Document Title	Specification of ECU Configuration
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	087
Document Classification	Standard

Document Version	3.5.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Description**
31.03.2014	3.5.0	AUTOSAR Release Management	• Various fixes and clarifications
24.10.2013	3.4.0	AUTOSAR Release Management	• Support unidirectional CDD communication • Adapted range of parameter MetaDataLength • Harmonization with TR_Methodology • Added "origin" attribute to the EcucContainerDef • Adapted CDD configuration to allow the configuration of the CDD interface type (IF/TP) • Adapted the upper limit of PduLength parameter • Stereotyped EcucChoiceReferenceDef.destination and EcucSymbolicNameReferenceDef.destination with atpUriDef

04.02.2013	3.3.0	AUTOSAR Administration	• Description of the variant handling approach to cope with PreCompile, Link and Post-Build Configuration parameters as alternative to the usage of multiple configuration containers • Made the CDD configuration postBuildConfigurable • Updated sorting criteria for EcucContainerValues • Extended CDD configuration with SoAd interaction • Clarified the production error configuration • The destination of EcucReferenceDef and EcucChoiceReferenceDef is changed to EcucContainerDef
			• Extended the Ecu Query Language to describe configuration validity rules • Added apiServicePrefix attribute to EcucModuleDef • Added EcucPartitionBswModule-Execution and EcucPartitionBswModuleDistin-guishedPartition • Updated section about the conversion of time parameters of main functions to ticks • Added EcucCoreDefinition to Ecuc module

08.11.2011	3.2.0	AUTOSAR Administration	• ecuc_sws_5001 removed. • Clarified modeling of destinationType and destinationContext. • Clarified scope of parameters. • Clarified postBuildChangeable and multipleConfigurationContainer. • Added annotation to EcucAbstractReferenceValue. • Updated semantics of definitionRef and introduced the term "pure VSMD" • Clarification of PostBuildSelectable, PostBuildLoadable in VSMD • Set configuration class affection support to deprecated • Support for ordering of EcucParameters and EcucReferences • Reworked CDD configuration to reflect the direction of the communication • Clarified usage of symbolic name references

22.07.2010	3.1.0	AUTOSAR Administration	• Updated "refvalue" function requirements • Added requirement sws6045 • Changed specification of PduLength parameter from bits to bytes • Added attribute "origin" to EcucEnumerationParamDef • Added "Template Glossary" to Appendix • Added "Rules for navigating in Ecu Configuration Artifacts" chapter • Removed restriction on hex-representation of integers • Updated description of refinedModuleDef within class ModuleDef • Changed calculation language key words to lower case • Changed structure of EcucQuery and EcucQueryExpression • Added section on Communication Channel ID • Removed section on EcucMemoryMappingCollection • Removed "annotation" from "EcucContainerValue"

04.12.2009	3.0.0	AUTOSAR Administration	• Implemented Variant Handling concept • Implemented Calculation Formula concept • Reworked Parameter Value representation • Reworked Service Component Methodology chapter • Updated rules for deriving VSMD from StMD • Implemented Documentation support concept • Implemented support for existence dependence of ECUC Parameter Definition elements • Added "Clock Tree Configuration" chapter • Added "CDD module" chapter
15.09.2008	2.1.0	AUTOSAR Administration	Fixed foreign reference to PduToFrameMapping
15.02.2008	2.0.2	AUTOSAR Administration	Legal disclaimer revised.
01.02.2008	2.0.1	AUTOSAR Administration	Added reference from Container to ContainerDef. Removed reference from Container to ParamConfContainerDef.
06.12.2007	2.0.0	AUTOSAR Administration	• Changed representation of a ChoiceContainerDef in an ECU Configuration Description • Moved sections from "ECU Configuration Parameter Definition" into the "Specification of ECU Configuration" (COM-Stack Configuration Patterns) • Updated interaction of ECU Configuration with BSW Module Description

			• Added specification items which define what is allowed when creating a Vendor Specific Module Definition (VSMD) • Correction of "InstanceParamRef" definition in ECU Configuration Specification • Refined the available character set of calculationFormula • Added clarification about the usage of ADMIN-DATA to track version information • Document meta information extended • Small layout adaptations made
31.01.2007	1.1.1	AUTOSAR Administration	• “Advice for users” revised • Legal disclaimer revised
06.12.2006	1.1.0	AUTOSAR Administration	• Methodology chapter revised (incl. introduction of support for AUTOSAR Services) • Added EcucElement, EcuSwComposition, configuration class affection, LinkerSymbolDef and LinkerSymbolValue to the metamodel • Support for multiple configuration sets added • Legal disclaimer revised
28.06.2006	1.0.1	AUTOSAR Administration	Layout Adaptations
09.05.2006	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

Table of Contents

References

[1] Methodology
AUTOSAR_TR_Methodology

[2] System Template
AUTOSAR_TPS_SystemTemplate

[3] Glossary
AUTOSAR_TR_Glossary

[4] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[5] Requirements on ECU Configuration
AUTOSAR_RS_ECUConfiguration

[6] Specification of Interoperability of AUTOSAR Tools
AUTOSAR_TR_InteroperabilityOfAutosarTools

[7] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[8] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules

[9] Specification of ECU Configuration Parameters (XML)
AUTOSAR_MOD_ECUConfigurationParameters

[10] IEEE standard for radix-independent floating-point arithmetic
(ANSI/IEEE Std 854-1987)

[11] Meta Model
AUTOSAR_MMOD_MetaModel

[12] Meta Model-generated XML Schema
AUTOSAR_MMOD_XMLSchema

[13] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[14] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[15] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

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

1 Introduction

According to the AUTOSAR Methodology (see figure 1.1) the configuration process is a major part of the ECU software integration that is represented by the activity `Intergrate Software for ECU`.

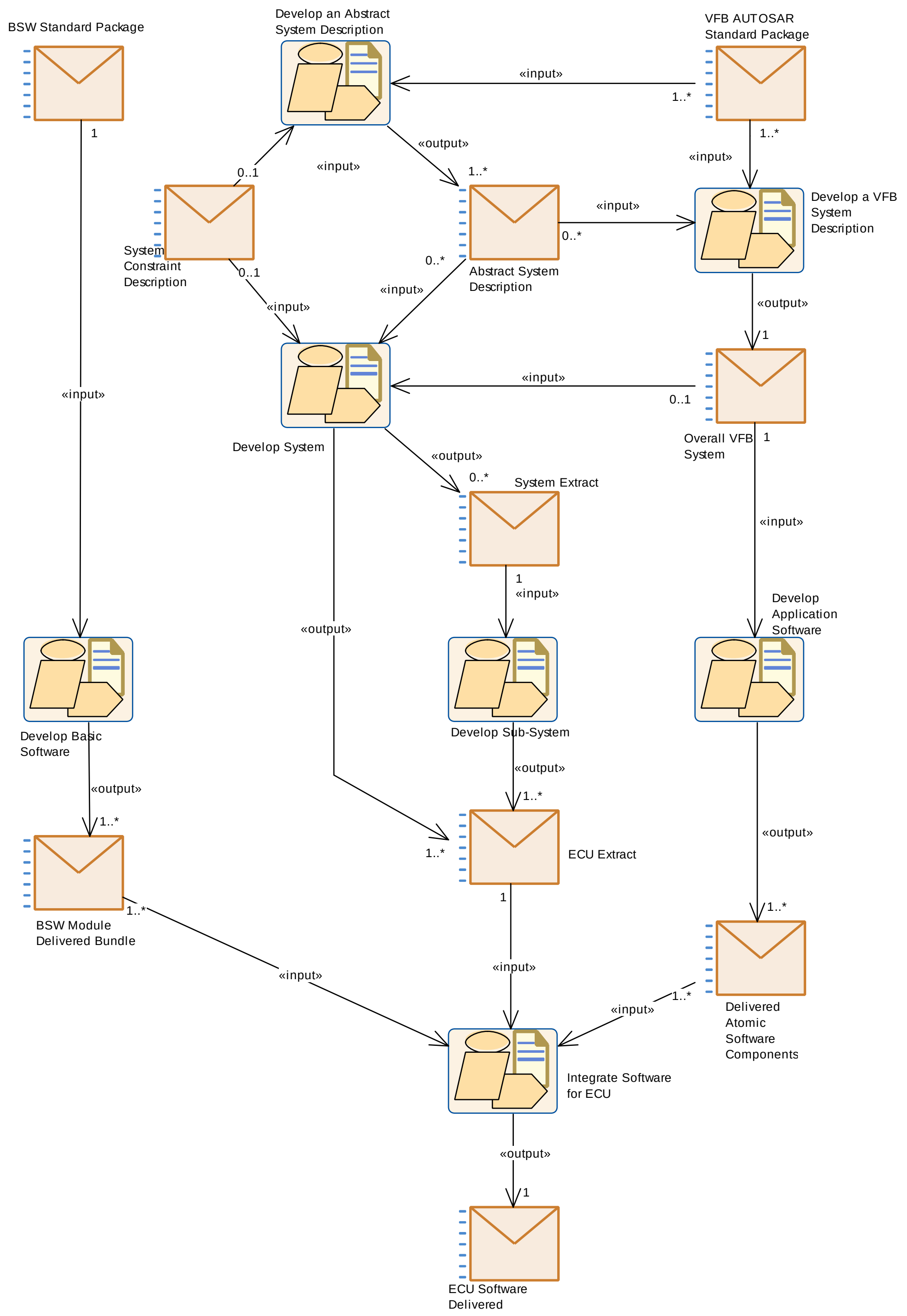

Figure 1.1: AUTOSAR Methodology Overview (from [1])

The configuration process of an ECU starts with the splitting of the System Description into several descriptions, whereas each contains all information about one single ECU. In figure 1.1 the artifact `System Description` is hidden in the activity `Develop System`. The creation of an `Ecu Extract` is described in detail in the System Template specification [2].

The `Ecu Extract` and the `BSW Module Delivered Bundle` are the inputs for the ECU configuration step. This is also visible in figure 1.2 where the ECU configuration is described by the activities `Prepare ECU Configuration` and `Configure BSW and RTE`.

A detailed description about this activities is given in the AUTOSAR Methodology [1], chapter 2.7.

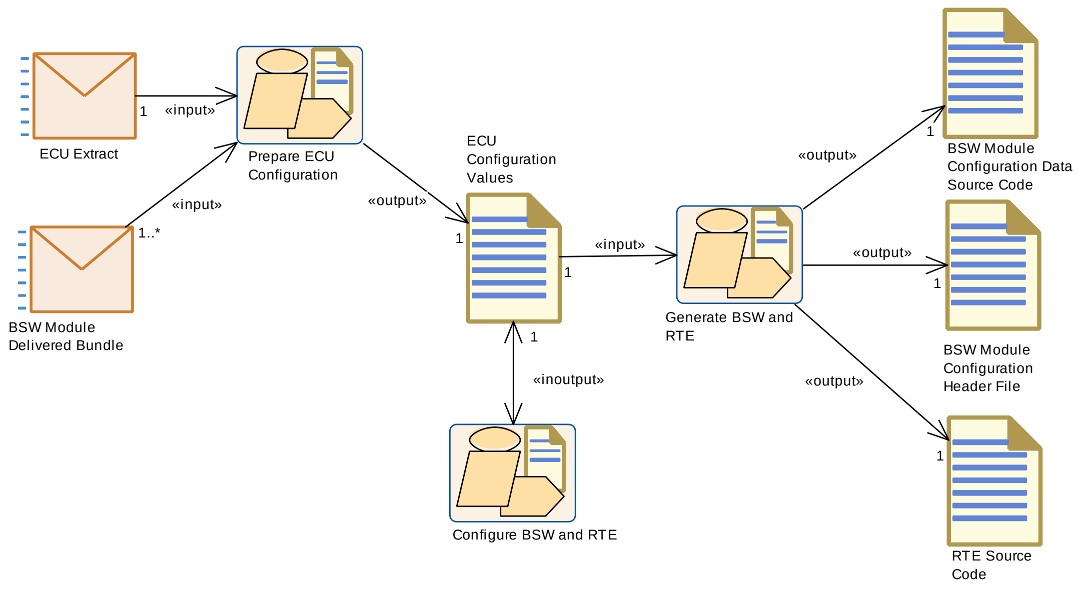

Figure 1.2: Ecu Configuration Overview (from [1])

Within the ECU Configuration process each single module of the AUTOSAR Architecture can be configured for the special needs of this ECU. Because of a quite complex AUTOSAR Architecture, modules and interdependencies between the modules, tool-support is required: AUTOSAR ECU Configuration Editor(s). Some basic rules for such Ecu Configuration Editor(s) are described in chapter 4.3.

The tool strategy and tooling details for the ECU Configuration are out of scope of this specification. Nevertheless tools need the knowledge about ECU Configuration Parameters and their constraints such as configuration class, value range, multiplicities etc. This description is the input for the tools. The description of configuration parameters is called ECU Configuration Parameter Definition and described in detail in this specification (chapter 2.3).

To make sure, that all tools are using the same output-format within the configured values of the parameters, the ECU Configuration Value description is also part of this specification and described in detail later on (chapter 2.4). The ECU Configuration Value description may be on one hand the input format for other configuration tools (within a tool-chain of several configuration editors) and on the other hand it is the basis of generators. The configured parameters are generated into ECU executables. This is the last step of the configuration process and again out of scope of this specification.

1.1 Abbreviations

This section describes abbreviations that are specific to the ECU Configuration Specification and that are not part of the official AUTOSAR Glossary [3].

Following abbreviations are mentioned that are specifically used in this specification:

ECUC	ECU Configuration
ECUC Value description	ECU Configuration Value Description
ECUC ParamDef	ECU Configuration Parameter Definition
ECUC Value	ECU Configuration Value
StMD	Standardized Module Definition
VSMD	Vendor Specific Module Definition

1.2 Document Conventions

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

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

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

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

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

Class	**AUTOSAR**			
Package	M2::AUTOSARTemplates::AutosarTopLevelStructure			
Note	Root element of an AUTOSAR description, also the root element in corresponding XML documents. **Tags:** xml.globalElement=true			
Base	ARObject			
Attribute	***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:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
introductio n	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 1.2: AUTOSAR

The first rows in the table have the following meaning:

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

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

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

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

The headers in the table have the following meaning:

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

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.

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([4]).

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([4]).

1.3 Requirements Tracing

The following table references the requirements specified in [5] and links to the fulfillments of these.

Requirement	Description	Satisfied by
[RS_ECUC_00002]	Support of vendor-specific ECU Configuration Parameters	[TPS_ECUC_01001] [TPS_ECUC_01011] [TPS_ECUC_01013] [TPS_ECUC_01014] [TPS_ECUC_01015] [TPS_ECUC_05002] [TPS_ECUC_05003] [TPS_ECUC_06007]
[RS_ECUC_00008]	Post-build time configuration of BSW	[TPS_ECUC_02019]
[RS_ECUC_00012]	One description mechanism for different configuration classes	[TPS_ECUC_02016]
[RS_ECUC_00015]	Configuration of multiple instances of BSW modules	[TPS_ECUC_02008] [TPS_ECUC_02059]
[RS_ECUC_00032]	ECU Configuration Description shall be the root for the whole configuration information of an ECU	[TPS_ECUC_02003]
[RS_ECUC_00043]	Duplication free description	[TPS_ECUC_02124]
[RS_ECUC_00046]	Support definition of configuration class	[TPS_ECUC_02016]
[RS_ECUC_00047]	Pre-compile time configuration of BSW	[TPS_ECUC_02017]
[RS_ECUC_00048]	Link time configuration of BSW	[TPS_ECUC_02018]
[RS_ECUC_00049]	ECU Configuration description shall be tool process-able	[TPS_ECUC_02001]
[RS_ECUC_00050]	Specify ECU Configuration Parameter Definition	[TPS_ECUC_02065]
[RS_ECUC_00055]	Support standardization of mandatory and optional configuration parameters	[TPS_ECUC_02008] [TPS_ECUC_02009] [TPS_ECUC_03010] [TPS_ECUC_03011] [TPS_ECUC_03030] [TPS_ECUC_06007]
[RS_ECUC_00065]	Development according to the AUTOSAR Generic Structure Template document	[TPS_ECUC_02000]
[RS_ECUC_00066]	Transformation of ECUC model according to the AUTOSAR Model Persistence Rules for XML	[TPS_ECUC_02001]
[RS_ECUC_00070]	Support mandatory and optional containers	[TPS_ECUC_02008] [TPS_ECUC_02009] [TPS_ECUC_06007]
[RS_ECUC_00071]	Support for Generic Configuration Editor	[TPS_ECUC_02124]
[RS_ECUC_00072]	Support for referencing from dependent containers	[TPS_ECUC_02039] [TPS_ECUC_03027] [TPS_ECUC_03033]
[RS_ECUC_00073]	Support Service Configuration of AUTOSAR SW Components	[TPS_ECUC_02087]
[RS_ECUC_00074]	Support Sequential ECU Configuration	[TPS_ECUC_02124]
[RS_ECUC_00076]	Support the configuration of which AUTOSAR Services are available on a specific ECU	[TPS_ECUC_06014]
[RS_ECUC_00078]	Variable existence of container on value side	[TPS_ECUC_02119] [TPS_ECUC_02120]
[RS_ECUC_00079]	Variable existence of value	[TPS_ECUC_02121] [TPS_ECUC_02122] [TPS_ECUC_02141]

Requirement	Description	Satisfied by
[RS_ECUC_00080]	Variable value	[TPS_ECUC_02142]
[RS_ECUC_00082]	Variable lower and upper multiplicity in ECU Configuration Parameter definition	[TPS_ECUC_02110] [TPS_ECUC_06009] [TPS_ECUC_06010] [TPS_ECUC_06013] [TPS_ECUC_06016]
[RS_ECUC_00083]	Variable default value in ECU Configuration Parameter definition	[TPS_ECUC_02111] [TPS_ECUC_02112] [TPS_ECUC_02114] [TPS_ECUC_02115]
[RS_ECUC_00084]	Variable min and max ranges in ECU Configuration Parameter definition	[TPS_ECUC_02116] [TPS_ECUC_02117]
[SRS_BSW_00167]	No description	[TPS_ECUC_06038]
[SRS_BSW_00171]	No description	[TPS_ECUC_02009] [TPS_ECUC_06007]
[SRS_BSW_00387]	The Basic Software Module specifications shall specify how the callback function is to be implemented	[TPS_ECUC_02016]
[SRS_BSW_00388]	Containers shall be used to group configuration parameters that are defined for the same object	[TPS_ECUC_02006]
[SRS_BSW_00389]	Containers shall have names	[TPS_ECUC_02043]
[SRS_BSW_00391]	No description	[TPS_ECUC_02014] [TPS_ECUC_02043]
[SRS_BSW_00392]	Parameters shall have a type	[TPS_ECUC_02014]
[SRS_BSW_00393]	Parameters shall have a range	[TPS_ECUC_02027] [TPS_ECUC_02028]
[SRS_BSW_00395]	The Basic Software Module specifications shall list all configuration parameter dependencies	[TPS_ECUC_02039]
[SRS_BSW_00396]	The Basic Software Module specifications shall specify one classe (of the three) to be supported	[TPS_ECUC_02016]
[SRS_BSW_00397]	No description	[TPS_ECUC_02017]
[SRS_BSW_00398]	No description	[TPS_ECUC_02018]
[SRS_BSW_00399]	No description	[TPS_ECUC_04005]

2 Configuration Metamodel

2.1 Introduction

AUTOSAR exchange formats are specified using a metamodel based approach (see also *Specification of Interoperability of Authoring Tools* [6]). The metamodel for the configuration of ECU artifacts uses an universal description language so that it is possible to specify different kinds of configuration aspects. This is important as it is possible to describe AUTOSAR-standardized and vendor-specific ECU Configuration Parameters with the same set of language elements. This eases the development of tools and introduces the possibility to standardize vendor-specific ECU Configuration Parameters at a later point in time.

In general the configuration language uses containers and actual parameters. Containers are used to group corresponding parameters. Parameters hold the relevant values that configure the specific parts of an ECU. Due to the flexibility that has to be achieved by the configuration language the configuration description is divided into two parts:

- ECU Configuration Parameter Definition
- ECU Configuration Values

A detailed description of these two parts and their relationships is presented in the following sections.

2.2 ECU Configuration Template Structure

In this section the relationships between the different AUTOSAR templates involved in the ECU Configuration are introduced. A template is defining the structure and possible content of an actual description. The concept is open to be implemented in several possible ways, in AUTOSAR XML files have been chosen to be used for the exchange formats. If XML files are used there is no conceptual limit in the number of files making up the description. All the contributing files are virtually merged to build the actual description[1].

The goal of the ECU Configuration Value Template is to specify an exchange format for the ECU Configuration Values of one ECU. The actual output of ECU Configuration editors is stored in the ECU Configuration Value description, which might be one or several XML files. But the ECU Configuration editors need to know how the content of an ECU Configuration Values should be structured (which parameters are available in which container) and what kind of restrictions are to be respected (e.g. the ECU Configuration Parameter is an integer value in the range between 0 and 255). This is specified in the ECU Configuration Parameter Definition which is also an XML file. The relationship between the two file types is shown in figure 2.1.

[1]The rules are defined in the Specification of Interoperability of Authoring Tools document [6].

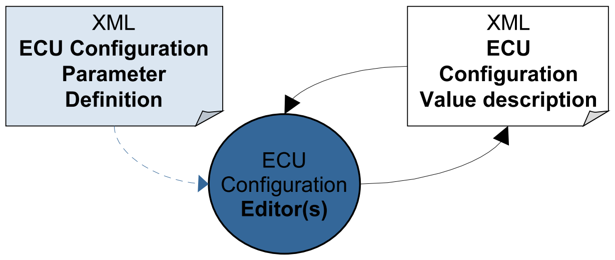

Figure 2.1: Parameter Definition and ECU Configuration Value files

For the ECU Configuration editors there are basically two possible approaches how to implement these definitions. Either the ECU Configuration Parameter Definition is read and interpreted directly from the XML file or the defined structures are hard-coded into the tool[2].

For the development of the ECU Configuration Parameter Definition and the ECU Configuration Value description a model-based approach has been chosen which already has been used during the development of other AUTOSAR template formats.

The main approach is to use a subset of UML to graphically model the desired entities and their relationships. Then, in a generation step, the actual XML formats are automatically generated out of the model.

[TPS_ECUC_02000] Modeling of ECU Configuration Value and ECU Configuration Parameter Definition metamodels ⌈ The modeling of the ECU Configuration Value and ECU Configuration Parameter Definition metamodels is done according to the Generic Structure Template [7]. ⌋*(RS_ECUC_00065)*

Please note that the Generic Structure Template [7] contains some fundamental infrastructure meta-classes and common patterns and provides details about:

- Autosar Top level structure,
- Commonly used metaclasses and primitives
- Variant Handling
- Documentation

[TPS_ECUC_02001] Transformation of the ECU Configuration Value and ECU Configuration Parameter Definition metamodels to schema definitions ⌈ The transformation of the ECU Configuration Value and ECU Configuration Parameter Definition metamodels to schema definitions is done according to the Model Persistence Rules for XML [8]. ⌋*(RS_ECUC_00049, RS_ECUC_00066)*

Because of these transformation rules there is a given discrepancy between the UML model and the generated XML-Schema names. This also affects this document. The major descriptions will be based on the UML model notations (figures and tables), although the corresponding XML notation might be given for reference purposes.

[2]The advantage of using the interpreter is that changes on the ECU Configuration Parameter Definition are directly available in the tool. But the hard-coded approach allows for more custom user support in the tool

In this section the application of the modeling approach for the ECU Configuration is described.

AUTOSAR uses the UML metamodel (M2-level) to describe the classes and objects that may be used in an AUTOSAR-compliant system. These metamodel elements may be used in an application model (M1-level) to describe the content of a real vehicle. ECU Configuration is a part of the AUTOSAR standard so the elements of ECU Configuration Description must be described in the UML metamodel at M2-level. The (M2) metamodel has therefore been populated with UML descriptions from which ECU Configuration Parameter models may be built.

With M2 definitions in place, it is possible to create AUTOSAR-conforming models of real application ECU Configuration Parameters (an ECU Configuration Parameter Definition Model) at M1-level. Certain aspects of real application configurations are already defined: BSW Modules have standard interfaces and configuration requirements. These 'real' configuration parameters have therefore already been modeled at M1-level for each defined BSW Module. These are described in detail in the SWS documents.

XML has been chosen as the technology that will be used by AUTOSAR-compliant tools in order to define and share information during an AUTOSAR-compliant system development. It must therefore be possible to transform the UML Configuration Parameter Definition Model (M1-level) into an XML Configuration Parameter Definition so that it may be used by ECU Configuration tools. This is the way that the tool gets a definition of exactly which ECU Configuration Parameters are available and how they may be configured. The Model Persistence Rules for XML [8] describes how the UML metamodel (M2-level) may be transformed into a schema that describes the format of XML to contain model elements.

This same formalization is also true for the ECU Configuration Parameter Definition Metamodel elements on M2-level: the Model Persistence Rules for XML dictates how ECU Configuration Parameter Definition elements will generate a schema to hold ECU Configuration Parameter Model (M1-level) elements in an XML ECU Configuration Parameter Definition, that can then be interpreted by ECU Configuration tools.

ECU Configuration editors allow a system designer to set ECU Configuration Parameter Values for their particular application. The actual values are then stored in an ECU Configuration Value description that conforms to the template described in the UML.

An ECU Configuration Value description is an XML file that conforms to an AUTOSAR schema called an ECU Configuration Value Template. The template in turn is an AUTOSAR standard defined by placing ECU Configuration Value Template elements into the UML Meta-Model (M2-level) such that the schema (the ECU Configuration Value Template) can be generated (using the Formalization Guide rules).

There are three different parts involved in the development of the ECU Configuration: UML models, Schema and XML content files. The overview is shown in figure 2.2.

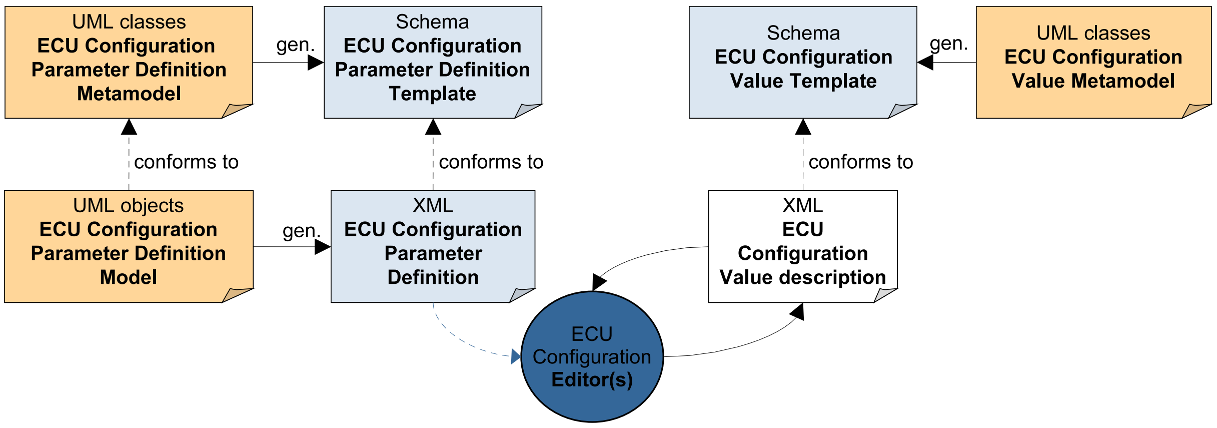
UML classes
ECU Configuration Parameter Definition Metamodel
gen.
Schema
ECU Configuration Parameter Definition Template
Schema
ECU Configuration Value Template
gen.
UML classes
ECU Configuration Value Metamodel
conforms to
conforms to
conforms to
UML objects
ECU Configuration Parameter Definition Model
gen.
XML
ECU Configuration Parameter Definition
XML
ECU Configuration Value description
ECU Configuration Editor(s)

Figure 2.2: Relationship between UML models and XML files

The following section describes one way to define ECU Configuration Parameter definitions. Other ways of defining and maintaining of ECU Configuration Parameter definitions are also possible.

The ECU Configuration Parameter Definition Model is used to specify the ECU Configuration Parameter Definition. This is done using object diagrams (this is the M1 level of metamodeling) with special semantics defined in section 2.3. What kind of UML elements are allowed in the ECU Configuration Parameter Definition Model is defined in the ECU Configuration Parameter Definition Metamodel which is conforming to the Generic Structure Template [7]. The definition is done using UML class diagrams (which is done on M2 level of metamodeling).

Out of the ECU Configuration Parameter Definition Metamodel a schema [3] is generated and the generated ECU Configuration Parameter Definition XML file has to conform to this schema. Vendor-specific ECU Configuration Parameter Definitions need to conform to this schema as well.

The ECU Configuration Value XML file needs to conform to the ECU Configuration Value Template schema which itself is generated out of the ECU Configuration Value Metamodel specified in UML class diagrams as well.

In the next section the ECU Configuration Parameter Definition Metamodel and its application toward the ECU Configuration Parameter Definition Model is described.

In the following figures and tables the names from the UML model are shown. In the generated XML-Schema the names may differ based on the Model Persistence Rules for XML [8]. For instance, the attribute `shortName` will become `SHORT-NAME` in the XML-Schema.

[3]Whether a DTD or an XML-Schema is used is not relevant for this explanation and is left to the formalization strategy defined in [8].

2.3 ECU Configuration Parameter Definition Metamodel

The two major building blocks for the specification of ECU Configuration Parameter Definitions are containers and parameters/references. With the ability to establish relationships between containers and parameters and the means to specify references, the definition of parameters has enough power for the needs of the ECU Configuration.

2.3.1 ECU Configuration Parameter Definition top-level structure

The definition of each Software Module's[4] configuration has at the top level the structure shown in figure 2.3. For an overview of the complete ECU Configuration top level structure please refer to chapter 2.4.1.

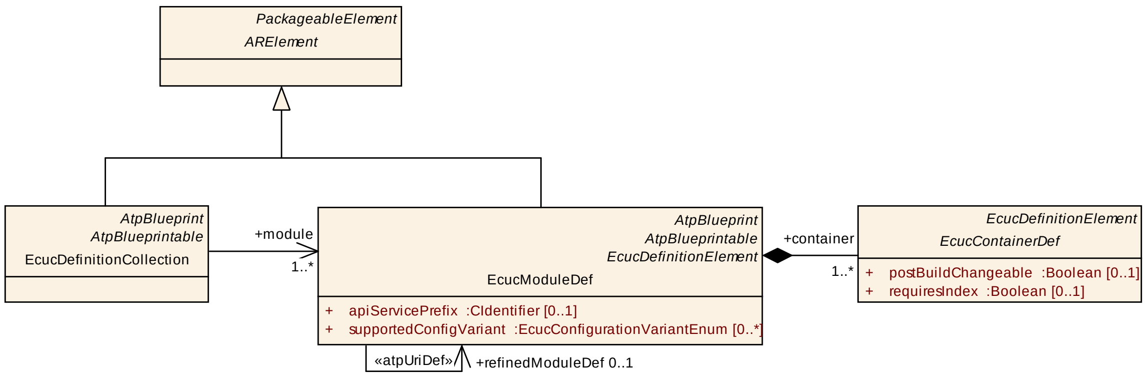

Figure 2.3: ECU Configuration Parameter Definition top-level structure

[TPS_ECUC_02002] Generic structure of all AUTOSAR templates ⌈ The generic structure of all AUTOSAR templates is described in detail in the AUTOSAR Generic Structure Template [7]. ⌋

[TPS_ECUC_02130] Standardized Module Definition package structure ⌈ The Standardized Module Definition (StMD) as delivered by AUTOSAR [9] shall be provided inside the package structure */AUTOSAR/EcucDefs/*. ⌋

[TPS_ECUC_06070] Sorting of Ecu Configuration Parameter Definitions ⌈ Ecu Configuration Parameter Definitions shall be sorted alphabetically. ⌋

[TPS_ECUC_02003] `EcucDefinitionCollection` class ⌈ First ECU Configuration specific class is the `EcucDefinitionCollection` which inherits from `ARElement`. Through this inheritance the `EcucDefinitionCollection` can be part of an AUTOSAR `ARPackage` and thus part of an AUTOSAR description. ⌋*(RS_ECUC_00032)*

[4]A Software Module might be Basic Software, RTE, Application Software Component or Complex Driver; see AUTOSAR Glossary [3]. The approach of Ecu configuration may be applied to non-standardized AUTOSAR Software modules (Application Software Component or Complex Driver) using the Vendor Specific Module Definition.

[TPS_ECUC_02065] EcucModuleDef class ⌈ The ECU Configuration Parameter Definition of one module is called EcucModuleDef and inherits from ARElement. ⌋*(RS_ECUC_00050)*

ARElement itself inherits from PackageableElement, Identifiable and Referrable which has two consequences: First, each Referrable has to have a machine readable shortName. Second, the Identifiable introduces the concept of a namespace for the contained Identifiable objects, so those objects need to have unique shortNames in the scope of that namespace. For additional information about the consequences of being a Referrable and Identifiable and the additional attributes please refer to the AUTOSAR Generic Structure Template [7].

[TPS_ECUC_02004] EcucDefinitionCollection collects all references to individual module configuration definitions ⌈ The use-case of the EcucDefinitionCollection class is to collect all references to individual module configuration definitions of the AUTOSAR ECU Configuration. Therefore the EcucDefinitionCollection specifies a reference relationship to the definition of several Software Modules in the module attribute. ⌋

Please note that it is allowed to have several EcucDefinitionCollections to collect the EcucModuleDefs based on various criteria e.g. modules from different vendors.

Class	**EcucDefinitionCollection**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	This represents the anchor point of an ECU Configuration Parameter Definition within the AUTOSAR templates structure. **Tags:** atp.recommendedPackage=EcucDefinitionCollections			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
module	EcucModuleDef	1..*	ref	References to the module definitions of individual software modules.

Table 2.1: EcucDefinitionCollection

2.3.1.1 Usage of the Admin Data

AdminData is an attribute of Identifiable [7] and can be used to set administrative information for an element (e.g. version information). Such administrative information can be set for the whole ECU Configuration Parameter Definition XML file and for each module definition.

