NvBlock provided by NvBlockSwComponentType.
romBlock	ParameterDataPrototype	0..1	aggr	Defines the ROM Block of the NvBlock provided by NvBlockSwComponentType.

Table 11.8: NvBlockDescriptor

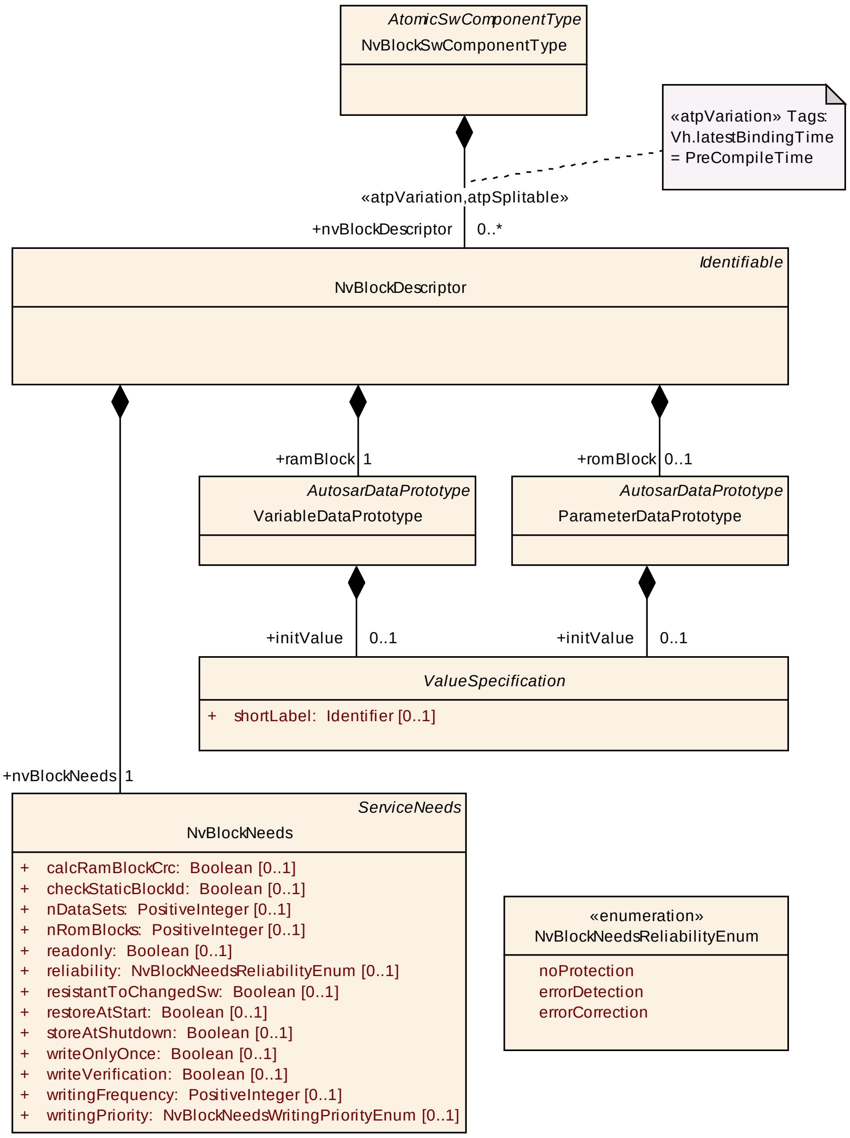

Figure 11.6: `NvBlockSwComponentType` and `NvBlockDescriptor`

Enumeration	**NvBlockNeedsReliabilityEnum**
Package	M2::AUTOSARTemplates::CommonStructure::ServiceNeeds
Note	Reliability against data loss on the non-volatile medium. These requirements give only a relative indication, for example on the required degree of redundancy for storage. They do however not specify by which means (e.g. software or hardware) the reliability is actually achieved.
Literal	***Description***
errorCorrection	Errors shall be corrected
errorDetection	Errors shall be detected
noProtection	Data need not to be handled with protection

Table 11.9: NvBlockNeedsReliabilityEnum

[constr_1095] Values of `nDataSets` vs. `reliability` ⌈ If the value of `nDataSets` is greater than 0 the value of `reliability` shall not be set to `errorCorrection`. ⌋

The reason for the existence of [constr_1095] is that the AUTOSAR NvM [27] does not support error correction for NV data sets.

If the value of `nDataSets` is equal to 0 the value of `reliability` can take any value out of `NvBlockNeedsReliabilityEnum`.

11.5.4.1 NvBlockNeeds

The requested *NvBlock* configuration of the *NVRAM Manager* is described by the `NvBlockNeeds` of the `NvBlockDescriptor`.

This information can be evaluated during ECU configuration similar to the `NvBlockNeeds` of an atomic software component or a BSW module. For further details see 7.11.3.

11.5.4.2 RAM Block and ROM Block

[TPS_SWCT_1145] RAM Block and the ROM Block are described by a `VariableDataPrototype` and a `ParameterDataProrotype` ⌈ The RAM Block and the ROM Block are described by a `VariableDataPrototype` and a `ParameterDataProrotype` which are typed by an `AutosarDataType`. ⌋

[TPS_SWCT_1146] ROM Block is optional ⌈ The ROM Block is optional. If an ROM block is configured, the RTE copies the ROM Block constants into the RAM Block in case of a block initialization notification (*NvMNotifyInitBlock*). ⌋

[TPS_SWCT_1147] No ROM Block is configured ⌈ If there is no ROM Block configured, the connected software components are either required to offer this functionality by a proper implementation of block initialization notification or the NvBlock has to be configured, that no ROM Block is needed. ⌋

[constr_2012] Compatibility of `ImplementationDataTypes` used for RAM and ROM Block ⌈

The RAM and the ROM Block shall have compatible `ImplementationDataTypes` to ensure, that the *NvBlock* default values in the ROM Block can be copied into the RAM Block.

⌋

Additionally it is possible that RAM Block and ROM Block are defined to be calibratable or measurable. Preceding `SwDataDefProps` might be defined with the means of an `InstantiationDataDefProps`.

11.5.4.3 NvBlockDataMapping

[TPS_SWCT_1148] NvBlockDataMapping ⌈ The meta-class NvBlockDataMapping specifies the mapping of VariableDataPrototypes of the NvBlockSwComponentType's ports (PPortPrototypes / RPortPrototypes) to VariableDataPrototypes inside the RAM Block. ⌋

This ensures a flexible but deterministic *NvBlock* memory structure given by the ImplementationDataType of the RAM Block and ROM Block and its association to the ports of the NvBlockSwComponentType.

[constr_2013] Compatibility of ImplementationDataTypes for NvBlockDataMapping ⌈ The NvBlockDataMapping is only valid if the ImplementationDataTypes of all referenced VariableDataPrototypes are compatible. ⌋

But nevertheless it is valid, that not all VariableDataPrototypes inside the RAM Block are mapped to ports. This enables to have fill elements or logistic data in the NvBlock which are not accessed by software components.

Class	**NvBlockDataMapping**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::NvBlockComponent			
Note	Defines the mapping between the VariableDataPrototypes in the NvBlockComponents ports and the VariableDataPrototypes of the RAM Block. The data types of the referenced VariableDataPrototypes in the ports and the referenced sub-element (inside a CompositeDataType) of the VariableDataPrototype representing the RAM Block shall be compatible.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
nvRamBlockElement	AutosarVariableRef	1	aggr	Reference to a VariableDataPrototype of a Ram Block.
readNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a pPort of the NvBlockComponent providing read access to the NvRam Mirror. If there is no port providing read access (write-only) the reference can be omitted.
writtenNvData	AutosarVariableRef	0..1	aggr	Reference to a VariableDataPrototype of a rPort of the NvBlockComponent providing write access to the NvRam Mirror. If there is no port providing write access (read-only) the reference can be omitted.

Table 11.10: NvBlockDataMapping

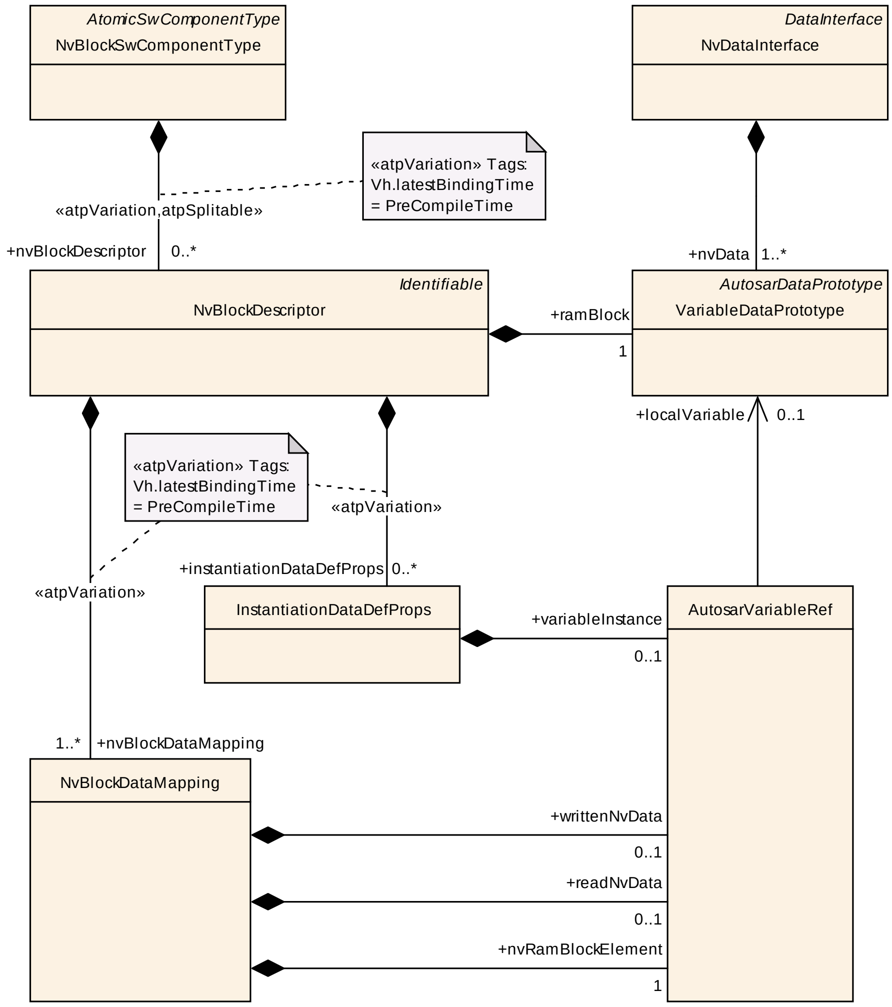

Figure 11.7: `NvBlockToPortMapping and InstantiationDataDefProps`

11.5.4.4 Client Server Ports

[TPS_SWCT_1149] `RoleBasedPortAssignment` of `NvBlockDescriptor` ⌈ The `RoleBasedPortAssignement` of the `NvBlockDescriptor` describes which client/server `PortPrototype` of the `NvBlockSwComponentType` serves for which purpose. The `role` specifies if the port serves for block-related services, administrative services or notification. ⌋

[constr_2014] Limitation of `RoleBasedPortAssignement.role` in `NvBlockDescriptors` ⌈ The `role` has to be set to a valid name of the *Standardized AUTOSAR Interface* used for the *NVRAM Manager* e.g. *NvMNotifyJobFinished* or *NvMNotifyInitBlock*. ⌋

In case of notifications one common callback function is provided by the RTE for each individual kind of notification defined by the `role`.