[TPS_ECUC_06004] AdminData field in ECU Configuration Parameter Definition XML file ⌈ An AdminData field is required at the beginning of every ECU Configuration Parameter Definition XML file (regardless whether it is the StMD or the VSMD file) to allow the setting of AdminData for the whole XML File. ⌋

Example 2.1 shows how AdminData can be used for the whole ECU Configuration Parameter Definition XML file. For the part that is provided by AUTOSAR the AdminData shall be filled out with the AUTOSAR release information (Release and Revision number).

Example 2.1

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-0-3.xsd">
  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <USED-LANGUAGES>
      <L-10 L="EN" xml:space="default">EN</L-10>
    </USED-LANGUAGES>
    <DOC-REVISIONS>
      <DOC-REVISION>
        <REVISION-LABEL>3.0.0_revision_0004</REVISION-LABEL>
        <ISSUED-BY>AUTOSAR GbR</ISSUED-BY>
      </DOC-REVISION>
    </DOC-REVISIONS>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AUTOSAR</SHORT-NAME>
        <AR-PACKAGES>
          <AR-PACKAGE>
            <SHORT-NAME>EcucDefs</SHORT-NAME>
            <ELEMENTS>
              <ECUC-DEFINITION-COLLECTION>
```

[TPS_ECUC_06005] Usage of AdminData on EcucModuleDef is mandatory ⌈ For each module definition, the revision of the StMD shall be provided. For the VSMD the AUTOSAR release version and the vendor's own version information shall be provided. The usage of AdminData on EcucModuleDef is mandatory. ⌋

Example 2.2 shows that there are possibilities to specify several elements for the AdminData. The initial one would be provided by AUTOSAR, the additional one is the vendor's information which is based on the AUTOSAR one.

Example 2.2

```
    <ECUC-MODULE-DEF>
      <SHORT-NAME>Rte</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Configuration Parameter Definition of the RTE</L-2>
      </DESC>
      <ADMIN-DATA>
        <DOC-REVISIONS>
          <DOC-REVISION>
            <REVISION-LABEL>3.0.0_revision_0004</REVISION-LABEL>
            <ISSUED-BY>AUTOSAR GbR</ISSUED-BY>
            <DATE>2007-05-09T00:00:00Z</DATE>
          </DOC-REVISION>
          <DOC-REVISION>
            <REVISION-LABEL>15.3.0</REVISION-LABEL>
            <!--predecessor -->
            <REVISION-LABEL-P-1>2.1.1</REVISION-LABEL-P-1>
            <ISSUED-BY>VendorX</ISSUED-BY>
            <DATE>2007-06-21T09:30:00+01:00</DATE>
          </DOC-REVISION>
        </DOC-REVISIONS>
      </ADMIN-DATA>
      <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
      <CONTAINERS>
```

2.3.1.2 Life Cycle definition

AUTOSAR provides support for life cycle handling, defined in the Generic Structure Template [7]. A standardized usage of this approach is defined in the Standardization Template [4].

For the definition of ECU Configuration Parameters there is support in the MetaModel to annotate the life cycle state of each EcucDefinitionElement. For the annotation the following tagged value pairs can be used:

- atp.Status
- atp.StatusComment
- atp.StatusRevisionBegin

Example 2.3

```
<AR-PACKAGE>
  <SHORT-NAME>LifeCycleInfoSets</SHORT-NAME>
  <ELEMENTS>
    <LIFE-CYCLE-INFO-SET>
      <SHORT-NAME>AUTOSARParameterDefinition</SHORT-NAME>
      <DEFAULT-LC-STATE-REF DEST="LIFE-CYCLE-STATE">/AUTOSAR/
         GeneralDefinitions/LifeCycleStateDefinitionGroups/
         AutosarLifeCycleStates/valid</DEFAULT-LC-STATE-REF>
      <DEFAULT-PERIOD-BEGIN>
        <AR-RELEASE-VERSION>4.1.1</AR-RELEASE-VERSION>
      </DEFAULT-PERIOD-BEGIN>
      <LIFE-CYCLE-INFOS>
        <LIFE-CYCLE-INFO>
          <LC-OBJECT-REF DEST="ECUC-DEFINITION-ELEMENT">/AUTOSAR/EcucDefs/
             EcuC/EcucPduCollection/Pdu/SysTPduToFrameMappingRef</LC-OBJECT
             -REF>
          <LC-STATE-REF DEST="LIFE-CYCLE-STATE">/AUTOSAR/GeneralDefinitions
             /LifeCycleStateDefinitionGroups/AutosarLifeCycleStates/
             obsolete</LC-STATE-REF>
          <PERIOD-BEGIN>
            <AR-RELEASE-VERSION>4.1.1</AR-RELEASE-VERSION>
          </PERIOD-BEGIN>
          <REMARK>
            <P>
              <L-1 L="EN">obsolete since R4.1.1</L-1>
            </P>
          </REMARK>
          <USE-INSTEAD-REFS>
            <USE-INSTEAD-REF DEST="ECUC-DEFINITION-ELEMENT">/AUTOSAR/
               EcucDefs/EcuC/EcucPduCollection/Pdu/
               SysTPduToFrameTriggeringRef</USE-INSTEAD-REF>
            <USE-INSTEAD-REF DEST="ECUC-DEFINITION-ELEMENT">/AUTOSAR/
               EcucDefs/EcuC/EcucPduCollection/Pdu/
               SysTPduToPduTriggeringRef</USE-INSTEAD-REF>
          </USE-INSTEAD-REFS>
        </LIFE-CYCLE-INFO>
      </LIFE-CYCLE-INFOS>
      <USED-LIFE-CYCLE-STATE-DEFINITION-GROUP-REF DEST="LIFE-CYCLE-STATE-
         DEFINITION-GROUP">/AUTOSAR/GeneralDefinitions/
         LifeCycleStateDefinitionGroups/AutosarLifeCycleStates</USED-LIFE-
         CYCLE-STATE-DEFINITION-GROUP-REF>
    </LIFE-CYCLE-INFO-SET>
  </ELEMENTS>
</AR-PACKAGE>
```

2.3.1.3 Documentation Support

AUTOSAR 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 [7].

The documentation can be specified within in the following levels:

- a single paragraph can be inserted in any `Identifiable` element using the `desc` element.
- a documentation block is available in any `Identifiable` element as `introduction`. This type of documentation is typically used to capture a short introduction about the role of an element or respectively how it is built.
- a standalone documentation structured into multiple chapters is also offered in AUTOSAR. It is provided as `Documentation` which is an `ARElement` of its own rights allowing for a reference to the documents context.

With the introduction of this concept the container and parameter notes in the ECU Configuration Parameter Definition XML file are split into a `desc` and an `introduction` field. The `desc` field contains a brief description about the element and the `introduction` field contains the documentation about how the element is built and used.

In the ECU Configuration Parameter Definition XML file of the current AUTOSAR Release the proper usage of the desc and the introduction fields is not guaranteed. Therefore the content of the desc and introduction shall be read as one cohesive note.

Example 2.4 shows the split of the desc and introduction.

Example 2.4

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>AdcHwUnit</SHORT-NAME>
  <DESC>
    <L-2 L="EN">This container contains the Driver configuration (
       parameters) depending on grouping of channels</L-2>
  </DESC>
  <INTRODUCTION>
    <P>
      <L-1 L="EN">This container could contain HW specific parameters which
          are not defined in the Standardized Module Definition. They must
         be added in the Vendor Specific Module Definition.</L-1>
    </P>
  </INTRODUCTION>
  <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
  <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-CONTAINER
      >
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

Example 2.5 shows the usage of the Documentation element to describe elements like chapters, lists, tables and figures. For details on this description means please refer to the AUTOSAR Generic Structure Template [7].

Example 2.5

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>EcucDefs</SHORT-NAME>
      <ELEMENTS>
        <DOCUMENTATION>
          <SHORT-NAME>Adc_AddInfo</SHORT-NAME>
          <CONTEXTS>
            <DOCUMENTATION-CONTEXT>
              <SHORT-NAME>AUTOSAR_Adc</SHORT-NAME>
              <IDENTIFIABLE-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/
                  Adc</IDENTIFIABLE-REF>
            </DOCUMENTATION-CONTEXT>
          </CONTEXTS>
          <DOCUMENTATION-CONTENT>
            <CHAPTER>
              <SHORT-NAME>Introduction</SHORT-NAME>
              <P><L-1 L="EN">The ADC module initializes and controls the
                  internal Analogue Digital Converter Unit(s) of the
```

```
                    microcontroller. It provides services to start and stop a
                    conversion respectively to enable and disable the trigger
                    source for a conversion.</L-1></P>
                 <P><L-1 L="EN">The consistency of the group channel results
                    can be obtained with the following methods on the
                    application side:</L-1></P>
                 <LIST>
                   <ITEM>
                     <P><L-1 L="EN">Using group notification mechanism</L-1></
                        P>
                   </ITEM>
                   <ITEM>
                     <P><L-1 L="EN">Polling via API function
                        Adc_GetGroupStatus</L-1></P>
                   </ITEM>
                 </LIST>
                 <TABLE>
                   <TGROUP COLS="2">
                     <THEAD>
                       <ROW>
                         <ENTRY>
                           <P><L-1 L="EN">column1</L-1></P>
                         </ENTRY>
                         <ENTRY>
                           <P><L-1 L="EN">column2</L-1></P>
                         </ENTRY>
                       </ROW>
                     </THEAD>
                     <TBODY>
                       <ROW>
                         <ENTRY>
                           <P><L-1 L="EN">element11</L-1></P>
                         </ENTRY>
                         <ENTRY>
                           <P><L-1 L="EN">element12</L-1></P>
                         </ENTRY>
                       </ROW>
                       <ROW>
                         <ENTRY>
                           <P><L-1 L="EN">element21</L-1></P>
                         </ENTRY>
                         <ENTRY>
                           <P><L-1 L="EN">element22</L-1></P>
                         </ENTRY>
                       </ROW>
                     </TBODY>
                   </TGROUP>
                 </TABLE>
               </CHAPTER>
             </DOCUMENTATION-CONTENT>
           </DOCUMENTATION>
         </ELEMENTS>
       </AR-PACKAGE>
     </AR-PACKAGES>
   </AR-PACKAGE>
```

2.3.2 ECU Configuration Module Definition

[TPS_ECUC_02005] EcucModuleDef class ⌈ The class EcucModuleDef is defining the ECU Configuration Parameters of one Software Module[5]. It is inheriting form ARElement, so each individual EcucModuleDef needs to have a unique name within its enclosing ARPackage. ⌋

[TPS_ECUC_02059] Number of instances of a BSW module in the ECU Configuration Value description ⌈ The EcucModuleDef is using the EcucDefinitionElement attributes to specify how many instances of that specific module are allowed in the ECU Configuration Value description (see section 2.3.4.2). ⌋*(RS_ECUC_00015)*

Class	**EcucModuleDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used as the top-level element for configuration definition for Software Modules, including BSW and RTE as well as ECU Infrastructure. **Tags:** atp.recommendedPackage=EcucModuleDefs			
Base	ARElement,ARObject,AtpBlueprint,AtpBlueprintable,AtpDefinition,Collectable Element,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
apiService Prefix	CIdentifier	0..1	ref	For CDD modules this attribute holds the apiServicePrefix. The shortName of the module definition of a Complex Driver is always "CDD". Therefore for CDD modules the module apiServicePrefix is described with this attribute.
container	EcucContainerD ef	1..*	aggr	Aggregates the top-level container definitions of this specific module definition. **Tags:** xml.sequenceOffset=11
refinedMod uleDef	EcucModuleDef	0..1	ref	Optional reference from the Vendor Specific Module Definition to the Standardized Module Definition it refines. In case this EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION this reference shall not be provided. In case this EcucModuleDef has the category VENDOR_SPECIFIC_MODULE_DEFINITION this reference is mandatory. **Stereotypes:** atpUriDef

[5]A Software Module is not restricted to the BSW Modules but also includes the RTE, Application Software Components and generic ECU Configuration.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
supported ConfigVari ant	EcucConfigurati onVariantEnum	*	attr	Specifies which ConfigurationVariants are supported by this software module. This attribute is optional if the EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_MODULE_DEFINITION then this attribute is mandatory.

Table 2.2: EcucModuleDef

[TPS_ECUC_02094] `EcucModuleDef` is able to aggregate container definitions ⌈ The `EcucModuleDef` aggregates container definitions (`EcucContainerDef`) with the role name `container` which may hold other container definitions, parameter definitions and reference definitions. ⌋

[TPS_ECUC_02095] VSMD refines the StMD ⌈ The reference `refinedModuleDef` from an `EcucModuleDef` with the `category` *VENDOR_SPECIFIC_MODULE_DEFINITION* to an `EcucModuleDef` with the `category` *STANDARDIZED_MODULE_DEFINITION* specifies that the source `EcucModuleDef` is the *Vendor Specific Module Definition* which refines the referenced Standardized `EcucModuleDef`. ⌋

[TPS_ECUC_06076] Use cases where the reference `refinedModuleDef` is mandatory ⌈ The `refinedModuleDef` reference is mandatory if the `EcucModuleDef` with the `category` *VENDOR_SPECIFIC_MODULE_DEFINITION* actually refines the `EcucModuleDef` with the `category` *STANDARDIZED_MODULE_DEFINITION* (e.g. Vendor Specific Module Definition of Com BSW module refines Standardized Module Definition of Com BSW module). ⌋

[TPS_ECUC_06077] Use cases where the reference `refinedModuleDef` is optional ⌈ The `refinedModuleDef` reference is not necessary if the `EcucModuleDef` with the `category` *VENDOR_SPECIFIC_MODULE_DEFINITION* does not actually refines any `EcucModuleDef`s with the `category` *STANDARDIZED_MODULE_DEFINITION* (e.g. Vendor Specific Module Definition of CDD which does not contribute to the ComStack configuration). ⌋

[TPS_ECUC_06044] `refinedModuleDef` reference in the StMD ⌈ The reference `refinedModuleDef` from an `EcucModuleDef` with the category *STANDARDIZED_MODULE_DEFINITION* shall not be used. ⌋

[TPS_ECUC_06043] `EcucModuleDef` categories ⌈ The `category` attribute shall be used to clearly distinguish between the different roles of the `EcucModuleDef` class. ⌋

category	*Meaning*
STANDARDIZED_ MODULE_DEFINITION	The `EcucModuleDef` class is used to describe the Standardized Module Definition (StMD)
VENDOR_SPECIFIC_ MODULE_DEFINITION	The `EcucModuleDef` class is used to describe Vendor Specific Module Definition

Table 2.3: `EcucModuleDef` class categories

[constr_3022] EcucModuleDef category restriction ⌈The category definition shall be restricted to exactly the two defined ones:

- VENDOR_SPECIFIC_MODULE_DEFINITION
- STANDARDIZED_MODULE_DEFINITION

⌋

[TPS_ECUC_02096] Supported configuration variants in a BSW module ⌈ The EcucModuleDef specifies which configuration variants are supported by this software modules configuration using the element supportedConfigVariant. For each configuration variant that is supported one entry shall be provided. ⌋

For a detailed description how the configuration variants are related to the configuration classes please refer to section 2.3.4.3.2.

In figure 2.4 an example of the top-level structure is provided and in the example 2.6 the corresponding ECU Configuration Parameter Definition XML file extract is shown. In the example XML also the overall XML structure of AUTOSAR descriptions is shown. The corresponding ECU Configuration Value description XML file extract is shown in example 2.29.

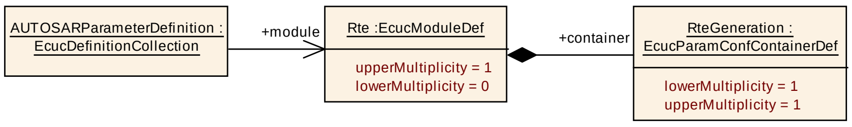

Figure 2.4: ECU Configuration Definition example

Example 2.6

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
    <AR-PACKAGES>
      <AR-PACKAGE>
        <SHORT-NAME>EcucDefs</SHORT-NAME>
        <ELEMENTS>
          <ECUC-DEFINITION-COLLECTION>
            <SHORT-NAME>AUTOSARParameterDefinition</SHORT-NAME>
            <MODULE-REFS>
              <MODULE-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Rte<
                  /MODULE-REF>
              <!-- Further references to module definitions -->
            </MODULE-REFS>
          </ECUC-DEFINITION-COLLECTION>
          <ECUC-MODULE-DEF>
            <SHORT-NAME>Rte</SHORT-NAME>
            <DESC>
              <L-2 L="EN">Configuration Parameter Definition of the RTE
                  </L-2>
            </DESC>
            <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
            <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
            <SUPPORTED-CONFIG-VARIANTS>
              <SUPPORTED-CONFIG-VARIANT>VARIANT-PRE-COMPILE</SUPPORTED-
                  CONFIG-VARIANT>
            </SUPPORTED-CONFIG-VARIANTS>
            <CONTAINERS>
```

In the next sections the structure of containers, individual parameters and references is introduced.

2.3.3 Container Definition

[TPS_ECUC_02006] Container definition ⌈ The container definition is used to group other parameter container definitions, parameter definitions and reference definitions. ⌋*(SRS_BSW_00388)*

There are two specializations of a container definition. The abstract class `EcucContainerDef` is used to gather the common features (see figure 2.5).

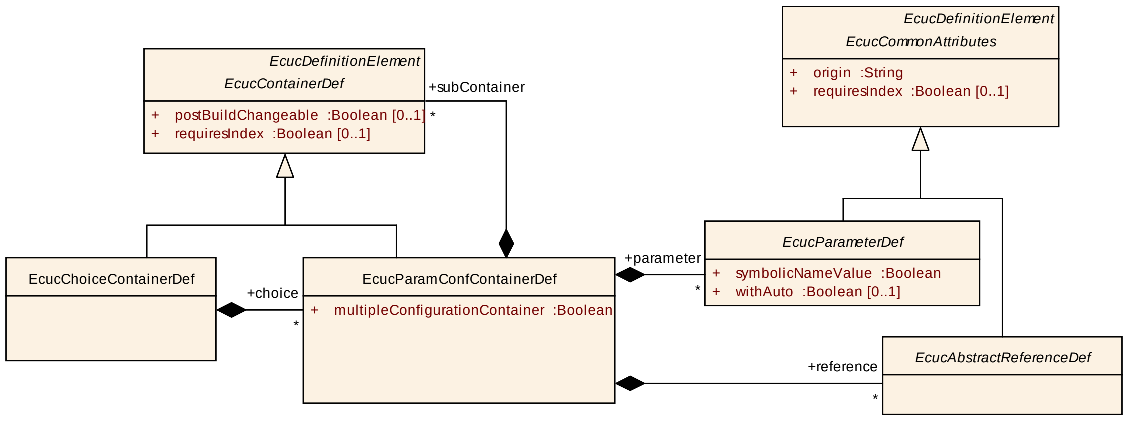
EcucDefinitionElement
EcucContainerDef
+ postBuildChangeable :Boolean [0..1]
+ requiresIndex :Boolean [0..1]
+subContainer
*
EcucDefinitionElement
EcucCommonAttributes
+ origin :String
+ requiresIndex :Boolean [0..1]
EcucChoiceContainerDef
+choice
*
EcucParamConfContainerDef
+ multipleConfigurationContainer :Boolean
+parameter
*
EcucParameterDef
+ symbolicNameValue :Boolean
+ withAuto :Boolean [0..1]
+reference
*
EcucAbstractReferenceDef

Figure 2.5: Class diagram for parameter container definition

Class	**EcucContainerDef (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Base class used to gather common attributes of configuration container definitions.			
Base	ARObject,AtpDefinition,EcucDefinitionElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
postBuildChangeable	Boolean	0..1	attr	Specifies if the number of instances of this container may be changed in post-build selectable and post-build loadable time in the ECU Configuration Value Description. This attribute is only applicable to containers which may appear multiple times in the configuration set, i.e. their upperMultiplicity is greater than lowerMultiplicity and their upperMultiplicity is greater than 1. If a value of this attribute is not defined in the EcucModuleDef with category STANDARDIZED_MODULE_DEFINITION, it shall be defined in the EcucModuleDef with category VENDOR_SPECIFIC_MODULE_DEFINITION for all containers with upperMultiplicity greater than lowerMultiplicity and upperMultiplicity greater than 1.
requiresIndex	Boolean	0..1	attr	Used to define whether the value element for this definition shall be provided with an index.

Table 2.4: EcucContainerDef

[TPS_ECUC_02043] Each `EcucContainerDef` is `Identifiable` ⌈ Each `EcucContainerDef` is an `Identifiable`. ⌋*(SRS_BSW_00389, SRS_BSW_00391)*

[TPS_ECUC_02044] Number of instances of a `EcucContainerDef` in the ECU Configuration Value description ⌈ Each `EcucContainerDef` also has the features of `EcucDefinitionElement` which enables to specify for each `EcucContainerDef` how often it is allowed to occur in the ECU Configuration Value description later on (see section 2.3.4.2). ⌋

[TPS_ECUC_08000] Number of instances of a `EcucContainerDef` may change at post-build time if attribute `postBuildChangeable` is set to true ⌈ The attribute `postBuildChangeable` specifies if the number of instances of this `EcucContainerDef` may be changed in post-build selectable and post-build loadable time in the `ECU Configuration Value Description`. ⌋

For post-build selectable configuration, setting the attribute `postBuildChangeable` to `true` means that different number of instances of this container may exist in different configuration sets.

For post-build loadable configuration, setting the attribute `postBuildChangeable` to `true` means that new instances of this container may be introduced or existing instances removed in the new configuration set.

[constr_5500] Applicability of `postBuildChangeable` attribute ⌈ The attribute `postBuildChangeable` is applicable only to `EcucContainerDef`s which have `upperMultiplicity` greater than `lowerMultiplicity` and `upperMultiplicity` is greater than 1. ⌋

[constr_5504] Removing an instance of the EcucContainerDef in post-build time ⌈ Only instances of EcucContainerDefs with the attribute postBuildChangeable set to true which are exclusively referenced by EcucAbstractReferenceDefs with PostBuild configuration class and have been introduced in post-build time (which were created in the initial configuration before post-build updates) can be removed in post-build time. ⌋

[TPS_ECUC_08003] Usage of postBuildChangeable attribute is independent of aggregated subContainers ⌈ An EcucContainerDef may have the attribute postBuildChangeable set to true even if one or more of its aggregated EcucContainerDefs in the role subContainer have the attribute postBuildChangeable set to false. ⌋

If a container "A" has the attribute postBuildChangeable set to true and its sub-container "B" set to false, it is not possible to add a new instance "b2" of sub-container "B" to the existing container instance "a1" of "A" in post-build time. However, it is allowed to add a new instance "a2" of the postBuildChangeable container "A" together with a new instance "b2" of its sub-container "B".

[TPS_ECUC_02007] EcucParamConfContainerDef class ⌈ A EcucParamConfContainerDef is the main container class definition and can contain other containers, configuration parameters and references. ⌋

Class	**EcucParamConfContainerDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used to define configuration containers that can hierarchically contain other containers and/or parameter definitions.			
Base	ARObject,AtpDefinition,EcucContainerDef,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
multipleConfigurationContainer	Boolean	1	attr	Specifies whether this container is used to define multiple configuration sets. Only one container in the whole EcucModuleDef shall have this enabled.
parameter	EcucParameterDef	*	aggr	The parameters defined within the EcucParamConfContainerDef.
reference	EcucAbstractReferenceDef	*	aggr	The references defined within the EcucParamConfContainerDef.
subContainer	EcucContainerDef	*	aggr	The containers defined within the EcucParamConfContainerDef.

Table 2.5: EcucParamConfContainerDef

For a detailed description of Multiple Configuration sets please refer to chapters 2.3.3.2 and sec:MultipleConfigSets.

One example of a EcucContainerDef and its embedding in the ECU Configuration Parameter Definition is shown in figure 2.6. One EcucModuleDef Rte is specified being part of the EcucDefinitionCollection. Two containers of type EcucParamConfContainerDef are specified as part of the module definition.

When specifying the containment relationship between the EcucModuleDef and containers the role name container is used. When specifying the containment relationship between two containers an aggregation with the role name subContainer at the contained container is used.

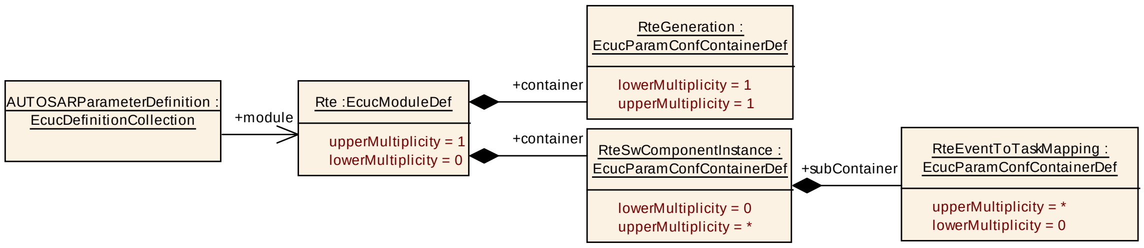

Figure 2.6: Example of an object diagram for container definition

In the XML outtake in example 2.7 only the relevant part from figure 2.6 is shown, not including the EcucDefinitionCollection[6]. The corresponding ECU Configuration Value description XML file extract is shown in example 2.33.

Example 2.7

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Rte</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>RteGeneration</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
          CONTAINER>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>SwComponentInstance</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
          CONTAINER>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

2.3.3.1 Choice Container Definition

[TPS_ECUC_02011] EcucChoiceContainerDef class ⌈ The EcucChoiceContainerDef can be used to specify that certain containers might occur exclusively in

[6]Note that in the figures of ECU Configuration Parameter Definition modeled in UML the infinite upper multiplicity is shown as *upperMultiplicity* = * resulting in *<UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>*

the ECU Configuration Value description. In the ECU Configuration Parameter Definition the potential containers are specified as part of the `EcucChoiceContainerDef` and the constraint is that in the actual ECU Configuration Value description only some of those specified containers will actually be present. ⌋

Class	**EcucChoiceContainerDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Used to define configuration containers that provide a choice between several EcucParamConfContainerDef. But in the actual ECU Configuration Values only one instance from the choice list will be present.			
Base	ARObject,AtpDefinition,EcucContainerDef,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
choice	EcucParamConfContainerDef	*	aggr	The choices available in a EcucChoiceContainerDef.

Table 2.6: EcucChoiceContainerDef

[TPS_ECUC_02067] Multiplicity of the *to be chosen* containers ⌈ The multiplicity of the *to be chosen* containers shall always be `0..1`, indicating that each time a choice is performed you can only choose exactly one of these *to be chosen* containers at a time. ⌋

[TPS_ECUC_02012] Allowed choice of available *to be chosen* containers in the ECU Configuration Value description ⌈ Each time a choice can be performed, the user is free to choose one of the available *to be chosen* containers. The `upperMultiplicity` of the `EcucChoiceContainerDef` specifies how many instances on the values side shall be allowed. ⌋

An example of the usage of a EcucChoiceContainerDef is shown in figure 2.7 and the XML definition is shown in example 2.8. The corresponding ECU Configuration Value description is shown in example 2.34.

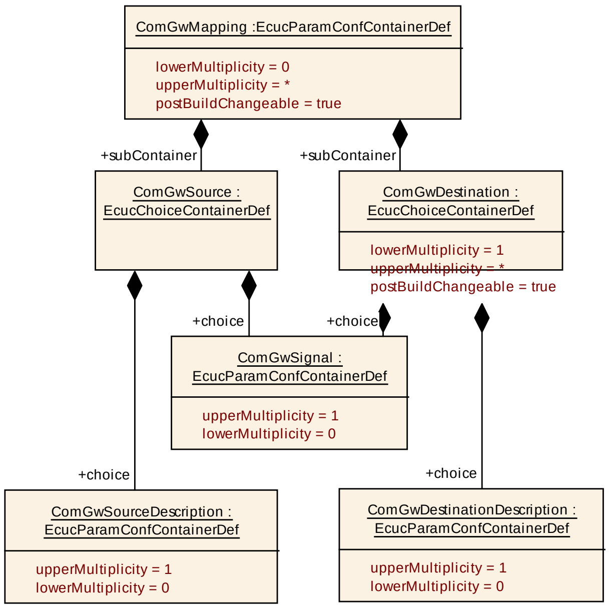
ComGwMapping :EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
postBuildChangeable = true
+subContainer
+subContainer
ComGwSource : EcucChoiceContainerDef
ComGwDestination : EcucChoiceContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
postBuildChangeable = true
+choice
+choice
ComGwSignal : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
+choice
+choice
ComGwSourceDescription : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
ComGwDestinationDescription : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0

Figure 2.7: Example of an object diagram for two choice container definitions

The example shows two use-cases of EcucChoiceContainerDef with different multiplicities of the EcucChoiceContainerDef.

The EcucChoiceContainerDef ComGwSource is defined to be able to hold one of the two given containers later in the ECU Configuration Value description. Since the upperMultiplicity of ComGwSource = 1 there can only be one choice taken.

The EcucChoiceContainerDef ComGwDestination is defined to be able to hold one of the two given containers later in the ECU Configuration Value description. Since the upperMultiplicity of ComGwDestination = * there can several choices be taken.

Example 2.8

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>ComGwMapping</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
  <SUB-CONTAINERS>
    <ECUC-CHOICE-CONTAINER-DEF>
      <SHORT-NAME>ComGwSource</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
      <CHOICES>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>ComGwSignal</SHORT-NAME>
          <!-- ... -->
        </ECUC-PARAM-CONF-CONTAINER-DEF>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
```

```
          <SHORT-NAME>ComGwSourceDescription</SHORT-NAME>
          <!-- ... -->
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </CHOICES>
    </ECUC-CHOICE-CONTAINER-DEF>
    <ECUC-CHOICE-CONTAINER-DEF>
      <SHORT-NAME>ComGwDestination</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
      <CHOICES>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>ComGwSignal</SHORT-NAME>
          <!-- ... -->
        </ECUC-PARAM-CONF-CONTAINER-DEF>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>ComGwDestinationDescription</SHORT-NAME>
          <!-- ... -->
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </CHOICES>
    </ECUC-CHOICE-CONTAINER-DEF>
  </SUB-CONTAINERS>
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

The containers from the example, which the choice is from, will of course have to be specified in more detail in an actual definition file.

2.3.3.2 Multiple Configuration Set Definition

To allow the description of several ECU configuration sets, either the approach using `multipleConfigurationContainer` (see section 2.4.7) or the approach using `Variant Handling` (see section 2.4.8) shall be used.

[TPS_ECUC_02091] `multipleConfigurationContainer` approach ⌈ In case `multipleConfigurationContainer` approach is used to allow the existence of multiple configuration sets, an `EcucModuleDef` which supports multiple configuration sets shall contain exactly one `EcucParamConfContainerDef` which is specified to be a `multipleConfigurationContainer`. ⌋

[TPS_ECUC_02133] `upperMultiplicity` of a `multipleConfigurationContainer` ⌈ The `upperMultiplicity` of a container that has the attribute `multipleConfigurationContainer` set to `true` shall always be `1`. ⌋

If a `EcucParamConfContainerDef` is specified to be the `multipleConfigurationContainer` there can be several `EcucContainerValue` descriptions of this container in the Ecu Configuration Values for PostBuild configurations[7]. The `shortName` of the `multipleConfigurationContainer` does define the name of the configuration set (see also section 2.4.7 for details on the Ecu Configuration Value de-

[7]In case of PreCompile and LinkTime configuration variants the features of the `multipleConfigurationContainer` are not used.

scription). The corresponding ECUC Value description XML file extract is shown in example 2.49.[8]

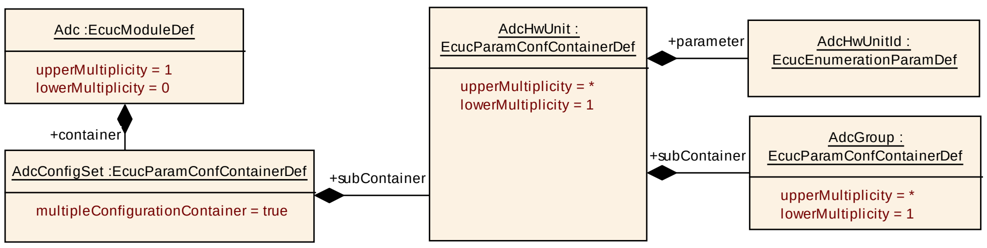

Figure 2.8: Example of an object diagram for multiple configuration container definition

Example 2.9

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Adc</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>AdcConfigSet</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
      <MULTIPLE-CONFIGURATION-CONTAINER>true</MULTIPLE-CONFIGURATION-
          CONTAINER>
      <SUB-CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>AdcHwUnit</SHORT-NAME>
          <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
              CONTAINER>
          <PARAMETERS>
            <ECUC-INTEGER-PARAM-DEF>
              <SHORT-NAME>AdcHwUnitIt</SHORT-NAME>
            </ECUC-INTEGER-PARAM-DEF>
          </PARAMETERS>
          <SUB-CONTAINERS>
            <ECUC-PARAM-CONF-CONTAINER-DEF>
              <SHORT-NAME>AdcGroup</SHORT-NAME>
              <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-
                  CONFIGURATION-CONTAINER>
              <!-- ... -->
            </ECUC-PARAM-CONF-CONTAINER-DEF>
          </SUB-CONTAINERS>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </SUB-CONTAINERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

[8]Please note that in the figures of ECU Configuration Parameter Definition the default value of `upperMultiplicity` and `lowerMultiplicity` is 1.

For the ECU Configuration Value description of this example please refer to section 2.4.7.

2.3.4 Common Configuration Elements

Configuration Containers, Parameters and References have some common attributes which are described in this section.

2.3.4.1 Variant Handling

Variant Handling has been introduced to AUTOSAR in a generic way. The major specification can be found in the AUTOSAR Generic Structure Template [7]. Every element which is subject to variability shall have the stereotype «atpVariation» set.

Variant Handling is used in both areas of ECU Configuration, the ECU Configuration Parameter Definition and ECU Configuration Value description. In this specification the semantics of variant handling are specified at the actual location where they occur individually.

2.3.4.2 Configuration Multiplicity

[TPS_ECUC_02008] Number of occurrences of containers, parameters and references in the ECU Configuration Value description ⌈ To be able to specify how often a specific configuration element (container, parameter or reference) may occur in the ECU Configuration Value description the class EcucDefinitionElement is introduced. With the two attributes lowerMultiplicity and upperMultiplicity the minimum and maximum occurrence of the configuration element is specified. ⌋*(RS_ECUC_00015, RS_ECUC_00055, RS_ECUC_00070)*

[TPS_ECUC_06016] Countably infinite number of containers, parameters and references in the ECU Configuration Value description ⌈ To express a countable infinite number of occurrences of this element the upperMultiplicityInfinite element shall exist and shall be set to true. [9] ⌋*(RS_ECUC_00082)*

[TPS_ECUC_06017] Existence of upperMultiplicityInfinite and upperMultiplicity is mutually exclusive ⌈ The existence of the elements upperMultiplicityInfinite and upperMultiplicity shall be mutually exclusive. ⌋

[TPS_ECUC_02110] Variable lower and upper multiplicity in ECU Configuration Parameter definition ⌈ The attributes lowerMultiplicity, upperMultiplicity and upperMultiplicityInfinite are subject to variant handling (see sec-

[9]Note that in the figures of ECU Configuration Parameter Definition modeled in UML the infinite upper multiplicity is shown as *upperMultiplicity = **

tion 2.3.4.1). The values can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00082)*

In this specification the literals `n` and `m` are used to represent some natural number in order to allow the definition of relations between the `lowerMultiplicity` and the `upperMultiplicity`.

[TPS_ECUC_02009] Expression of optionality of containers, parameters and references ⌈ When there is no multiplicity specified the default is exactly '`1`' meaning the element is mandatory in the ECU Configuration Value description and has to occur exactly once. To express an optional element the `lowerMultiplicity` has to be set to '`0`'. ⌋*(RS_ECUC_00055, RS_ECUC_00070, SRS_BSW_00171)*

Configuration Parameter and Reference definitions with an `upperMultiplicity` > `1` have to be considered with care, since it is not possible to reference to individual parameters. So such multiple occurrences of a parameter in the Value description will just be mere collections, it is neither guaranteed that the order will be preserved nor that individual elements do have a special semantics.

[TPS_ECUC_02010] Multiplicity attributes in ECU Configuration Parameter Model diagrams ⌈ In the specification object diagrams (ECU Configuration Parameter Model) the multiplicity attributes may be omitted if both values are equal to the default value of '`1`'. Otherwise both attributes are shown. ⌋

Class	**EcucDefinitionElement (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Common class used to express the commonalities of configuration parameters, references and containers. If not stated otherwise the default multiplicity is exactly one mandatory occurrence of the specified element.			
Base	ARObject,AtpDefinition,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucCond	EcucConditionSpecification	0..1	aggr	If it evaluates to true the Ecu Parameter definition shall be processed as specified. Otherwise the parameter definition shall be ignored. **Tags:** xml.sequenceOffset=100
ecucValidationCond	EcucValidationCondition	*	aggr	Collection of validation conditions which all need to evaluate to true in order to indicate a valid validation condition of the EcucDefinitionElement.
lowerMultiplicity	PositiveInteger	1	attr	The lower multiplicity of the specified element. 0: optional 1: at least one occurrence n: at least n occurrences atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=110
scope	EcucScopeEnum	0..1	attr	Specifies the scope of this configuration element. **Tags:** xml.sequenceOffset=150

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
upperMultiplicity	PositiveInteger	0..1	attr	The upper multiplicity of the specified element. 0: no occurrence (used for VSMD) 1: at most one occurrence m: at most m occurrences If upperMultiplicity is set than upperMultiplicityInfinite shall not be used. atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=120
upperMultiplicityInfinite	Boolean	0..1	attr	To express an infinite number of occurrences of this element this attribute has to be set to true. If upperMultiplicityInfinite is set than upperMultiplicity shall not be used. atpVariation: [RS_ECUC_00082] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime xml.sequenceOffset=130

Table 2.7: EcucDefinitionElement

Enumeration	**EcucScopeEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible scope settings for a configuration element.
Literal	***Description***
ECU	An element may be shared with other modules.
local	An element is only be applicable for the module it is defined in.

Table 2.8: EcucScopeEnum

[constr_3509] Applicability of `scope` attribute ⌈ The usage of the attribute `scope` is prohibited for `EcucModuleDef` and for sub-classes of `EcucContainerDef` (i.e. `EcucChoiceContainerDef` and `EcucParamConfContainerDef`). ⌋

For examples please refer to figure 2.6 and example 2.7

2.3.4.3 Common Configuration Attributes

Several attributes are available on both, parameters and references. These common attributes are shown in figure 2.9.

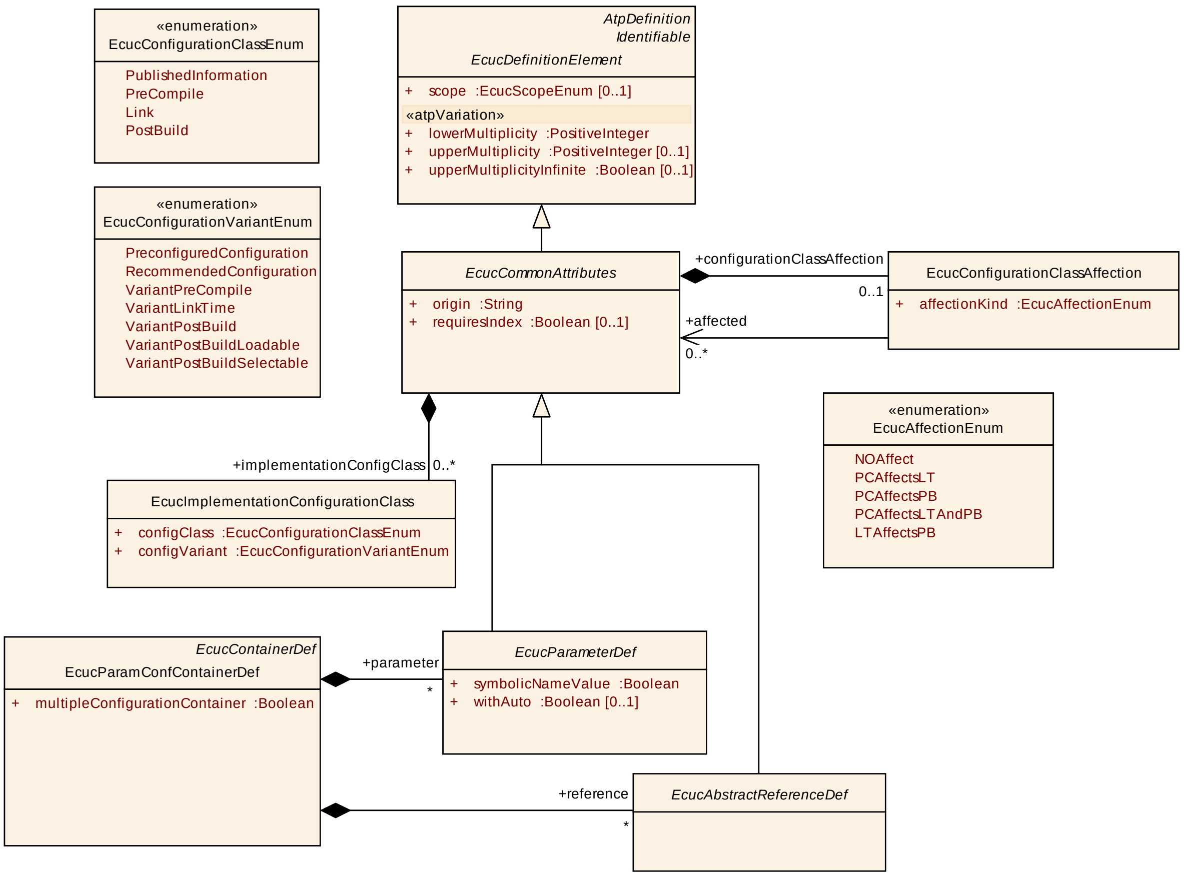
«enumeration»
EcucConfigurationClassEnum
PublishedInformation
PreCompile
Link
PostBuild
«enumeration»
EcucConfigurationVariantEnum
PreconfiguredConfiguration
RecommendedConfiguration
VariantPreCompile
VariantLinkTime
VariantPostBuild
VariantPostBuildLoadable
VariantPostBuildSelectable
AtpDefinition
Identifiable
EcucDefinitionElement
+ scope :EcucScopeEnum [0..1]
«atpVariation»
+ lowerMultiplicity :PositiveInteger
+ upperMultiplicity :PositiveInteger [0..1]
+ upperMultiplicityInfinite :Boolean [0..1]
EcucCommonAttributes
+ origin :String
+ requiresIndex :Boolean [0..1]
+configurationClassAffection
0..1
+affected
0..*
EcucConfigurationClassAffection
+ affectionKind :EcucAffectionEnum
«enumeration»
EcucAffectionEnum
NOAffect
PCAffectsLT
PCAffectsPB
PCAffectsLTAndPB
LTAffectsPB
+implementationConfigClass
0..*
EcucImplementationConfigurationClass
+ configClass :EcucConfigurationClassEnum
+ configVariant :EcucConfigurationVariantEnum
EcucContainerDef
EcucParamConfContainerDef
+ multipleConfigurationContainer :Boolean
+parameter
*
EcucParameterDef
+ symbolicNameValue :Boolean
+ withAuto :Boolean [0..1]
+reference
*
EcucAbstractReferenceDef

Figure 2.9: Common Attributes for parameters and references

Class	**EcucCommonAttributes (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Attributes used by Configuration Parameters as well as References.			
Base	ARObject,AtpDefinition,EcucDefinitionElement,Identifiable,Multilanguage Referrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
configurationClassAffection	EcucConfigurationClassAffection	0..1	aggr	Specifies whether changes on this parameter have some affection on other parameters.
implementationConfigClass	EcucImplementationConfigurationClass	*	aggr	Specifies in which ConfigurationClass this parameter or reference is available for which ConfigurationVariant. This aggregation is optional if the surrounding EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_MODULE_DEFINITION then this aggregation is mandatory. **Tags:** xml.namePlural=IMPLEMENTATION-CONFIG-CLASSES

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
origin	String	1	attr	String specifying if this configuration parameter is an AUTOSAR standardized configuration parameter or if the parameter is hardware- or vendor-specific.
requiresIndex	Boolean	0..1	attr	Used to define whether the value element for this definition shall be provided with an index.

Table 2.9: EcucCommonAttributes

2.3.4.3.1 Parameter Origin

[TPS_ECUC_02015] Origin information in parameter and reference definitions ⌈ Each parameter type has to provide information on its `origin`, which contains a string describing if the parameter is defined in the AUTOSAR standard (`'AUTOSAR_ECUC'`) or if the parameter is defined as a vendor specific parameter (e.g. `'VendorXYZ_v1.3'`). ⌋

Example 2.10

```
<ECUC-INTEGER-PARAM-DEF>
  <SHORT-NAME>ClockRate</SHORT-NAME>
  <ORIGIN>AUTOSAR_ECUC</ORIGIN>
</ECUC-INTEGER-PARAM-DEF>
<ECUC-BOOLEAN-PARAM-DEF>
  <SHORT-NAME>VendorExtensionEnabled</SHORT-NAME>
  <ORIGIN>VendorXYZ_v1.3</ORIGIN>
</ECUC-BOOLEAN-PARAM-DEF>
```

In example 2.10 two parameters are defined, one which belongs to the `AUTOSAR` standard and one which is introduced by the module vendor in a specific version of his own ECU Configuration tools.

2.3.4.3.2 Implementation Configuration Classes

[TPS_ECUC_02016] Configuration class of parameter and reference definitions ⌈ The attribute `implementationConfigClass` provides information what kind of configuration class this parameter shall be implemented for each of the supported configuration variants. The different configuration classes defined within AUTOSAR are[10]: ⌋*(SRS_BSW_00396, RS_ECUC_00012, RS_ECUC_00046, SRS_BSW_00387)*

- **[TPS_ECUC_02070] Configuration class "PubilishedInformation"** ⌈ `PublishedInformation` ⌋

[10]In the XML-Schema the values are represented as `PUBLISHED-INFORMATION`, `PRE-COMPILE`, `LINK`, `POST-BUILD`.

- **[TPS_ECUC_02017] Configuration class "PreCompile"** ⌈ PreCompile ⌋*(RS_ECUC_00047, SRS_BSW_00397)*
- **[TPS_ECUC_02018] Configuration class "Link"** ⌈ Link ⌋*(RS_ECUC_00048, SRS_BSW_00398)*
- **[TPS_ECUC_02019] Configuration class "PostBuild"** ⌈ PostBuild[11] ⌋*(RS_ECUC_00008)*

The element PublishedInformation is used to specify the fact that certain information is fixed even before the pre-compile stage.

[TPS_ECUC_02071] Usage of PublishedInformation configuration class ⌈ If PublishedInformation is selected as configuration class it has to be the same for all configuration variants. ⌋

[constr_3091] Multiplicity of implementationConfigClass ⌈ The multiplicty of the attribute implementationConfigClass shall not exceed the following values:

- 3 for STMD
- 4 for VSMD

⌋

[constr_3091] means that the implementer of the module does not have complete freedom how the configuration classes are chosen for each individual configuration parameter but needs to select one of the specified variants.

[TPS_ECUC_02097] Supported configuration variants in the StMD ⌈ The supported configuration variants in the StMD are[12]: ⌋

- **[TPS_ECUC_02098] StMD Configuration variant "VariantPreCompile"** ⌈ VariantPreCompile ⌋
- **[TPS_ECUC_02099] StMD Configuration variant "VariantLinkTime"** ⌈ VariantLinkTime ⌋
- **[TPS_ECUC_02100] StMD Configuration variant "VariantPostBuild"** ⌈ VariantPostBuild ⌋

[TPS_ECUC_06052] Supported configuration variants in the VSMD ⌈ The supported configuration variants in the VSMD are: ⌋

- **[TPS_ECUC_06053] VSMD Configuration variant "VariantPreCompile"** ⌈ VariantPreCompile ⌋
- **[TPS_ECUC_06054] VSMD Configuration variant "VariantLinkTime"** ⌈ VariantLinkTime ⌋

[11]The configuration classes PostBuildLoadable and PostBuildSelectable are no longer required for the ECU Configuration Parameter Definition because the difference between these two configuration classes is only relevant for the memory mapping of the configuration data during linking.

[12]In the XML-Schema the values are represented as VARIANT-PRE-COMPILE, VARIANT-LINK-TIME, VARIANT-POST-BUILD.

- **[TPS_ECUC_06055] VSMD Configuration variant "VariantPostBuildLoadable"** ⌈ VariantPostBuildLoadable ⌋
- **[TPS_ECUC_06056] VSMD Configuration variant "VariantPostBuildSelectable"** ⌈ VariantPostBuildSelectable ⌋

The mapping of the EcucConfigurationVariantEnum to the EcucConfigurationClassEnum is done using the EcucImplementationConfigurationClass:

Class	**EcucImplementationConfigurationClass**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specifies which ConfigurationClass this parameter has in the individual ConfigurationVariants.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
configClass	EcucConfigurationClassEnum	1	attr	Specifies the ConfigurationClass for the given ConfigurationVariant.
configVariant	EcucConfigurationVariantEnum	1	attr	Specifies the ConfigurationVariant the ConfigurationClass is specified for.

Table 2.10: EcucImplementationConfigurationClass

Enumeration	**EcucConfigurationClassEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible configuration classes for the AUTOSAR configuration parameters.
Literal	***Description***
Link	Link Time: parts of configuration are delivered from another object code file
PostBuild	PostBuildTime: the configuration parameter has to be stored at a known memory location.
PreCompile	PreCompile Time: after compilation a configuration parameter can not be changed any more.
Published Information	PublishedInformation is used to specify the fact that certain information is fixed even before the pre-compile stage.

Table 2.11: EcucConfigurationClassEnum

Enumeration	**EcucConfigurationVariantEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Specifies the possible Configuration Variants used for AUTOSAR BSW Modules.
Literal	***Description***
Preconfigured Configuration	Preconfigured (i.e. fixed) configuration which cannot be changed.
Recommended Configuration	Recommended configuration for a module.
VariantLink Time	Specifies that the BSW Module implementation may use PreCompileTime and LinkTime configuration parameters.
VariantPost Build	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild configuration parameters.

VariantPost BuildLoad-able	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild loadable configuration parameters (supported in the VSMD).
VariantPost BuildSe-lectable	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild selectable configuration parameters (supported in the VSMD).
VariantPre Compile	Specifies that the BSW Module implementation uses only PreCompileTime configuration parameters.

Table 2.12: EcucConfigurationVariantEnum

[TPS_ECUC_02101] `EcucImplementationConfigurationClass` usage ⌈ For each `EcucConfigurationVariantEnum` the `EcucModuleDef` supports there shall be one `EcucImplementationConfigurationClass` element. ⌋

The supported configuration variants of the module are described in section 2.3.2.

[constr_3092] Usage of `configVariant` and `configClass` attributes ⌈ `configVariant` and `configClass` shall always exist as a pair for each existing `EcucImplementationConfigurationClass`. ⌋

[TPS_ECUC_02102] Configuration class selection for parameters and references for supported configuration variants ⌈ Every `EcucImplementationConfigurationClass` specifies which `EcucConfigurationClassEnum` this parameter or reference shall be implemented for the `EcucConfigurationVariantEnum` the `EcucModuleDef` supports. ⌋

The example 2.11 shows how the `EcucImplementationConfigurationClass` is provided in XML for three configuration variants of some module. The integer configuration parameter `SignalSize` shall be implemented as a `PRE-COMPILE` parameter for the configuration variants `VARIANT-PRE-COMPILE` and `VARIANT-LINK-TIME`. It shall be `POST-BUILD` for the configuration variant `VARIANT-POST-BUILD`.

Example 2.11

```
<ECUC-INTEGER-PARAM-DEF>
  <SHORT-NAME>SignalSize</SHORT-NAME>
  <IMPLEMENTATION-CONFIG-CLASSES>
    <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
      <CONFIG-CLASS>PRE-COMPILE</CONFIG-CLASS>
      <CONFIG-VARIANT>VARIANT-PRE-COMPILE</CONFIG-VARIANT>
    </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
    <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
      <CONFIG-CLASS>PRE-COMPILE</CONFIG-CLASS>
      <CONFIG-VARIANT>VARIANT-LINK-TIME</CONFIG-VARIANT>
    </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
    <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
      <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
      <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
    </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
  </IMPLEMENTATION-CONFIG-CLASSES>
</ECUC-INTEGER-PARAM-DEF>
```

The configuration tools are now able to derive the configuration class of each configuration parameter and reference from the ECU Configuration Parameter Definition XML file [9].

2.3.4.3.3 Configuration Class Affection

The configuration class affection is deprecated and will be removed in future versions!

The EcucConfigurationClassAffection is used to describe whether a specific configuration parameter is affecting any other configuration parameters in the ECU Configuration Value description and in which configuration phase this affection occurs. The actual affection will be described in the Vendor Specific Module Definition based on the actual implementation.

The possible values for the affectionKind of a EcucConfigurationClassAffection are[13]:

- **[TPS_ECUC_02076] "NOAffect" affection** ⌈ NOAffect ⌋
- **[TPS_ECUC_02077] "PCAffectsLT" affection** ⌈ PCAffectsLT ⌋
- **[TPS_ECUC_02078] "PCAffectsPB" affection** ⌈ PCAffectsPB ⌋
- **[TPS_ECUC_02079] "PCAffectsLTAndPB" affection** ⌈ PCAffectsLTAndPB ⌋
- **[TPS_ECUC_02080] "LTAffectsPB" affection** ⌈ LTAffectsPB ⌋

[13] In the XML-Schema the values are represented as NO-AFFECT, PC-AFFECTS-LT, PC-AFFECTS-PB, PC-AFFECTS-LT-AND-PB, LT-AFFECTS-PB.

[TPS_ECUC_02081] Parameters or references which are affected may be referenced with the `affected` reference ⌈ The reference `affected` from the `EcucConfigurationClassAffection` to any subclass of `EcucCommonAttributes` is used to define which actual parameters and references are affected. ⌋

Class	**EcucConfigurationClassAffection**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specifies in the "VendorSpecificModuleDefinition" whether changes on this parameter do affect other parameters in a later configuration step.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
affected	EcucCommonAttributes	*	ref	Optional reference to parameters or references which are affected by the ConfigurationClassAffection.
affectionKind	EcucAffectionEnum	1	attr	Specifies which affect do changes in this parameter have on other parameters. This attribute is deprecated and will be removed in future versions. **Tags:** atp.Status=obsolete

Table 2.13: EcucConfigurationClassAffection

2.3.5 Parameter Definition

[TPS_ECUC_02013] Definition of parameters within a `EcucParamConfContainerDef` ⌈ Parameters are defined within a `EcucParamConfContainerDef` using an aggregation with the role name `parameter` at the parameter side. ⌋

[TPS_ECUC_02014] Parameter types ⌈ The possible parameter types are specified using one of the specialized classes derived from `EcucParameterDef`. The `EcucParameterDef` does inherit from `Identifiable`, `EcucCommonAttributes` and `EcucDefinitionElement`. ⌋*(SRS_BSW_00391, SRS_BSW_00392)*

The available parameter types are shown in figure 2.10.

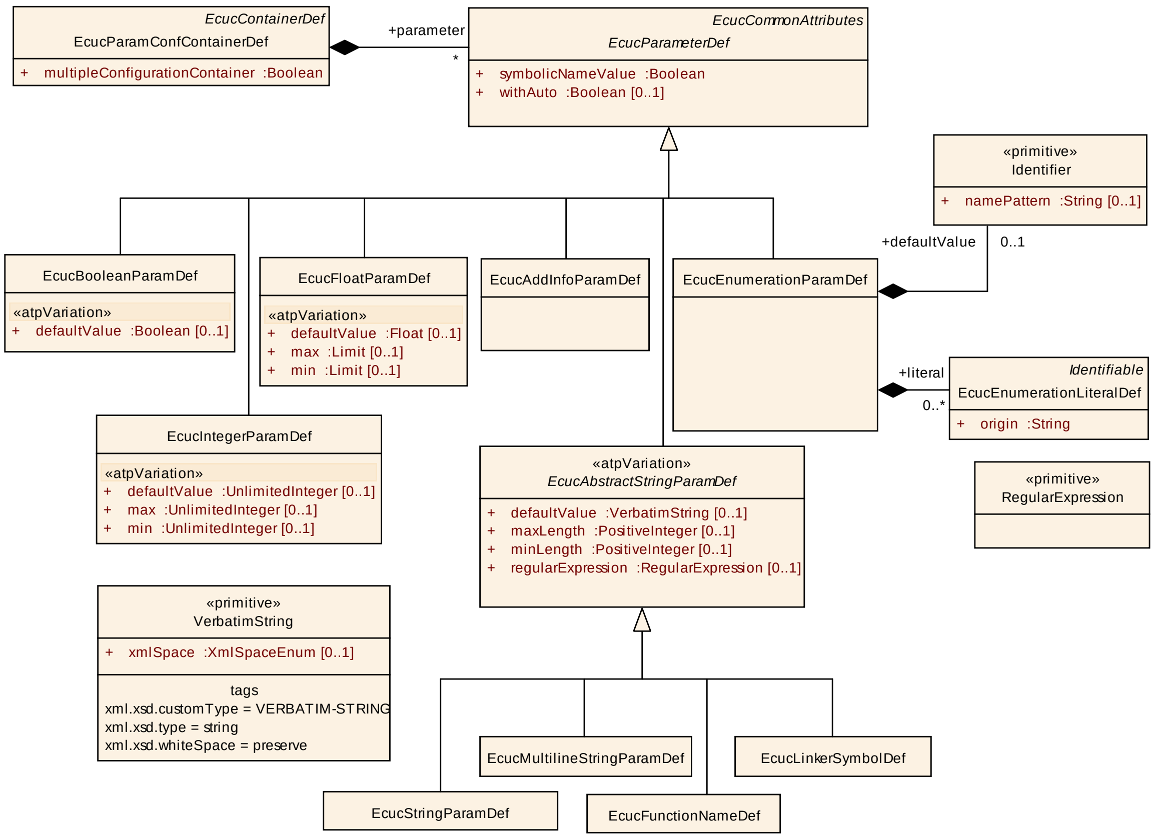
EcucContainerDef
EcucParamConfContainerDef
+ multipleConfigurationContainer :Boolean
+parameter
*
EcucCommonAttributes
EcucParameterDef
+ symbolicNameValue :Boolean
+ withAuto :Boolean [0..1]
«primitive»
Identifier
+ namePattern :String [0..1]
+defaultValue
0..1
EcucBooleanParamDef
«atpVariation»
+ defaultValue :Boolean [0..1]
EcucFloatParamDef
«atpVariation»
+ defaultValue :Float [0..1]
+ max :Limit [0..1]
+ min :Limit [0..1]
EcucAddInfoParamDef
EcucEnumerationParamDef
+literal
0..*
Identifiable
EcucEnumerationLiteralDef
+ origin :String
«primitive»
RegularExpression
EcucIntegerParamDef
«atpVariation»
+ defaultValue :UnlimitedInteger [0..1]
+ max :UnlimitedInteger [0..1]
+ min :UnlimitedInteger [0..1]
«atpVariation»
EcucAbstractStringParamDef
+ defaultValue :VerbatimString [0..1]
+ maxLength :PositiveInteger [0..1]
+ minLength :PositiveInteger [0..1]
+ regularExpression :RegularExpression [0..1]
«primitive»
VerbatimString
+ xmlSpace :XmlSpaceEnum [0..1]
tags
xml.xsd.customType = VERBATIM-STRING
xml.xsd.type = string
xml.xsd.whiteSpace = preserve
EcucMultilineStringParamDef
EcucLinkerSymbolDef
EcucStringParamDef
EcucFunctionNameDef

Figure 2.10: Class diagram for parameter definition

Class	**EcucParameterDef (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Abstract class used to define the similarities of all ECU Configuration Parameter types defined as subclasses.			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
derivation	EcucDerivationSpecification	0..1	aggr	A derivation of a Configuration Parameter value can be specified by an informal Calculation Formula or by a formal language that can be used to specify the computational rules.
symbolicNameValue	Boolean	1	attr	Specifies that this parameter's value is used, together with the aggregating container, to derive a symbolic name definition. See chapter "Representation of Symbolic Names" in Ecuc specification for more details.

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
withAuto	Boolean	0..1	attr	Specifies whether it shall be allowed on the value side to specify this parameter value as "AUTO". If withAuto is "true" it shall be possible to set the "isAutoValue" attribute of the respective parameter to "true". This means that the actual value will not be considered during ECU Configuration but will be (re-)calculated by the code generator and stored in the value attribute afterwards. These implicit updated values might require a re-generation of other modules which reference these values. If withAuto is "false" it shall not be possible to set the "isAutoValue" attribute of the respective parameter to "true". If withAuto is not present the default is "false".

Table 2.14: EcucParameterDef

[TPS_ECUC_08001] Configuration class of parameters and references within `postBuildChangeable` containers. ⌈ `EcucParameterDef`s and `EcucAbstractReferenceDef`s within an `EcucParamConfContainerDef` and its aggregated `EcucParamConfContainerDef`s in the role `subContainer` which have the attribute `postBuildChangeable` set to `true` may have either `PreCompile`, `Link` or `Post-Build` configuration class. ⌋

The use-case for the attribute `symbolicNameValue` is described in section 2.3.6.5.

The use-case for the attribute `withAuto` is described in section 3.4.1.

In the next sections the different parameter types will be described in detail. The examples for the individual parameters are taken from figure 2.11.

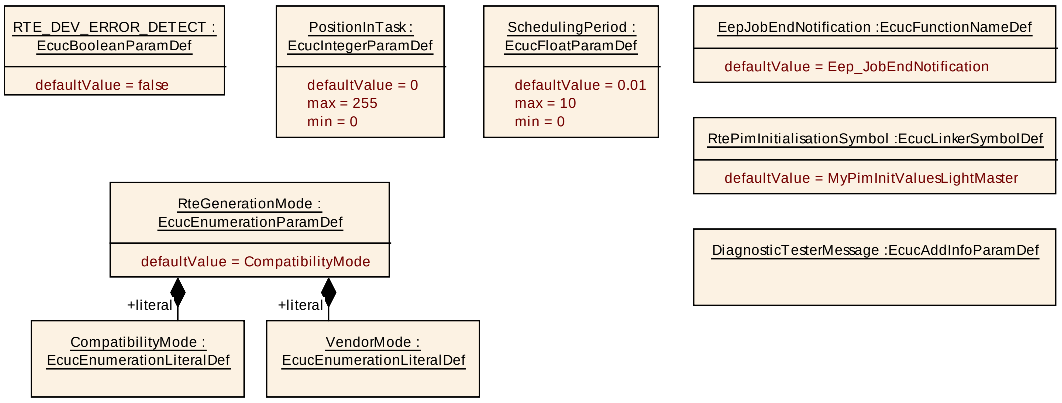
RTE_DEV_ERROR_DETECT : EcucBooleanParamDef
defaultValue = false
PositionInTask : EcucIntegerParamDef
defaultValue = 0
max = 255
min = 0
SchedulingPeriod : EcucFloatParamDef
defaultValue = 0.01
max = 10
min = 0
EepJobEndNotification :EcucFunctionNameDef
defaultValue = Eep_JobEndNotification
RtePimInitialisationSymbol :EcucLinkerSymbolDef
defaultValue = MyPimInitValuesLightMaster
RteGenerationMode : EcucEnumerationParamDef
defaultValue = CompatibilityMode
DiagnosticTesterMessage :EcucAddInfoParamDef
+literal
+literal
CompatibilityMode : EcucEnumerationLiteralDef
VendorMode : EcucEnumerationLiteralDef

Figure 2.11: Example of parameter definitions using different types

2.3.5.1 Boolean Type

[TPS_ECUC_02026] EcucBooleanParamDef properties ⌈ With the EcucBooleanParamDef parameter a 'true' or 'false' parameter can be specified. The only additional attribute is the defaultValue which may be specified while defining the parameter. ⌋

[TPS_ECUC_02127] Possible values for EcucBooleanParamDef parameters ⌈ The alternative representation of 'true' and 'false' are '1' and '0' which allows the usage of a numerical representation of the value in order to be computed in the variant handling. ⌋

This parameter is also to be used for other 'boolean'-type configuration parameters which might result into values like:

- ON / OFF
- ENABLE / DISABLE
- 1 / 0

Please note that the representation of an boolean parameter value or an attribute which supports ≪atpVariation≫ as true / false already requires the processing of the BooleanLiteral true /false by the formula processor.

On the ECU Configuration Value description side boolean parameter values are represented as EcucNumericalParamValues (see chapter 2.4.4.2). The attribute "value" in the EcucNumericalParamValue supports ≪atpVariation≫ and therefore the BooleanLiteral true /false is supported by the formula language as well. Please note that true evaluates to 1 and false to 0 (see [7] for more details).

[TPS_ECUC_02111] Variable default value in EcucBooleanParamDef ⌈ The attribute defaultValue of EcucBooleanParamDef is subject to variant handling (see section 2.3.4.1). The value can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00083)*

Class	**EcucBooleanParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Boolean. Allowed values are true and false. **Tags:** xml.sequenceOffset=0			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,Ecuc ParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValu e	Boolean	0..1	attr	Default value of the boolean configuration parameter. atpVariation: [RS_ECUC_00083] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime

Table 2.15: EcucBooleanParamDef

Example 2.12 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.38.

Example 2.12

```
<ECUC-BOOLEAN-PARAM-DEF>
  <SHORT-NAME>RTE_DEV_ERROR_DETECT</SHORT-NAME>
  <DEFAULT-VALUE>false</DEFAULT-VALUE>
</ECUC-BOOLEAN-PARAM-DEF>
```

2.3.5.2 Integer Type

[TPS_ECUC_02027] `EcucIntegerParamDef` properties ⌈ With the `EcucIntegerParamDef` parameter a signed/unsigned whole number can be specified. With the additional attributes `min` and `max` the range of this parameters values in the ECU Configuration Value description can be limited[14]. Also the `defaultValue` can be specified. ⌋*(SRS_BSW_00393)*

[TPS_ECUC_02114] Variable default value in `EcucIntegerParamDef` ⌈ The attribute `defaultValue` of `EcucIntegerParamDef` is subject to variant handling (see section 2.3.4.1). The value can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00083)*

[TPS_ECUC_02116] Variable min, max values in `EcucIntegerParamDef` ⌈ The attributes `min` and `max` of `EcucIntegerParamDef` are subject to variant handling (see section 2.3.4.1). The values can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00084)*

[14]The `min` and `max` values are defined optional, however in the 'Vendor Specific Module Definition' these values are mandatory.

The value range of the EcucIntegerParamDef has two use-cases, signed and unsigned, which both have to fit in a 64-bit number space.

[TPS_ECUC_02072] Signed EcucIntegerParamDef value range ⌈ If a signed value is represented the min value can be down to −9223372036854775808 (in hex 0x8000000000000000) and the max value can be up to 9223372036854775807 (in hex 0x7FFFFFFFFFFFFFFF). ⌋

[TPS_ECUC_02073] Unsigned EcucIntegerParamDef value range ⌈ If an unsigned value is represented the min value can be down to 0 and the max value can be up to 18446744073709551615 (in hex 0xFFFFFFFFFFFFFFFF). ⌋

[TPS_ECUC_06032] Min and max values in EcucIntegerParamDef ⌈ The max value must be equal or bigger than the min value and the min value must be equal or less than the max value. ⌋

[TPS_ECUC_02074] +/- sign in the EcucNumericalParamValue is optional ⌈ If the optional +/- sign in the EcucNumericalParamValue is omitted, "+" is assumed. ⌋

[TPS_ECUC_03040] The value of an EcucNumericalParamValue shall be unambiguously an integer value ⌈ The value of an EcucNumericalParamValue shall be specified such that it is unambiguously an integer value. In particular the result of the NumericalValueVariationPoint shall yield an integer, not a float. ⌋

Class	**EcucIntegerParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Integer. **Tags:** xml.sequenceOffset=0			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,Ecuc ParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValu e	UnlimitedInteger	0..1	attr	Default value of the integer configuration parameter. atpVariation: [RS_ECUC_00083] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
max	UnlimitedInteger	0..1	attr	Max value allowed for the parameter defined. atpVariation: [RS_ECUC_00084] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
min	UnlimitedInteger	0..1	attr	Min value allowed for the parameter defined. atpVariation: [RS_ECUC_00084] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime

Table 2.16: EcucIntegerParamDef

Example 2.13 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.39.

Example 2.13