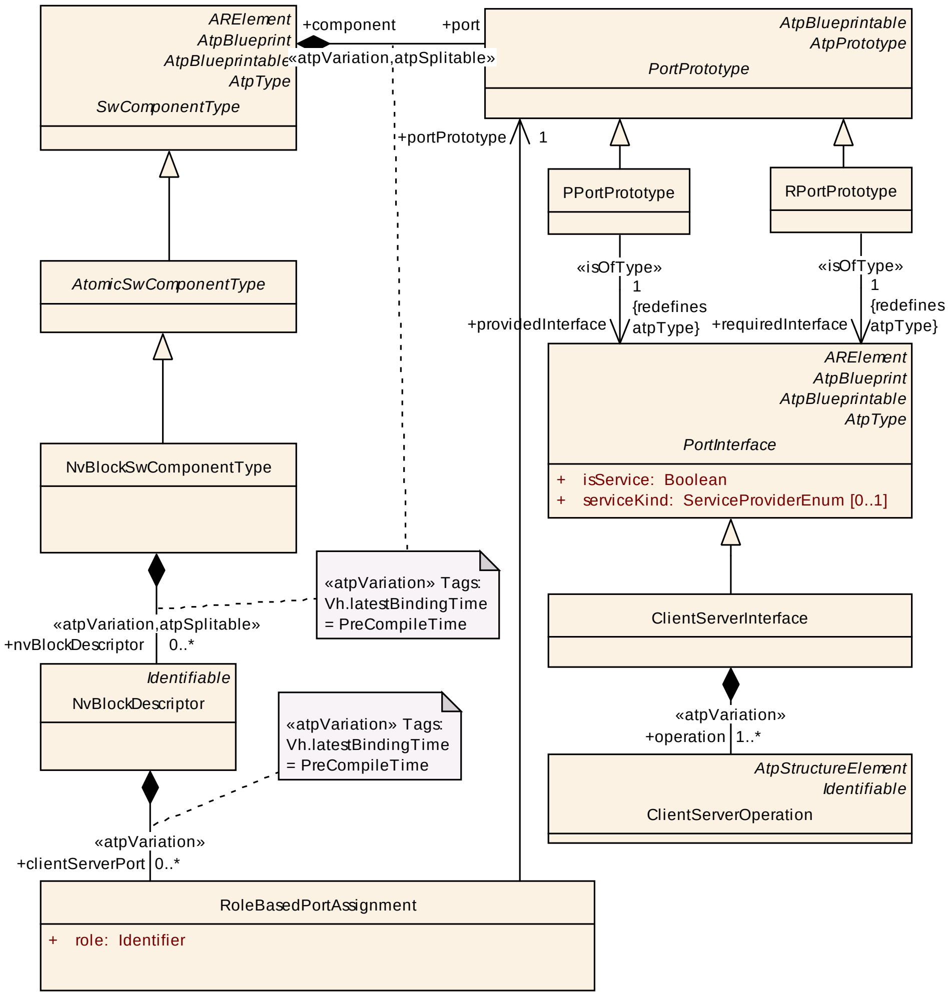

Figure 11.8: `NvBlockNotification`

11.5.4.5 SwcInternalBehavior of an NvBlockSwComponentType

[TPS_SWCT_1150] **`InternalBehavior` of a `NvBlockSwComponentType`** ⌈ The `InternalBehavior` of a `NvBlockSwComponentType` is only used for an limited scope. It is required, if the `NvBlockSwComponentType` defines server ports to enable access to the NvBlock management API. To enable the configuration of the server invocation in the RTE's ECU configuration the *NvBlockComponent* needs:

- OperationInvokedEvent(s)
- server runnable
- Port defined argument values to defined the *nv block* ID which has to be passed to the NvM

⌋

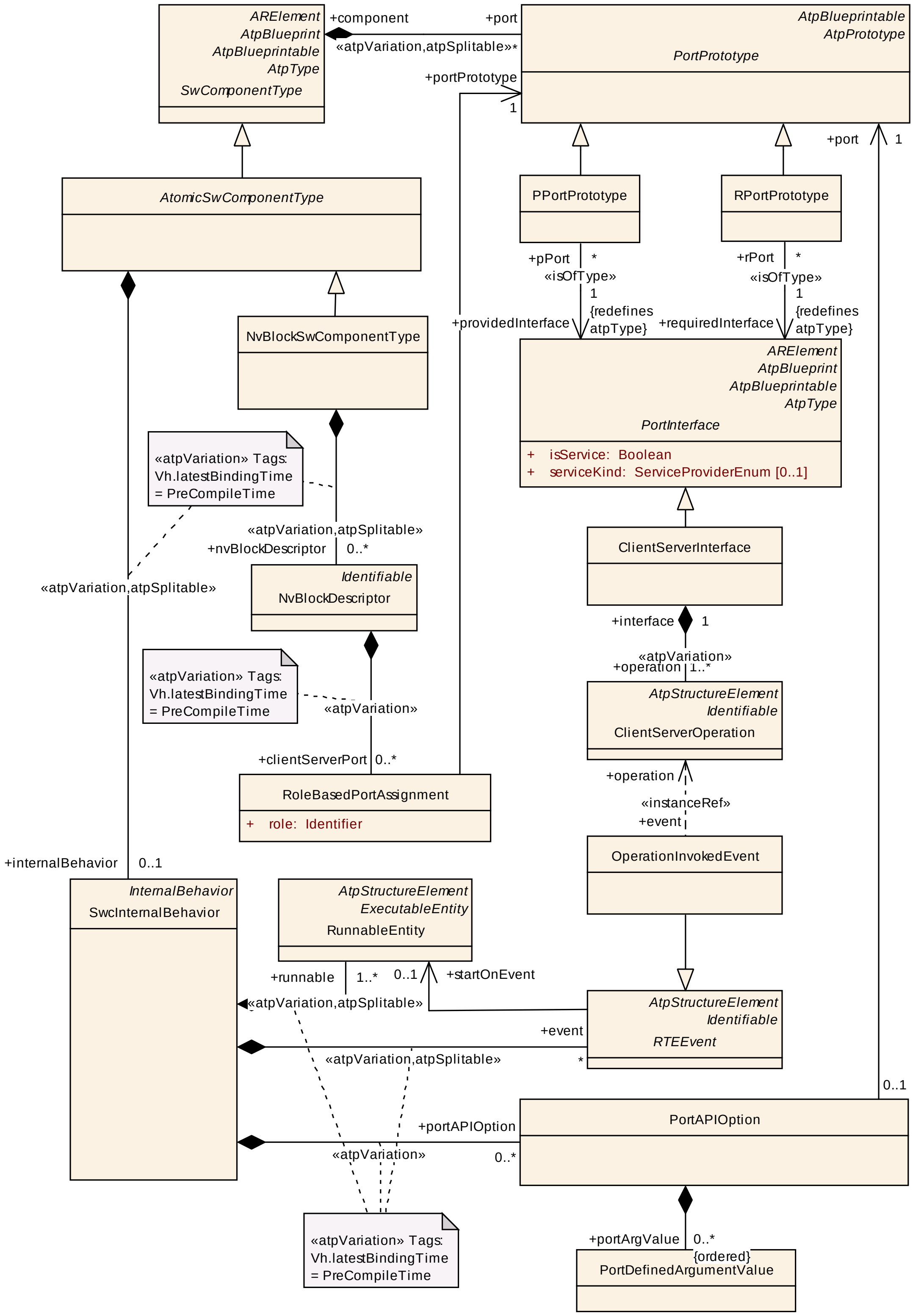

Figure 11.9: `NvBlockNotification`

[TPS_SWCT_1152] `InternalBehavior` does not have further attributes ⌈ It is not expected, that such `InternalBehavior` do have further attributes like exclusive areas, per instance memory or inter runnable variables, etc. ⌋

[TPS_SWCT_1151] `RunnableEntity`s do not have further attributes ⌈ The same condition exists for the `RunnableEntity`s of such `InternalBehavior` which shall not define further attributes, e.g. data access points or server call points. ⌋

[constr_2015] Limitation of `SwcInternalBehavior` of a `NvBlockSwComponentType` ⌈ The `SwcInternalBehavior` of a `NvBlockSwComponentType` is only permitted to define

- `OperationInvokedEvent`s
- `RunnableEntity`s triggered by `OperationInvokedEvent`s (server runnables)
- `RunnableEntity`s which defines only the mandatory attributes `symbol` and `canBeInvokedConcurrently`
- `PortAPIOption`s defining `PortDefinedArgumentValue`s

⌋

12 Software Component Documentation

AUTOSAR supports documentation of software component types by adopting the principles of ASAM-FSX [37] Standard to AUTOSAR. With AUTOSAR Release 4.0 the AUTOSAR XML schema provides support for integrated and well structured documentation. More details about the AUTOSAR Documentation Support Concept can be found in the AUTOSAR Generic Structure Template [13].

[TPS_SWCT_1062] Documentation of software-components ⌈ As shown in figure 12.1, the documentation of a `software component` is composed of several chapters. Some chapters are predefined, describing the component from the perspective of different activities performed on the component like testing it (`swTestDesc`), maintaining it(`swMaintenanceNotes`), calibrating it (`swCalibrationNotes`) or performing diagnostic (`swDiagnosticsNotes`) on the component. ⌋*(RS_SWCT_2110, RS_SWCT_3230)*

Two other predefined chapters describe the component (`swFeatureDesc`) and define its physical functionality (`swFeatureDef`). In order to describe additional aspects of a software component, an arbitrary number of free chapters can be defined.

The predefined chapters typically provide informal guideline (e.g., recommendation) or documentation. Formal information can be captured using special data groups [13] or annotating documentation construct with semantic information. This could be used to extend the predefined chapters or in separate free chapters.

Note that the documentation of a software component can be stored in a different file than the component itself (i.e., it is ≪`atpSplitable`≫ from the component).

Each of the predefined and free chapters follows the ≪`atpVariation`≫ stereotype to support variant handling (see [13]) on the documentation at the chapter level. These variation points have a post-build as latest binding time, because the decision to include or exclude a chapter as well as the decision which variant of this chapter should be included can be made when the component has been built.