```
<ECUC-INTEGER-PARAM-DEF>
  <SHORT-NAME>PositionInTask</SHORT-NAME>
  <DEFAULT-VALUE>0</DEFAULT-VALUE>
  <MAX>255</MAX>
  <MIN>0</MIN>
</ECUC-INTEGER-PARAM-DEF>
```

2.3.5.3 Float Type

[TPS_ECUC_02028] **`EcucFloatParamDef`** **properties** ⌈ To be able to specify parameters with floating number values the `EcucFloatParamDef` can be used.

The additional attributes `min`, `max` and `defaultValue` can be specified as well[15].
⌋*(SRS_BSW_00393)*

[TPS_ECUC_02115] Variable default value in `EcucFloatParamDef` ⌈ The attribute `defaultValue` of `EcucFloatParamDef` is subject to variant handling (see section 2.3.4.1). The value can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00083)*

[TPS_ECUC_02117] Variable min, max values in `EcucFloatParamDef` ⌈ The attributes `min` and `max` of `EcucFloatParamDef` are subject to variant handling (see section 2.3.4.1). The values can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00084)*

[TPS_ECUC_06033] Min and max values in `EcucFloatParamDef` ⌈ The `max` value must be equal or bigger than the `min` value and the `min` value must be equal or less than the `max` value. ⌋

[TPS_ECUC_06034] Special float values ⌈ The notation of the special float values "Not a Number" and positive/negative "infinity" shall be:

- NaN
- INF
- -INF

⌋

[TPS_ECUC_02075] Representation of `EcucFloatParamDefs` ⌈ For the representation the `IEEE double-precision 64-bit floating point` of the *IEEE 754-1985* standard [10] is used. ⌋

Float values that exist on a target ECU which does not support 64 bit have to be converted to the nearest approximation of the value in float 32 for the target. In AUTOSAR XML the value shall be kept in 64 bit representation.

Class	**EcucFloatParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Float. **Tags:** xml.sequenceOffset=0			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	Float	0..1	attr	Default value of the float configuration parameter. atpVariation: [RS_ECUC_00083] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime

[15]The `min` and `max` values are defined optional, however in the 'Vendor Specific Module Definition' these values are mandatory.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
max	Limit	0..1	ref	Max value allowed for the parameter defined. atpVariation: [RS_ECUC_00084] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime
min	Limit	0..1	ref	Min value allowed for the parameter defined. atpVariation: [RS_ECUC_00084] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=codeGenerationTime

Table 2.17: EcucFloatParamDef

Example 2.14 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.40.

Example 2.14

```
<ECUC-FLOAT-PARAM-DEF>
  <SHORT-NAME>SchedulingPeriod</SHORT-NAME>
  <ORIGIN>AUTOSAR_ECUC</ORIGIN>
  <DEFAULT-VALUE>NaN</DEFAULT-VALUE>
  <MAX>10</MAX>
  <MIN>0</MIN>
</ECUC-FLOAT-PARAM-DEF>
<ECUC-LINKER-SYMBOL-DEF>
```

2.3.5.4 String Parameter

[TPS_ECUC_02029] Subclasses of `EcucAbstractStringParamDef` ⌈ The subclasses of the class `EcucAbstractStringParamDef` provide means to specify strings in the ECUC Value description. Additionally an optional `defaultValue` can be provided. ⌋

[TPS_ECUC_02112] Variable default value in `EcucAbstractStringParamDef` ⌈ The attribute `defaultValue` of `EcucAbstractStringParamDef` and its subclasses is subject to variant handling (see section 2.3.4.1). The value can be computed using the variant handling mechanism. ⌋*(RS_ECUC_00083)*

[TPS_ECUC_06035] Regular expression ⌈ The regular expression is provided according to the Generic Structure Template [7]. ⌋

Class	≪**atpVariation**≫ **EcucAbstractStringParamDef (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Abstract class that is used to collect the common properties for StringParamDefs, LinkerSymbolDef, FunctionNameDef and MultilineStringParamDefs. atpVariation: [RS_ECUC_0083] **Tags:** vh.latestBindingTime=codeGenerationTime			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	VerbatimString	0..1	ref	Default value of the string configuration parameter.
maxLength	PositiveInteger	0..1	attr	Max length allowed for this string.
minLength	PositiveInteger	0..1	attr	Min length allowed for this string.
regularExpression	RegularExpression	0..1	attr	This represents the regular expression which shall be used to validate the string parameter value.

Table 2.18: EcucAbstractStringParamDef

Class	≪**atpVariation**≫ **EcucStringParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for String. **Tags:** xml.sequenceOffset=0			
Base	ARObject,AtpDefinition,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.19: EcucStringParamDef

Class	≪**atpVariation**≫ **EcucMultilineStringParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for multiline Strings (including "carriage return").			
Base	ARObject,AtpDefinition,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.20: EcucMultilineStringParamDef

2.3.5.5 Linker Symbol Parameter

[TPS_ECUC_06006] `EcucLinkerSymbolDef` properties ⌈ When a parameter represents a linker symbol in the configured software the `EcucLinkerSymbolDef` shall be used. The actual values of the symbol defined will be specified by the implementing software and are not subject to configuration. ⌋

[TPS_ECUC_02030] Programming language identifier limitations ⌈ The restriction on the `defaultValue` and the value of a `EcucLinkerSymbolDef` and its subclass are the common programming language identifier limitations: start with a letter or a special character (sc) followed by upper- and lower-case letters, digits and special characters:

```
identifier := (letter | sc) ( letter | digit | sc )*
```

where letter is [a-z] or [A-Z], sc is (_ | . | $ | %) and digit is [0-9]. ⌋

[TPS_ECUC_02031] Restriction on the length of `EcucLinkerSymbolDef` values and defaultValue ⌈ The restriction on the length of the default value and the value of a `EcucLinkerSymbolDef` is set to 255 characters. ⌋

The class `EcucLinkerSymbolDef` does not introduce any additional attributes.

The `EcucLinkerSymbolDef` in fact represents the C-compiler symbol which later is translated into a linker symbol. With this element the usage of the `external` declaration of symbols (e.g. variables, constants) is possible.

Class	≪**atpVariation**≫ **EcucLinkerSymbolDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Linker Symbol Names like those used to specify memory locations of variables and constants.			
Base	ARObject,AtpDefinition,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.21: EcucLinkerSymbolDef

Example 2.15 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.35.

Example 2.15

```
<ECUC-LINKER-SYMBOL-DEF>
  <SHORT-NAME>RtePimInitializationSymbol</SHORT-NAME>
  <ECUC-LINKER-SYMBOL-DEF-VARIANTS>
    <ECUC-LINKER-SYMBOL-DEF-CONDITIONAL>
      <DEFAULT-VALUE>MyPimInitValuesLightMaster</DEFAULT-VALUE>
    </ECUC-LINKER-SYMBOL-DEF-CONDITIONAL>
  </ECUC-LINKER-SYMBOL-DEF-VARIANTS>
</ECUC-LINKER-SYMBOL-DEF>
```

2.3.5.6 Function Name Parameter

[TPS_ECUC_02033] EcucFunctionNameDef properties ⌈ When a parameter represents a function name in the configured software the EcucFunctionNameDef shall be used. With this feature functions (like callbacks) can be specified. The class EcucFunctionNameDef does not introduce any additional attributes. ⌋

Class	≪atpVariation≫ **EcucFunctionNameDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Function Names like those used to specify callback functions.			
Base	ARObject,AtpDefinition,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.22: EcucFunctionNameDef

Example 2.16 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.36.

Example 2.16

```
<ECUC-FUNCTION-NAME-DEF>
  <SHORT-NAME>EepJobEndNotification</SHORT-NAME>
  <ECUC-FUNCTION-NAME-DEF-VARIANTS>
    <ECUC-FUNCTION-NAME-DEF-CONDITIONAL>
      <DEFAULT-VALUE>Eep_JobEndNotification</DEFAULT-VALUE>
    </ECUC-FUNCTION-NAME-DEF-CONDITIONAL>
  </ECUC-FUNCTION-NAME-DEF-VARIANTS>
</ECUC-FUNCTION-NAME-DEF>
```

[TPS_ECUC_06075] EcucFunctionNameDef shall represent a valid C Identifier ⌈ The defaultValue and the value of a EcucFunctionNameDef shall follow the pattern [a-zA-Z_][a-zA-Z0-9_]* defined in the context of the CIdentifier. ⌋

2.3.5.7 Enumeration Parameter

[TPS_ECUC_02034] EcucEnumerationParamDef properties ⌈ When the parameter can be one choice of several possibilities the EcucEnumerationParamDef shall be used. It defines the parameter that will hold the actual value and may also define the defaultValue for the enumeration. ⌋

The specification of variable default value for the enumeration is currently not supported.

Class	**EcucEnumerationParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Enumeration. **Tags:** xml.sequenceOffset=0			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
defaultValue	Identifier	0..1	aggr	Default value of the enumeration configuration parameter. This string needs to be one of the literals specified for this enumeration.
literal	EcucEnumerationLiteralDef	*	aggr	Aggregation on the literals used to define this enumeration parameter. This aggregation is optional if the surrounding EcucModuleDef has the category STANDARDIZED_MODULE_DEFINITION. If the category attribute of the EcucModuleDef is set to VENDOR_SPECIFIC_MODULE_DEFINITION then this aggregation is mandatory.

Table 2.23: EcucEnumerationParamDef

2.3.5.8 Enumeration Literal Definition

[TPS_ECUC_02035] Available choices of `EcucEnumerationParamDef`s are defined by aggregated `literals` ⌈ To provide the available choices for the `EcucEnumerationParamDef` the `EcucEnumerationLiteralDef` is used. For each available choice there needs to be one `literal` defined. ⌋

[TPS_ECUC_02036] The shortName of an `EcucEnumerationLiteralDef` is used to define the literal ⌈ For the text used to define the `EcucEnumerationLiteralDef` no additional attribute is needed because the `shortName` inherited from `Identifiable` is used to define the `literal`s. ⌋

[TPS_ECUC_02054] Allowed literal strings ⌈ For the allowed string in `shortName` the restrictions apply as defined in the Generic Structure Template [7], in the primitive `Identifier`. ⌋

This basically restricts the `shortName` to only containing the characters `[a-zA-Z][a-zA-Z0-9_]` and have a maximum length of 128 characters. If a more human readable text shall be provided the `longName` can be used which has much more freedom. This requires that configuration tools will show the optional `longName` to the users, see also requirement [TPS_ECUC_02088].

The relationship between the `EcucEnumerationParamDef` and the available `EcucEnumerationLiteralDef` is established using aggregations with the role name `literal` at the side of the `EcucEnumerationLiteralDef`.

[TPS_ECUC_02131] Origin information in literal definitions ⌈ Each `EcucEnumerationLiteralDef` has to provide information on its `origin`, which contains a string describing if the parameter is defined in the AUTOSAR standard (`'AUTOSAR_ECUC'`) or if the parameter is defined as a vendor specific parameter (e.g. `'VendorXYZ_v1.3'`). ⌋

Class	**EcucEnumerationLiteralDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for enumeration literals definition.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucCond	EcucConditionSpecification	0..1	aggr	If it evaluates to true the literal definition shall be processed as specified. Otherwise the literal definition shall be ignored.
origin	String	1	attr	String specifying if this literal is an AUTOSAR standardized literal or if the literal is vendor-specific.

Table 2.24: EcucEnumerationLiteralDef

Example 2.17 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.37.

Example 2.17

```
<ECUC-ENUMERATION-PARAM-DEF>
  <SHORT-NAME>RteGenerationMode</SHORT-NAME>
  <LITERALS>
    <ECUC-ENUMERATION-LITERAL-DEF>
      <SHORT-NAME>CompatibilityMode</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Generate in Compatibility Mode</L-4>
      </LONG-NAME>
    </ECUC-ENUMERATION-LITERAL-DEF>
    <ECUC-ENUMERATION-LITERAL-DEF>
      <SHORT-NAME>VendorMode</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">Generate in Vendor Mode</L-4>
      </LONG-NAME>
    </ECUC-ENUMERATION-LITERAL-DEF>
  </LITERALS>
</ECUC-ENUMERATION-PARAM-DEF>
```

2.3.5.9 AddInfo

[TPS_ECUC_02118] EcucAddInfoParamDef properties ⌈ The parameter EcucAddInfoParamDef is used to specify the need for formated text in the ECU Configuration Value description. The specification of the details on formated text can be found in the AUTOSAR Generic Structure Template [7]. ⌋

Class	**EcucAddInfoParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration Parameter Definition for the specification of formatted text in the ECU Configuration Parameter Description.			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.25: EcucAddInfoParamDef

Example 2.18 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 2.41.

Example 2.18

```
<ECUC-ADD-INFO-PARAM-DEF>
  <SHORT-NAME>DiagnosticTesterMessage</SHORT-NAME>
</ECUC-ADD-INFO-PARAM-DEF>
```

2.3.6 References in Parameter Definition

There are five kinds of references available for the definition of configuration parameters referring to other entities.

- Reference to other configuration containers within the ECU Configuration Value description (see section 2.3.6.1).
- A choice in the referenced configuration container can be specified and the ECU Configuration Value description has the freedom (with restrictions) to choose to which target type the reference is pointing to (see section 2.3.6.2).
- Entities outside the ECU Configuration Value description can be referenced when they have been specified in a different AUTOSAR Template (see section 2.3.6.3).
- Entities outside the ECU Configuration Value description can be referenced using the `instanceRef` semantics defined in the Generic Structure Template [7] (see section 2.3.6.4).
- A container can be referenced to achieve a symbolic name semantics (see section 2.3.6.5).

The metamodel of those references is shown in figure 2.12.

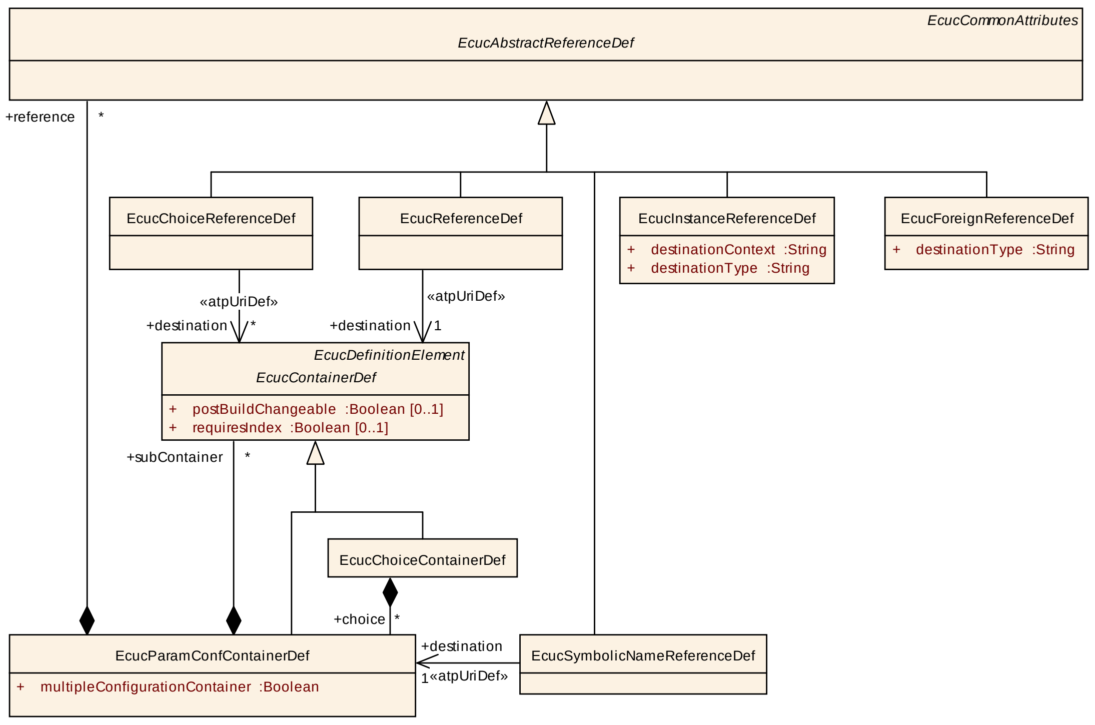

Figure 2.12: Class diagram for parameter references

[TPS_ECUC_02037] EcucAbstractReferenceDef properties ⌈ The abstract class EcucAbstractReferenceDef is used to specify the common parts of all reference definitions. EcucAbstractReferenceDef is an Identifiable so it is mandatory to give each reference definition a name. Also EcucAbstractReferenceDef is inheriting from EcucDefinitionElement so for each reference definition it can be specified how many such references might be present in the same configuration container later in the ECU Configuration Value description. ⌋

Class	**EcucAbstractReferenceDef (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Common class to gather the attributes for the definition of references.			
Base	ARObject,AtpDefinition,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 2.26: EcucAbstractReferenceDef

2.3.6.1 Reference

[TPS_ECUC_02039] References between containers are established with the EcucReferenceDef ⌈ The EcucReferenceDef is used to establish references from one EcucParamConfContainerDef to one other specific EcucParamConf-

ContainerDef or EcucChoiceContainerDef within the same ECU Configuration Value description. For this purpose an object representing the reference has to be used. ⌋*(RS_ECUC_00072, SRS_BSW_00395)*

[TPS_ECUC_02038] Destination of EcucReferenceDef and EcucChoiceReferenceDefs is the EcucContainerDef ⌈ The destination for the EcucReferenceDef and the EcucChoiceReferenceDef is both the EcucContainerDef. So it is not possible to reference to a specific EcucParameterDef, EcucReferenceDef or EcucModuleDef. ⌋

The reason is that there is no use-case where a direct reference to a parameter would be needed.

Class	**EcucReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify references within the ECU Configuration Description between parameter containers.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
destination	EcucContainerDef	1	ref	Exactly one reference to a parameter container is allowed as destination. **Stereotypes:** atpUriDef

Table 2.27: EcucReferenceDef

The role name at the EcucReferenceDef has to be reference and the role name at the referenced container has to be destination (see figure 2.13 for an example).

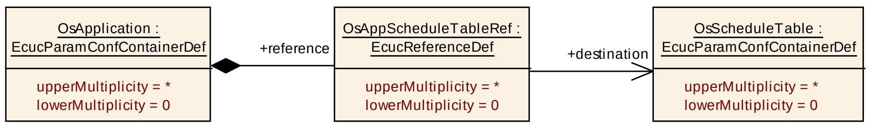

Figure 2.13: Example of an object diagram for a reference

In the example in figure 2.13 the 'OsApplication' is defined to contain references to the 'OsScheduleTable'. The references are called 'OsAppScheduleTableRef' and there can be several such references in the actual ECU Configuration Value description document. For the multiplicity of references the multiplicity definition on the EcucReferenceDef are relevant (in the example the lowerMultiplicity is '0' and the upperMultiplicity is '*'). The multiplicity of the referenced container is not considered for references.

In the ECU Configuration Parameter Definition XML file the destination has to be identified unambiguously because the names of configuration parameters are not required to be unique throughout the whole ECU Configuration Parameter Definition. So there might be a parameter defined in the CAN-Driver with the same name as one parameter defined in the ADC-Driver. For this reason the containment hierarchy of the referenced configuration parameter has to be denoted in the definition XML file, as shown in example 2.19. In this example the referenced parameter will be found in the definition of the Os module directly in the AUTOSARParameterDefinition. The corresponding ECUC Value description XML file extract is shown in example 2.42.

Example 2.19

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>OsApplication</SHORT-NAME>
  <REFERENCES>
    <ECUC-REFERENCE-DEF>
      <SHORT-NAME>OsAppScheduleTableRef</SHORT-NAME>
      <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Os/OsScheduleTable</DESTINATION-REF>
    </ECUC-REFERENCE-DEF>
  </REFERENCES>
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

2.3.6.2 Choice Reference

[TPS_ECUC_02040] EcucChoiceReferenceDef properties ⌈ With the EcucChoiceReferenceDef it is possible to define one reference where the destination is specified to be one of several possible kinds. To be able to define such a choice an object of the class EcucChoiceReferenceDef has to be aggregated in a container with the role name reference at the EcucChoiceReferenceDef object. The destinations of a EcucChoiceReferenceDef may be EcucParamConfContainerDef and EcucChoiceContainerDef. ⌋

Class	**EcucChoiceReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify alternative references where in the ECU Configuration description only one of the specified references will actually be used.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
destination	EcucContainerDef	*	ref	All the possible parameter containers for the reference are specified. **Stereotypes:** atpUriDef

Table 2.28: EcucChoiceReferenceDef

All the available choices are connected via associations with the role name destination at the referenced object (see example in figure 2.14).

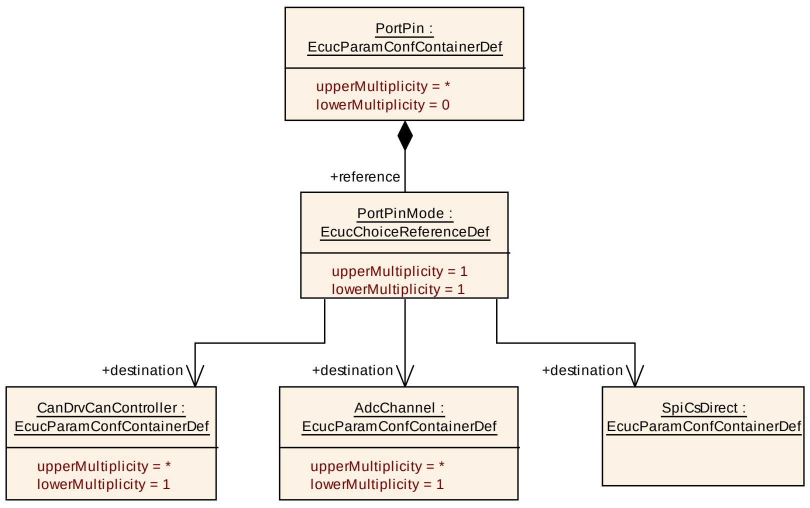

Figure 2.14: Example of an object diagram for a choice reference

In this example an actual instance of the 'PortPinMode' container can reference one of the three defined containers. Once again the multiplicity is defined by the `EcucChoiceReferenceDef` (here the default '1' for lower and upper) and the multiplicities of the referenced containers are not relevant for choice references.

Also the destination needs to be defined unambiguously in the ECU Configuration Parameter Definition XML file like shown in example 2.20. The corresponding ECUC Value description XML file extract is shown in example 2.43.

Example 2.20

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>PortPin</SHORT-NAME>
  <REFERENCES>
    <ECUC-CHOICE-REFERENCE-DEF>
      <SHORT-NAME>PortPinMode</SHORT-NAME>
      <DESTINATION-REFS>
        <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
           EcucDefs/Can/CanDrvCanController</DESTINATION-REF>
        <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
           EcucDefs/Adc/AdcChannel</DESTINATION-REF>
        <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
           EcucDefs/Spi/SpiCsDirect</DESTINATION-REF>
      </DESTINATION-REFS>
    </ECUC-CHOICE-REFERENCE-DEF>
  </REFERENCES>
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

In the ECU Configuration Value description the actual choice will be taken and there will be only one reference destination left[16].

[16]The `EcucDefinitionElement` is used to specify the possible occurrences of each reference later in the ECU Configuration Description. The `EcucChoiceReferenceDef` specifies multiple possible destinations for one reference but later in the ECU Configuration Value description there can only be exactly one destination described. So the freedom of multiple destinations is only available on the

2.3.6.3 Foreign Reference

[TPS_ECUC_02041] EcucForeignReferenceDef properties ⌈ To be able to reference to descriptions of other AUTOSAR templates the parameter definition EcucForeignReferenceDef is used. With the attribute destinationType the type of the referenced entity has to be specified. ⌋

Class	**EcucForeignReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify a reference to an XML description of an entity described in another AUTOSAR template.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
destinationType	String	1	attr	The type in the AUTOSAR Metamodel to which instance this reference is allowed to point to.

Table 2.29: EcucForeignReferenceDef

[TPS_ECUC_02042] Specification of the destinationType in a EcucForeignReferenceDef ⌈ Since the AUTOSAR Generic Structure Template [7] requires the class names of all Identifiables to be unique within the AUTOSAR 'M2::AUTOSAR Templates' metamodel, it is sufficient to provide only the actual class name of the referenced class in the destinationType, as shown in example 2.21. ⌋

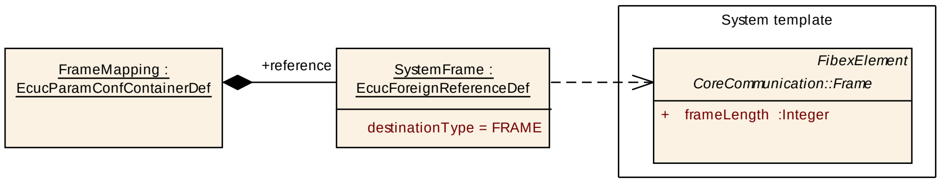

Figure 2.15: Example of an object diagram for a foreign reference

In the example in figure 2.15 the reference is defined to be pointing to a description of a Frame. The Frame is defined in the System Template metamodel [2] and is derived from Identifiable. The corresponding ECUC Value description XML file extract is shown in example 2.44.

definition of references, if several containers need to be referenced the EcucDefinitionElement has to be set to more than 1, even for the EcucChoiceReferenceDef.

Example 2.21

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>FrameMapping</SHORT-NAME>
  <REFERENCES>
    <ECUC-FOREIGN-REFERENCE-DEF>
      <SHORT-NAME>SystemFrame</SHORT-NAME>
      <DESTINATION-TYPE>FRAME</DESTINATION-TYPE>
    </ECUC-FOREIGN-REFERENCE-DEF>
  </REFERENCES>
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

2.3.6.4 Instance Reference

[TPS_ECUC_02060] EcucInstanceReferenceDef properties ⌈ To be able to reference to descriptions of other AUTOSAR templates with the instanceRef semantics[17] the parameter definition EcucInstanceReferenceDef is used. With the attribute destinationType the type of the referenced entity has to be specified. With the attribute destinationContext the context expression has to be specified. ⌋

[TPS_ECUC_02082] Specification of the destinationType in a EcucInstanceReferenceDef ⌈ The string entered as destinationType shall have the name of a M2 class defined in the metamodel [11] under 'M2::AUTOSAR Templates' as it is represented in the XML-Schema [12] and the referenced class needs to be derived (directly or indirectly) from Identifiable. In the generated Parameter Definition XML file [9] the XML-Schema name shall be used. ⌋

[TPS_ECUC_02083] Specification of the destinationContext in a EcucInstanceReferenceDef ⌈ The string entered as destinationContext shall be an ordered list of M2 class names defined in the metamodel [11] under 'M2::AUTOSAR Templates' as it is represented in the XML schema [12] separated by the SPACE character. Additionally the '*' character can be used to indicate none or multiple occurrence of the M2 class BEFORE the '*' character. ⌋

Examples of destinationContext expressions are:

```
    SW-COMPONENT-PROTOTYPE R-PORT-PROTOTYPE

    ROOT-SW-COMPOSITION-PROTOTYPE SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE

    ROOT-SW-COMPOSITION-PROTOTYPE SW-COMPONENT-PROTOTYPE PORT-PROTOTYPE
DATA-PROTOTYPE*
```

[17]For a detailed description of the instanceRef concept please refer to the Generic Structure Template [7]

Class	**EcucInstanceReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify a reference to an XML description of an entity described in another AUTOSAR template using the INSTANCE REFERENCE semantics.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,Ecuc DefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
destination Context	String	1	attr	The context in the AUTOSAR Metamodel to which' this reference is allowed to point to.
destination Type	String	1	attr	The type in the AUTOSAR Metamodel to which' instance this reference is allowed to point to.

Table 2.30: EcucInstanceReferenceDef

[TPS_ECUC_02061] Specification of the destinationType in a EcucInstanceReferenceDef ⌈ Since the AUTOSAR Generic Structure Template [7] requires the class names of all `Identifiable`s to be unique within the AUTOSAR 'M2:: AUTOSAR Templates' metamodel, it is sufficient to provide only the actual class name of the referenced class in the `destinationType`, as shown in example 2.22. ⌋

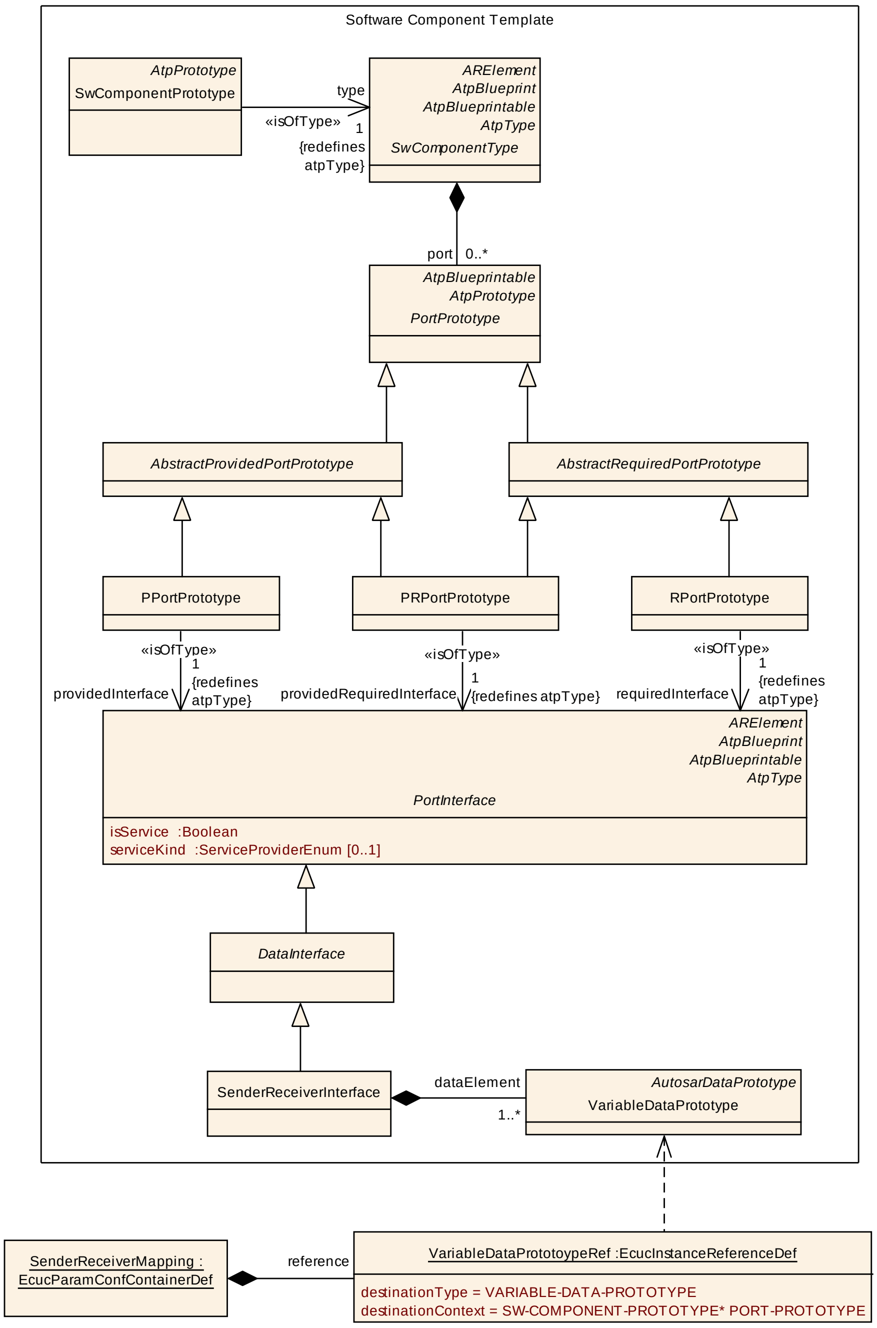

Figure 2.16: Example of an object diagram for an instance reference

In the example in figure 2.16 the reference is defined to be pointing to a description of a ’VARIABLE-DATA-PROTOTYPE’. The ’VARIABLE-DATA-PROTOTYPE’ is defined in the Software Component Template metamodel [13] and is derived from `Identifiable`. Via the `destinationContext` it is specified that each ’VARIABLE-DATA-PROTOTYPE’ exists in the context of a ’PORT-PROTOTYPE’, which itself is in the context of the ’SW-COMPONENT-PROTOTYPE’. The corresponding ECUC Value description XML file extract is shown in example 2.45.

Example 2.22

```
<ECUC-PARAM-CONF-CONTAINER-DEF>
  <SHORT-NAME>SenderRecieverMapping</SHORT-NAME>
  <REFERENCES>
    <ECUC-INSTANCE-REFERENCE-DEF>
      <SHORT-NAME>VariableDataPrototypeRef</SHORT-NAME>
      <DESTINATION-CONTEXT>SW-COMPONENT-PROTOTYPE* PORT-PROTOTYPE</
         DESTINATION-CONTEXT>
      <DESTINATION-TYPE>VARIABLE-DATA-PROTOTYPE</DESTINATION-TYPE>
    </ECUC-INSTANCE-REFERENCE-DEF>
  </REFERENCES>
</ECUC-PARAM-CONF-CONTAINER-DEF>
```

Although the ECU Configuration Parameter Definition of the `EcucForeignReferenceDef` and `EcucInstanceReferenceDef` are similar there is a difference how those references are represented in the ECU Configuration Value description (see section 2.4.5).

2.3.6.5 Symbolic Name Reference

[TPS_ECUC_02032] `EcucSymbolicNameReferenceDef` properties ⌈ The `EcucSymbolicNameReferenceDef` is used to establish the relationship between the user of a symbolic name and the provider of a symbolic name. The object defining the `EcucSymbolicNameReferenceDef` is the user and the `destination` of the reference is the provider of the symbolic name. ⌋

The `EcucSymbolicNameReferenceDef` can only be used to point to elements of the kind of `EcucParamConfContainerDef` within the ECU Configuration Value description.

Class	**EcucSymbolicNameReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	This meta-class specifies that the implementation of the reference is done using a symbolic name defined by the referenced Container's shortName.			
Base	ARObject,AtpDefinition,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
destination	EcucParamConfContainerDef	1	ref	Exactly one reference to a parameter container is allowed as destination. **Stereotypes:** atpUriDef

Table 2.31: EcucSymbolicNameReferenceDef

[TPS_ECUC_02063] Parameters with `symbolicNameValue` = true ⌈ If the attribute `symbolicNameValue` of a configuration parameter (see section 2.3.5) is set to `true` this configuration parameter is used as the actual value for the symbolic name. Only one configuration parameter within a container may have this attribute set to `true`. ⌋

If the attribute symbolicNameValue is not present it shall be assumed to be set to false.

In the example definition shown in figure 2.17 the CorTst module can contain a CorTstDemEventParameterRefs. Those errors need to be defined in the Dem module. And only the Dem module is able to define actual numbers associated with these errors when all errors have been specified and collected in the Dem module. Those associated values can be stored in the DemEventId parameter which belongs to each DemEventParameter.

For an example how this is used in the ECU Configuration Value description refer to section 2.4.5.2. The corresponding ECUC Value description XML file extract is shown in example 2.46.

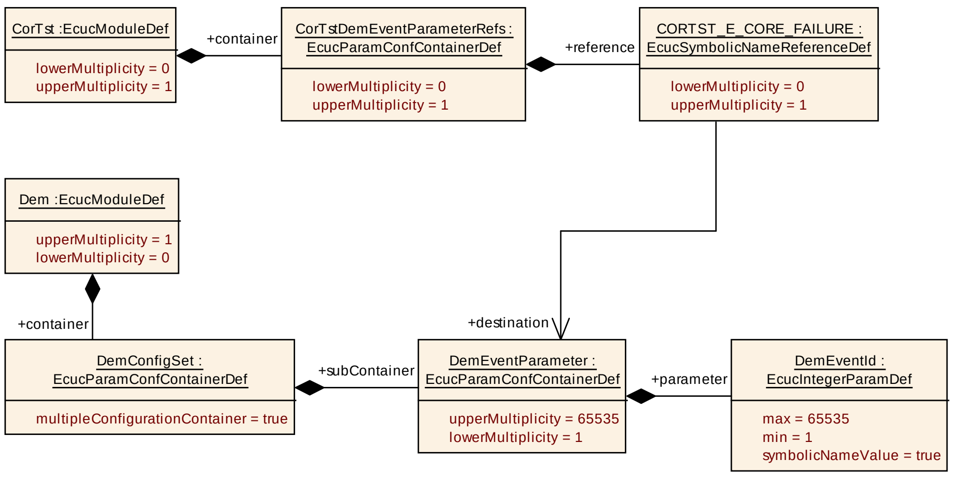

Figure 2.17: Example of an object diagram for a Symbolic Name Reference

Example 2.23

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>CorTst</SHORT-NAME>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>CorTstDemEventParameterRefs</SHORT-NAME>
      <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
          CONTAINER>
      <REFERENCES>
        <ECUC-SYMBOLIC-NAME-REFERENCE-DEF>
          <SHORT-NAME>CORTST_E_CORE_FAILURE</SHORT-NAME>
          <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Dem/DemEventParameter</DESTINATION-REF>
        </ECUC-SYMBOLIC-NAME-REFERENCE-DEF>
      </REFERENCES>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Dem</SHORT-NAME>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>DemEventParameter</SHORT-NAME>
      <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
         CONTAINER>
      <PARAMETERS>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>DemEventId</SHORT-NAME>
          <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
          <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
          <SYMBOLIC-NAME-VALUE>true</SYMBOLIC-NAME-VALUE>
        </ECUC-INTEGER-PARAM-DEF>
      </PARAMETERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
```

2.3.7 Derived Parameter Specification

The parameter definitions introduced in the previous sections are meant to define configuration parameter types regardless how the actual values will be captured. But since the ECU Configuration is dependent on lots of other input information many values for the configuration of the BSW and the RTE can be taken over or calculated from other values already available in the description (e.g. the System Extract or the Software-Component description) or other sections of the ECU Configuration. Such configuration parameters are called Derived Configuration Parameters.

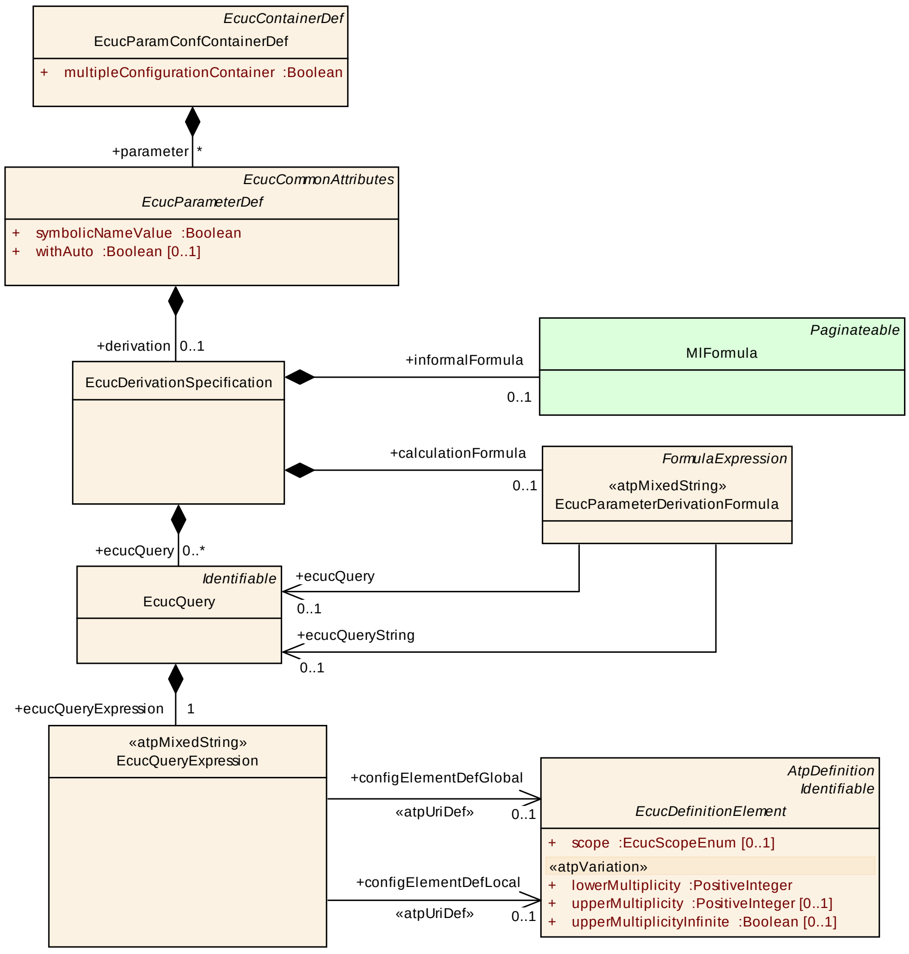
EcucContainerDef
EcucParamConfContainerDef
+ multipleConfigurationContainer :Boolean
+parameter *
EcucCommonAttributes
EcucParameterDef
+ symbolicNameValue :Boolean
+ withAuto :Boolean [0..1]
+derivation 0..1
EcucDerivationSpecification
+informalFormula 0..1
Paginateable
MlFormula
+calculationFormula 0..1
FormulaExpression
«atpMixedString»
EcucParameterDerivationFormula
+ecucQuery 0..*
Identifiable
EcucQuery
+ecucQuery 0..1
+ecucQueryString 0..1
+ecucQueryExpression 1
«atpMixedString»
EcucQueryExpression
+configElementDefGlobal «atpUriDef» 0..1
+configElementDefLocal «atpUriDef» 0..1
AtpDefinition
Identifiable
EcucDefinitionElement
+ scope :EcucScopeEnum [0..1]
«atpVariation»
+ lowerMultiplicity :PositiveInteger
+ upperMultiplicity :PositiveInteger [0..1]
+ upperMultiplicityInfinite :Boolean [0..1]

Figure 2.18: Definition of Derived Parameters

Class	**EcucDerivationSpecification**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Allows to define configuration items that are calculated based on the value of • other parameter values • elements (attributes/classes) defined in other AUTOSAR templates such as System template and SW component template			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
calculation Formula	EcucParameter DerivationForm ula	0..1	aggr	Definition of the formula used to calculate the value of the configuration element.
ecucQuery	EcucQuery	*	aggr	Query to the ECU Configuration Description.
informalFo rmula	MlFormula	0..1	aggr	Informal description of the derivation used to calculate the value of the configuration element.

Table 2.32: EcucDerivationSpecification

[TPS_ECUC_02047] Derivation of parameter values ⌈ For each EcucParameter-Def it can be specified how the parameter value will be computed. This is captured in the element EcucDerivationSpecification. ⌋

[TPS_ECUC_02129] Informal description of the derivation ⌈ For all EcucParame-terDef types an informal description of the derivation can be specified in the element informalFormula. ⌋

[TPS_ECUC_02128] Formal description of the derivation ⌈ For the EcucParame-terDef types

- EcucBooleanParamDef
- EcucIntegerParamDef
- EcucFloatParamDef

a formal calculationFormula can be specified in the element EcucParameter-DerivationFormula. ⌋

Note: The application of the formal calculation formula to the above mentioned types is due to the fact that the result of the calculation formula is numerical.

2.3.7.1 Derived Parameter Calculation Formula

A derivation of a Configuration Parameter value can be specified by an informal Calculation Formula or by a formal language that can be used to specify the computational rules (see figure 2.18). The formal language is defined in the Generic Structure Template [7]. With this formal language it is possible to express dependencies between parameters and e.g. to calculate a value of one parameter based on other parameter values.

Class	≪atpMixedString≫ **EcucParameterDerivationFormula**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	This formula is intended to specify how an ecu parameter can be derived from other information in the Autosar Templates.			
Base	ARObject,FormulaExpression			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
ecucQuery	EcucQuery	0..1	ref	This is one particular EcucQuery used in the calculation formula.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucQuery String	EcucQuery	0..1	ref	This indicates that the referenced query shall return a string.

Table 2.33: EcucParameterDerivationFormula

The informal Calculation Formula (`MlFormula`) can be used for the same purpose. But here, the rules how the derived values are computed are not defined. Different representations can be used to specify such an informal computational rule. More details can be found in MSRSW. Although the `MlFormula` is informal there can be some programming language syntax and semantics interpreted.

To derive Configuration Parameter values with the formal calculation formula one or several `EcucQuery`s can be defined. An `EcucQuery` is `Identifiable` and aggregates one `EcucQueryExpression`. The `EcucQueryExpression` defines a query to the ECU Configuration Value description and outputs the result as a numerical value. Four functions are currently supported by the `EcucQueryExpression`: *count*, *value*, *deref* and *refvalue*. Due the `atpMixedString` nature of the `EcucQueryExpression` several function keywords mixed with several local and global references[18] can be defined within an `EcucQueryExpression`.

Class	**EcucQuery**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Defines a query to the ECUC Description.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucQuery Expression	EcucQueryExpr ession	1	aggr	This is the EcucQuery used in the calculation formula or the condition formula.

Table 2.34: EcucQuery

Class	≪**atpMixedString**≫ **EcucQueryExpression**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Defines a query expression to the ECUC Description and output the result as an numerical value. Due to the "mixedString" nature of the formula there can be several EcuQueryExpressions used.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

[18] `configElementDefLocal`, `configElementDefGlobal`

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
configElementDefGlobal	EcucDefinitionElement	0..1	ref	The EcucQueryExpression points to an EcucDefinitionElement that is used to find an element in the EcucDescription. In order to find the right element in the EcucDescription a search is necessary. If the complete EcucDescription needs to be searched this global reference shall be used. Due to the "mixedString" nature of the EcucQueryExpression several references to EcucDefintionElements can be used in one EcucQueryExpression. **Stereotypes:** atpUriDef
configElementDefLocal	EcucDefinitionElement	0..1	ref	The EcucQueryExpression points to an EcucDefinitionElement that is used to find an element in the EcucDescription. In order to find the right element in the EcucDescription a search is necessary. If the search is executed inside of the same module that contains the EcucQuery this local reference shall be used. Due to the "mixedString" nature of the EcucQueryExpression several references to EcucDefintionElements can be used in one EcucQueryExpression. **Stereotypes:** atpUriDef

Table 2.35: EcucQueryExpression

[constr_5505] Configuration class of the elements of the `EcucQueryExpression` ⌈ The elements of the `EcucQueryExpression` involved in one calculation formula shall have lower or equal configuration class (where `PreCompile` configuration class is considered to be the lowest and `PostBuild` the highest) with respect to the context element in which the calculation is performed (e.g. a `Link` configuration parameter can not calculate its value based on a `PostBuild` parameters value). ⌋

[TPS_ECUC_06018] Input and Output of the *refvalue* function ⌈ The *refvalue* function is provided with a `EcucDefinitionElement` and delivers a set of elements from the ECU Configuration Value description which share the `definition` role of the provided `EcucDefinitionElement`. ⌋

[TPS_ECUC_06019] Output of the refvalue function if the EcucDefinitionElement points to a not existing element in the ECU Configuration Parameter Definition ⌈ The *refvalue* function shall result in an error if the `EcucDefinitionElement` points to a not existing element in the ECU Configuration Parameter Definition. ⌋

[TPS_ECUC_06020] Output of the refvalue function if no element in the ECU Configuration Value description is found ⌈ The *refvalue* function shall return an empty set if the `EcucDefinitionElement` points to an existing element in the ECU Configuration Parameter Definition but no element in the ECU Configuration Value description has been found. ⌋

[TPS_ECUC_06021] Input and Output of the *deref* function ⌈ The *deref* function takes two parameters

1. result of another *deref* function or *refvalue* function, which is an element set
2. reference to a member of the first parameter

and returns the member of the first parameter that is denoted by the second parameter.
⌋

[TPS_ECUC_06022] Output of the *deref* function in case the first input parameter is a reference ⌈ In case the member of the first parameter is a reference the *deref* function returns the referenced element as a set. ⌋

[TPS_ECUC_06023] Cases where the *deref* function reports an error ⌈ The *deref* function shall result in an error if

- the first parameter is an empty set
- the first parameter is a set with more than 1 elements[19]
- the first parameter contains one element which is a value (e.g. 5)
- second parameter points to a not existing element in the ECU Configuration Parameter Definition or to the AUTOSAR Schema.

⌋

[TPS_ECUC_06024] Input of the *value* function ⌈ The *value* function takes the result of a *deref* function or *refvalue* function, which is an element set. ⌋

[TPS_ECUC_06025] Output of the *value* function ⌈ The *value* function returns the parameter's value as numerical value. ⌋

[TPS_ECUC_06026] Cases where the *value* function reports an error ⌈ The *value* function shall result in an error if

- the parameter is an empty set
- the parameter is a set with more than 1 elements[20]
- the parameter's single element does not have a value (e.g. is a container)

⌋

[TPS_ECUC_06057] Input of the *strValue* function ⌈ The *strValue* function takes the result of a *deref* function or *refvalue* function, which is an element set. ⌋

[TPS_ECUC_06058] Output of the *strValue* function ⌈ The *strValue* function returns the parameter's value as string. ⌋

[19]The *deref* function shall only be applied to element sets which are guaranteed to contain only up to 1 element.
[20]The *value* function shall only be applied to element sets which are guaranteed to contain only up to 1 element.

[TPS_ECUC_06059] Cases where the *strValue* function reports an error ⌈ The *strValue* function shall result in an error if

- the parameter is an empty set
- the parameter is a set with more than 1 elements[21]
- the parameter's single element does not have a value (e.g. is a container)

⌋

[TPS_ECUC_06060] Input of the *valueAt* function ⌈ The *valueAt* function takes the result of a *deref* function or *refvalue* function, which is an element set and a zero-based position argument. ⌋

[TPS_ECUC_06061] Output of the *valueAt* function ⌈ The *valueAt* function returns the value of the parameter as numerical value at the position according to the sorting criteria defined in section xxx ⌋

[TPS_ECUC_06062] Cases where the *valueAt* function reports an error ⌈ The *valueAt* function function shall result in an error if

- the parameter is an empty set
- the parameter is a set with more than 1 elements
- the parameter's single element does not have a value (e.g. is a container)
- the position is larger than the count-1

⌋

[TPS_ECUC_06063] Input of the *strValueAt* function ⌈ The *strValueAt* function takes the result of a *deref* function or *refvalue* function, which is an element set and a zero-based position argument. ⌋

[TPS_ECUC_06064] Output of the *strValueAt* function ⌈ The *strValueAt* function returns the value of the parameter as string at the position according to the sorting criteria defined in section x.x.x. ⌋

[TPS_ECUC_06065] Cases where the *strValueAt* function reports an error ⌈ The *strValueAt* function function shall result in an error if

- the parameter is an empty set
- the parameter is a set with more than 1 elements
- the parameter's single element does not have a value (e.g. is a container)
- the position is larger than the count-1

⌋

[21]The *strValue* function shall only be applied to element sets which are guaranteed to contain only up to 1 element.

[TPS_ECUC_06027] Input of the *count* function ⌈ The *count* function gets the result of the *deref* or *refvalue* function as input parameter. ⌋

[TPS_ECUC_06028] Output of the *count* function ⌈ The *count* function returns the number of elements in the input parameter set. ⌋

[TPS_ECUC_06029] Output of the *count* function in case the input parameter set is empty ⌈ The *count* function returns zero if the input parameter set is empty. ⌋

In order to find the referenced element in the ECUC Value description the reference to the `EcucDefinitionElement` needs to be traced. If the complete ECUC Value description needs to be searched a global reference (`configElementDefGlobal`) shall be used. If the search is executed inside of the same module a local reference (`configElementDefLocal`) is sufficient.

The following section shows the `EcucQueryExpression` syntax:

```
ecuQueryExpr : (valueExpr|stringValueExpr|valueAtExpr|stringValueAtExpr|countExpr);
valueExpr : 'value('(derefExpr | refValueExpr) ')';
stringValueExpr : 'strValue('(derefExpr | refValueExpr) ')';
valueAtExpr : 'valueAt('(derefExpr | refValueExpr) ',' index ')'
stringValueAtExpr : 'strValueAt('(derefExpr | refValueExpr) ',' index ')'
countExpr : 'count('(derefExpr | refValueExpr) ')';
refValueExpr : 'refvalue(' refExpr ')';
derefExpr : 'deref('(derefExpr| refValueExpr) ',' refString ')';
refExpr : (localRef | globalRef);
localRef : '<CONFIG-ELEMENT-DEF-LOCAL-REF DEST="' NCName* '">'
             refString '</CONFIG-ELEMENT-DEF-LOCAL-REF>';
globalRef : '<CONFIG-ELEMENT-DEF-GLOBAL-REF DEST="' NCName* '">'
             refString '</CONFIG-ELEMENT-DEF-GLOBAL-REF>';
refString : '/'NCName('/'NCName)*;
index: '0' | ('1'..'9')('0'..'9')*;
NCName : (Letter) (Letter | ('0'..'9') | '_')*;
```

Figure 2.19 shows a COM Gateway example where the `CheckConsistency` boolean parameter is calculated. This parameter checks the length of the Source Signal and compares it with the length of the Destination Signal. If the length of both signals is equal this parameter is set to true, otherwise to false. An XML extract from an ECUC Parameter Definition file is is shown in example 2.25.

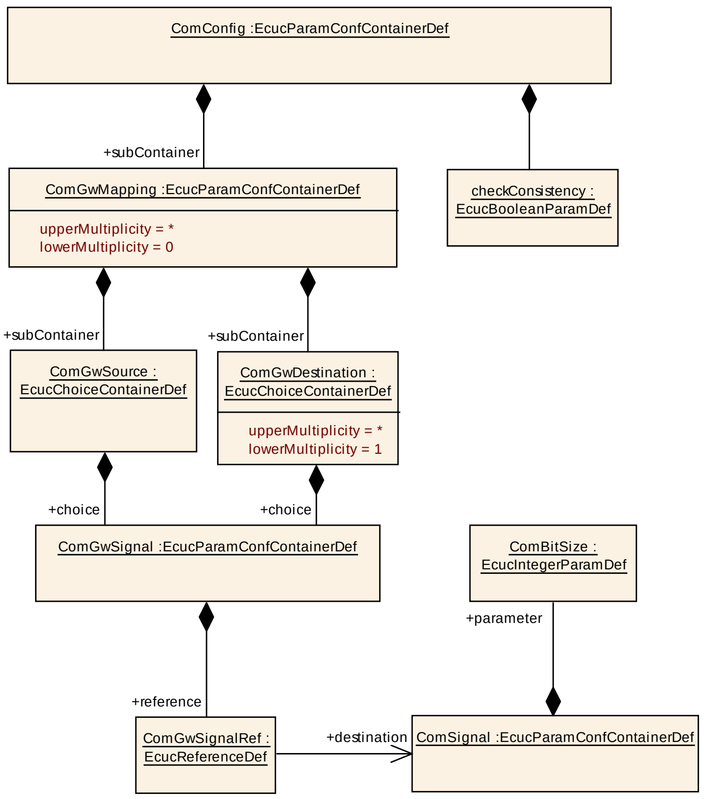
ComConfig :EcucParamConfContainerDef
+subContainer
ComGwMapping :EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
checkConsistency : EcucBooleanParamDef
+subContainer
+subContainer
ComGwSource : EcucChoiceContainerDef
ComGwDestination : EcucChoiceContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+choice
+choice
ComGwSignal :EcucParamConfContainerDef
ComBitSize : EcucIntegerParamDef
+parameter
+reference
ComGwSignalRef : EcucReferenceDef
+destination
ComSignal :EcucParamConfContainerDef

Figure 2.19: Calculation Formula Example

To determine the parameter value the EcucDerivationSpecification within the CheckConsistency parameter aggregates two EcucQueries.

The first EcucQuery "getSourceSignalLength" contains a EcucQueryExpression with a local reference to the ComGwSignalRef element. To get the signal length from the referenced ComGwSignal two *deref* functions are used. The first *deref* function takes the reference to the ComGwSignalRef element as input and returns the ComGwSignal that is searched by the second input parameter. The second *deref* function takes the ComGwSignal as the first input parameter and the reference to the searched ECUC parameter within the ComGwSignal as the second input parameter and returns the ComBitSize parameter. The value of the ComBitSize parameter is provided by the *value* function.

To find the right source signal in the ECUC Value description the biggest common prefix from the local reference and from the CheckConsistency parameter path is used as entry point to the ECUC Value description. In this example the biggest common prefix is the following path: /AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/.

The second EcucQuery "getDestinationSignalLength" provides the ComBitSize Parameter Value of the destination Signal accordingly.

The CalculationFormula compares both values and determines the value for the CheckConsistency parameter. The corresponding ECUC Value description XML file extract is shown in example 2.48.

Example 2.24

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Com</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>ComNetworkSignal</SHORT-NAME>
      <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY-INFINITE>true</UPPER-MULTIPLICITY-INFINITE>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
          CONTAINER>
      <PARAMETERS>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>SignalSize</SHORT-NAME>
          <IMPLEMENTATION-CONFIG-CLASSES>
            <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
              <CONFIG-CLASS>PRE-COMPILE</CONFIG-CLASS>
              <CONFIG-VARIANT>VARIANT-LINK-TIME</CONFIG-VARIANT>
            </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
          </IMPLEMENTATION-CONFIG-CLASSES>
          <ORIGIN>AUTOSAR_ECUC</ORIGIN>
        </ECUC-INTEGER-PARAM-DEF>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>BitPosition</SHORT-NAME>
          <IMPLEMENTATION-CONFIG-CLASSES>
            <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
              <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
              <CONFIG-VARIANT>VARIANT-PRE-COMPILE</CONFIG-VARIANT>
            </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
          </IMPLEMENTATION-CONFIG-CLASSES>
          <ORIGIN>AUTOSAR_ECUC</ORIGIN>
        </ECUC-INTEGER-PARAM-DEF>
      </PARAMETERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

Example 2.25

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Com</SHORT-NAME>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>ComConfig</SHORT-NAME>
      <SUB-CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>ComGwMapping</SHORT-NAME>
          <PARAMETERS>
```

```
            <ECUC-BOOLEAN-PARAM-DEF>
              <SHORT-NAME>CheckConsistency</SHORT-NAME>
              <DERIVATION>
                <CALCULATION-FORMULA>
(<ECUC-QUERY-REF DEST="ECUC-QUERY">getSourceSignalLength</ECUC-QUERY-REF>
   ==
   <ECUC-QUERY-REF DEST="ECUC-QUERY">getDestinationSignalLength</ECUC-QUERY
      -REF>)
                </CALCULATION-FORMULA>
                <ECUC-QUERYS>
                  <ECUC-QUERY>
                    <SHORT-NAME>getSourceSignalLength</SHORT-NAME>
                    <ECUC-QUERY-EXPRESSION>
  value(
    deref(
      deref(
        refvalue(<CONFIG-ELEMENT-DEF-LOCAL-REF DEST="ECUC-REFERENCE-DEF">/
            AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwSource/
            ComGwSignal/ComGwSignalRef</CONFIG-ELEMENT-DEF-LOCAL-REF>),
      /AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwSource/ComGwSignal/
          ComGwSignalRef),
    /ComBitSize)
  )
                    </ECUC-QUERY-EXPRESSION>
                  </ECUC-QUERY>
                  <ECUC-QUERY>
                    <SHORT-NAME>getDestinationSignalLength</SHORT-NAME>
                    <ECUC-QUERY-EXPRESSION>
  value(
    deref(
      deref(
        refvalue(<CONFIG-ELEMENT-DEF-LOCAL-REF DEST="ECUC-REFERENCE-DEF">/
            AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwDestination/
            ComGwSignal/ComGwSignalRef</CONFIG-ELEMENT-DEF-LOCAL-REF>),
      /AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwDestination/
          ComGwSignal/ComGwSignalRef),
    /ComBitSize)
  )
                    </ECUC-QUERY-EXPRESSION>
                  </ECUC-QUERY>
                </ECUC-QUERYS>
              </DERIVATION>
            </ECUC-BOOLEAN-PARAM-DEF>
          </PARAMETERS>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </SUB-CONTAINERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

The next example 2.26 shows the usage of the *count* operation. Within the COM module an Integer Parameter `countNoOfCanDrv` is introduced which counts the available CanDrv modules. To cover all CanDrv modules a global reference is used.

Example 2.26

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Com</SHORT-NAME>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>ComConfig</SHORT-NAME>
      <PARAMETERS>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>numberOfCanDrivers</SHORT-NAME>
          <DERIVATION>
            <CALCULATION-FORMULA>
              <ECUC-QUERY-REF DEST="ECUC-QUERY">countNoOfCanDrv</ECUC-QUERY
                  -REF>
            </CALCULATION-FORMULA>
            <ECUC-QUERYS>
              <ECUC-QUERY>
                <SHORT-NAME>countNoOfCanDrv</SHORT-NAME>
                <ECUC-QUERY-EXPRESSION>
  count(
    refvalue(<CONFIG-ELEMENT-DEF-GLOBAL-REF DEST="ECUC-MODULE-DEF">/AUTOSAR
        /EcucDefs/Can</CONFIG-ELEMENT-DEF-GLOBAL-REF>)
  )
                </ECUC-QUERY-EXPRESSION>
              </ECUC-QUERY>
            </ECUC-QUERYS>
          </DERIVATION>
        </ECUC-INTEGER-PARAM-DEF>
      </PARAMETERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

A third example 2.20 shows a reference into the System Description. The referenced `ComSignal` contains a `ForeignReference` into the System Template (SystemTemplateSystemSignalRef). The searched `startPosition` attribute is defined in the System Template and describes a bitposition of a `SystemSignal` within a `PDU`.

To get the value of this attribute three *deref* functions are used. The first *deref* function provides the `ComSignal`. The second *deref* function provides the `ISignalToPduMapping` element of the System Description and the third *deref* function returns the `startPosition` attribute of the `ISignalToPduMapping` element. The attribute value is provided by the *value* function and is used in the calculation formula.

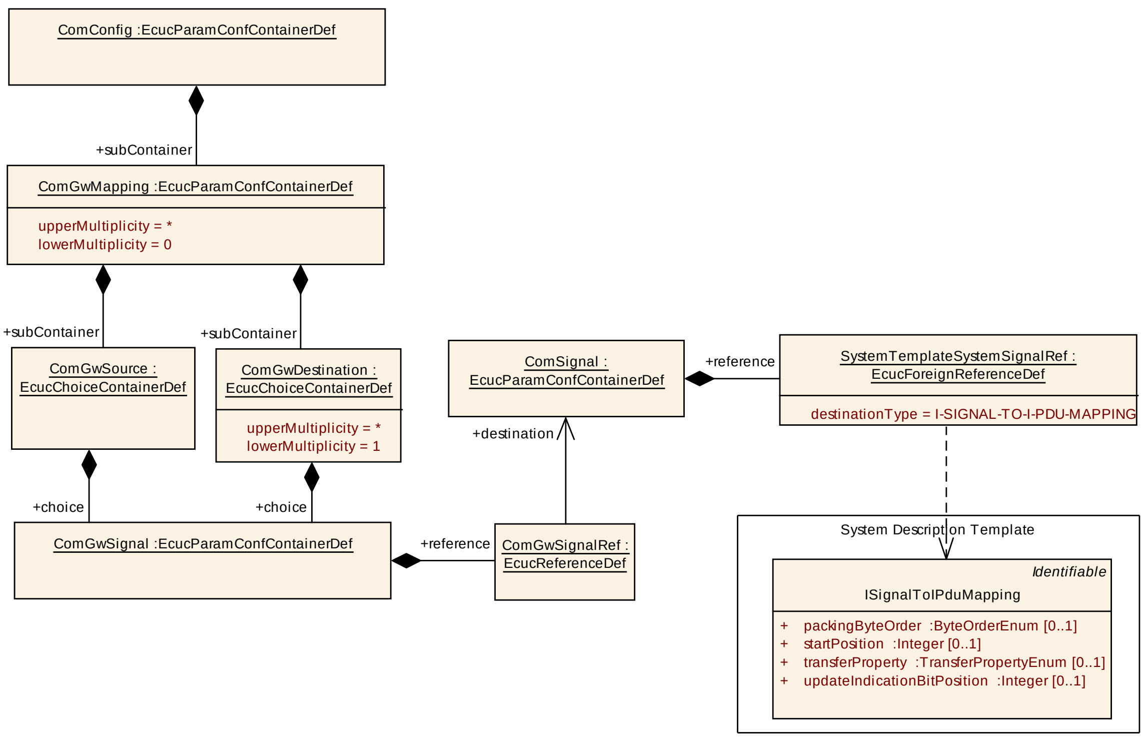
ComConfig :EcucParamConfContainerDef
+subContainer
ComGwMapping :EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
+subContainer
+subContainer
ComGwSource : EcucChoiceContainerDef
ComGwDestination : EcucChoiceContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
+choice
+choice
ComGwSignal :EcucParamConfContainerDef
+reference
ComGwSignalRef : EcucReferenceDef
+destination
ComSignal : EcucParamConfContainerDef
+reference
SystemTemplateSystemSignalRef : EcucForeignReferenceDef
destinationType = I-SIGNAL-TO-I-PDU-MAPPING
System Description Template
Identifiable
ISignalToIPduMapping
+ packingByteOrder :ByteOrderEnum [0..1]
+ startPosition :Integer [0..1]
+ transferProperty :TransferPropertyEnum [0..1]
+ updateIndicationBitPosition :Integer [0..1]

Figure 2.20: Calculation Formula Example

Example 2.27

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Com</SHORT-NAME>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>ComConfig</SHORT-NAME>
      <SUB-CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>ComGwMapping</SHORT-NAME>
          <PARAMETERS>
            <ECUC-INTEGER-PARAM-DEF>
              <SHORT-NAME>startPositionBits</SHORT-NAME>
              <DERIVATION>
                <CALCULATION-FORMULA>
                  <ECUC-QUERY-REF DEST="ECUC-QUERY">
                     getSourceSignalStartPosition</ECUC-QUERY-REF>* 8
                </CALCULATION-FORMULA>
                <ECUC-QUERYS>
                  <ECUC-QUERY>
                    <SHORT-NAME>getSourceSignalStartPosition</SHORT-NAME>
                    <ECUC-QUERY-EXPRESSION>
  value(
    deref(
      deref(
        deref(
          refvalue(<CONFIG-ELEMENT-DEF-LOCAL-REF DEST="ECUC-REFERENCE-DEF">
              /AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwSource/
              ComGwSignal/ComGwSignalRef</CONFIG-ELEMENT-DEF-LOCAL-REF>),
          /AUTOSAR/EcucDefs/Com/ComConfig/ComGwMapping/ComGwSource/
             ComGwSignal/ComGwSignalRef),
        /SystemTemplateSystemSignalRef),
      /SystemTemplateSystemSignalRef),
    /startPosition)
  )
                    </ECUC-QUERY-EXPRESSION>
                  </ECUC-QUERY>
                </ECUC-QUERYS>
              </DERIVATION>
            </ECUC-INTEGER-PARAM-DEF>
          </PARAMETERS>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </SUB-CONTAINERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

2.3.7.2 Restrictions on Configuration Class of Derived Parameters

Derived Parameters have to be defined similar to plain configuration parameters which means that also the configuration class has to be specified in the actual implementation of the configuration. But since derived parameters do depend on other information there are certain restrictions applicable which reduce the degree of freedom what kind of configuration class a derived parameter might be.

[TPS_ECUC_02055] Derivation of information from AUTOSAR Templates ⌈ If the derived parameter is only derived from information coming from other AUTOSAR templates using the `EcucForeignReferenceDef` or `EcucInstanceReferenceDef` relationship it is assumed that those documents do not change during the configuration process and therefore there are no restrictions on the configuration class of derived parameters. ⌋

If the derived parameter is derived from other Configuration Parameters in the ECU Configuration Value description then certain rules have to be applied:

- **[TPS_ECUC_02058] Derivation of information from `PreCompile` parameters** ⌈ If the derived parameter uses information from parameters defined as `PreCompile` then the derived parameter can be any configuration class. ⌋
- **[TPS_ECUC_02056] Derivation of information from `Link-time` parameters** ⌈ If the derived parameter uses information from parameters defined as `Link` then the derived parameter needs to be `Link` or `PostBuild` configurable. ⌋
- **[TPS_ECUC_02057] Derivation of information from `PostBuild` parameters** ⌈ If the derived parameter uses information from parameters defined as `PostBuild` then the derived parameter needs to be `PostBuild` configurable as well. ⌋

2.3.8 Existence dependence of ECUC Parameter Definition elements

ECUC Parameter Values can be calculated from other parameter values that are available in other sections of the ECU Configuration. Such derived configuration parameters are described in detail in chapter 2.3.7. But also the existence of a ECUC Container, Parameter and Reference definition elements can depend on the setting of ECUC Parameter Values. Such it is for example possible to define parameters that are only considered if a specific switch parameter is set to a certain value. Otherwise these parameters are ignored.