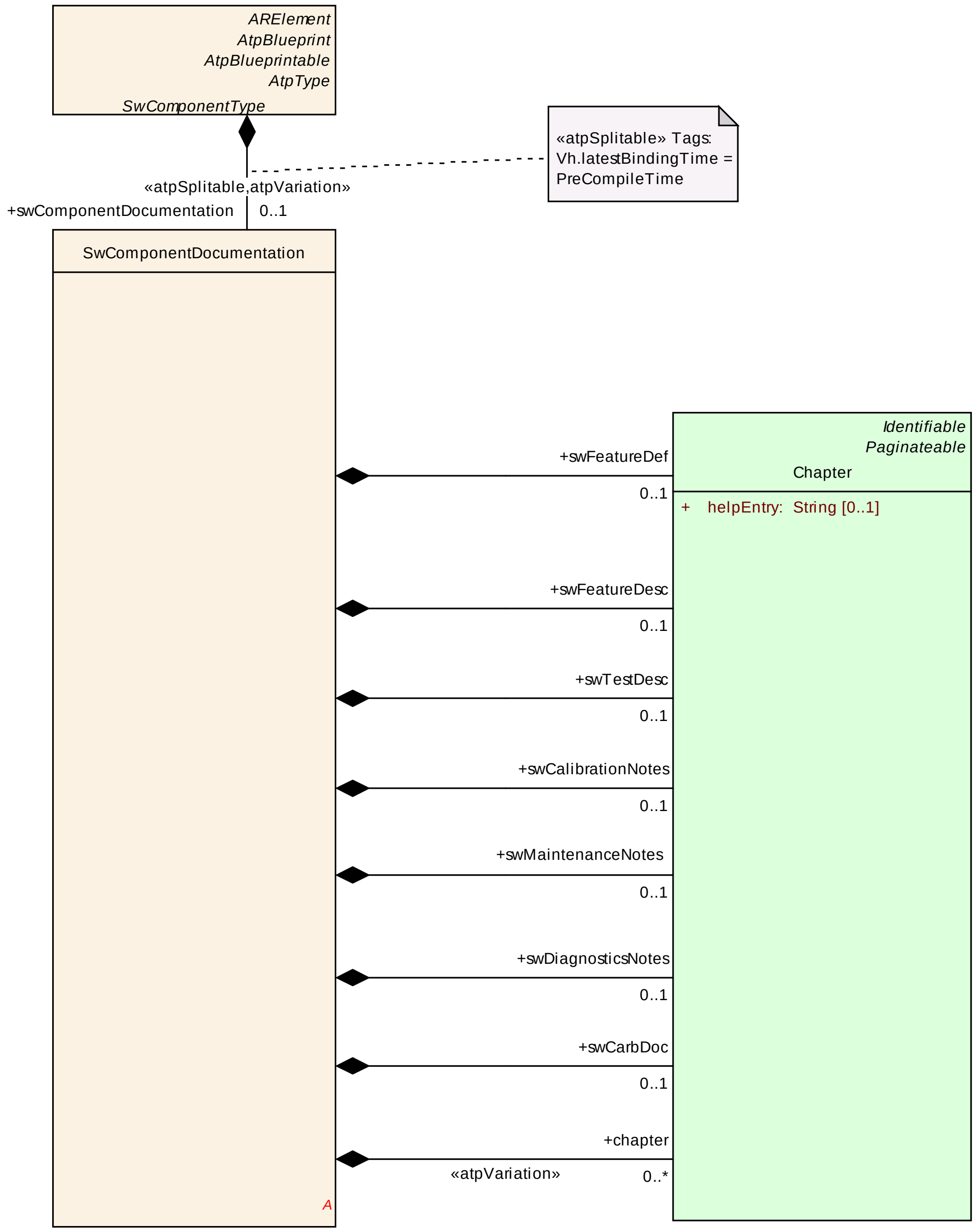

Figure 12.1: Software component documentation

Class	**SwComponentDocumentation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::SoftwareComponent Documentation			
Note	This class specifies the ability to write dedicated documentation to a component type according to ASAM FSX.			
Base	ARObject			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	Datatype	Mul.	Kind	Note
chapter	Chapter	*	aggr	These chapters provide additional information about the software component that do not fit in the other chapters. Note that this is subject to variation because Chapter aggregations in the role chapter are variant within the documentation in general. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PostBuild xml.roleElement=true; xml.roleWrapper Element=false; xml.sequenceOffset=100; xml.type Element=true
swCalibrationNotes	Chapter	0..1	aggr	This element contains calibration instructions and hints for a calibration engineer. **Tags:** xml.roleElement=true; xml.sequence Offset=60; xml.typeElement=false
swCarbDoc	Chapter	0..1	aggr	This element records the documentation requested by CARB. **Tags:** xml.roleElement=true; xml.sequence Offset=80; xml.typeElement=false
swDiagnosticsNotes	Chapter	0..1	aggr	This element contains general information about diagnostics issues within the component. **Tags:** xml.roleElement=true; xml.sequence Offset=75; xml.typeElement=false
swFeatureDef	Chapter	0..1	aggr	This element contains the definition of the physical functionality of this software component. This definition is more or less formal and is intended to be delivered from modeling tools. **Tags:** xml.roleElement=true; xml.sequence Offset=20; xml.typeElement=false
swFeatureDesc	Chapter	0..1	aggr	This element contains the textual description of the software functionality of this software component. Expert should write this description. **Tags:** xml.roleElement=true; xml.sequence Offset=30; xml.typeElement=false
swMaintenanceNotes	Chapter	0..1	aggr	This element contains information regarding the software maintenance of the component. **Tags:** xml.roleElement=true; xml.sequence Offset=70; xml.typeElement=false
swTestDesc	Chapter	0..1	aggr	This element contains suggestions and hints for the test of the software functionality of this software component. **Tags:** xml.roleElement=true; xml.sequence Offset=50; xml.typeElement=false

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

Table 12.1: SwComponentDocumentation

A Renamed Meta-Model Elements

A.1 Introduction

In the course of preparing AUTOSAR Release 4.0 some of the existing meta-model elements (as of R 3.x) have been renamed for a better clarity and consistency with respect to other meta-mode elements. This chapter provides an overview of the changed meta-model elements in order to allow readers with a background in R3.x specifications to understand changes made by mere renaming.

A.2 Renamed Meta-Model Elements

Old Name	*New Name*
ApplicationSoftwareComponentType	ApplicationSwComponentType
ArCalprmRef	AutosarParameterRef
ArgumentPrototype	ArgumentDataPrototype
ArrayElement	ApplicationArrayElement
ArrayType	ApplicationArrayDataType
AssemblyConnectorPrototype	AssemblySwConnector
AtomicSoftwareComponentType	AtomicSwComponentType
CalibrationPortAnnotation	ParameterPortAnnotation
CalprmAccess	ParameterAccess
CalprmComponentType	ParameterSwComponentType
CalprmElementPrototype	ParameterDataPrototype
CalprmInterface	ParameterInterface
ComplexDeviceDriverComponentType	ComplexDeviceDriverSwComponentType
ComponentPrototype	SwComponentPrototype
ComponentType	SwComponentType
CompositeType	ApplicationCompositeDataType
CompositionType	CompositionSwComponentType
ConnectorPrototype	SwConnector
DelegationConnectorPrototype	DelegationSwConnector
DataElementPrototype	VariableDataPrototype
DataPrototype	AutosarDataPrototype
Datatype	ApplicationDataType
DirectionKind	ArgumentDirectionEnum
EcuAbstractionComponentType	EcuAbstractionSwComponentType
HandleInvalidType	HandleInvalidEnum
InternalBehavior	SwcInternalBehavior
LimitKind	DataLimitKindEnum
ModeInterface	ModeSwitchInterface
ModeSwitchComSpec	ModeSwitchSenderComSpec
ModeSwitchEvent	SwcModeSwitchEvent
OperationPrototype	ClientServerOperation
PrimitiveType	ApplicationPrimitiveDataType
ProcessingKind	ProcessingKindEnum
RecordElement	ApplicationRecordElement
RecordType	ApplicationRecordDataType
SensorActuatorSoftwareComponentType	SensorActuatorSwComponentType

ServiceComponentType	ServiceSwComponentType
SoftwareComposition	RootSwCompositionPrototype
SwCalprmAxisCommonAxis	SwAxisIndividual
SwCalprmAxisIndividualAxis	SwAxisGrouped
UnqueuedReceiverComSpec	NonqueuedReceiverComSpec
UnqueuedSenderComSpec	NonqueuedSenderComSpec

Table A.1: Renamed meta-model elements

Please note that InternalBehavior has been moved out of the scope of this specification toward a more general meaning. The original semantics of InternalBehavior within the scope of this specification is now implemented by SwcInternalBehavior.

B Glossary

Artifact This is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts ([38]).

At a high level, an artifact is represented as a single conceptual file.

AUTOSAR Tool This is a software tool which supports one or more tasks defined as AUTOSAR tasks in the methodology. Depending on the supported tasks, an AUTOSAR tool can act as an authoring tool, a converter tool, a processor tool or as a combination of those (see separate definitions).

AUTOSAR Authoring Tool An AUTOSAR Tool used to create and modify AUTOSAR XML Descriptions. Example: System Description Editor.

AUTOSAR Converter Tool An AUTOSAR Tool used to create AUTOSAR XML files by converting information from other AUTOSAR XML files. Example: ECU Flattener

AUTOSAR Definition This is the definition of parameters which can have values. One could say that the parameter values are Instances of the definitions. But in the meta model hierarchy of AUTOSAR, definitions are also instances of the meta model and therefore considered as a description. Examples for AUTOSAR definitions are: `EcucParameterDef`, `PostBuildVariantCriterion`, `SwSystemconst`.

AUTOSAR XML Description In AUTOSAR this means "filled Template". In fact an AUTOSAR XML description is the XML representation of an AUTOSAR model.

The AUTOSAR XML description can consist of several files. Each individual file represents an AUTOSAR partial model and shall validate successfully against the AUTOSAR XML schema.

AUTOSAR Meta-Model This is an UML2.0 model that defines the language for describing AUTOSAR systems. The AUTOSAR meta-model is an UML representation of the AUTOSAR templates. UML2.0 class diagrams are used to describe the attributes and their interrelationships. Stereotypes, UML tags and OCL expressions (object constraint language) are used for defining specific semantics and constraints.

AUTOSAR Model This is a representation of an AUTOSAR product. The AUTOSAR model represents aspects suitable to the intended use according to the AUTOSAR methodology.

Strictly speaking, this is an instance of the AUTOSAR meta-model. The information contained in the AUTOSAR model can be anything that is representable according to the AUTOSAR meta-model.

AUTOSAR Partial Model In AUTOSAR, the possible partitioning of models is marked in the meta-model by ≪`atpSplitable`≫. One partial model is represented in an AUTOSAR XML description by one file. The partial model does not need to fulfill all semantic constraints applicable to an AUTOSAR model.

AUTOSAR Processor Tool An AUTOSAR Tool used to create non-AUTOSAR files by processing information from AUTOSAR XML files. Example: RTE Generator

AUTOSAR Template The term "Template" is used in AUTOSAR to describe the format different kinds of descriptions. The term template comes from the idea, that AUTOSAR defines a kind of form which shall be filled out in order to describe a model. The filled form is then called the description.

In fact the AUTOSAR templates are now defined as a meta model.

AUTOSAR XML Schema This is a W3C XML schema that defines the language for exchanging AUTOSAR models. This Schema is derived from the AUTOSAR meta model. The AUTOSAR XML Schema defines the AUTOSAR data exchange format.

Blueprint This is a model from which other models can be derived by copy and refinement. Note that in contrast to meta model resp. types, this process is *not* an instantiation.

Instance Generally this is a particular exemplar of a model or of a type.

Meta-Model This defines the building blocks of a model. In that sense, a Meta-Model represents the language for building models.

Meta-Data This includes pertinent information about data, including information about the authorship, versioning, access-rights, timestamps etc.

Model A Model is an simplified representation of reality. The model represents the aspects suitable for an intended purpose.

Partial Model This is a part of a model which is intended to be persisted in one particular artifact.

Pattern in GST : This is an approach to simplify the definition of the meta model by applying a model transformation. This transformation crates an enhanced model out of an annotated model.

Property A property is a structural feature of an object. As an example a “connector” has the properties “receive port” and “send port”

Properties are made variant by the ≪`atpVariation`≫.

Prototype This is the implementation of a role of a type within the definition of another type. In other words a type may contain Prototypes that in turn are typed by "Types". Each one of these prototypes becomes an instance when this type is instantiated.

Type A type provides features that can appear in various roles of this type.

Value This is a particular value assigned to a “Definition”.

Variability Variability of a system is its quality to describe a set of variants. These variants are characterized by variant specific property settings and / or selections.

As an example, such a system property selection manifests itself in a particular "receive port" for a connection.

This is implemented using the ≪atpVariation≫.

Variant A system variant is a concrete realization of a system, so that all its properties have been set respectively selected. The software system has no variability anymore with respect to the binding time.

This is implemented using EvaluatedVariantSet.

Variation Binding A variant is the result of a variation binding process that resolves the variability of the system by assigning particular values/selections to all the system's properties.

This is implemented by VariationPoint.

Variation Binding Time The variation binding time determines the step in the methodology at which the variability given by a set of variable properties is resolved.

This is implementing by vh.LatestBindingtime at the related properties .

Variation Definition Time The variation definition time determines the step in the methodology at which the variation points are defined.

Variation Point A variation point indicates that a property is subject to variation. Furthermore, it is associated with a condition and a binding time which define the system context for the selection / setting of a concrete variant.

This is implemented by VariationPoint.

C History of Constraints and Specification Items

C.1 Constraint History of this Document according to AUTOSAR R4.0.1

C.1.1 Changed Constraints in R4.0.1

N/A

C.1.2 Added Constraints in R4.0.1

Please note that constraints listed using an italicized typeface have been deleted in later versions of the document.