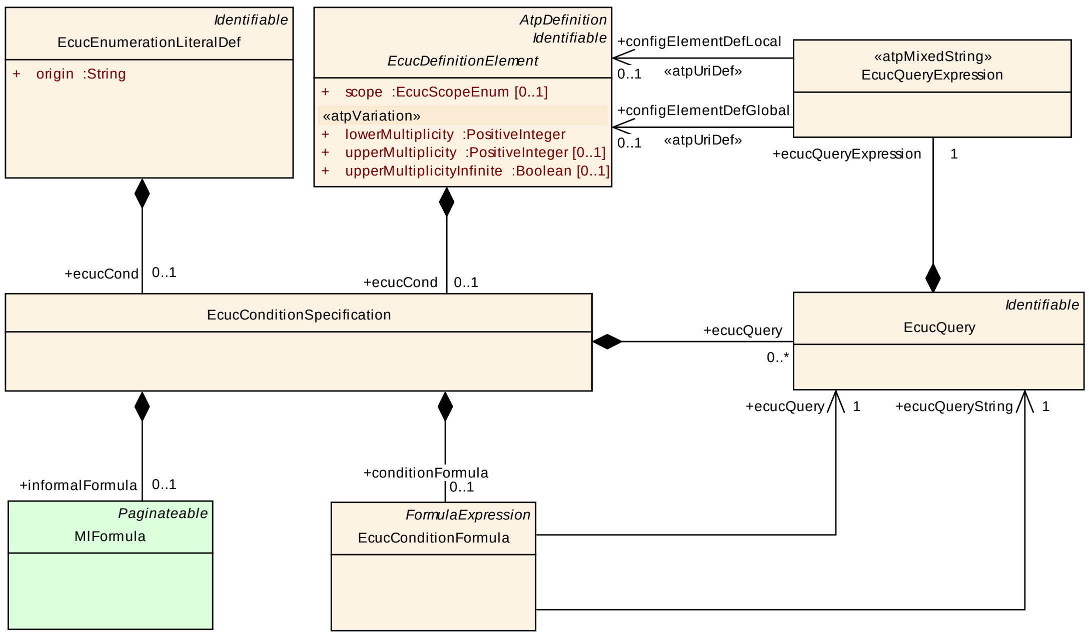

Figure 2.21: Existence dependence of parameter definitions and literal definitions

To allow the description of such existence dependencies the EcucDefinitionElement and the EcucEnumerationLiteralDef aggregate the EcucConditionSpecification. The EcucConditionSpecification aggregates an EcucConditionFormula or a informal Calculation Formula (MlFormula). If the EcucConditionFormula evaluates to true the parameter definition/literal definition shall be processed as specified. Otherwise the parameter definition/literal definition shall be ignored. The informal Calculation Formula (MlFormula) can be used for the same purpose. But here, the rules how the condition is evaluated are not defined.

An EcucQuery to the ECU Configuration Value Description serves as an argument for the EcucConditionFormula. Due the atpMixedString nature of the EcucConditionFormula several EcucQueries can be defined within an EcucConditionFormula.

An EcucQuery is Identifiable and aggregates one EcucQueryExpression. The EcucQueryExpression outputs the result as a numerical value. The EcucQueryExpression syntax is described in chapter 2.3.7.1.

Class	**EcucConditionSpecification**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Allows to define existence dependencies based on the value of parameter values.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
conditionFormula	EcucConditionFormula	0..1	aggr	Definition of the formula used to define existence dependencies.
ecucQuery	EcucQuery	*	aggr	Query to the ECU Configuration Description.
informalFormula	MlFormula	0..1	aggr	Informal description of the condition used to to define existence dependencies.

Table 2.36: EcucConditionSpecification

Class	≪**atpMixedString**≫ **EcucConditionFormula**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	This formula must yield a boolean expression depending on ecuc queries. Note that the EcucConditionFormula is a mixed string. Therefore, the properties have the upper multiplicity 1.			
Base	ARObject,FormulaExpression			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucQuery	EcucQuery	1	ref	The EcucQuery serves as a argument for the formula.
ecucQuery String	EcucQuery	1	ref	This indicates that the referenced query shall return a string.

Table 2.37: EcucConditionFormula

In the following example in figure 2.22 – taken from the Can Interface module – a possible usage of the condition formula is shown.

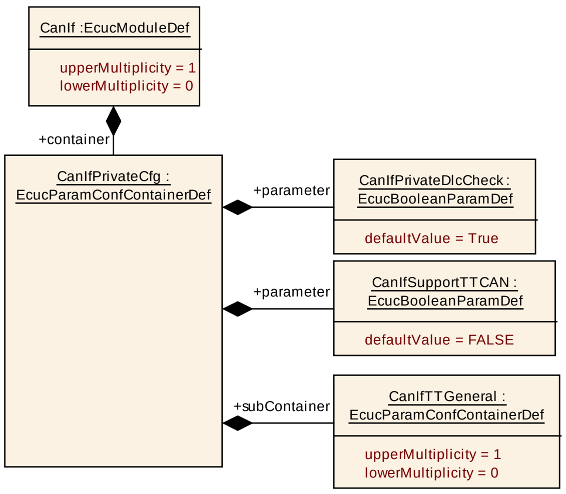

Figure 2.22: Example for condition formula

The container `CanIfPrivateCfg` contains 2 parameters and one sub container. The use case is to make the existance of the container `CanIfTTGeneral` dependent on the value configured in the parameter `CanIfSupportTTCAN`. If the value of `CanIf-SupportTTCAN` is set to `true` the container `CanIfTTGeneral` and its content shall be available for configuration. If the value of `CanIfSupportTTCAN` is set to `false` the container `CanIfTTGeneral` shall not be considered for configuration.

Example 2.28

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>CanIf</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>CanIfPrivateCfg</SHORT-NAME>
      <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
```

```
        <PARAMETERS>
          <ECUC-BOOLEAN-PARAM-DEF>
            <SHORT-NAME>CanIfPrivateDlcCheck</SHORT-NAME>
            <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
            <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
            <!-- ... -->
          </ECUC-BOOLEAN-PARAM-DEF>
          <ECUC-BOOLEAN-PARAM-DEF>
            <SHORT-NAME>CanIfSupportTTCAN</SHORT-NAME>
            <LOWER-MULTIPLICITY>1</LOWER-MULTIPLICITY>
            <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
            <DEFAULT-VALUE>false</DEFAULT-VALUE>
          </ECUC-BOOLEAN-PARAM-DEF>
        </PARAMETERS>
        <SUB-CONTAINERS>
          <ECUC-PARAM-CONF-CONTAINER-DEF>
            <SHORT-NAME>CanIfTTGeneral</SHORT-NAME>
            <ECUC-COND>
              <CONDITION-FORMULA>
                <ECUC-QUERY-REF DEST="ECUC-QUERY">GetTTCanEnabled</ECUC-QUERY
                    -REF>
              </CONDITION-FORMULA>
              <ECUC-QUERYS>
                <ECUC-QUERY>
                  <SHORT-NAME>GetTTCanEnabled</SHORT-NAME>
                  <ECUC-QUERY-EXPRESSION>
                    value(
                      refvalue(<CONFIG-ELEMENT-DEF-LOCAL-REF DEST="ECUC-
                          BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/CanIf/
                          CanIfPrivateCfg/CanIfSupportTTCAN</CONFIG-ELEMENT-
                          DEF-LOCAL-REF>)
                    )
                  </ECUC-QUERY-EXPRESSION>
                </ECUC-QUERY>
              </ECUC-QUERYS>
            </ECUC-COND>
            <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
            <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
            <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
                CONTAINER>
            <PARAMETERS>
              <!-- ... -->
            </PARAMETERS>
          </ECUC-PARAM-CONF-CONTAINER-DEF>
        </SUB-CONTAINERS>
      </ECUC-PARAM-CONF-CONTAINER-DEF>
    </CONTAINERS>
</ECUC-MODULE-DEF>
```

The condition formula is part of the `CanIfTTGeneral` container definition (see example 2.28). The formula itself is pretty simple, it just returns the value of the `EcucQuery` with the name `GetTTCanEnabled`.

The EcucQuery looks for an element in the ECU Configuration Value description which matches the definition
(/AUTOSAR/EcucDefs/CanIf/CanIfPrivateCfg/CanIfSupportTTCAN)
in the local context using the refvalue function.

The EcucQuery then takes the value of the element and returns. Since the element is of boolean type the result of the EcucQuery is already a boolean value which can be processed by the condition formula.

2.3.9 Validation conditions

In order to describe validity constrains on a configuration element the ecucValidationCond can define a set of EcucValidationConditions which can be aggregated by any subclass of EcucDefinitionElement.

[TPS_ECUC_02135] Validation of EcucValidationCondition ⌈ An EcucValidationCondition of an EcucDefinitionElement is considered *valid* if the validationFormula of that EcucValidationCondition evaluates to true. ⌋

[TPS_ECUC_02136] Validation of multiple EcucValidationConditions ⌈ A configuration of an EcucDefinitionElement is considered *valid* if all of the defined ecucValidationConds of that EcucDefinitionElement are *valid*. ⌋

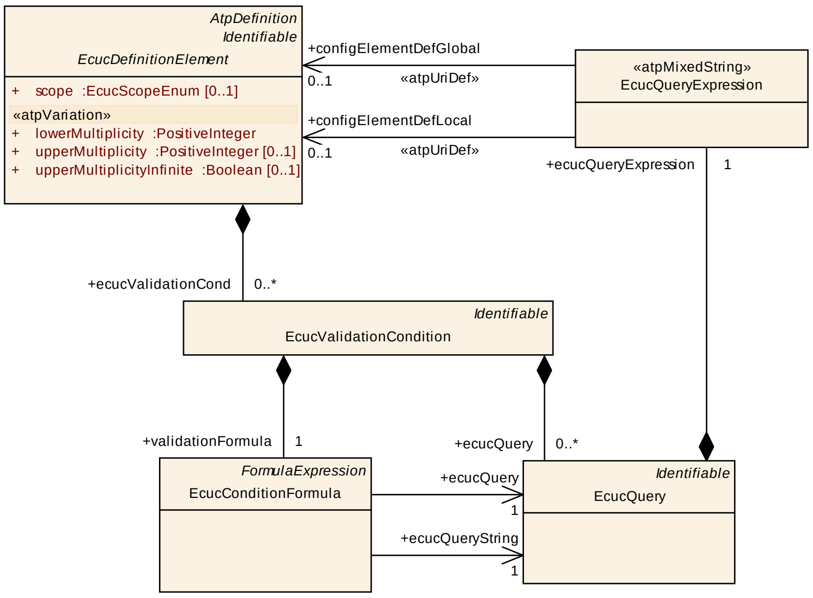

Figure 2.23: Validation condition

Class	**EcucValidationCondition**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Validation condition to perform a formula calculation based on EcucQueries.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucQuery	EcucQuery	*	aggr	Query to the ECU Configuration Description.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
validationFormula	EcucConditionFormula	1	aggr	Definition of the formula used to define validation condition.

Table 2.38: EcucValidationCondition

2.4 ECU Configuration Value Metamodel

As mentioned in section 2.2 the ECU Configuration Definition metamodel provides the means to declare the parameters and their permitted occurrences within a configuration file. This section will specify the complement to that ECU Configuration Parameter Definition on the actual Value description side, namely the ECU Configuration Value description.

The following sections will depict the ECU Configuration Value metamodel. Sections 2.4.1 and 2.4.2 will introduce the top-level structure of a configuration Value description and the module configurations, whereas the sections 2.4.3, 2.4.4 and 2.4.5 will describe the means to file and structure the actual configuration values.

2.4.1 ECU Configuration Value Top-Level Structure

The top-level entry point to an AUTOSAR ECU Configuration Value description is the `EcucValueCollection` (see figure 2.24). Because of the inheritance from `ARElement` the `EcucValueCollection` can be part of an AUTOSAR description like its counterpart the `EcucDefinitionCollection` does. Please note that the `EcucValueCollection` and the `EcucDefinitionCollection` are independent from each other.

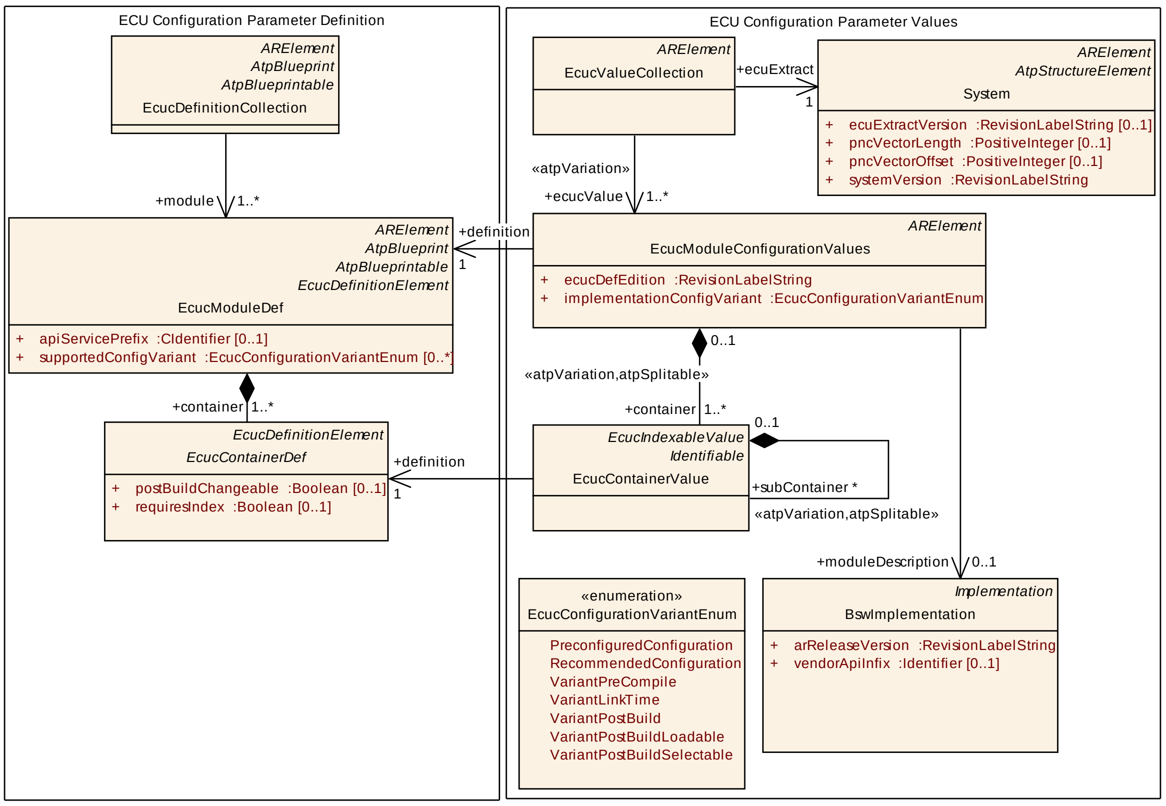
ECU Configuration Parameter Definition
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionCollection
+module
1..*
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionElement
EcucModuleDef
+ apiServicePrefix :CIdentifier [0..1]
+ supportedConfigVariant :EcucConfigurationVariantEnum [0..*]
+container
1..*
EcucDefinitionElement
EcucContainerDef
+ postBuildChangeable :Boolean [0..1]
+ requiresIndex :Boolean [0..1]
+definition
1
+definition
1
ECU Configuration Parameter Values
ARElement
EcucValueCollection
+ecuExtract
1
ARElement
AtpStructureElement
System
+ ecuExtractVersion :RevisionLabelString [0..1]
+ pncVectorLength :PositiveInteger [0..1]
+ pncVectorOffset :PositiveInteger [0..1]
+ systemVersion :RevisionLabelString
«atpVariation»
+ecucValue
1..*
ARElement
EcucModuleConfigurationValues
+ ecucDefEdition :RevisionLabelString
+ implementationConfigVariant :EcucConfigurationVariantEnum
0..1
«atpVariation,atpSplitable»
+container
1..*
0..1
EcucIndexableValue
Identifiable
EcucContainerValue
+subContainer *
«atpVariation,atpSplitable»
+moduleDescription
0..1
«enumeration»
EcucConfigurationVariantEnum
PreconfiguredConfiguration
RecommendedConfiguration
VariantPreCompile
VariantLinkTime
VariantPostBuild
VariantPostBuildLoadable
VariantPostBuildSelectable
Implementation
BswImplementation
+ arReleaseVersion :RevisionLabelString
+ vendorApiInfix :Identifier [0..1]

Figure 2.24: ECU Configuration Value Top-Level Structure

A valid EcucValueCollection needs to reference the System description (provided as an ecuExtract) [2] that specifies the environment in which the configured ECU operates. Additionally it references all Software Module configurations (see section 2.4.2) that are part of this ECU Configuration. It shall be noted that several EcucValueCollections are allowed in the context of one ecuExtract.

Class	**EcucValueCollection**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	This represents the anchor point of the ECU configuration description. **Tags:** atp.recommendedPackage=EcucValueCollections			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecuExtract	System	1	ref	Represents the extract of the System Configuration that is relevant for the ECU configured with that ECU Configuration Description.
ecucValue	EcucModuleConfigurationValues	1..*	ref	References to the configuration of individual software modules that are present on this ECU. atpVariation: [RS_ECUC_0079] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 2.39: EcucValueCollection

[TPS_ECUC_02141] Variable reference EcucValueCollection.ecucValue ⌈ The reference EcucValueCollection.ecucValue is subject to variant handling (see section 2.3.4.1). The existence can be evaluated using the variant handling mechanism. ⌋*(RS_ECUC_00079)*

2.4.2 Module Configurations

[TPS_ECUC_03016] EcucModuleConfigurationValues properties ⌈ The EcucModuleConfigurationValues subsumes all configuration objects that belong to one managed Software Module, namely Application Software Components, BSW modules, RTE and generic ECU Configuration artifacts (e.g. memory maps). ⌋

[TPS_ECUC_02089] The content of EcucModuleConfigurationValues is splitable among several XML-Files ⌈ The EcucModuleConfigurationValues aggregates the EcucContainerValue with the role container and the stereotype ≪atpSplitable≫ which allows the content of a EcucModuleConfigurationValues to be split among several XML-Files (see also section 2.4.2.1). ⌋

[TPS_ECUC_02119] Variable existence of container on value side ⌈ The aggregated container is subject to variant handling (see section 2.3.4.1). The existence can be evaluated using the variant handling mechanism. ⌋*(RS_ECUC_00078)*

[TPS_ECUC_03017] EcucModuleConfigurationValues reference to BswImplementation ⌈ If the EcucModuleConfigurationValues holds the configuration values of a BSW module, a reference to the according BswImplementation shall be provided. ⌋

The reference is established to the BswImplementation because this is the most detailed information available for the configuration.

[TPS_ECUC_03035] Assignment of EcucModuleConfigurationValues to an EcucModuleDef ⌈ The reference definition assigns the EcucModuleConfigurationValues to the according EcucModuleDef it is depending on. ⌋

[TPS_ECUC_06066] Order of Container-, Parameter- and Reference-Values ⌈ Container-, Parameter- and Reference-Values shall be ordered according to the shortName of the parameter definition (which is the last chunk of DEFINITION-REF). ⌋

[TPS_ECUC_06067] Sorting criteria for Containers on the Values side ⌈ Containers on the Values side which have the same parameter definition shall be sorted according to the following criteria: primary sorting criterion is the index. Containers without an index are to be sorted after the containers with index. Secondary sorting criterion is the shortName of the EcucContainerValue. ⌋

[TPS_ECUC_06068] Sorting criteria for References on the Values side ⌈ References on the Values side which have the same definition shall be sorted according to the following criteria: primary sorting criterion is the index. Values without an index are to be sorted after the values with index. Secondary sorting criterion is the reference value (Base + reference). ⌋

[TPS_ECUC_06069] Sorting criteria for Parameters on the Values side ⌈ Parameters on the Values side which have the same definition shall be sorted according to the following criteria: primary sorting criterion is the index. ⌋ Values without an index are to be sorted after the values with index. Secondary sorting criterion is the parameter value.

The index is defined in the EcucIndexableValue class. EcucParameterValue and EcucAbstractReferenceValue inherit from EcucIndexableValue.

Class	**EcucIndexableValue (abstract)**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Used to support the specification of ordering of parameter values.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
index	PositiveInteger	0..1	attr	Used to support the specification of ordering of parameter values. **Tags:** xml.sequenceOffset=-5

Table 2.40: EcucIndexableValue

[TPS_ECUC_06072] Container-, Parameter-, and Reference-Values with requiresIndex set to true ⌈ Container-, Parameter-, and Reference-Values which have the requiresIndex set to true in their definition shall provide an index. ⌋

[TPS_ECUC_03031] EcucModuleDef includes standardized and vendor-specific parameter definitions ⌈ The EcucModuleDef, to which the EcucModuleConfigurationValues is associated to, is specified by the implementor of the according Software Module. Therefore the EcucModuleDef includes standardized as well as vendor-specific parameter definitions. ⌋

Class	**EcucModuleConfigurationValues**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Head of the configuration of one Module. A Module can be a BSW module as well as the RTE and ECU Infrastructure. As part of the BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:** atp.recommendedPackage=EcucModuleConfigurationValuess			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
container	EcucContainerValue	1..*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild xml.sequenceOffset=10
definition	EcucModuleDef	1	ref	Reference to the definition of this EcucModuleConfigurationValues element. Typically, this is a vendor specific module configuration. **Tags:** xml.sequenceOffset=-10

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
ecucDefEdition	RevisionLabelString	1	attr	This is the version info of the ModuleDef ECUC Parameter definition to which this values conform to / are based on. For the Definition of ModuleDef ECUC Parameters the AdminData shall be used to express the semantic changes. The compatibility rules between the definition and value revision labels is up to the module's vendor.
implementationConfigVariant	EcucConfigurationVariantEnum	1	attr	Specifies the kind of deliverable this EcucModuleConfigurationValues element provides. If this element is not used in a particular role (e.g. preconfiguredConfiguration or recommendedConfiguration) then the value must be one of VariantPreCompile, VariantLinkTime, VariantPostBuild.
moduleDescription	BswImplementation	0..1	ref	Referencing the BSW module description, which this EcucModuleConfigurationValues element is configuring. This is optional because the EcucModuleConfigurationValues element is also used to configure the ECU infrastructure (memory map) or Application SW-Cs. However in case the EcucModuleConfigurationValues are used to configure the module, the reference is mandatory in order to fetch module specific "common" published information.

Table 2.41: EcucModuleConfigurationValues

Figure 2.25 depicts the different associations between the `EcucModuleConfigurationValues` and the `Basic Software Module Description`. The `BswImplementation` may specify a vendor specific pre-configured configuration Value description (`preconfiguredConfiguration`) that includes the configuration values already assigned by the implementor of the Software Module and a vendor specific recommended configuration Value description (`recommendedConfiguration`) that can be used to initialize configuration editors.

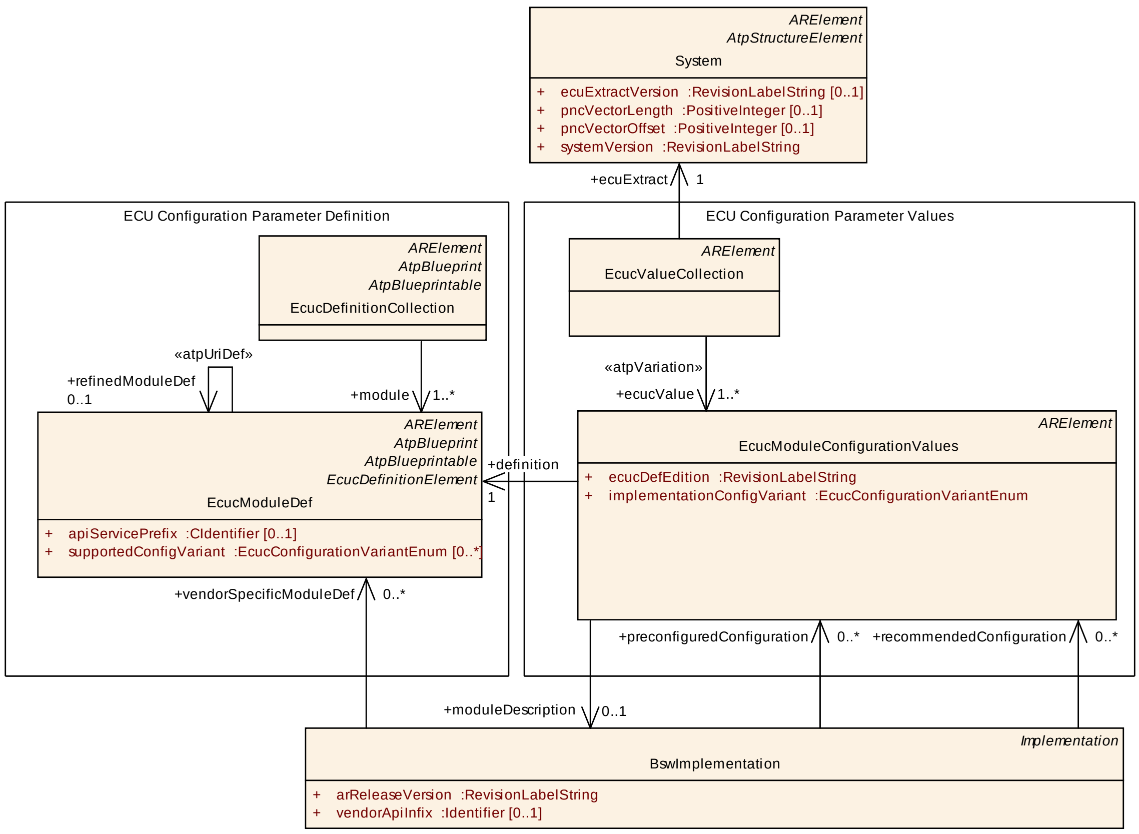
ARElement
AtpStructureElement
System
+ ecuExtractVersion :RevisionLabelString [0..1]
+ pncVectorLength :PositiveInteger [0..1]
+ pncVectorOffset :PositiveInteger [0..1]
+ systemVersion :RevisionLabelString
+ecuExtract 1
ECU Configuration Parameter Definition
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionCollection
«atpUriDef»
+refinedModuleDef 0..1
+module 1..*
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionElement
EcucModuleDef
+ apiServicePrefix :CIdentifier [0..1]
+ supportedConfigVariant :EcucConfigurationVariantEnum [0..*]
+vendorSpecificModuleDef 0..*
+definition 1
ECU Configuration Parameter Values
ARElement
EcucValueCollection
«atpVariation»
+ecucValue 1..*
ARElement
EcucModuleConfigurationValues
+ ecucDefEdition :RevisionLabelString
+ implementationConfigVariant :EcucConfigurationVariantEnum
+preconfiguredConfiguration 0..*
+recommendedConfiguration 0..*
+moduleDescription 0..1
Implementation
BswImplementation
+ arReleaseVersion :RevisionLabelString
+ vendorApiInfix :Identifier [0..1]

Figure 2.25: Dependencies of ModuleConfigurations

[TPS_ECUC_02103] Configuration variant of EcucModuleConfigurationValues ⌈ The implementationConfigVariant specifies which configuration variant has been chosen for this EcucModuleConfigurationValues. The choice is taken from the supportedConfigVariant elements specified in the EcucModuleDef associated to this EcucModuleConfigurationValues. The values preconfiguredConfiguration and recommendedConfiguration are for documentation purposes and cannot be used for code generation. ⌋

The element supportedConfigVariant is described in section 2.3.2 and section 2.3.4.3.2.

To illustrate the structure of an ECU Configuration Value description example 2.29 depicts the top-level structure of an ECU Configuration Value description XML file that conforms to the ECU Configuration Definition XML file that was presented in example 2.6. Please note that it is allowed to have an arbitrary number of packages before a module package definition (e.g. /AUTOSAR/Ecuc_VendorX/CanIf/...).

The only supportedConfigVariant of example 2.6 is taken for the implementationConfigVariant element.

Example 2.29

```
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://autosar.org/schema/r4.0␣AUTOSAR_4-0-3.xsd">
  <ADMIN-DATA>
    <LANGUAGE>EN</LANGUAGE>
    <USED-LANGUAGES>
      <L-10 L="EN" xml:space="default">EN</L-10>
    </USED-LANGUAGES>
    <DOC-REVISIONS>
      <DOC-REVISION>
        <REVISION-LABEL>3.0.0_revision_0004</REVISION-LABEL>
        <ISSUED-BY>AUTOSAR GbR</ISSUED-BY>
      </DOC-REVISION>
    </DOC-REVISIONS>
  </ADMIN-DATA>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ECUC</SHORT-NAME>
      <ELEMENTS>
        <ECUC-VALUE-COLLECTION>
          <SHORT-NAME>Configuration</SHORT-NAME>
          <ECU-EXTRACT-REF DEST="SYSTEM">/some_package/some_path/
             theEcuExtractForEcuXY</ECU-EXTRACT-REF>
          <ECUC-VALUES>
            <ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
              <ECUC-MODULE-CONFIGURATION-VALUES-REF DEST="ECUC-MODULE-
                 CONFIGURATION-VALUES">/ECUC/theRteConfig</ECUC-MODULE-
                 CONFIGURATION-VALUES-REF>
            </ECUC-MODULE-CONFIGURATION-VALUES-REF-CONDITIONAL>
          </ECUC-VALUES>
        </ECUC-VALUE-COLLECTION>
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>theRteConfig</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Rte</
             DEFINITION-REF>
          <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-PRE-COMPILE</
             IMPLEMENTATION-CONFIG-VARIANT>
          <MODULE-DESCRIPTION-REF DEST="BSW-IMPLEMENTATION">/some_package/
             some_path/theUsed_Rte_BSWModuleImplementation</MODULE-
             DESCRIPTION-REF>
          <CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>theGeneration</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                 /EcucDefs/Rte/RteGeneration</DEFINITION-REF>
              <SUB-CONTAINERS>
                <!-- ... -->
              </SUB-CONTAINERS>
            </ECUC-CONTAINER-VALUE>
          </CONTAINERS>
        </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

2.4.2.1 Splitable ModuleConfiguration

In the document *Generic Structure Template* [7] it is specified that the elements of an aggregation are allowed to be split over several XML files if the relationship is marked with the stereotype ≪atpSplitable≫.

The stereotype ≪atpSplitable≫ has been introduced to support the delivery of *one* module's EcucModuleConfigurationValues in *several* XML files, see also Autosar Methodology [1] chapter 2.7.8.3 and 2.7.8.4 for use-cases.

Each splitable property (attribute, aggregation, reference) need to be uniquely identifiable. This happens usually by shortName. The DEFINITION-REF can also be used. For example, the EcucParameterValues of an EcucContainerValue are allowed to be split over several XML files. Each EcucParameterValue is uniquely identifiable via the reference to the EcucParameterDef. More details can be found in the Generic Structure Template [7].

In Example 2.30 a simple definition of a module's configuration parameters is shown. It just consists of one container which has two parameters, one parameter defined to be PRE-COMPILE time configurable, the other parameter is POST-BUILD time configurable. The values for these parameters are defined in different process steps and therefore two XML files can be used to describe both values.

In example 2.31 the value for the PRE-COMPILE time parameter ComSignalLength is specified, while in example 2.32 the POST-BUILD parameter's ComSignalInitValue value is given.

The XML structure in both EcucModuleConfigurationValues XML files is equivalent with respect to the packages and containers. In both XML files a container with the name theSignal is defined. It is up to the configuration tool to *merge* the content of these two files into one model. Also is the number of possible XML files not limited, so it would be possible (although probably not reasonable) to put each parameter value into one XML file.

Example 2.30

```
<ECUC-MODULE-DEF>
  <SHORT-NAME>Com</SHORT-NAME>
  <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
  <UPPER-MULTIPLICITY>1</UPPER-MULTIPLICITY>
  <SUPPORTED-CONFIG-VARIANTS>
    <SUPPORTED-CONFIG-VARIANT>VARIANT-POST-BUILD</SUPPORTED-CONFIG-VARIANT>
  </SUPPORTED-CONFIG-VARIANTS>
  <CONTAINERS>
    <ECUC-PARAM-CONF-CONTAINER-DEF>
      <SHORT-NAME>ComSignal</SHORT-NAME>
      <LOWER-MULTIPLICITY>0</LOWER-MULTIPLICITY>
      <UPPER-MULTIPLICITY>*</UPPER-MULTIPLICITY>
      <MULTIPLE-CONFIGURATION-CONTAINER>false</MULTIPLE-CONFIGURATION-
          CONTAINER>
      <PARAMETERS>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>ComSignalLength</SHORT-NAME>
          <IMPLEMENTATION-CONFIG-CLASSES>
            <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
              <CONFIG-CLASS>PRE-COMPILE</CONFIG-CLASS>
              <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
            </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
          </IMPLEMENTATION-CONFIG-CLASSES>
          <ORIGIN>AUTOSAR_ECUC</ORIGIN>
        </ECUC-INTEGER-PARAM-DEF>
        <ECUC-INTEGER-PARAM-DEF>
          <SHORT-NAME>ComSignalInitValue</SHORT-NAME>
          <IMPLEMENTATION-CONFIG-CLASSES>
            <ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
              <CONFIG-CLASS>POST-BUILD</CONFIG-CLASS>
              <CONFIG-VARIANT>VARIANT-POST-BUILD</CONFIG-VARIANT>
            </ECUC-IMPLEMENTATION-CONFIGURATION-CLASS>
          </IMPLEMENTATION-CONFIG-CLASSES>
          <ORIGIN>AUTOSAR_ECUC</ORIGIN>
        </ECUC-INTEGER-PARAM-DEF>
      </PARAMETERS>
    </ECUC-PARAM-CONF-CONTAINER-DEF>
  </CONTAINERS>
</ECUC-MODULE-DEF>
```

Example 2.31

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>theComConfig</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Com</DEFINITION-
      REF>
  <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-POST-BUILD</IMPLEMENTATION-CONFIG-
      VARIANT>
  <MODULE-DESCRIPTION-REF DEST="BSW-IMPLEMENTATION">/some_package/
      theUsed_Com_BSWModuleImplementation</MODULE-DESCRIPTION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>theSignal</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Com/ComSignal</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
              Com/ComSignal/ComSignalLength</DEFINITION-REF>
          <VALUE>2</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

Example 2.32

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>theComConfig</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Com</DEFINITION-
      REF>
  <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-POST-BUILD</IMPLEMENTATION-CONFIG-
      VARIANT>
  <MODULE-DESCRIPTION-REF DEST="BSW-IMPLEMENTATION">/some_package/
      theUsed_Com_BSWModuleImplementation</MODULE-DESCRIPTION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>theSignal</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Com/ComSignal</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
              Com/ComSignal/ComSignalInitValue</DEFINITION-REF>
          <VALUE>0</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

2.4.3 Parameter Container Description

Symmetrically to the parameter container definition (see section 2.3.3) the parameter container description is specified to group other containers, parameter values and references. Figure 2.26 depicts the general structure of the configuration container Value description and its association to the configuration definition. The dependencies reflect the direct relationship between a `EcucContainerValue` and a `EcucContainerDef` as well as a `EcucParameterValue` and a `ParameterType`.

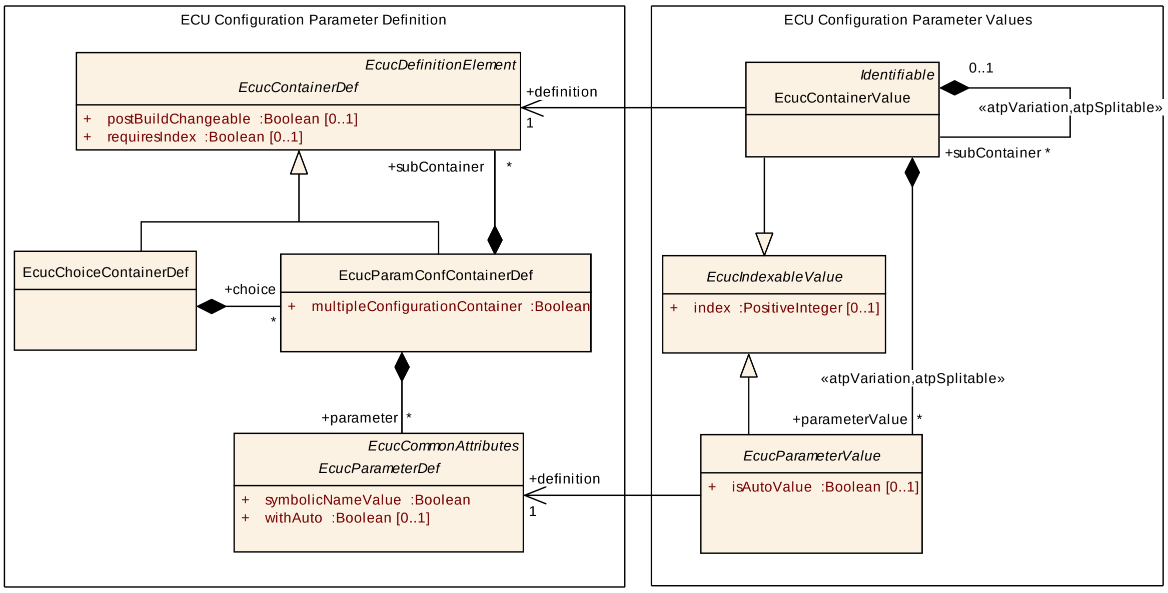
ECU Configuration Parameter Definition
EcucDefinitionElement
EcucContainerDef
+ postBuildChangeable :Boolean [0..1]
+ requiresIndex :Boolean [0..1]
+subContainer *
+definition 1
EcucChoiceContainerDef
+choice *
EcucParamConfContainerDef
+ multipleConfigurationContainer :Boolean
+parameter *
EcucCommonAttributes
EcucParameterDef
+ symbolicNameValue :Boolean
+ withAuto :Boolean [0..1]
+definition 1
ECU Configuration Parameter Values
Identifiable
EcucContainerValue
0..1
«atpVariation,atpSplitable»
+subContainer *
EcucIndexableValue
+ index :PositiveInteger [0..1]
«atpVariation,atpSplitable»
+parameterValue *
EcucParameterValue
+ isAutoValue :Boolean [0..1]

Figure 2.26: Parameter container Value description

[TPS_ECUC_03012] `EcucContainerValue` defines a namespace for all included containers, parameters and references ⌈ The `EcucContainerValue` inherits from `Identifiable` defining a namespace for all `EcucContainerValue`, `EcucParameterValue` and `EcucReferenceValue` that belong to that `EcucContainerValue`.
⌋

[TPS_ECUC_03019] `EcucContainerValue definition` reference ⌈ The reference `definition` assigns the `EcucContainerValue` to the according `EcucContainerDef`[22] it is depending on. ⌋

If the configuration Value description would be provided without an according configuration definition an editor could not reconstruct what kind of `EcucContainerDef` a `EcucContainerValue` is based upon.

[TPS_ECUC_03011] `EcucContainerDef`s with `lowerMultiplicity` $<$ 1 and the effect on the corresponding `EcucContainerValues` ⌈ If a `EcucContainerDef` has specified a `lowerMultiplicity` $<$ 1 the corresponding `EcucContainerValue` may be omitted in the ECU Configuration Value description because of being treated as optional. ⌋*(RS_ECUC_00055)*

[22] including all `EcucContainerDef`'s decendants

Class	**EcucContainerValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Represents a Container definition in the ECU Configuration Description.			
Base	ARObject,EcucIndexableValue,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
definition	EcucContainerDef	1	ref	Reference to the definition of this Container in the ECU Configuration Parameter Definition. **Tags:** xml.sequenceOffset=-10
parameterValue	EcucParameterValue	*	aggr	Aggregates all ECU Configuration Values within this Container. atpVariation: [RS_ECUC_00079] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, variationPoint.shortLabel vh.latestBindingTime=postBuild
referenceValue	EcucAbstractReferenceValue	*	aggr	Aggregates all References with this container. atpVariation: [RS_ECUC_00079] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, variationPoint.shortLabel vh.latestBindingTime=postBuild
subContainer	EcucContainerValue	*	aggr	Aggregates all sub-containers within this container. atpVariation: [RS_ECUC_00078] **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=definition, shortName, variationPoint.shortLabel vh.latestBindingTime=postBuild

Table 2.42: EcucContainerValue

[TPS_ECUC_02120] Variable `subContainers` ⌈ The aggregated `subContainer` is subject to variant handling (see section 2.3.4.1). The existence can be evaluated using the variant handling mechanism. ⌋*(RS_ECUC_00078)*

[TPS_ECUC_02121] Variable `parameterValues` ⌈ The aggregated `parameter-Value` is subject to variant handling (see section 2.3.4.1). The existence can be evaluated using the variant handling mechanism. ⌋*(RS_ECUC_00079)*

[TPS_ECUC_02122] Variable `referenceValues` ⌈ The aggregated `reference-Value` is subject to variant handling (see section 2.3.4.1). The existence can be evaluated using the variant handling mechanism. ⌋*(RS_ECUC_00079)*

[TPS_ECUC_02092] `multipleConfigurationContainer` allows several `Ecuc-ContainerValue` elements in the ECU Configuration ⌈ If a `EcucParamConfCon-`

tainerDef is specified to be the multipleConfigurationContainer there can be several EcucContainerValue elements defined in the ECU Configuration. Each EcucContainerValue shortName does specify the name of the configuration set it contains. ⌋

The multipleConfigurationContainer is further detailed in section 2.4.7.

In example 2.33 a snippet of an ECU Configuration Value description XML file is shown that conforms to the ECU Configuration Parameter Definition described in example 2.7. The container RteGeneration is specified to have an upperMultiplicity of 1, so there can only be one EcucContainerValue representation. The container SwComponentInstance has an upperMultiplicity of *, so there can be several representations of this EcucContainerValue.

Example 2.33

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>theRteConfig</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Rte</DEFINITION-
     REF>
  <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-PRE-COMPILE</IMPLEMENTATION-CONFIG
     -VARIANT>
  <MODULE-DESCRIPTION-REF DEST="BSW-IMPLEMENTATION">/some_package/some_path
     /theUsed_Rte_BSWModuleImplementation</MODULE-DESCRIPTION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>theGeneration</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Rte/RteGeneration</DEFINITION-REF>
      <SUB-CONTAINERS>
        <!-- ... -->
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>SwcInstance1</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Rte/SwComponentInstance</DEFINITION-REF>
      <SUB-CONTAINERS>
        <!-- ... -->
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>SwcInstance2</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Rte/SwComponentInstance</DEFINITION-REF>
      <SUB-CONTAINERS>
        <!-- ... -->
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

2.4.3.1 Choice Containers

[TPS_ECUC_03020] EcucChoiceContainerDef on the value side ⌈ In the ECU Configuration Parameter Definition the container choices are specified as part of the EcucChoiceContainerDef. On the Value side a EcucChoiceContainerDef is treated as a usual container, though it depends on the upperMultiplicity of the EcucChoiceContainerDef how often the choice can be taken. Which choice has been taken is defined by the <DEFINITION-REF> of the <SUB-CONTAINER>. ⌋

Example 2.34 depicts the notation of a filled out EcucChoiceContainerDef as described in example 2.8.

For the myGwSource001 only one choice is possible, in this case the ComGwSignal has been selected.

For the second part (ComGwDestination) three choices have been taken, myGwDestination021 has chosen ComGwSignal, then myGwDestination022 has chosen ComGwDestinationDescription and then myGwDestination023 has chosen another ComGwSignal again.

Example 2.34

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>myChoiceExample</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Com</DEFINITION-
      REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ComGwMapping001</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Com/ComGwMapping</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>myGwSource001</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CHOICE-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Com/ComGwMapping/ComGwSource</DEFINITION-REF>
          <SUB-CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>myGwSource001_1</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/Com/ComGwMapping/ComGwSource/ComGwSignal</
                  DEFINITION-REF>
              <!--...-->
            </ECUC-CONTAINER-VALUE>
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>myGwDestination021</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CHOICE-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Com/ComGwMapping/ComGwDestination</DEFINITION-REF>
          <SUB-CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>myGwDestination021a</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/Com/ComGwMapping/ComGwDestination/ComGwSignal</
                  DEFINITION-REF>
              <!--...-->
            </ECUC-CONTAINER-VALUE>
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>myGwDestination022</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CHOICE-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Com/ComGwMapping/ComGwDestination</DEFINITION-REF>
          <SUB-CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>myGwDestination022a</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/Com/ComGwMapping/ComGwDestination/
                  ComGwDestinationDescription</DEFINITION-REF>
              <!--...-->
            </ECUC-CONTAINER-VALUE>
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>myGwDestination023</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-CHOICE-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Com/ComGwMapping/ComGwDestination</DEFINITION-REF>
          <SUB-CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>myGwDestination023a</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/Com/ComGwMapping/ComGwDestination/ComGwSignal</
                  DEFINITION-REF>
              <!--...-->
            </ECUC-CONTAINER-VALUE>
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

2.4.4 Parameter Values

In the ECU Configuration Parameter Definition exist individual elements for the different types of parameters (e.g. Boolean, Integer, String, see section 2.3.5). On the ECU Configuration Value description side this distinction is no longer needed, because every parameter value element references the corresponding definition element and therefore has its type bound.

However there is a different distinction for the parameter values based on the variant handling implementation (see section 2.3.4.1) and the documentation support (see section 2.3.5.9).

[TPS_ECUC_03006] `EcucParameterValue` is the base class for all parameter values ⌈ All metamodel classes specifying parameter values are derived from `EcucParameterValue` (see figure 2.27). ⌋

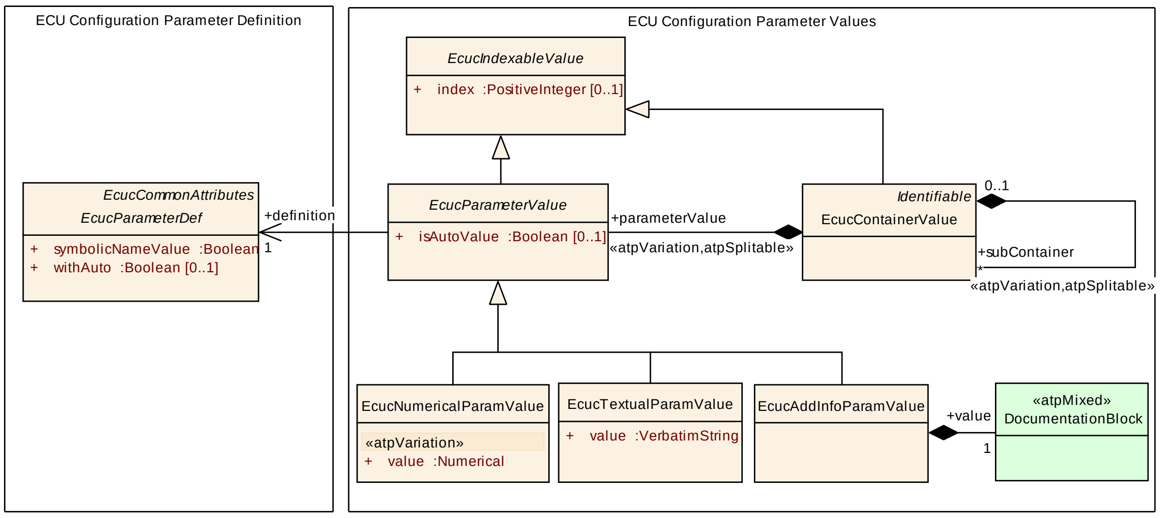

Figure 2.27: Parameter description

[TPS_ECUC_03007] Attribute `value` stores the configuration value in XML-based description ⌈ All inheriting metamodel classes representing an ECU Configuration Value specify an attribute `value` that stores the configuration value in XML-based description. ⌋

[TPS_ECUC_03038] Assignment of an `EcucParameterValue` to the corresponding `EcucParameterDef` ⌈ The reference `definition` assigns the `EcucParameterValue`[23] to the according `EcucParameterDef` it is providing the value for. ⌋

[TPS_ECUC_03009] A `defaultValue` that is specified in the ECU Configuration Parameter Definition may be used as the initial value in the ECU Configuration Value description ⌈ If a `defaultValue` is specified in the ECU Configuration Parameter Definition that given value can be used as the initial `value` of the according `EcucParameterValue` for the ECU Configuration Value description as explained in section 4.2. ⌋

[23]and all its sub-classes

[TPS_ECUC_03034] Each parameter in an Ecuc Configuration Value description shall have a `value` ⌈ In a well-formed and completed ECU Configuration Value description each provided parameter needs to have a `value` specified even if it is just copied from the `defaultValue` of the ECU Configuration Definition. ⌋

For further rules how a `value` can be provided if no `defaultValue` is specified in the ECU Configuration Definition see section 4.2.

[TPS_ECUC_03010] Parameters that are declared as optional in the ECU Configuration Definition may be left out in the ECU Configuration Value description ⌈ If an ECU Configuration Parameter has specified a `lowerMultiplicity` $<$ 1 an ECU Configuration Value may be left out in the ECU Configuration Value description because of being treated as optional. ⌋*(RS_ECUC_00055)*

Class	**EcucParameterValue (abstract)**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Common class to all types of configuration values.			
Base	ARObject,EcucIndexableValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining the ECU Configuration Parameter Values. These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=10
definition	EcucParameterDef	1	ref	Reference to the definition of this EcucParameterValue subclasses in the ECU Configuration Parameter Definition. **Tags:** xml.sequenceOffset=-10
isAutoValue	Boolean	0..1	attr	If withAuto is set to "true" for this parameter definition the isAutoValue can be set to "true". If isAutoValue is set to "true" the actual value will not be considered during ECU Configuration but will be (re-)calculated by the code generator and stored in the value attribute afterwards. These implicit updated values might require a re-generation of other modules which reference these values. If isAutoValue is not present the default is "false". **Tags:** xml.sequenceOffset=20

Table 2.43: EcucParameterValue

[constr_5501] `EcucParameterValue`s and `EcucAbstractReferenceValue`s in `EcucContainerValue`s that exist in multiple configuration sets ⌈ The values of `EcucParameterDef`s and `EcucAbstractReferenceDef`s with `PreCompile` or `Link` configuration class within identical `EcucParamConfContainerDef` instances that exist in multiple configuration sets shall have equal value in all of the configuration sets. Two EcucParamConfContainerDef instances are identical if they have the same qualified shortName path that exhibits exactly the same elements starting from the shortName of the mulitpleConfigurationContainer (not including the shortName of the mulitpleConfigurationContainer) of the configuration set containing the respective EcucParamConfContainerDef instance. ⌋

[constr_5503] symbolicNameValue parameters in post-build configuration sets ⌈ In both post-build selectable and post-build loadable configuration sets in case an EcucContainerValue contains an EcucParameterValue defined in the corresponding EcucParameterDef with symbolicNameValue attribute set to true, EcucParameterValue shall be equal in all configuration sets. ⌋

[TPS_ECUC_08002] Introduction of new EcucParamConfContainerDef instances in a post-build loadable configuration set ⌈ If a new EcucParamConfContainerDef instance is introduced according to the [TPS_ECUC_08000] in a post-build loadable configuration set, each EcucParameterValue and EcucReferenceValue within that EcucParamConfContainerDef instance and its aggregated EcucParamConfContainerDefs instanced in the role subContainer may be assigned a new value regardless of its configuration class (PreCompile, Link or PostBuild). ⌋

Example: HandleId value for an existing ComIPdu shall not be changed in post-build time as it is link-time configurable. However if a new ComIPdu instance is introduced in post-build time, it shall receive a new HandleId value. This basically means that configuration classes are applicable only to parameters in container instances which already exist in the configuration before the post-build updates.

[constr_5502] EcucParameterValues of type EcucFunctionNameDef ⌈ The exception to the [TPS_ECUC_08002] are EcucParameterValues of type EcucFunctionNameDef (see Chapter 2.3.5.6) in a post-build loadable configuration set where it is only allowed to choose one of the existing function names for the new function (e.g. callout), i.e. it is not allowed to introduce a new one. ⌋

[TPS_ECUC_08004] Changing of values and multiplicities of EcucParameterValues at post-build time ⌈ If there may exist multiple EcucParameterValues inside one EcucContainerValue (i.e. the defining EcucDefinitionElement.upperMultiplicity is greater than EcucDefinitionElement.lowerMultiplicity) and the configuration class of the defining EcucParameterDef is PostBuild, both value and multiplicity of this EcucParameterValue may be changed in post-build time. ⌋

This means that it is allowed to change a value of a parameter and also introduce new instances of this parameter or remove some instances in different post-build loadable and selectable configuration sets.

2.4.4.1 Textual Parameter Value

For the storage of values of parameters which do not have a numerical representation the element EcucTextualParamValue shall be used.

[TPS_ECUC_02126] Values for parameter types stored in the element EcucTextualParamValue ⌈ Values for parameter types

- EcucEnumerationParamDef
- EcucAbstractStringParamDef and its sub-classes

shall be stored in the element EcucTextualParamValue. ⌋

The actual value is stored in the element value as VerbatimString and shall conform to the definition of the ECU Configuration Parameter Definition which is referenced in the definition element. The restrictions on the textual representation specified in section 2.3.5.4, section 2.3.5.5, section 2.3.5.6 and section 2.3.5.7 are applicable to the corresponding value specifications.

In case the value of the EcucTextualParamValue shall be affected by the variant handling, the existence of the individual alternative EcucTextualParamValue elements shall be made variant. The value element itself can not be affected by variant handling.

Class	**EcucTextualParamValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Holding a value which is not subject to variation.			
Base	ARObject,EcucIndexableValue,EcucParameterValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	VerbatimString	1	ref	Value of the parameter, not subject to variant handling.

Table 2.44: EcucTextualParamValue

2.4.4.1.1 Examples of `EcucTextualParamValue`

Example 2.35 depicts the configuration description of definition type EcucLinkerSymbolDef for example 2.15.

Example 2.35

```
<ECUC-TEXTUAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-LINKER-SYMBOL-DEF">/AUTOSAR/EcucDefs/Rte/
     Resource/Pim/RtePimInitializationSymbol</DEFINITION-REF>
  <VALUE>MyPimInitValuesLightMaster</VALUE>
</ECUC-TEXTUAL-PARAM-VALUE>
```

Example 2.36 depicts the configuration description of definition type EcucFunctionNameDef for example 2.16.

Example 2.36

```
<ECUC-TEXTUAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-FUNCTION-NAME-DEF">/AUTOSAR/EcucDefs/Eep/
     EepInitConfiguration/EepJobEndNotification</DEFINITION-REF>
  <VALUE>Eep_VendorXY_JobEndNotification</VALUE>
</ECUC-TEXTUAL-PARAM-VALUE>
```

Example 2.37 depicts the configuration description of definition type EcucEnumerationParamDef for example 2.17.

Example 2.37

```
<ECUC-TEXTUAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
     RteGeneration/RteGenerationMode</DEFINITION-REF>
  <VALUE>CompatibilityMode</VALUE>
</ECUC-TEXTUAL-PARAM-VALUE>
```

2.4.4.2 Numerical Parameter Value

If the value of a configuration parameter shall be provided as subject to variant handling the element EcucNumericalParamValue shall be used. The value element of EcucNumericalParamValue is defined as «atpVariation» (see section 2.3.4.1).

Class	**EcucNumericalParamValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Holding the value which is subject to variant handling.			
Base	ARObject,EcucIndexableValue,EcucParameterValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	Numerical	1	attr	Value which is subject to variant handling. atpVariation: [RS_ECUC_00080] **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime

Table 2.45: EcucNumericalParamValue

[TPS_ECUC_02142] Variable value of EcucNumericalParamValue.value ⌈ The value of EcucNumericalParamValue.value is subject to variant handling (see section 2.3.4.1). ⌋*(RS_ECUC_00080)*

2.4.4.2.1 Examples of `EcucNumericalParamValue`

Example 2.38 depicts the configuration description of definition type `EcucBooleanParamDef` for example 2.12.

Example 2.38

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
     RteGeneration/RTE_DEV_ERROR_DETECT</DEFINITION-REF>
  <VALUE>1</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

Example 2.39 depicts the configuration description of definition type `EcucIntegerParamDef` for example 2.13.

Example 2.39

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
     RunnableEntityMapping/PositionInTask</DEFINITION-REF>
  <VALUE>5</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

Example 2.40 depicts the configuration description of definition type `EcucFloatParamDef` for example 2.14.

Example 2.40

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-FLOAT-PARAM-DEF">/AUTOSAR/EcucDefs/Rte/
     RunnableEntityMapping/SchedulingPeriod</DEFINITION-REF>
  <VALUE>74.8</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

2.4.4.3 AddInfo Parameter Value

The only type-specific distinction for the values is the ECU Configuration Parameter Type `EcucAddInfoParamDef` (see section 2.3.5.9).

[TPS_ECUC_02123] The value of the parameter type `EcucAddInfoParamDef` ⌈
The value of the parameter type `EcucAddInfoParamDef` shall be provided in the element `EcucAddInfoParamValue`. This allows the usage of formated text (see AUTOSAR Generic Structure Template [7] for further information). ⌋

Class	**EcucAddInfoParamValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	This parameter corresponds to EcucAddInfoParamDef.			
Base	ARObject,EcucIndexableValue,EcucParameterValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	Documentation Block	1	aggr	Holds the content of the formated text.

Table 2.46: EcucAddInfoParamValue

Example 2.41 depicts the configuration description of definition type `EcucAddInfoParamDef` for example 2.18.

Example 2.41

```
<ECUC-ADD-INFO-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-ADD-INFO-PARAM-DEF">/AUTOSAR/EcucDefs/Dcm/Dtc<
      /DEFINITION-REF>
  <VALUE>
    <P>
      <L-1 L="EN">Description of the Dtc 0815.</L-1>
    </P>
  </VALUE>
</ECUC-ADD-INFO-PARAM-VALUE>
```

2.4.5 References in the ECU Configuration Metamodel

Figure 2.28 depicts the ECU Configuration Metamodel to reference other description elements.

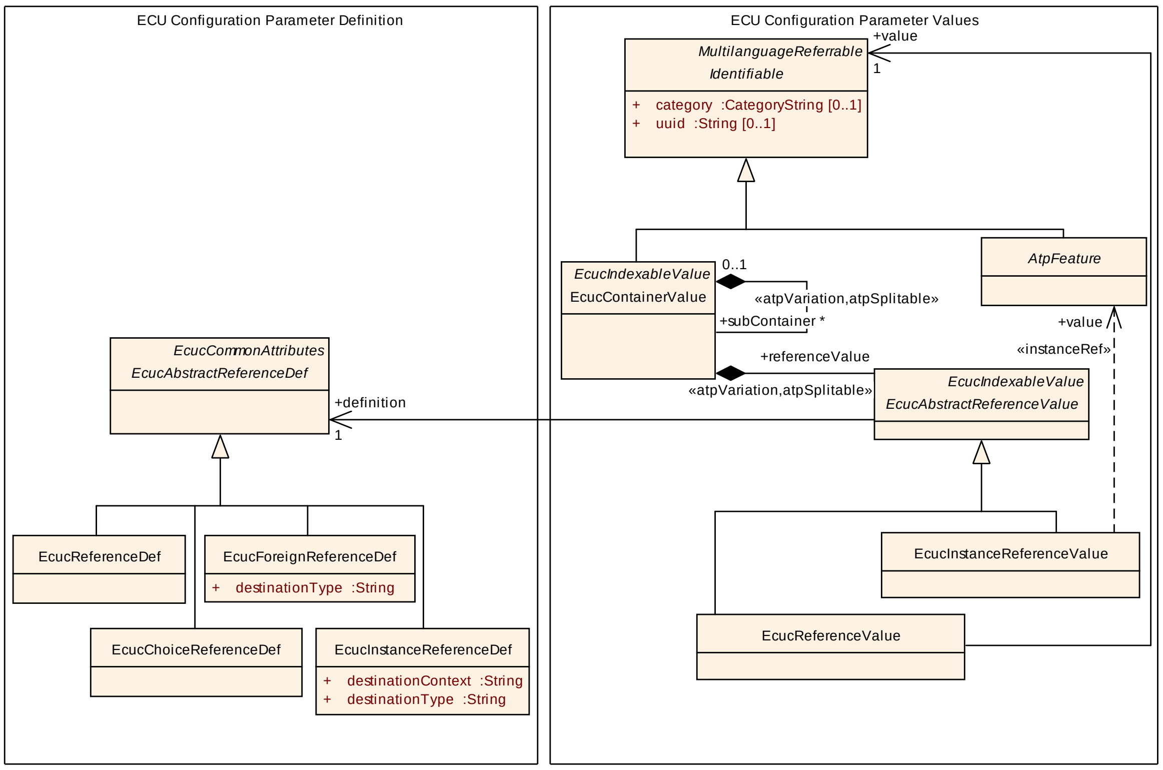

Figure 2.28: Parameter references

[TPS_ECUC_03032] Generalization of all reference types ⌈ The metamodel class EcucAbstractReferenceValue acts as the generalization of all reference types in the ECU Configuration Value description. ⌋

[TPS_ECUC_03039] EcucAbstractReferenceValue definition reference ⌈ The reference definition assigns the EcucAbstractReferenceValue[24] to the according EcucAbstractReferenceDef it is depending on. ⌋

[TPS_ECUC_03030] EcucAbstractReferenceDefs with lowerMultiplicity $<$ 1 and the effect on the corresponding EcucAbstractReferenceValues ⌈ If a EcucAbstractReferenceDef has specified a lowerMultiplicity < 1 an according EcucAbstractReferenceValue may be omitted in the ECU Configuration Value description because of being treated as optional. ⌋*(RS_ECUC_00055)*

Class	**EcucAbstractReferenceValue (abstract)**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Abstract class to be used as common parent for all reference values in the ECU Configuration Description.			
Base	ARObject,EcucIndexableValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

[24] and all its descendants

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes.
definition	EcucAbstractReferenceDef	1	ref	Reference to the definition of this EcucAbstractReferenceValue subclasses in the ECU Configuration Parameter Definition. **Tags:** xml.sequenceOffset=-10

Table 2.47: EcucAbstractReferenceValue

[TPS_ECUC_03027] `EcucReferenceValue` provides the mechanism to reference model elements that are `Identifiable` ⌈ The metamodel class `EcucReferenceValue` provides the mechanism to reference to any model element of type `Identifiable`. ⌋*(RS_ECUC_00072)*

[TPS_ECUC_03028] `EcucReferenceValue` describes `EcucReferenceDef`s, `EcucChoiceReferenceDef`s, `EcucForeignReferenceDef`s and `EcucSymbolicNameReferenceDef`s in the Ecu Configuration Value description ⌈ `EcucReferenceValue` provides the means to describe all kinds of reference definitions except an `EcucInstanceReferenceDef`, which is described in section 2.4.5.1 in more detail. ⌋

[TPS_ECUC_03029] `EcucChoiceReferenceDef` translates to a `EcucReferenceValue` in the ECU Configuration Value description ⌈ A `EcucChoiceReferenceDef` translates to a `EcucReferenceValue` in the ECU Configuration Value description because the choice has to be resolved in that description. Therefore no special configuration Value description type is introduced. ⌋

Class	**EcucReferenceValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Used to represent a configuration value that has a parameter definition of type EcucAbstractReferenceDef(used for all of its specializations excluding EcucInstanceReferenceDef).			
Base	ARObject,EcucAbstractReferenceValue,EcucIndexableValue			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
value	Identifiable	1	ref	Specifes the destination of the reference.

Table 2.48: EcucReferenceValue

[TPS_ECUC_02093] Referenced containers shall be part of the same `EcucValueCollection` as the reference itself ⌈ If a `EcucAbstractReferenceValue` references a container within some `EcucModuleConfigurationValues` the referenced container shall be part of a `EcucModuleConfigurationValues` which is itself part of the `EcucValueCollection`. ⌋

According to figure 2.24 a EcucModuleConfigurationValues is part of the EcucValueCollection if it is referenced with the ecucValue role.

The following examples will picture that EcucReferenceValue can be used to represent most of the specializations of EcucAbstractReferenceDef (namely EcucReferenceDef, EcucChoiceReferenceDef, EcucForeignReferenceDef and EcucSymbolicNameReferenceDef).

Example 2.42 depicts the configuration description of definition type EcucReferenceDef for example 2.19.

Example 2.42

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myOsApplication</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/Os
      /OsApplication</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/EcucDefs/Os/
          OsApplication/OsAppScheduleTableRef</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ECUC/myOs/myOsScheduleTable1<
          /VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Example 2.43 depicts the configuration description of definition type EcucChoiceReferenceDef for example 2.20. To illustrate the usage of a EcucChoiceReferenceDef in more detail, this example takes advantage of the fact that a PortPin may be used in several modes at once. Therefore it has multiple references of different type.

Example 2.43

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myPortPin</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Port/PortPin</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-CHOICE-REFERENCE-DEF">/AUTOSAR/EcucDefs/
          Port/PortPin/PortPinMode</DEFINITION-REF>
      <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ECUC/mySpi/
          aSpiExternalDevice1</VALUE-REF>
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

Example 2.44 depicts the configuration description of definition type EcucForeign-ReferenceDef for example 2.21.

Example 2.44

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myFrameMapping</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
     Rte/Communication/FrameMapping</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-FOREIGN-REFERENCE-DEF">/AUTOSAR/EcucDefs/
         Rte/Communication/FrameMapping/SystemFrame</DEFINITION-REF>
      <VALUE-REF DEST="FRAME">/SystemDescription/SystemFrameNo42</VALUE-REF
         >
    </ECUC-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

2.4.5.1 Instance Reference Values

Due to the formalization of prototypes in the AUTOSAR Templates (see [7]) the reference to the instance of a prototype needs to declare the complete context in which the instance is residing.

[TPS_ECUC_03033] EcucInstanceReferenceValue provides the mechanism to reference an instance of a prototype ⌈ The metamodel class EcucInstanceReferenceValue provides the mechanism to reference to an actual instance of a prototype. This is achieved by specifying a relation with the stereotype <<instanceRef>>. ⌋*(RS_ECUC_00072)*

In figure 2.29 the detailed modeling of the EcucInstanceReferenceValue <<instanceRef>> is specified.

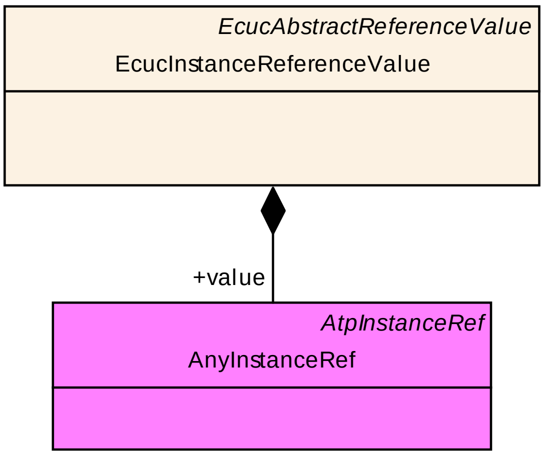

Figure 2.29: Instance Reference Value details

Class	**EcucInstanceReferenceValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	InstanceReference representation in the ECU Configuration.			
Base	ARObject,EcucAbstractReferenceValue,EcucIndexableValue			
Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
value	AtpFeature	1	iref	InstanceReference representation in the ECU Configuration.

Table 2.49: EcucInstanceReferenceValue

Example 2.45 depicts the configuration description of definition type `EcucInstanceReferenceDef` for example 2.22. As one can see in the example the reference value is decomposed of the context path of the instance and the reference to the instance itself.