Number	Heading
[constr_1002]	End-to-end protection does not support n:1 communication
[constr_1005]	Compatibility of ImplementationDataTypes mapped to the same ApplicationDataType
[constr_1006]	applicable data categories
[constr_1007]	Allowed attributes of SwDataDefProps
[constr_1008]	Applicability of categories STRUCTURE and ARRAY
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes
[constr_1010]	If nativeDeclaration does note exist
[constr_1011]	CATEGORY of SwBaseType
[constr_1012]	Value of CATEGORY is FIXED_LENGTH
[constr_1013]	Value of CATEGORY is VARIABLE_LENGTH
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDatDefProps
[constr_1016]	invalidValue is restricted
[constr_1017]	Supported combinations of SwImplPolicy and SwCalibrationAccess
[constr_1018]	measurementPoint shall not be referenced in DataReadAccess
[constr_1019]	Compatibility of input value and axis
[constr_1020]	ParameterDataPrototype needs to be of compatible data type as referenced in sharedAxisType
[constr_1021]	A CompuMethod shall specify instructions for both directions
[constr_1022]	Limits shall be defined for each direction of CompuMethod
[constr_1023]	Specification of Units in CompuMethods
[constr_1024]	Stepwise definition of CompuMethods
[constr_1025]	Avoid division by zero in rational formula
[constr_1026]	Compatibility of Units
[constr_1027]	Types for record layouts
[constr_1029]	ConstantSpecificationMapping and ConstantSpecification
[constr_1030]	ParameterSwComponentType references ConstantSpecificationMappingSet
[constr_1031]	NvBlockSwComponentType references ConstantSpecificationMappingSet
[constr_1032]	DelegationSwConnector can only connect PortPrototypes of the same kind
[constr_1033]	Communication scenarios for sender/receiver communication
[constr_1035]	Recursive definition of CompositionSwComponentType
[constr_1036]	Connect kinds of PortInterfaces
[constr_1037]	Client may not connect to multiple servers

[constr_1038]	Reference to ApplicationError
[constr_1039]	Relevance of SwImplPolicy
[constr_1040]	Conversion of SenderReceiverInterfaces
[constr_1041]	Conversion of ClientServerInterfaces
[constr_1042]	Definition of a linear data scaling
[constr_1043]	PortInterface vs. ComSpec
[constr_1044]	Applicability of DataFilter
[constr_1045]	Supported value encodings for SwBaseType in the context of PortInterfaces
[constr_1046]	Applicability of [constr_1045]
[constr_1047]	Compatibility of ApplicationPrimitiveDataTypes
[constr_1048]	Compatibility of ApplicationRecordDataTypes
[constr_1049]	Compatibility of ApplicationArrayDataTypes
[constr_1050]	Compatibility of ImplementationDataTypes
[constr_1051]	Compatibility of SwDataDefProps
[constr_1052]	Compatibility of Units
[constr_1053]	Compatibility of PhysicalDimensions
[constr_1054]	No DataConstr available at the provider
[constr_1055]	ImplementationDataType has category VALUE
[constr_1056]	ImplementationDataType has category TYPE_REFERENCE
[constr_1057]	ImplementationDataType has category DATA_REFERENCE
[constr_1058]	ImplementationDataType has category FUNCTION_REFERENCE
[constr_1059]	Compatibility of data types with category VALUE
[constr_1060]	Compatibility of data types with category ARRAY, VAL_BLK, or STRING
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_1062]	Compatibility of data types with category BIT
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1064]	Compatibility of data types with category COM_AXIS, RES_AXIS, CURVE or MAP
[constr_1066]	ApplicationDataType is or is not compatible to specific ImplementationDataType
[constr_1067]	ApplicationDataType is or is not compatible to specific ImplementationDataType
[constr_1068]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes
[constr_1069]	Compatibility of PortPrototypes of different DataInterfaces in the context of AssemblySwConnectors
[constr_1070]	Compatibility of PortPrototypes of different DataInterfaces in the context of DelegationSwConnectors
[constr_1071]	compatibility of ParameterDataPrototype and VariableDataPrototype
[constr_1072]	Compatibility of ModeSwitchInterfaces in the context of an AssemblySwConnector
[constr_1073]	Compatibility of ModeSwitchInterfaces in the context of an DelegationSwConnector
[constr_1074]	Compatibility of ModeDeclarationGroupPrototypes
[constr_1075]	Compatibility of ModeDeclarationGroups
[constr_1076]	Compatibility of ArgumentDataPrototypes
[constr_1077]	Compatibility of ApplicationErrors
[constr_1078]	Compatibility of ClientServerOperations
[constr_1079]	Compatibility of ClientServerInterfaces in the context of an AssemblySwConnector
[constr_1080]	Compatibility of ClientServerInterfaces in the context of an DelegationSwConnector
[constr_1081]	Compatibility of TriggerInterfaces in the context of an AssemblySwConnector
[constr_1082]	Compatibility of TriggerInterfaces in the context of an DelegationSwConnector

[constr_1083]	Compatibility of Triggers
[constr_1084]	delegation of an provided outer PortPrototype
[constr_1085]	Compatibility in the case of a flat ECU extract
[constr_1086]	SwConnector between two specific PortPrototypes
[constr_1087]	AssemblySwConnector inside CompositionSwComponentType
[constr_1088]	DelegationSwConnector inside CompositionSwComponentType
[constr_1090]	WaitPoint and RunnableEntity
[constr_1091]	RTEEvents that can unblock a WaitPoint
[constr_1092]	ParameterSwComponentType
[constr_1093]	Definition of textual strings
[constr_1094]	Usage of symbol of RunnableEntity
[constr_1095]	Values of nDataSets v.s reliability
[constr_1096]	ModeSwitchEvent and WaitPoint
[constr_1097]	RunnableEntity that has a WaitPoint
[constr_1098]	Mode switch and mode disabling
[constr_1099]	Data type of inter-runnable variables
[constr_1100]	Unconnected RPortPrototype typed by a DataInterface
[constr_1101]	Mode-related communication
[constr_1102]	ApplicationError in the scope of one SwComponentType
[constr_1103]	NonqueuedReceiverComSpec and enableUpdate
[constr_1104]	Trigger sink and trigger source
[constr_1105]	Value of arraySize
[constr_1106]	Structure shall have at least one element
[constr_1107]	Union shall have at least one element
[constr_1108]	Value of ApplicationError.errorCode
[constr_1109]	Mapping of SensorActuatorSwComponentType
[constr_1110]	Value of category in EndToEndDescription
[constr_1111]	Constraints of dataId in PROFILE_01
[constr_1112]	Constraints of dataIdMode in PROFILE_01
[constr_1113]	Existence of attributes in PROFILE_01
[constr_1114]	Constraints of crcOffset in PROFILE_01
[constr_1115]	Constraints of counterOffset in PROFILE_01
[constr_1116]	Constraints of dataLength in PROFILE_01
[constr_1117]	Constraints of maxDeltaCounterInit in PROFILE_01
[constr_1118]	Existence of attributes in PROFILE_02
[constr_1119]	Constraints of dataLength in PROFILE_02
[constr_1120]	Constraints of dataId in PROFILE_02
[constr_1121]	Constraints of maxDeltaCounterInit in PROFILE_02
[constr_1122]	Existence of attributes in PROFILE_03
[constr_1123]	Constraints of dataLength in PROFILE_03
[constr_1124]	Constraints of dataId in PROFILE_03
[constr_1125]	Constraints of maxDeltaCounterInit in PROFILE_03
[constr_1126]	Compatibility of DataConstrs
[constr_2000]	Compatibility of ClientServerOperations
[constr_2001]	Initial value for a specific InterRunnableVariable
[constr_2002]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReadAccess
[constr_2003]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataWriteAccess
[constr_2004]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataSendPoint

[constr_2005]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role dataReceivePointByValue or dataReceivePointByArgument
[constr_2006]	Number of AsynchronousServerCallResultPoint referencing to one AsynchronousServerCallPoint
[constr_2007]	Consistency of typeDefinition attribute
[constr_2009]	Supported kinds of ports of a NvBlockSwComponentType
[constr_2010]	Connections between SwComponentPrototypes oftype NvBlockSwComponentType
[constr_2011]	Connections between SwComponentPrototypes typed by NvBlockSwComponentType and SwComponentPrototypes typed by other AtomicSwComponentTypes
[constr_2012]	Compatibility of ImplementationDataTypes used for RAM and ROM Block
[constr_2013]	Compatibility of ImplementationDataTypes for NvBlockDataMapping
[constr_2014]	Limitation of RoleBasedPortAssignement.role in NvBlockDescriptors
[constr_2015]	Limitation of SwcInternalBehavior of a NvBlockSwComponentType
[constr_2016]	Connections between SwComponentPrototypes oftype ServiceProxySwComponentType
[constr_2017]	Ports of ServiceProxySwComponentTypes
[constr_2018]	Supported remote communication of a ServiceProxySwComponentType
[constr_2019]	ServiceSwComponentType shall have service ports only
[constr_2020]	dataReadAccess can not be used for queued communication
[constr_2021]	WaitPoint referencing a DataReceivedEvent can not be used for non-queued communication
[constr_2022]	Mutually exclusive use of SynchronousServerCallPoints and AsynchronousServerCallPoints
[constr_2023]	Consistency of timeout values
[constr_2024]	enableTakeAddress is restricted to single instantiation
[constr_2025]	Uniqueness of symbol attributes
[constr_2026]	Referenced VariableDataPrototype from AutosarVariableRef of VariableAccess in role writtenLocalVariable and readLocalVariable
[constr_2027]	SwcServiceDependency shall be defined for service ports only
[constr_2028]	staticMemory is restricted to single instantiation
[constr_2029]	shortName of constantMemory and staticMemory
[constr_2030]	AsynchronousServerCallResultPoint combined with WaitPoint shall belong to the same RunnableEntity
[constr_2031]	Period of TimingEvent shall be greater than 0
[constr_2032]	transmissionAcknowledge requires a DataSendCompletedEvent
[constr_2033]	Timeout of DataSendCompletedEvent
[constr_2500]	PortInterfaces shall be of same kind
[constr_2526]	PortInterfaces need to be compatible to the blueprints
[constr_2527]	Blueprints shall live in package of a proper category
[constr_2528]	PortPrototypes shall not refer to blueprints of PortInterfaces
[constr_2529]	Blueprints of ports and interfaces shall be compatible
[constr_2533]	Iteration along output axis is only supported for VALUE and VAL_BLK
[constr_4000]	Local communication of mode switches
[constr_4001]	Content of ModeRequestTypeMap
[constr_4002]	Unambiguous mapping of modes to data types
[constr_4003]	Semantics of ModeSwitchEvent
[constr_4004]	Context of SenderReceiverAnnotation
[constr_4005]	Context of ClientServerAnnotation
[constr_4006]	Context of ParameterPortAnnotation
[constr_4007]	Context of ModePortAnnotation

[constr_4008]	Context of TriggerAnnotation
[constr_4009]	Context of NvDataPortAnnotation
[constr_4010]	Context of DelegatedPortAnnotation
[constr_4011]	ComSpec and ModeSwitchedAckEvent
[constr_4012]	Timeout of ModeSwitchedAckEvent
[constr_4035]	ValueSpecification shall fit into data type
[constr_1001]	Value of dataId shall be unique
[constr_1004]	Mapping of ApplicationDataTypes
[constr_1000]	End-to-end protection is limited to sender/receive communication

Table C.1: Added Constraints in R4.0.1

C.1.3 Deleted Constraints

N/A

C.2 Constraint History of this Document according to AUTOSAR R4.0.2

C.2.1 Changed Constraints in R4.0.2

Please note that constraints listed using an italicized typeface have been deleted in later versions of the document.

Number	Heading
[constr_1007]	Allowed attributes of SwDataDefProps for ApplicationDataTypes
[constr_1061]	Compatibility of data types with category STRUCTURE
[constr_2001]	Initial value for a specific implicitInterRunnableVariable or explicitInterRunnableVariable

Table C.2: Changed Constraints in R4.0.2

C.2.2 Added Constraints in R4.0.2

Please note that constraints listed using an italicized typeface have been deleted in later versions of the document.

Number	Heading
[constr_1127]	ServiceSwComponentType shall not have ServiceNeeds
[constr_1128]	Queue length of OperationPrototypes associated with the same RunnableEntity
[constr_1129]	swImplPolicy
[constr_1130]	swImplPolicy and NonqueuedReceiverComSpec
[constr_1131]	swImplPolicy and NonqueuedSenderComSpec
[constr_1132]	swImplPolicy and NonqueuedSenderComSpec
[constr_1133]	Values shall be unique

[constr_1134]	Allowed structure of TEXTTABLE
[constr_1135]	Limit of `vt` in BITFIELD_TEXTTABLE
[constr_1136]	Compatibility of `introduction` of blueprint and blueprinted element
[constr_1137]	Applicability of `ParameterInterface`
[constr_1138]	`assignedPort` and `DiagEventDebounceMonitorInternal`
[constr_1139]	`assignedPort` of `DiagEventDebounceMonitorInternal` shall refer to an `RPortPrototype`
[constr_2034]	`SwAddrMethod` referenced by `RunnableEntity`s or `BswSchedulableEntity`s
[constr_2035]	`swImplPolicy` for `VariableDataPrototype` in `SenderReceiverInterface`
[constr_2036]	`swImplPolicy` for `VariableDataPrototype` in `NvDataInterface`
[constr_2037]	`swImplPolicy` for `VariableDataPrototype` in therole `ramBlock`
[constr_2038]	`swImplPolicy` for `VariableDataPrototype` in therole `implicitInter-RunnableVariable`
[constr_2039]	`swImplPolicy` for `VariableDataPrototype` in therole `explicitInter-RunnableVariable`
[constr_2040]	`swImplPolicy` for `VariableDataPrototype` in therole `arTypedPerInstance-Memory`
[constr_2041]	`swImplPolicy` for `VariableDataPrototype` in therole `staticMemory`
[constr_2042]	`swImplPolicy` for `ParameterDataPrototoype` in `ParameterInterface`
[constr_2043]	`swImplPolicy` for `ParameterDataPrototoype` in therole `staticMemory`
[constr_2044]	`swImplPolicy` for `ParameterDataPrototoype` in therole `sharedParameter`
[constr_2045]	`swImplPolicy` for `ParameterDataPrototoype` in therole `perInstanceParam-eter`
[constr_2046]	`swImplPolicy` for `ParameterDataPrototoype` in therole `constantMemory`
[constr_2047]	`swImplPolicy` for `ArgumentDataPrototype`
[constr_2048]	`swImplPolicy` for `SwServiceArg`
[constr_2535]	Target of an `autosarParameter` in `AutosarParameterRef` shall refer to a param-eter
[constr_2536]	Target of an `autosarVariable` in `AutosarVariableRef` shall refer to a variable

Table C.3: Added Constraints in R4.0.2

C.2.3 Deleted Constraints in R4.0.2

Number	Heading
[constr_1015]	Prioritization of `SwDatDefProps`
[constr_1099]	Data type of inter-runnable variables

Table C.4: Deleted Constraints in R4.0.2

C.3 Constraint History of this Document according to AUTOSAR R4.0.3

C.3.1 Changed Constraints in R4.0.3

Please note that constraints listed using an italicized typeface have been deleted in later versions of the document.

Number	Heading

[constr_1006]	applicable data categories
[constr_1009]	SwDataDefProps applicable to ImplementationDataTypes[1]
[constr_1014]	Supported value encodings for SwBaseType
[constr_1015]	Prioritization of SwDatDefProps
[constr_1043]	PortInterface vs. ComSpec
[constr_1051]	Compatibility of SwDataDefProps
[constr_1053]	Compatibility of PhysicalDimensions
[constr_1063]	Compatibility of data types with category BOOLEAN
[constr_1110]	Value of category in EndToEndDescription
[constr_1113]	Existence of attributes in PROFILE_01
[constr_1118]	Existence of attributes in PROFILE_02
[constr_1134]	Allowed structure of TEXTTABLE
[constr_2000]	Compatibility of ClientServerOperations
[constr_2027]	SwcServiceDependency shall be defined for service ports only

Table C.5: Changed Constraints in R4.0.3

C.3.2 Added Constraints in R4.0.3

Number	Heading
[constr_1140]	Combination of initValue with the attribute handleInvalid
[constr_1141]	Applicability of the scope attribute
[constr_1142]	category of compuMethod shall not be extended
[constr_1143]	category of AutosarDataType shall not be extended
[constr_1144]	SensorActuatorSwComponentType, EcuAbstractionSwComponentType, and ComplexDeviceDriverSwComponentType may only reference a HwType
[constr_1145]	Finding the symbol for the representation of a CompuScale in C code
[constr_1146]	Applicability of a symbol for a CompuScale in C code
[constr_1147]	Standardized values for the attribute category of meta-class PortGroup
[constr_1148]	PortInterfaces of PortPrototypes used to connect to NvBlockSwComponentTypes
[constr_1149]	PortPrototypes used for NV data management
[constr_1150]	Usage of valueType for PortDefinedArgumentValue
[constr_1151]	Applicability of PortInterfaceMapping
[constr_1151]	category of ApplicationArrayElement and AutosarDataType referenced in the role type shall be kept in sync
[constr_1153]	Applicability of compatibility requirements for CompuScales
[constr_1154]	Compatibility of CompuScales for sender-receiver communication and similar use cases
[constr_1155]	Compatibility of CompuScales for client-server communication
[constr_1156]	Relevance of "names" of CompuScales
[constr_1157]	Applicability of constraints of CompuScales
[constr_1158]	Applicable categorys for attribute compuMethod
[constr_1159]	Consistency of VariableAndParameterInterfaceMapping with respect to the referenced DataInterfaces
[constr_1160]	Size of “compound primitive” is variant

[1] Technically, the text of the constraint did not change. However, as the constraint is mainly saying that table 5.17 has the characteristics of a constraint and on the same time the contents of table 5.17 changed the constraint can also be considered changed.

[constr_1161]	Applicability of the index attribute of Ref
[constr_1162]	Compatibility of SwRecordLayouts
[constr_1163]	Compatibility of CompuMethods
[constr_1164]	Number of arguments owned by a RunnableEntity
[constr_1165]	Applicability of RunnableEntityArgument
[constr_1166]	Restrictions of ModeRequestTypeMap
[constr_1167]	ImplementationDataTypes used as ModeRequestTypeMap.implementationDataType
[constr_1168]	Compatibility of ImplementationDataTypes used used in the ModeRequestTypeMap
[constr_1169]	Allowed values for Trigger.swImplPolicy
[constr_1170]	Interpretation of attribute maxDeltaCounterInit of EndToEndDecription
[constr_1171]	Interpretation of attribute maxDeltaCounterInit of EndToEndDecription
[constr_1172]	Allowed values of SwCalibrationAccessEnum for ModeDeclarationGroupPrototype
[constr_1173]	Applicability of AutosarParameterRef referencing a VariableDataPrototype
[constr_1174]	PortInterfaces used in the context of CompositionSwComponentTypes cannot refer to AUTOSAR services
[constr_1175]	Depending on its category, CompuMethod shall refer to a unit
[constr_1176]	Compatibility of CompuScales of category LINEAR and RAT_FUNC
[constr_1177]	Allowed category for SwPointerTargetProps
[constr_1178]	Existence of attributes of SwDataDefProps in the context of ImplementationDataType
[constr_1179]	Existence of ModeDeclaration.value within a ModeDeclarationGroup
[constr_1180]	Existence of ModeDeclarationGroup.onTransitionValue
[constr_1181]	Numerical values used in ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue
[constr_1182]	Allowed values for InternalTriggeringPoint.swImplPolicy
[constr_1183]	EndToEndProtectionVariableDataPrototypes aggregated by EndToEndProtection
[constr_1184]	Consistency of rootDataPrototype and base in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1185]	Consistency of data types in the context of ApplicationCompositeElementInPortInterfaceInstanceRef
[constr_1186]	Consistency of data types in the context of ArVariableInImplementationDataInstanceRef
[constr_1187]	Compatibility of VariableDataPrototypes or ParameterDataPrototypes typed by composite data types
[constr_1188]	Existence of externalReplacement
[constr_1189]	Allowed targets of externalReplacement
[constr_1190]	Only one mapping for composite to primitive use case
[constr_2000]	Compatibility of ClientServerOperations triggering the same RunnableEntity
[constr_2049]	Different ModeDeclarationGroups shall have different shortNames.
[constr_2050]	Mandatory information of a SwAxisCont
[constr_2051]	Mandatory information of a SwValueCont
[constr_2052]	Values of swArraySize and the number of values provided by swValuesPhys shall be consistent.
[constr_2053]	Consistency between role IUMPRNumeratorand ConnectionType
[constr_2544]	Limits need to be consistent
[constr_2545]	invalidValue shall fit in the specified range
[constr_2548]	Data constraint of value axis shall match
[constr_2549]	Units of input axis shall be consistent
[constr_2550]	Units of value axis shall be consistent

[constr_2551]	SwCalprmAxis.baseType shall be ignored
[constr_2561]	Application of DataConstrRule.constrLevel

Table C.6: Added Constraints in R4.0.3

C.3.3 Added Specification Items in R4.0.3

Number	Heading
[TPS_SWCT_1000]	Usage of attribute symbol of the symbolProps
[TPS_SWCT_1001]	Prefix symbols generated for the RunnableEntity
[TPS_SWCT_1002]	SwComponentTypes may only interact by means of their PortPrototypes
[TPS_SWCT_1003]	Inconsistencies regarding the value of serviceKind and the actual implementation of the PortInterface
[TPS_SWCT_1004]	Default value if serviceKind is not defined
[TPS_SWCT_1005]	Usage of SwcServiceDependencys for vendor-specific services
[TPS_SWCT_1006]	arraySize of ImplementationDataType shall be used to define the size of the array
[TPS_SWCT_1007]	Semantics of array index
[TPS_SWCT_1008]	Definition of positive integer values that are directly taken over by the RTE generator for creating the programmatic representations of the ModeDeclaration
[TPS_SWCT_1009]	The numerical values used to define the values of ModeDeclaration.value and ModeDeclarationGroup.onTransitionValue can be arbitrarily defined
[TPS_SWCT_1010]	categorys for the definition of a ModeDeclarationGroup
[TPS_SWCT_1011]	Default category of a ModeDeclarationGroup
[TPS_SWCT_1013]	AtomicSwComponentType needs to keep the ECU alive or needs to execute operations before the ECU is shut down
[TPS_SWCT_1014]	AtomicSwComponentType wants to select a shutdown target
[TPS_SWCT_1015]	AtomicSwComponentType wants to select a boot target
[TPS_SWCT_1016]	AtomicSwComponentType wants to select a shutdown target
[TPS_SWCT_1017]	AtomicSwComponentType wants to select a boot target
[TPS_SWCT_1018]	AtomicSwComponentType wants to use an alarm clock
[TPS_SWCT_1019]	AtomicSwComponentType reads the current ComM mode
[TPS_SWCT_1020]	AtomicSwComponentType requests a ComM mode. It may also check later whether the requested ComM mode has become effective
[TPS_SWCT_1021]	AtomicSwComponentType acts as a mode manager that influences the ECU state
[TPS_SWCT_1022]	Queued processing of internal trigger
[TPS_SWCT_1023]	Mapping of elements of composite data types
[TPS_SWCT_1024]	Combination of ApplicationCompositeDataType and nested ImplementationDataType
[TPS_SWCT_1025]	The role of PortPrototypes in the AUTOSAR architecture
[TPS_SWCT_1026]	The role of PortInterfaces in the AUTOSAR architecture
[TPS_SWCT_1027]	Different flavors of PortInterfaces
[TPS_SWCT_1028]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_1029]	AtomicSwComponentType implements a Diagnostic Monitor
[TPS_SWCT_1030]	RunnableEntity
[TPS_SWCT_1031]	ExclusiveArea
[TPS_SWCT_1032]	CompositionSwComponentType
[TPS_SWCT_1033]	Nested definition of CompositionSwComponentTypes
[TPS_SWCT_1034]	CompositionSwComponentTypes do not have any binary footprint
[TPS_SWCT_1035]	CompositionSwComponentType aggregates SwComponentPrototypes
[TPS_SWCT_1036]	SwComponentPrototype implements a specific role

[TPS_SWCT_1037]	arbitrary numbers of SwComponentPrototypes can be created
[TPS_SWCT_1038]	Support for Variant Handling in the in Software Component Template
[TPS_SWCT_1039]	Purpose of variant handling
[TPS_SWCT_1040]	SwConnector exists depending on a *PostBuild* condition
[TPS_SWCT_1041]	API functions of not existing SwConnector are still part of the software-component's implementation
[TPS_SWCT_1042]	Four types of locations in the meta-model which may exhibit variability
[TPS_SWCT_1043]	ApplicationSwComponentTypes are independent from actual ECU Hardware
[TPS_SWCT_1044]	ServiceNeeds
[TPS_SWCT_1045]	Actual values of ECU configuration parameters fulfill the requirements given by the ServiceNeeds
[TPS_SWCT_1046]	ServiceNeeds are defined in the scope of the SwcInternalBehavior
[TPS_SWCT_1047]	Reference from the software representation of a sensor/actuator to the actual hardware element
[TPS_SWCT_1048]	SensorActuatorSwComponentType may use the I/O hardware abstraction directly
[TPS_SWCT_1049]	Two ways to use the ExclusiveAreas
[TPS_SWCT_1050]	RunnableEntity always runs inside an ExclusiveArea
[TPS_SWCT_1051]	RunnableEntity explicitly enters and leaves a specific ExclusiveArea
[TPS_SWCT_1052]	Inter-runnable variable
[TPS_SWCT_1053]	Relationship of interchanged data with RunnableEntitys
[TPS_SWCT_1054]	Semantics of the explicitInterRunnableVariable
[TPS_SWCT_1055]	Semantics of implicitInterRunnableVariable
[TPS_SWCT_1056]	Physical dimension
[TPS_SWCT_1057]	Unit references one physical dimension
[TPS_SWCT_1058]	UnitGroup
[TPS_SWCT_1059]	Exponent for each of the seven fundamental dimensions
[TPS_SWCT_1060]	Units can be grouped with the help of UnitGroup
[TPS_SWCT_1061]	Conversion of units
[TPS_SWCT_1062]	Documentation of software-components
[TPS_SWCT_1063]	PortGroup
[TPS_SWCT_1064]	PortGroups have to be defined on the VFB level
[TPS_SWCT_1065]	PortPrototype may belong to more than one PortGroups
[TPS_SWCT_1066]	ModeDeclaration
[TPS_SWCT_1067]	Initial mode
[TPS_SWCT_1068]	Units can be grouped with the help of UnitGroup
[TPS_SWCT_1069]	DataInterface is defined as abstract base class
[TPS_SWCT_1070]	PortInterface acts as a *type* for a PortPrototype
[TPS_SWCT_1071]	ModeDeclaration
[TPS_SWCT_1072]	ApplicationDataType and Implementation-DataTypeg««««««««««««««««««««aa
[TPS_SWCT_1073]	Composite ApplicationDataType
[TPS_SWCT_1074]	Composite ImplementationDataType
[TPS_SWCT_1075]	SwcInternalBehavior
[TPS_SWCT_1076]	Number of elements of a specific ApplicationArrayDataType might vary at run-time
[TPS_SWCT_1077]	Configure the response to mode changes
[TPS_SWCT_1078]	Configurable array size
[TPS_SWCT_1079]	SwConnector
[TPS_SWCT_1080]	Implications of being a delegation port
[TPS_SWCT_1081]	DelegationSwConnector
[TPS_SWCT_1082]	AssemblySwConnector

[TPS_SWCT_1083]	DelegationSwConnector
[TPS_SWCT_1084]	Outer PortPrototype is referenced by multiple DelegationSwConnectors
[TPS_SWCT_1085]	Variation on the behavior level
[TPS_SWCT_1086]	Request mode change
[TPS_SWCT_1087]	Propagation of mode information
[TPS_SWCT_1088]	ComSpecs defined by CompositionSwComponents
[TPS_SWCT_1089]	end-to-end communication protection
[TPS_SWCT_1090]	EndToEndProtection
[TPS_SWCT_1091]	Two cases for end-to-end protection
[TPS_SWCT_1092]	EndToEndProtectionSet
[TPS_SWCT_1093]	Definition of end-to-end protection is splitable
[TPS_SWCT_1094]	category of EndToEndDescription
[TPS_SWCT_1095]	category set to NONE
[TPS_SWCT_1096]	PortGroup
[TPS_SWCT_1097]	CompositionSwComponentType cannot have RunnableEntitys
[TPS_SWCT_1098]	Only AtomicSwComponentType can have RunnableEntitys
[TPS_SWCT_1099]	PortInterfaceMapping
[TPS_SWCT_1100]	Precedence of PortInterfaceMapping
[TPS_SWCT_1101]	Unmapped elements of PortInterfaces
[TPS_SWCT_1102]	VariableAndParameterInterfaceMapping
[TPS_SWCT_1103]	Mapping between different kinds of PortInterfaces
[TPS_SWCT_1104]	Possible mappings are restricted by the SwImplPolicy
[TPS_SWCT_1105]	ClientServerInterfaceMapping
[TPS_SWCT_1106]	ClientServerOperation
[TPS_SWCT_1107]	swMinAxisPoints and swMaxAxisPoints represent variation points
[TPS_SWCT_1109]	Adding the SwcInternalBehavior in a later process step
[TPS_SWCT_1110]	Symbolic name of a software-component
[TPS_SWCT_1111]	PortPrototypes need an additional model artifact, the PortInterface
[TPS_SWCT_1112]	PortPrototypes are either *require*- or *provide*-ports.
[TPS_SWCT_1113]	Connecting two PortPrototypes
[TPS_SWCT_1114]	SenderReceiverInterface
[TPS_SWCT_1115]	InvalidationPolicy
[TPS_SWCT_1116]	swImplPolicy
[TPS_SWCT_1117]	Communication patterns for sender-receiver communication
[TPS_SWCT_1118]	ClientServerInterface
[TPS_SWCT_1119]	Direction of ArgumentDataPrototypes
[TPS_SWCT_1120]	Client needs to provide ArgumentDataPrototypes
[TPS_SWCT_1121]	Pass correct data type
[TPS_SWCT_1122]	Synchronous call of ClientServerOperation
[TPS_SWCT_1123]	No default values for ArgumentDataPrototypes
[TPS_SWCT_1124]	Definition of ArgumentDataPrototypes within the context of a ClientServerOperation is ordered
[TPS_SWCT_1125]	serverArgumentImplPolicy
[TPS_SWCT_1126]	Access to partial networking via BswM
[TPS_SWCT_1127]	Byte arrary with variable size
[TPS_SWCT_1128]	RecordLayout needed for special cases
[TPS_SWCT_1129]	Express diagnostic capabilities
[TPS_SWCT_1130]	Measurement and calibration access to model elements is defined by swCalibrationAccess
[TPS_SWCT_1131]	AtomicSwComponentType accepts a request to restart an entire function
[TPS_SWCT_1132]	AtomicSwComponentType provides information about operating cycles
[TPS_SWCT_1133]	AtomicSwComponentType provides information about aging cycles
[TPS_SWCT_1134]	AtomicSwComponentType enables storage of DTCs in general

[TPS_SWCT_1135]	AtomicSwComponentType enables storage of subsequent DTCs
[TPS_SWCT_1136]	AtomicSwComponentType retrieves information from the fault storage
[TPS_SWCT_1137]	Dem provides information that the fault storage overflows
[TPS_SWCT_1138]	AtomicSwComponentType suppresses the storage of DTCs within the Dem
[TPS_SWCT_1139]	AtomicSwComponentType informs the Dem that the PTO is active
[TPS_SWCT_1140]	AtomicSwComponentType needs information about specific DTC without being a diagnostic monitor
[TPS_SWCT_1141]	AtomicSwComponentType may have RPortPrototypes typed by an NvDataInterface
[TPS_SWCT_1142]	non-volatile data are provided by a specialized AtomicSwComponentType
[TPS_SWCT_1143]	Non-volatile data represented by an *NvBlockComponent* can be read and written
[TPS_SWCT_1144]	NvBlockDescriptor specifies the properties of exactly one *NvBlock*
[TPS_SWCT_1145]	RAM Block and the ROM Block are described by a VariableDataPrototype and a ParameterDataProrotype
[TPS_SWCT_1146]	ROM Block is optional
[TPS_SWCT_1147]	No ROM Block is configured
[TPS_SWCT_1148]	NvBlockDataMapping
[TPS_SWCT_1149]	RoleBasedPortAssignment of NvBlockDescriptor
[TPS_SWCT_1150]	InternalBehavior of a NvBlockSwComponentType
[TPS_SWCT_1151]	RunnableEntitys do not have further attributes
[TPS_SWCT_1152]	InternalBehavior does not have further attributes
[TPS_SWCT_1153]	IncludedModeDeclarationGroupSet
[TPS_SWCT_1154]	Attribute prefix of IncludedModeDeclarationGroupSet
[TPS_SWCT_1155]	IncludedDataTypeSet
[TPS_SWCT_1156]	Required if the AutosarDataType is not used for any DataPrototype
[TPS_SWCT_1157]	Attribute literalPrefix of IncludedDataTypeSet
[TPS_SWCT_1158]	Three cases for PortInterfaceMapping
[TPS_SWCT_1159]	Mapping is described separately from the SwConnector as reusable ARElement
[TPS_SWCT_1160]	Validity of ModeInterfaceMapping
[TPS_SWCT_1161]	Linear conversion factor can be calculated
[TPS_SWCT_1162]	Conditional existence of TextTableMapping
[TPS_SWCT_1163]	Conversion from firstValue to secondValue
[TPS_SWCT_1164]	Conversion from secondValue to firstValue
[TPS_SWCT_1165]	Invertible mapping
[TPS_SWCT_1166]	Non-invertible mapping
[TPS_SWCT_1167]	Validity of ModeInterfaceMapping
[TPS_SWCT_1168]	Linear conversion factor can be calculated
[TPS_SWCT_1169]	Support for partial networking
[TPS_SWCT_1170]	Purpose of Virtual Function Cluster
[TPS_SWCT_1171]	Purpose of a control port
[TPS_SWCT_1172]	Requesting and releasing partial networks
[TPS_SWCT_1173]	Actively query the status of a partial network
[TPS_SWCT_1174]	Status port shall **not** become a member of the PortGroup
[TPS_SWCT_1175]	Actively query the status of a partial network
[TPS_SWCT_1176]	last-is-best semantics for sender-receiver communication
[TPS_SWCT_1177]	Assignment of constant values
[TPS_SWCT_1178]	Specialized subclasses of ValueSpecification
[TPS_SWCT_1179]	Compound primitive type
[TPS_SWCT_1180]	Maximum possible size of compound primitive type
[TPS_SWCT_1181]	Bound model specifies a primitive which is smaller than the maximum defined by the range of the involved SwSystemconst

[TPS_SWCT_1182]	Conceptual levels for the definition of initial values
[TPS_SWCT_1183]	Actual value of an initValue shall be interpreted according to the AutosarDataType
[TPS_SWCT_1184]	ApplicationPrimitiveDataTypes with category VALUE
[TPS_SWCT_1185]	initValues for compound primitive types
[TPS_SWCT_1186]	ConstantSpecificationMapping
[TPS_SWCT_1187]	ConstantSpecificationMappingSet referenced by the InternalBehavior
[TPS_SWCT_1188]	Definition of calibration data sets through RTE-generator and compiler
[TPS_SWCT_1190]	ModeRequestTypeMap
[TPS_SWCT_1192]	Meta-classes that have an association to a DataTypeMappingSet
[TPS_SWCT_1193]	Mappings between application and implementation types do not necessarily have to form a 1:1 relation
[TPS_SWCT_1194]	Symbolic name of an ImplementationDataType
[TPS_SWCT_1195]	Mapping of composite element to primitive DataPrototype
[TPS_SWCT_1196]	Semantics of an external trigger event communication
[TPS_SWCT_1197]	TriggerInterface
[TPS_SWCT_1198]	Period for periodic triggering
[TPS_SWCT_1199]	Queued processing of Triggers
[TPS_SWCT_1200]	ModeDeclarationGroupPrototype per ModeSwitchInterface
[TPS_SWCT_1201]	CompositionSwComponentType requires and provides the modes that are required or provided by its contained SwComponentPrototypes
[TPS_SWCT_1202]	ApplicationDataType defines a subset of the values used in the ModeDeclarationGroup
[TPS_SWCT_1203]	PortPrototype may own port annotations
[TPS_SWCT_1204]	GeneralAnnotation
[TPS_SWCT_1205]	Typical annotations for sender/receiver communication
[TPS_SWCT_1206]	Min and Max annotations are valid for a certain amount of time
[TPS_SWCT_1207]	VariableDataPrototypes use the same application-level SenderReceiverAnnotation
[TPS_SWCT_1208]	ClientServerAnnotation
[TPS_SWCT_1209]	ClientServerAnnotation
[TPS_SWCT_1210]	IoHwAbstractionServerAnnotation
[TPS_SWCT_1211]	Assign several annotations to ArgumentDataPrototype
[TPS_SWCT_1212]	ParameterPortAnnotation
[TPS_SWCT_1213]	ModePortAnnotation
[TPS_SWCT_1214]	TriggerPortAnnotation
[TPS_SWCT_1215]	NvDataPortAnnotation
[TPS_SWCT_1216]	DelegatedPortAnnotation
[TPS_SWCT_1217]	Semantics of DelegatedPortAnnotation.signalFan
[TPS_SWCT_1218]	Big picture of ComSpec
[TPS_SWCT_1219]	ComSpec for queued and non-queued sender-receiver communication
[TPS_SWCT_1220]	initValue defines an initial value that shall be taken if the corresponding dataElement has not yet been received
[TPS_SWCT_1221]	DataFilter
[TPS_SWCT_1222]	Applicability of DataFilter
[TPS_SWCT_1223]	networkRepresentation defines how a specific dataElement is represented on a communication bus
[TPS_SWCT_1224]	CompuMethods of dataElement and the networkRepresentation are used for conversion purposes
[TPS_SWCT_1225]	RunnableEntity implements the functionality of two or more ClientServerOperations

[TPS_SWCT_1226]	initValue on the level of a ComSpec is relevant for connections to the corresponding PortPrototype
[TPS_SWCT_1227]	Unconnected RPortPrototype typed by NvDataInterface
[TPS_SWCT_1228]	NvProvideComSpec
[TPS_SWCT_1229]	Three different levels of abstraction regarding the definition of data types
[TPS_SWCT_1230]	Application Data Level
[TPS_SWCT_1231]	Application level may impose strong requirements on the design of the corresponding implementation level
[TPS_SWCT_1232]	Implementation Data Level
[TPS_SWCT_1233]	Use case for the Implementation Data Level
[TPS_SWCT_1234]	Base Level
[TPS_SWCT_1235]	Mapping of data defined on the *Application* level to the *Implementation* and *Base Type* level
[TPS_SWCT_1236]	Big picture of data types
[TPS_SWCT_1237]	SwDataDefProps
[TPS_SWCT_1238]	Attribute category used in the context of AutosarDataType
[TPS_SWCT_1239]	default value for attribute category used in the context of AutosarDataType
[TPS_SWCT_1240]	Subclasses of ApplicationDataType
[TPS_SWCT_1241]	Applicable categorys for subclasses ApplicationDataType
[TPS_SWCT_1242]	category characterizes the nature of a data type on application level
[TPS_SWCT_1243]	Definition of enumeration types
[TPS_SWCT_1244]	Data types for calibration parameters are also described as primitive types
[TPS_SWCT_1245]	SwDataDefProps control the structure of calibration parameters
[TPS_SWCT_1246]	RecordLayout may be required for A2L generation
[TPS_SWCT_1247]	ApplicationArrayDataType and ApplicationRecordDataType
[TPS_SWCT_1248]	Nested definition of ImplementationDataType
[TPS_SWCT_1249]	ApplicationRecordDataType
[TPS_SWCT_1250]	ImplementationDataType has been introduced to optimize the formal support for data type handling on the implementation level
[TPS_SWCT_1251]	Limited set of values for category are applicable for Implementation-DataType
[TPS_SWCT_1252]	ImplementationDataType can express concepts not available on application level
[TPS_SWCT_1253]	Rules applies for the usage of the attribute Implementation-DataType.typeEmitter
[TPS_SWCT_1254]	ImplementationDataType with array semantics
[TPS_SWCT_1255]	Indicate whether the array is supposed to have a fixed size or whether the actual size might change during run-time
[TPS_SWCT_1256]	Definition of multi-dimensional array data types
[TPS_SWCT_1257]	ImplementationDataType or the aggregated Implementation-DataTypeElements do not form closed sets
[TPS_SWCT_1258]	Definition of a pointer to data
[TPS_SWCT_1259]	Definition of a pointer to a function
[TPS_SWCT_1260]	SwBaseType
[TPS_SWCT_1261]	Further use cases for SwBaseType
[TPS_SWCT_1262]	memAlignment and byteOrder are platform specific
[TPS_SWCT_1263]	Further use cases for SwBaseType
[TPS_SWCT_1264]	Data prototypes implement a role of a data type
[TPS_SWCT_1265]	DataPrototype aggregates an own set of SwDataDefProps
[TPS_SWCT_1266]	Three non-abstract classes derived from AutosarDataPrototype
[TPS_SWCT_1267]	DataPrototype can be aggregated in different roles
[TPS_SWCT_1268]	Definition of initValue for a VariableDataPrototype or a Parameter-DataPrototype

[TPS_SWCT_1269]	In PortInterfaces, initial values defined for DataPrototypes are ignored
[TPS_SWCT_1270]	AutosarVariableRef
[TPS_SWCT_1271]	AutosarParameterRef
[TPS_SWCT_1272]	Semantics of swComparisonVariable
[TPS_SWCT_1273]	Precedence rules for the application of SwDataDefProps
[TPS_SWCT_1274]	SwDataDefProps used to support calibration and measurement
[TPS_SWCT_1275]	values of the attribute swImplPolicy are restricted depending on the context
[TPS_SWCT_1276]	Computation methods
[TPS_SWCT_1277]	Computation methods are used for the conversion of *internal* values into their *physical* representation and vice versa
[TPS_SWCT_1278]	CompuMethods can also be used to assign symbolic names to internal values
[TPS_SWCT_1279]	Preferred conversion direction depends on the use case
[TPS_SWCT_1280]	CompuMethod applied to values outside of its limits
[TPS_SWCT_1281]	Physical unit associated with a data type
[TPS_SWCT_1282]	Number of intervals in which a given conversion applies
[TPS_SWCT_1283]	Rational function
[TPS_SWCT_1284]	CompuScale might require a representation in the generated RTE C code
[TPS_SWCT_1285]	Physical dimension
[TPS_SWCT_1286]	DataConstr
[TPS_SWCT_1287]	Standard limits and extended limits in the ASAM-MCD2 (ASAP2) specification
[TPS_SWCT_1288]	Interpretation of PhysConstrs and InternalConstrs by tools
[TPS_SWCT_1289]	Semantics of Limit
[TPS_SWCT_1290]	SwAddrMethod
[TPS_SWCT_1291]	Association of MemorySection with SwAddrMethod
[TPS_SWCT_1292]	Usage of SwAddrMethod in the context of a DataPrototype
[TPS_SWCT_1293]	RTE Generator has to derive the Memory Allocation Keyword
[TPS_SWCT_1294]	Missing SwDataDefProps.swAddrMethod
[TPS_SWCT_1295]	RecordLayout
[TPS_SWCT_1296]	Different approaches of ASAM MCD-2MC and AUTOSAR with respect to RecordLayout
[TPS_SWCT_1297]	Compliance of ApplicationDataTypes or ImplementationDataTypes to swDataDefProps
[TPS_SWCT_1298]	Computing SwRecordLayout from ImplementationDataTypes is not possible
[TPS_SWCT_1299]	Relation of swRecordLayoutGroup to subElement
[TPS_SWCT_1300]	Relationship between record layouts and interpolation routines
[TPS_SWCT_1301]	Importance of initial values
[TPS_SWCT_1302]	Semantics of minimumStartInterval
[TPS_SWCT_1303]	Conditions for a transition from suspended to to be started
[TPS_SWCT_1305]	RunnableEntity as one that cannot be invoked concurrently
[TPS_SWCT_1307]	supportsMultipleInstantiation vs. canBeInvokedConcurrently
[TPS_SWCT_1308]	Combination of supportsMultipleInstantiation=FALSE and canBeInvokedConcurrently=FALSE
[TPS_SWCT_1309]	signature of a RunnableEntity depends on the connected RTEEvent
[TPS_SWCT_1310]	Categories of RunnableEntitys
[TPS_SWCT_1311]	Name of an operation argument
[TPS_SWCT_1312]	RunnableEntity has a mapping to BswModuleEntry
[TPS_SWCT_1313]	Conditions for a transition from suspended to to be started
[TPS_SWCT_1314]	RTEEvent
[TPS_SWCT_1315]	Interaction of RunnableEntity with RTEEvent
[TPS_SWCT_1317]	RTE triggers RunnableEntity in response to occurring RTEEvent
[TPS_SWCT_1318]	RunnableEntity and WaitPoint
[TPS_SWCT_1319]	RTEEvent can be used to trigger WaitPoints in different RunnableEntitys

[TPS_SWCT_1320]	Mode switches need to be completed in finite time
[TPS_SWCT_1321]	Communication among RunnableEntitys
[TPS_SWCT_1322]	Interaction patterns for the application of the sender-receiver paradigm
[TPS_SWCT_1323]	Read and write access to a dataElement
[TPS_SWCT_1324]	Mode switches need to be completed in finite time
[TPS_SWCT_1325]	Read and write access is only applicable for RunnableEntitys of category 1
[TPS_SWCT_1326]	Constrain the scope of a specific communication
[TPS_SWCT_1327]	RTE generator can omit the creation of checks at run-time
[TPS_SWCT_1328]	Default value of attribute scope
[TPS_SWCT_1329]	Access to specific data is implemented by means of aggregating the meta-class VariableAccess in specific roles
[TPS_SWCT_1330]	RunnableEntity can also have dataSendPoints
[TPS_SWCT_1331]	dataWriteAccess vs. dataSendPoint
[TPS_SWCT_1332]	dataReceivePointByValue vs. dataReceivePointByArgument
[TPS_SWCT_1333]	dataReceivePointByValue/dataReceivePointByArgument vs. dataReadAccess
[TPS_SWCT_1334]	RunnableEntitys of category 1 may have dataReceivePoints
[TPS_SWCT_1335]	Combine dataReceivePointByValue or dataReceivePointByArgument with a WaitPoint
[TPS_SWCT_1336]	dataSendPoint also allows for the definition of a DataSendCompletedEvent
[TPS_SWCT_1337]	DataReceivedEvent
[TPS_SWCT_1338]	DataReceiveErrorEvent
[TPS_SWCT_1339]	RTE activates RunnableEntity in response to DataReceiveErrorEvent
[TPS_SWCT_1340]	DataReceiveErrorEvent cannot be combined with a WaitPoint
[TPS_SWCT_1341]	DataReceiveErrorEvent is directly associated with the corresponding VariableDataPrototype
[TPS_SWCT_1341]	DataReceiveErrorEvent is directly associated with the corresponding VariableDataPrototype
[TPS_SWCT_1342]	Invocation of a server operation
[TPS_SWCT_1343]	Synchronous vs. asynchronous invocation
[TPS_SWCT_1344]	Consistency of values of timeout
[TPS_SWCT_1345]	Synchronous operation invocation
[TPS_SWCT_1346]	Asynchronous operation invocation
[TPS_SWCT_1347]	Blocking access to operation result in an asynchronous operation invocation
[TPS_SWCT_1348]	Trigger source
[TPS_SWCT_1350]	Calibration Parameters shared among several SwComponentTypes
[TPS_SWCT_1351]	Access to a ParameterDataPrototype
[TPS_SWCT_1352]	Requested mode is just sent and received as an ordinary data value
[TPS_SWCT_1353]	RunnableEntitys react on a mode request via a corresponding RTEEvent
[TPS_SWCT_1354]	PortAPIOption
[TPS_SWCT_1355]	enableTakeAddress = TRUE
[TPS_SWCT_1356]	indirectAPI option switches the generation of the RTE's indirect API functionality
[TPS_SWCT_1357]	Definition of implicit values that are passed by the RTE to the server's entry point
[TPS_SWCT_1358]	Values are hidden from the client components
[TPS_SWCT_1359]	Private memory per instance
[TPS_SWCT_1360]	Arbitrary number of per-instance memory blocks
[TPS_SWCT_1361]	attribute supportsMultipleInstantiation == FALSE
[TPS_SWCT_1362]	Initialization of PerInstanceMemory
[TPS_SWCT_1363]	PerInstanceMemory typed by 'C' Data Types
[TPS_SWCT_1364]	Initial value of a PerInstanceMemory typed by 'C' Data Types

[TPS_SWCT_1365]	PerInstanceMemory typed by AUTOSAR Data Types
[TPS_SWCT_1366]	Initial value of a PerInstanceMemory typed by AUTOSAR Data Types
[TPS_SWCT_1367]	Typed by AUTOSAR data type vs. typed by C data type
[TPS_SWCT_1368]	Describe static and constant memory
[TPS_SWCT_1369]	Static and constant memory is not instantiated by the RTE
[TPS_SWCT_1370]	VariationPointProxy
[TPS_SWCT_1371]	VariationPointProxy vs. VariationPoint
[TPS_SWCT_1372]	bindingTime = PreCompileTime
[TPS_SWCT_1373]	RTE generator shall evaluate the SwSystemconstDependantFormula
[TPS_SWCT_1374]	Implementation of AutosarVariableRef
[TPS_SWCT_1375]	Implementation of AutosarVariableRef
[TPS_SWCT_1376]	Software-components need to be capable of reacting to state changes
[TPS_SWCT_1377]	Two mechanisms to define how SwcInternalBehavior should interact with the mode management
[TPS_SWCT_1378]	AtomicSwComponentType can define an SwcModeSwitchEvent to execute RunnableEntity
[TPS_SWCT_1379]	AtomicSwComponentType can indicate whether an RTEEvent that starts an associated RunnableEntity is disabled in a certain mode
[TPS_SWCT_1380]	ModeSwitchPoint
[TPS_SWCT_1381]	Read the currently active mode
[TPS_SWCT_1382]	Mode switch requests are handled asynchronously by the RTE
[TPS_SWCT_1383]	ModeSwitchPoint
[TPS_SWCT_1384]	Execution of initialization code for software-components
[TPS_SWCT_1385]	Execution of initialization code for software-components
[TPS_SWCT_1386]	Initialization by mode management
[TPS_SWCT_1387]	Initial modes of AtomicSwComponentTypes are defined by the initialMode
[TPS_SWCT_1388]	Initial modes of AtomicSwComponentTypes are defined by the initialMode
[TPS_SWCT_1389]	I/O Hardware Abstraction interfaces MCAL drivers
[TPS_SWCT_1390]	I/O Hardware Abstraction might have sub-structures
[TPS_SWCT_1391]	I/O Hardware Abstraction abstracts from the location of peripheral I/O devices
[TPS_SWCT_1392]	Mapping between the EcuAbstractionSwComponentType and the corresponding BswModuleDescription
[TPS_SWCT_1393]	Complex Driver
[TPS_SWCT_1394]	Complex Driver is represented by the ComplexDeviceDriverSwComponentType
[TPS_SWCT_1395]	ComplexDeviceDriverSwComponentType has dependencies to ECU Hardware
[TPS_SWCT_1396]	Mapping between the ComplexDeviceDriverSwComponentType and the corresponding BswModuleDescription
[TPS_SWCT_1397]	Hybrid concept between Basic Software Modules and a SwComponentType
[TPS_SWCT_1398]	Impact of AUTOSAR services on the methodology
[TPS_SWCT_1399]	Dependency is modeled by aggregating required and provided PortPrototypes
[TPS_SWCT_1400]	PortInterface selected from the set of standardized Service Interfaces
[TPS_SWCT_1401]	Form a top-level RootSwCompositionPrototype
[TPS_SWCT_1402]	Mapping of all AtomicSwComponentType instances to ECUInstances
[TPS_SWCT_1403]	Impact of AUTOSAR services on the methodology
[TPS_SWCT_1404]	Creation of the EcuExtract

[TPS_SWCT_1405]	Creation of the ServiceSwComponentTypes
[TPS_SWCT_1406]	Creation of SwComponentPrototype typed by a ServiceSwComponentType
[TPS_SWCT_1407]	Creation of InternalBehavior typed by a ServiceSwComponentType
[TPS_SWCT_1408]	Creation of SwcBswMapping
[TPS_SWCT_1409]	Update of Port Defined Argument Values
[TPS_SWCT_1410]	Dcm and Dem can directly access dataElements in PPortPrototypes typed by a SenderReceiverInterface
[TPS_SWCT_1411]	Use cases for a ServiceSwComponentType to express ServiceNeeds
[TPS_SWCT_1412]	ServiceSwComponentType shall be added in ECU Configuration phase
[TPS_SWCT_1413]	Local communication with services
[TPS_SWCT_1414]	Mode manager needs to communicate with application software components located on other ECUs
[TPS_SWCT_1415]	Interfaces of ServiceProxySwComponentType
[TPS_SWCT_1416]	Difference between a ServiceProxySwComponentType and an ApplicationSwComponentType
[TPS_SWCT_1417]	Define calibration parameters common to all SwComponentPrototypes of the same SwComponentType
[TPS_SWCT_1418]	Ways to define a calibration parameter
[TPS_SWCT_1419]	ParameterSwComponentType shall never aggregate a SwcInternalBehavior
[TPS_SWCT_1420]	SwComponentType requiring access to shared calibration parameters needs RPortPrototype typed by a ParameterInterface
[TPS_SWCT_1421]	ParameterInterface is not restricted to parameters which can actually can be calibrated
[TPS_SWCT_1422]	Delegation of PortPrototypes typed bay aParameterInterface
[TPS_SWCT_1423]	ParameterDataPrototype aggregated in the role constantMemory
[TPS_SWCT_1424]	ParameterDataPrototype aggregated in the role perInstanceParameter
[TPS_SWCT_1425]	AtomicSwComponentType provides one callback per event if diagnostic event data change
[TPS_SWCT_1426]	AtomicSwComponentType provides callback if any diagnostic event data and/or status changed
[TPS_SWCT_1427]	AtomicSwComponentType provides data for diagnostic purposes via ClientServerInterface
[TPS_SWCT_1428]	ServiceSwComponentType representing the Dem provides a PPortPrototype for the Dcm
[TPS_SWCT_1429]	[constr_1135] only applies for BITFIELD_TEXTTABLE
[TPS_SWCT_2000]	Default value for attribute swImplPolicy
[TPS_SWCT_1430]	Conversion specification from internal to physical values as well as the reverse conversion
[TPS_SWCT_2001]	Values of SwAxisCont with the CATEGORY CURVE_AXIS, COM_AXIS, RES_AXIS are for display only
[TPS_SWCT_2002]	AtomicSwComponentType offers a server ports to read/write current value via diagnostic services
[TPS_SWCT_2003]	AtomicSwComponentType offers sender receiver ports to read/write current values via diagnostic services
[TPS_SWCT_2004]	AtomicSwComponentType offers a server port to start/stop or request routine results of diagnostic routines
[TPS_SWCT_2005]	AtomicSwComponentType offers a server ports to adjust the IO signal via diagnostic services
[TPS_SWCT_2006]	AtomicSwComponentType offers sender receiver ports to adjust the IO signal via diagnostic services

[TPS_SWCT_2007]	`AtomicSwComponentType` implements a OBD system monitor with In-Use-Monitor Performance Ratio
[TPS_SWCT_2008]	`AtomicSwComponentType` offers a server ports to read/write current value via OBD services
[TPS_SWCT_2009]	`AtomicSwComponentType` offers sender receiver ports to read/write current values via OBD services
[TPS_SWCT_2010]	`AtomicSwComponentType` offers a server port to read vehicle information values via OBD services
[TPS_SWCT_2011]	`AtomicSwComponentType` offers a server ports to read DTR value via OBD services
[TPS_SWCT_2012]	`AtomicSwComponentType` offers a server port for request control of on-board system, test or component via OBD services
[TPS_SWCT_2013]	`AtomicSwComponentType` offers a server ports to get protocol, session and security information or to request a Reset to Default Session
[TPS_SWCT_2014]	`AtomicSwComponentType` supports Response On Event (ROE) via diagnostic services
[TPS_SWCT_2015]	`AtomicSwComponentType` verifies the access to security level via diagnostic services
[TPS_SWCT_2016]	`AtomicSwComponentType` requires information on the status of the protocol communication and may disallow a protocol
[TPS_SWCT_2017]	`AtomicSwComponentType` requires the notification about a Service Request via diagnostic services
[TPS_SWCT_2018]	`AtomicSwComponentType` contains a Supervised Entity
[TPS_SWCT_2019]	`AtomicSwComponentType` requires *Global Supervision Status* notification
[TPS_SWCT_2020]	`AtomicSwComponentType` uses the hash calculation of the Crypto Service
[TPS_SWCT_2021]	`AtomicSwComponentType` uses the message authentication code (MAC) calculation of the Crypto Service
[TPS_SWCT_2022]	`AtomicSwComponentType` uses the message authentication code (MAC) verification of the Crypto Service
[TPS_SWCT_2023]	`AtomicSwComponentType` uses the generation of random seed of the Crypto Service
[TPS_SWCT_2024]	`AtomicSwComponentType` uses the generation of random numbers of the Crypto Service
[TPS_SWCT_2025]	`AtomicSwComponentType` uses the symmetrical block encryption of the Crypto Service
[TPS_SWCT_2026]	`AtomicSwComponentType` uses the symmetrical block decryption of the Crypto Service
[TPS_SWCT_2027]	`AtomicSwComponentType` uses the symmetrical encryption of the Crypto Service
[TPS_SWCT_2028]	`AtomicSwComponentType` uses the symmetrical decryption of the Crypto Service
[TPS_SWCT_2029]	`AtomicSwComponentType` uses the asymmetrical encryption of the Crypto Service
[TPS_SWCT_2030]	`AtomicSwComponentType` uses the asymmetrical decryption of the Crypto Service
[TPS_SWCT_2031]	`AtomicSwComponentType` uses the signature generation of the Crypto Service
[TPS_SWCT_2032]	`AtomicSwComponentType` uses the signature verification of the Crypto Service
[TPS_SWCT_2033]	`AtomicSwComponentType` uses the checksum calculation of the Crypto Service
[TPS_SWCT_2034]	`AtomicSwComponentType` uses the key derivation of the Crypto Service
[TPS_SWCT_2035]	`AtomicSwComponentType` uses the symmetric key derivation of the Crypto Service

[TPS_SWCT_2036]	`AtomicSwComponentType` uses the key exchange interface for public value calculation of the Crypto Service
[TPS_SWCT_2037]	`AtomicSwComponentType` uses the key exchange interface for secret value calculation of the Crypto Service
[TPS_SWCT_2038]	`AtomicSwComponentType` uses the key exchange interface to calculate symmetric key with the Crypto Service
[TPS_SWCT_2039]	`AtomicSwComponentType` uses the symmetrical key extraction of the Crypto Service
[TPS_SWCT_2040]	`AtomicSwComponentType` uses the symmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a symmetric key
[TPS_SWCT_2041]	`AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a symmetrical key structure with a asymmetric key
[TPS_SWCT_2042]	`AtomicSwComponentType` uses the asymmetrical public key extraction of the Crypto Service
[TPS_SWCT_2043]	`AtomicSwComponentType` uses the asymmetrical private key extraction of the Crypto Service
[TPS_SWCT_2044]	`AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a symmetrical wrapping key
[TPS_SWCT_2045]	`AtomicSwComponentType` uses the asymmetrical key wrapping of the Crypto Service to export a (asymmetric) private key structure with a asymmetrical wrapping key

Table C.7: Added Specification Items in 4.0.3

C.3.4 Deleted Constraints in R4.0.3

Number	Heading
[constr_1023]	Specification of `Unit`s in `CompuMethod`s[2]
[constr_1062]	Compatibility of data types with `category` BIT
[constr_1122]	Existence of attributes in PROFILE_03
[constr_1123]	Constraints of `dataLength` in PROFILE_03
[constr_1124]	Constraints of `dataId` in PROFILE_03
[constr_1125]	Constraints of `maxDeltaCounterInit` in PROFILE_03
[constr_1127]	`ServiceSwComponentType` shall not have `ServiceNeeds`
[constr_1136]	Compatibility of `introduction` of blueprint and blueprinted element
	the following constraints are moved to [1]
[constr_2500]	`PortInterface`s shall be of same kind
[constr_2526]	`PortInterfaces` need to be compatible to the blueprints
[constr_2527]	Blueprints shall live in package of a proper category
[constr_2528]	`PortPrototype`s shall not refer to blueprints of `PortInterface`s
[constr_2529]	Blueprints of ports and interfaces shall be compatible
[constr_4001]	Content of `ModeRequestTypeMap`

Table C.8: Deleted Constraints in R4.0.3

C.3.5 Deleted Specification Items

N/A

[2]The text is still there but it does no longer represent a constraint.