Example 2.45

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>mySenderReceiverMapping</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Rte/DataMappings/DataSRMapping</DEFINITION-REF>
  <REFERENCE-VALUES>
    <ECUC-INSTANCE-REFERENCE-VALUE>
      <DEFINITION-REF DEST="ECUC-INSTANCE-REFERENCE-DEF">/AUTOSAR/EcucDefs/
          Rte/DataMappings/DataSRMapping/DataElementPrototypeRef</DEFINITION
          -REF>
      <VALUE-IREF>
        <CONTEXT-ELEMENT-REF DEST="SW-COMPONENT-PROTOTYPE">/DoorFR</CONTEXT
            -ELEMENT-REF>
        <CONTEXT-ELEMENT-REF DEST="R-PORT-PROTOTYPE">/DoorAntennaReceiver</
            CONTEXT-ELEMENT-REF>
        <TARGET-REF DEST="VARIABLE-DATA-PROTOTYPE">/AntennaStatus</TARGET-
            REF>
      </VALUE-IREF>
    </ECUC-INSTANCE-REFERENCE-VALUE>
  </REFERENCE-VALUES>
</ECUC-CONTAINER-VALUE>
```

2.4.5.2 Representation of Symbolic Names

[TPS_ECUC_03036] `EcucSymbolicNameReferenceDef` translates to a `EcucReferenceValue` in the ECU Configuration Value description ⌈ A `EcucSymbolicNameReferenceDef` is represented by an usual `EcucReferenceValue` in the ECU Configuration Value description. ⌋

[TPS_ECUC_03037] The `shortName` of the referenced container provides the symbolic name in the implementation ⌈ The `shortName` of the referenced `destination` is expected to be the provided symbolic name in the implementation later on. Therefore the code generator of the providing module has the responsibility to associate the provided symbolic name[25] to its actual value. ⌋

[TPS_ECUC_02107] Values of parameters with the `symbolicNameValue` set to `TRUE` that are assigned by the configuration editor or module generator shall be stored in the XML file ⌈ Configuration parameter values which represent symbolic name values shall be stored in the corresponding XML file after the configuration editor or module generator assigned the actual value. ⌋

Example 2.46 depicts the configuration description of definition type `EcucSymbolicNameReferenceDef` for example 2.23. To give a better impression how the referencing mechanism and code generation may work the `EcucModuleConfigurationValues` of the using and the providing modules are shown here.

Example 2.46

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>myCorTst</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/CorTst</
      DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>Dem_PLL_lock_error</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/CorTst/CorTstDemEventParameterRefs</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-SYMBOLIC-NAME-REFERENCE-DEF">/AUTOSAR/
              EcucDefs/CorTst/CorTstDemEventParameterRefs/
              CORTST_E_CORE_FAILURE</DEFINITION-REF>
          <VALUE-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/ECUC/myDem/
              CORTST_E_CORE_FAILURE_1</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>myDem</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Dem</DEFINITION-
      REF>
```

[25] The one that is referenced to

```
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>CORTST_E_CORE_FAILURE_1</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
         EcucDefs/Dem/DemEventParameter</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
             Dem/DemEventParameter/DemEventId</DEFINITION-REF>
          <VALUE>17</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

[TPS_ECUC_02108] Rule for the creation of `#define` symbols in the header file for parameters with the `symbolicNameValue` set to `TRUE` ⌈ The values of `EcucParameterDef`s with effective configuration class `PreCompile` and `symbolicNameValue` set to `TRUE` shall be generated into the header file of the declaring module as `#define`s. The symbol shall be composed of

- the module abbreviation <MA> of the declaring BSW Module (according to BswModuleList [14]) followed by the literal "Conf_" followed by
- the `shortName` of the `EcucParamConfContainerDef` of the declaring module followed by "_" followed by
- the `shortName` of the `EcucContainerValue` container which holds the `symbolicNameValue` configuration parameter value.

⌋

Taking the specification requirements above the configuration snippet results in the according symbolic name definition in the header file of the providing `Dem` module:

```
...
#define DemConf_DemEventParameter_CORTST_E_CORE_FAILURE_1 17
...
```

Especially in case of production error reporting this pattern is used extensively: The integrator has the freedom to call the DemEventParameter container name arbitrarily. In general it is reasonable to name the DemEventParameter like the actual production error (e.g. FLS_E_ERASE_FAILED), however there are use-cases where the same production error shall be reported for several instances / channels individually, thus it is required to distinguish between these different production error occurrences (e.g. FRIF_E_NIT_CH_A_CLUSTER_1 where FRIF_E_NIT_CH_A is the production error name and _CLUSTER_1 encodes one specific FlexRay cluster). This needs to be kept in mind when accessing the production error symbolic name from the reporting module, e.g. FrIf shall call:

```
Dem_ReportErrorStatus(DemConf_DemEventParameter_FRIF_E_NIT_CH_A_CLUSTER_1,
DEM_EVENT_STATUS_PASSED);
```

In figure 2.30 another example of a symbolic name value configuration setup is shown. The example 2.47 shows a possible value description for this definition.

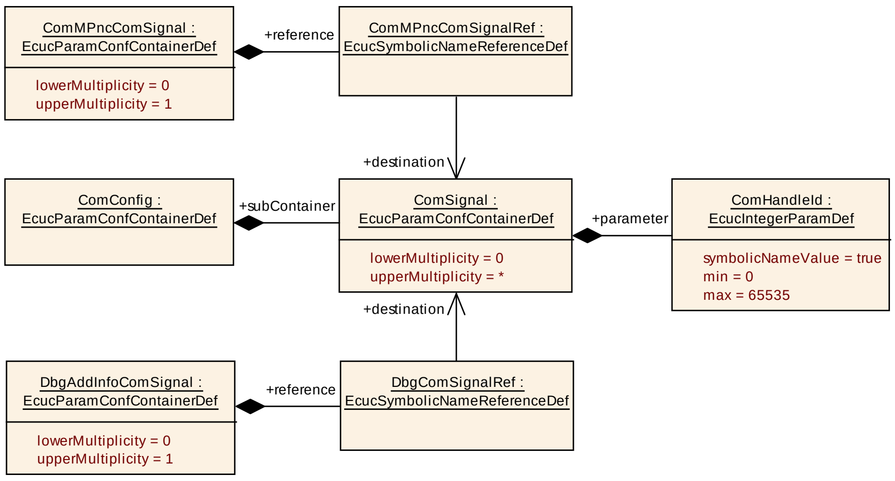
ComMPncComSignal : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
ComMPncComSignalRef : EcucSymbolicNameReferenceDef
+destination
ComConfig : EcucParamConfContainerDef
+subContainer
ComSignal : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+parameter
ComHandleId : EcucIntegerParamDef
symbolicNameValue = true
min = 0
max = 65535
+destination
DbgAddInfoComSignal : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
DbgComSignalRef : EcucSymbolicNameReferenceDef

Figure 2.30: Example of ComSignal symbolic name definition

Example 2.47

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myComConfig</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/
      Com/ComConfig</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>PNC_02</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Com/ComConfig/ComSignal</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
              Com/ComConfig/ComSignal/ComHandleId</DEFINITION-REF>
          <VALUE>231</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>Debuging_Sig5</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Com/ComConfig/ComSignal</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/EcucDefs/
              Com/ComConfig/ComSignal/ComHandleId</DEFINITION-REF>
          <VALUE>245</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

This leads to the generation of the following definitions in the Com header file:

```
#define ComConf_ComSignal_PNC_02 231

#define ComConf_ComSignal_Debuging_Sig5 245
```

Such that the other BSW Modules - which include the Com header file - can call the Com module using these symbols:

```
ComM: Com_SendSignal(ComConf_ComSignal_PNC_02, value)

Dgb:  Com_SendSignal(ComConf_ComSignal_Debuging_Sig5, value)
```

Invalid configuration due to symbolic name values

[TPS_ECUC_06074] Invalid configuration due to symbolic name values ⌈ Due to the hierarchical structure of the `EcucParameterValue`s, it is possible that the same `shortName` (in independent `EcucContainerValue` structures) is the base for multiple `symbolicNameValue` definitions. If the respective value is equal in all occurrences of the `shortName` according to [TPS_ECUC_02108], the generation of the `#define` shall only be done once. If the respective value is different in any of the occurrences of the `shortName` according to [TPS_ECUC_02108], the configuration is invalid. ⌋

Example 2.31 shows a valid and an invalid configuration.

Please note that the requirement and the example is not only applicable for `multipleConfigurationContainer`s but for all other container structures as well.

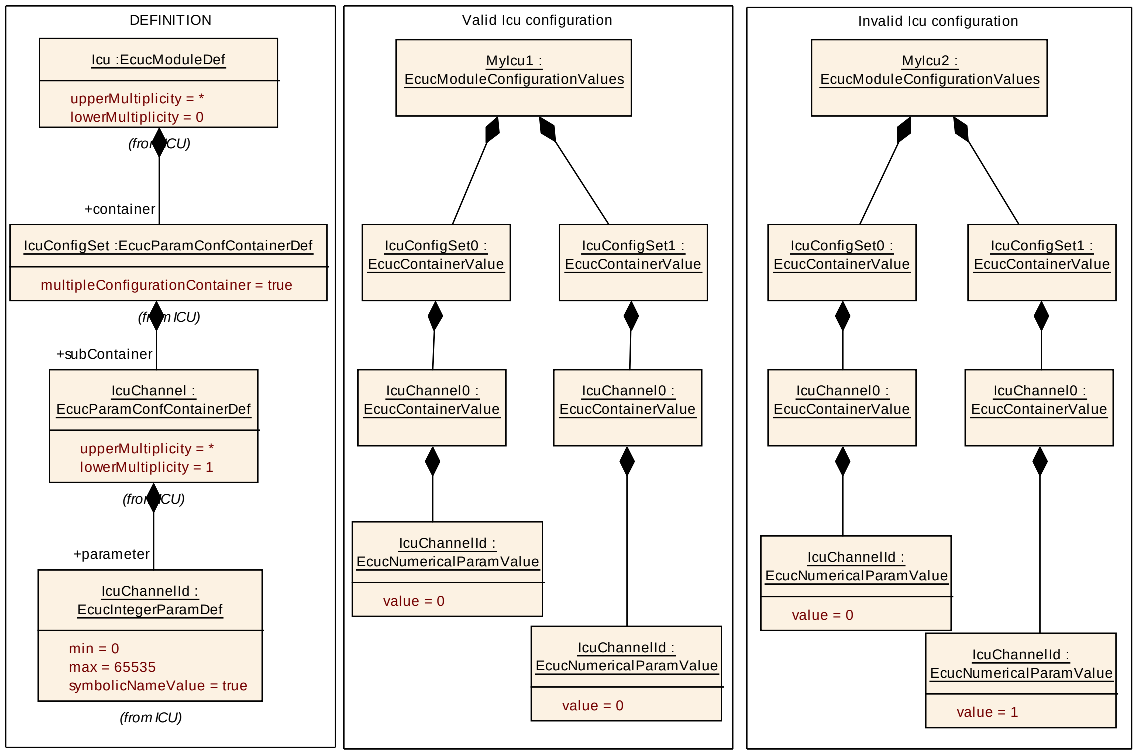
DEFINITION
Icu :EcucModuleDef
upperMultiplicity = *
lowerMultiplicity = 0
(from ICU)
+container
IcuConfigSet :EcucParamConfContainerDef
multipleConfigurationContainer = true
(from ICU)
+subContainer
IcuChannel : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
(from ICU)
+parameter
IcuChannelId : EcucIntegerParamDef
min = 0
max = 65535
symbolicNameValue = true
(from ICU)
Valid Icu configuration
MyIcu1 : EcucModuleConfigurationValues
IcuConfigSet0 : EcucContainerValue
IcuConfigSet1 : EcucContainerValue
IcuChannel0 : EcucContainerValue
IcuChannel0 : EcucContainerValue
IcuChannelId : EcucNumericalParamValue
value = 0
IcuChannelId : EcucNumericalParamValue
value = 0
Invalid Icu configuration
MyIcu2 : EcucModuleConfigurationValues
IcuConfigSet0 : EcucContainerValue
IcuConfigSet1 : EcucContainerValue
IcuChannel0 : EcucContainerValue
IcuChannel0 : EcucContainerValue
IcuChannelId : EcucNumericalParamValue
value = 0
IcuChannelId : EcucNumericalParamValue
value = 1

Figure 2.31: SymbolicNameValues and the generation of #defines: valid and invalid configurations

The *valid* example in figure 2.31 does lead to the following definition:

```
#define IcuConf_IcuChannel_IcuChannel0 0
```

The *invalid* example in figure 2.31 would possibly lead to the following definitions:

```
#define IcuConf_IcuChannel_IcuChannel0 0

#define IcuConf_IcuChannel_IcuChannel0 1
```

where a different value would be assigned to the same symbol. This is an invalid configuration.

2.4.6 Derived Parameters in an ECU Configuration Description

[TPS_ECUC_03021] `EcucParameterDef`s with `EcucDerivationSpecification` result in a `EcucNumericalParamValue` in the ECUC Value description ⌈ Providing the configuration value for an instance of an `EcucParameterDef` which has as `EcucDerivationSpecification` results in a `EcucNumericalParamValue`. ⌋

[TPS_ECUC_02125] Value of parameters with a defined derivation specification ⌈ The value of a parameter shall be provided even when the defining `EcucParameterDef` has a `EcucDerivationSpecification`. ⌋

In this way it is guaranteed that even when a tool does not support the derivation formula the value is still available.

With the storage of the value it is also possible to implement consistency checks whether the actually provided value matches the result of the derivation formula.

Example 2.48 depicts the configuration description of derived parameters for example 2.25.

Example 2.48

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/AUTOSAR/EcucDefs/Com/
     ComConfig/ComGwMapping/CheckConsistency</DEFINITION-REF>
  <VALUE>1</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

2.4.7 Multiple Configuration Sets

As mentioned in section 2.3.3 each EcucModuleDef may specify exactly one EcucContainerValue which represents the root container of the multiple configuration set.[26]

[TPS_ECUC_03042] Definition of multiple configuration sets ⌈ The definition of this EcucContainerValue has to reference the EcucParamConfContainerDef which has the attribute multipleConfigurationContainer set to true. ⌋

[TPS_ECUC_03043] Occurrence of multiple configuration containers in the ECUC Value description ⌈ The EcucContainerValue may occur as often as configuration sets exist. Even if the upperMultiplicity of the corresponding EcucParamConfContainerDef is exactly "1". The configuration tools have to check that the multiplicity of this EcucContainerValue results from the presence of multiple configuration sets. ⌋

[TPS_ECUC_08008] Usage of the multiple configuration container in EcucModuleDefs with supportedConfigVariant of VariantPostBuild ⌈ For EcucModuleDefs with supportedConfigVariant of VariantPostBuild, the name of the C-init-data struct containing the configuration of a BSW module shall be the shortName of the EcucContainerValue belonging to that EcucModuleDef that is marked as multipleConfigurationContainer in the Ecu Configuration Parameter Definition. ⌋

Rationale for the [TPS_ECUC_08008]: The EcuM/BswM modules call the module's init function with the address (i.e. the address operator "&" is applied) of the C-init-data struct as a parameter, for EcucModuleDefs with supportedConfigVariant of VariantPostBuild.

[TPS_ECUC_03044] Name of the configuration set ⌈ The shortName of the EcucContainerValue, which is marked as multipleConfigurationContainer, has to be the name of the configuration set, i.e. the configuration set is part of the namespace path. ⌋

[TPS_ECUC_08009] Names of containers inside a multiple configuration set ⌈ The shortName of an EcucContainerValue inside a multipleConfigurationContainer structure shall be considered as PreCompile configuration class. The value of the shortName shall be the same throughout all configuration sets. ⌋

[TPS_ECUC_02104] Valid configuration set names ⌈ The shortName of the EcucContainerValue, which is marked as multipleConfigurationContainer, shall be a valid C-literal for defining a symbol. ⌋

The shortName shall be a valid C-literal to be used as a symbol for references within the ECU.

[26]Please note that for supporting multiple configuration sets as opposed to the approach using multipleConfigurationContainer, another approach using Variant Handling (described in section 2.4.8) can be used.

[TPS_ECUC_02105] Uniqueness of configuration set names ⌈ The shortName of the EcucContainerValue, which is marked as multipleConfigurationContainer, shall be unique for the whole ECU. That uniqueness includes all software modules' symbols used on that ECU. ⌋

The shortName shall be unique for the whole ECU to allow distinct addressing of the configuration data in the ECU memory.

Because the configuration set name is used in the shortName of the EcucContainerValue each configuration set can be addressed individually. So it is possible to define which configuration set shall be used for a certain initialization of the module.

[TPS_ECUC_03045] Parameter value description structure underneath the multiple configuration container ⌈ The parameter value description structure underneath the EcucContainerValue will be copied for each configuration set, that includes all pre-compile time and link time parameters. ⌋

[TPS_ECUC_03046] Values of pre-compile time and link time parameters in different configuration sets ⌈ Configuration tools have to check that pre-compile time and link time parameters have the same values throughout all configuration sets. ⌋

[TPS_ECUC_03047] EcucReferenceValue in multiple configuration sets ⌈ EcucReferenceValue have to include absolute paths when used in multiple configuration sets. Otherwise it cannot be distinguished which Identifiable in which configuration set is referenced. ⌋

Example 2.49 depicts a EcucContainerValue with according EcucParamConfContainerDef AdcConfigSet (see example 2.9) is defined as a multiple configuration set container:

Example 2.49

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>myAdcConfig</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Adc</DEFINITION-
      REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ConfDoorFrontLeft</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Adc/AdcConfigSet</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>hwUnit1</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Adc/AdcConfigSet/AdcHwUnit</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                  EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcHwUnitId</
                  DEFINITION-REF>
              <VALUE>5</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
          </PARAMETER-VALUES>
          <SUB-CONTAINERS>
            <!-- ... -->
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ConfDoorFrontRight</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
          EcucDefs/Adc/AdcConfigSet</DEFINITION-REF>
      <SUB-CONTAINERS>
        <ECUC-CONTAINER-VALUE>
          <SHORT-NAME>hwUnit1</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/
              EcucDefs/Adc/AdcConfigSet/AdcHwUnit</DEFINITION-REF>
          <PARAMETER-VALUES>
            <ECUC-NUMERICAL-PARAM-VALUE>
              <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/AUTOSAR/
                  EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcHwUnitId</
                  DEFINITION-REF>
              <VALUE>7</VALUE>
            </ECUC-NUMERICAL-PARAM-VALUE>
          </PARAMETER-VALUES>
          <SUB-CONTAINERS>
            <!-- ... -->
          </SUB-CONTAINERS>
        </ECUC-CONTAINER-VALUE>
      </SUB-CONTAINERS>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

In this example the `Adc` module is used to illustrate the two configuration sets. The `Adc` module is initialized by the `EcuM` module. So the `EcuM` needs to know which configuration set to be used for the initialization of the `Adc`. So in the configuration Value description of the `EcuM` there needs to be a reference defined to choose the configuration set. For the example 2.49 the references to the two configuration set are:

`<VALUE-REF>/myAdcConfig/ConfDoorFrontLeft</VALUE-REF>` and

`<VALUE-REF>/myAdcConfig/ConfDoorFrontRight</VALUE-REF>`.

With such references any configuration set can be addressed explicitly.

2.4.8 Usage of Variant Handling to Cope with PreCompile, Link and PostBuild Configuration Parameters

The goal of this feature is to provide modeling support to handle several binding times of the ECU Configuration Value Description in one model.[27]

Motivation: One major drawback of the current modeling approach of Ecuc Values for post-build configuration is the usage of multipleConfigurationContainers. This approach requires that multiple configuration sets of one BSW Module have differently named multipleConfigurationContainers on the value side.

The shortNames of these containers are then used to enable the EcuM to initialize the respective BSW Module with the proper configuration structure during initialization.

This leads to a problem since references to containers which are structurally included in the multipleConfigurationContainer have the shortNames of the multipleConfigurationContainer included in the formal shortNameReference expression.

Thus, it is not specified how a PreCompile/Link configuration reference shall refer to a container which is part of the post-build configuration structure. Additionally, parameter values which are not post-build configurable and have the same value in all configuration sets are replicated inside each multipleConfigurationContainer.

The idea of this feature is to utilize the Variant Handling approach in order to allow the distinction between configuration parameter values for different binding times.

2.4.8.1 Declaration of PostBuildVariantCriterion

In order to utilize the post-build feature, it is required to declare at least one PostBuildVariantCriterion which is used to define a common selecting element for different post-build variants.

Example 2.50

```
<POST-BUILD-VARIANT-CRITERION>
  <SHORT-NAME>PostBuildConfigSet</SHORT-NAME>
</POST-BUILD-VARIANT-CRITERION>
```

Then different variants can be specified using several PostBuildVariantCriterionValues for this PostBuildVariantCriterion.

Example 2.51

```
<POST-BUILD-VARIANT-CRITERION-VALUE-SET>
  <SHORT-NAME>PostBuildVariants</SHORT-NAME>
```

[27] Please note that for supporting multiple configuration sets as opposed to the approach using Variant Handling, another approach using multipleConfigurationContainer (described in section 2.4.7) can be used. However this approach is plan to be removed in future releases.

```
          <POST-BUILD-VARIANT-CRITERION-VALUES>
            <POST-BUILD-VARIANT-CRITERION-VALUE>
              <VARIANT-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
                 EcucDemo/PostBuildConfigSet</VARIANT-CRITERION-REF>
              <VALUE>1</VALUE>
            </POST-BUILD-VARIANT-CRITERION-VALUE>
            <POST-BUILD-VARIANT-CRITERION-VALUE>
              <VARIANT-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
                 EcucDemo/PostBuildConfigSet</VARIANT-CRITERION-REF>
              <VALUE>2</VALUE>
            </POST-BUILD-VARIANT-CRITERION-VALUE>
          </POST-BUILD-VARIANT-CRITERION-VALUES>
        </POST-BUILD-VARIANT-CRITERION-VALUE-SET>
```

[TPS_ECUC_08011] Pattern for creating a C symbol used by the EcuM/BswM to initialize post-build Bsw modules ⌈ For the name mangling of symbols of different post-build variants (configuration sets) of one BSW module, the following pattern is suggested:

<MSN>_PBVCV_<VariantCriterion.shortName>_<value>

where "MSN" represents the symbolic name of the `EcucModuleDef`, "VariantCriterion.shortName"' the name of the `PostBuildVariantCriterion`, "value" the `PostBuildVariantCriterionValue` and "PBVCV" is short from PostBuildVariantCriterionValue (e.g. Adc_PBVCV_PostBuildConfigSet_2).

Rationale for the [TPS_ECUC_08011]: The EcuM/BswM modules call the module's init function with the address (i.e. the address operator "&" is applied) of the C-init-data struct as a parameter, for `EcucModuleDef`s with `supportedConfigVariant` of `VariantPostBuild`. ⌋

2.4.8.2 Example of parameters configuration using `Variant Handling`

This section contains an example of how `PreCompile` and `PostBuild` parameters inside containers with both `postBuildChangeable` attribute set to `true` and `false` can be configured using `Variant Handling`. This approach does not utilize the `multipleConfigurationContainer` structure but instead provides ONE model for all variants.

As an example, a part of the Adc module configuration parameters is taken (see Figure 2.32). `AdcDevErrorDetect` parameter of the `AdcGeneral` container and `AdcChannelLimitCheck` parameter of the `AdcChannel` container are `PreCompile`.

`AdcChannelId` and `AdcChannelResolution` parameters of the `AdcChannel` container are `PostBuild`. The container `AdcGeneral` has `postBuildChangeable` attribute set to `false` while the container `AdcChannel` has `postBuildChangeable` attribute set to `true`.

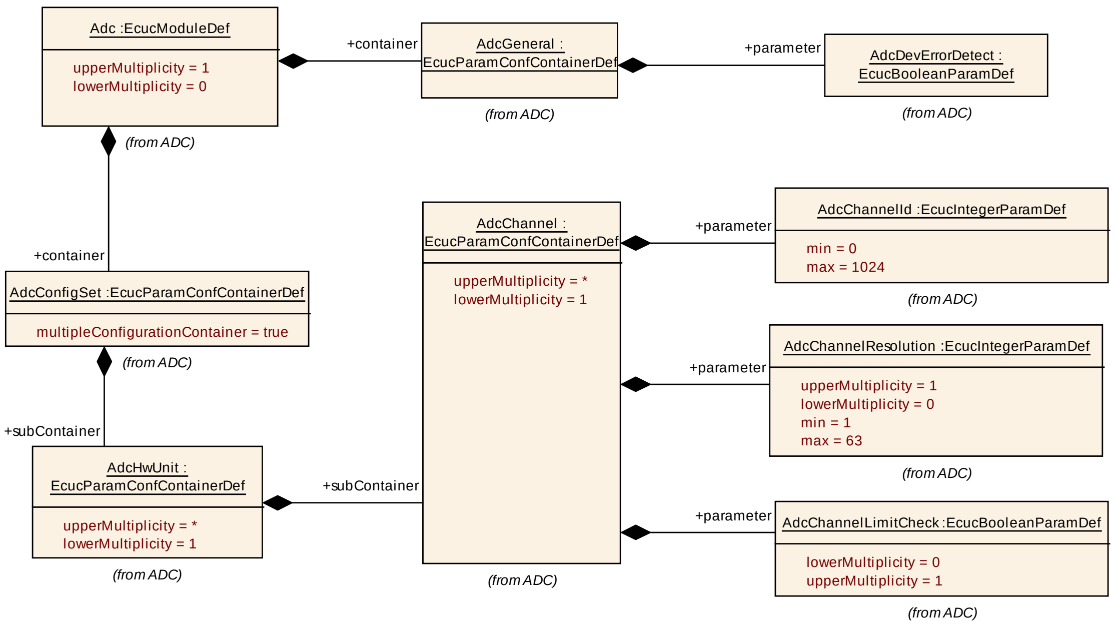

Figure 2.32: Example of Parameters Configuration Using Variant Handling

2.4.8.2.1 Values of PreCompile parameters

The PreCompile parameters are provided inside the container structure they are defined in, just like in case of using the multipleConfigurationContainer approach.

Example 2.52

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>theAdc</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/EcucDemo/Adc</DEFINITION-
      REF>
  <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-POST-BUILD-SELECTABLE</
      IMPLEMENTATION-CONFIG-VARIANT>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>myGeneralValue</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
          EcucDemo/Adc/AdcGeneral</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-NUMERICAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/EcucDemo/
              Adc/AdcGeneral/AdcDevErrorDetect</DEFINITION-REF>
          <VALUE>1</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
      <SUB-CONTAINERS/>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

2.4.8.2.2 Values of PreCompile parameters in postBuildChangeable containers

In case of using the Variant Handling approach, the PreCompile parameters inside the postBuildChangeable containers do not have to be duplicated throughout every configuration set, which was the case when using the multipleConfigurationContainer approach.

It is enough to define PreCompile parameters once for all configuration sets. This way, it is also guaranteed that the value of these parameters will be the same in all configuration sets.

Example 2.53

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myHwUnit</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/EcucDemo/
     Adc/AdcConfigSet/AdcHwUnit</DEFINITION-REF>
  <SUB-CONTAINERS>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>myChannel1</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/EcucDemo/
       Adc/AdcConfigSet/AdcHwUnit/AdcChannel</DEFINITION-REF>
    <PARAMETER-VALUES>
    <!-- PreCompile value -->
    <ECUC-NUMERICAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/EcucDemo/Adc/
         AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelLimitCheck</
         DEFINITION-REF>
      <VALUE>0</VALUE>
    </ECUC-NUMERICAL-PARAM-VALUE>
    </PARAMETER-VALUES>
  </ECUC-CONTAINER-VALUE>
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
```

2.4.8.2.3 Values of PostBuild parameters which shall be configured in a pre-compile semantics

There are use-cases where a pre-compile module's configuration shall refer to a postBuildChangeable container. In this case some identification element needs to be fixed for all post-build variants. In this example the AdcChannelId is defined as post-build configurable, but it is the intention of this specific values definition that the AdcChannelId value is fixed for all configuration sets.

Example 2.54

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>myGeneralValue</SHORT-NAME>
```

```
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/EcucDemo/Adc/
        AdcGeneral</DEFINITION-REF>
    <PARAMETER-VALUES>
    <ECUC-NUMERICAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/EcucDemo/Adc/
          AdcGeneral/AdcDevErrorDetect</DEFINITION-REF>
      <VALUE>1</VALUE>
    </ECUC-NUMERICAL-PARAM-VALUE>
    </PARAMETER-VALUES>
    <SUB-CONTAINERS/>
  </ECUC-CONTAINER-VALUE>
```

2.4.8.2.4 Values of PostBuild parameters

For the definition of the different values of different post-build configuration sets the post-build Variant Handling shall be applied. The applicability of a specific value to a post-build configurable parameter is defined by the usage of the PostBuildVariantCriterion which specifies a condition for which this specific value is applicable.

The PostBuildVariantCriterion is referenced to define the selector. A specific value for the selector is defined to specify which configuration set this parameter value is associated.

Example 2.55

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/EcucDemo/Adc/
      AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelResolution</
      DEFINITION-REF>
  <VARIATION-POINT>
  <POST-BUILD-VARIANT-CONDITIONS>
    <POST-BUILD-VARIANT-CONDITION>
    <MATCHING-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
        EcucDemo/PostBuildConfigSet</MATCHING-CRITERION-REF>
    <VALUE>1</VALUE>
    <!-- PostBuildFrontLeft -->
    </POST-BUILD-VARIANT-CONDITION>
  </POST-BUILD-VARIANT-CONDITIONS>
  </VARIATION-POINT>
  <VALUE>10</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

A different configuration set might be defined like this for the same configuration parameter:

Example 2.56

```
<ECUC-NUMERICAL-PARAM-VALUE>
```

```
    <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/EcucDemo/Adc/
       AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelResolution</
       DEFINITION-REF>
    <VARIATION-POINT>
    <POST-BUILD-VARIANT-CONDITIONS>
      <POST-BUILD-VARIANT-CONDITION>
      <MATCHING-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
         EcucDemo/PostBuildConfigSet</MATCHING-CRITERION-REF>
      <VALUE>2</VALUE>
      <!-- PostBuildFrontRight -->
      </POST-BUILD-VARIANT-CONDITION>
    </POST-BUILD-VARIANT-CONDITIONS>
    </VARIATION-POINT>
    <VALUE>40</VALUE>
  </ECUC-NUMERICAL-PARAM-VALUE>
```

2.4.8.2.5 Containers which are present in all post-build configuration sets

In case a container shall be used in all configuration variants (because there are pre-compile configurations pointing to this container), it shall not define a variation point and thus indicate its "pre-compile" nature.

2.4.8.2.6 Adding further containers in case of `postBuildChangeable` = `true`

In case a container exists only in some configuration variants, it shall define a variation point. Also all included elements shall then define the respective variation point.

Example 2.57

```
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>myChannel5</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/EcucDemo/Adc/
       AdcConfigSet/AdcHwUnit/AdcChannel</DEFINITION-REF>
    <PARAMETER-VALUES>
    <!-- PreCompile value for newly created container -->
    <ECUC-NUMERICAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/EcucDemo/Adc/
         AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelLimitCheck</
         DEFINITION-REF>
      <VALUE>1</VALUE>
    </ECUC-NUMERICAL-PARAM-VALUE>
    <!-- PostBuild value -->
    <ECUC-NUMERICAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/EcucDemo/Adc/
         AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelId</DEFINITION-REF
         >
      <VALUE>5</VALUE>
    </ECUC-NUMERICAL-PARAM-VALUE>
    <!-- -->
    <ECUC-NUMERICAL-PARAM-VALUE>
      <DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/EcucDemo/Adc/
         AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelResolution</
         DEFINITION-REF>
      <VARIATION-POINT>
      <POST-BUILD-VARIANT-CONDITIONS>
        <POST-BUILD-VARIANT-CONDITION>
        <MATCHING-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
           EcucDemo/PostBuildConfigSet</MATCHING-CRITERION-REF>
        <VALUE>2</VALUE>
        <!-- PostBuildFrontRight -->
        </POST-BUILD-VARIANT-CONDITION>
      </POST-BUILD-VARIANT-CONDITIONS>
      </VARIATION-POINT>
      <VALUE>30</VALUE>
    </ECUC-NUMERICAL-PARAM-VALUE>
    </PARAMETER-VALUES>
    <VARIATION-POINT>
    <POST-BUILD-VARIANT-CONDITIONS>
      <POST-BUILD-VARIANT-CONDITION>
      <MATCHING-CRITERION-REF DEST="POST-BUILD-VARIANT-CRITERION">/
         EcucDemo/PostBuildConfigSet</MATCHING-CRITERION-REF>
      <VALUE>2</VALUE>
      <!-- PostBuildFrontRight -->
      </POST-BUILD-VARIANT-CONDITION>
    </POST-BUILD-VARIANT-CONDITIONS>
    </VARIATION-POINT>
  </ECUC-CONTAINER-VALUE>
```

3 ECU Configuration Parameter Definition SWS implications

In this section several aspects of applying the ECU Configuration Specification to AUTOSAR specifications are described.

The ECU Configuration Parameter Definitions are distributed over the BSW SWS documents. How these parameters are specified in the documents is described in section 3.1.

How the AUTOSAR COM-Stack is configured from an inter-module perspective is described in section 3.4.

3.1 Formalization aspects

The goal of this section is to describe how the ECU Configuration Parameter Definitions of BSW modules are specified in the SWS documents. Therefore there is not necessarily a simple translation of the ECU Configuration Parameter's values in the ECU Configuration Value description (XML file) into the module's configuration (header file). It is the duty of the module's generation tool to transform the configuration information from the XML file into a header file.

The ECU Configuration Parameter Definitions are formalized in an UML model. This UML model is used to partly generate the specification tables of the BSW SWS and to generate the ECU Configuration Parameter Definition XML file. [1]

Some formalization patterns have been applied when developing the ECU Configuration Parameter Definition:

- *Modified parameter names*: Due to the limitations imposed by the AUTOSAR XML format (32 character limit starting with a letter, etc.) the names of parameters and containers have been redefined. Also a different naming schema has been applied. The original names from the SWS are provided in this document as well.
- *Added parameter multiplicities*: In the original tables from the BSW SWS there is no possibility to specify the optionality and multiplicity of parameters. The parameter multiplicities have been added.
- *Added references*: To allow a better interaction of the configuration Value descriptions of several modules references between the configuration have been introduced.
- *Harmonized parameter types*:

[1]The generation from the UML model is only one way to create the ECU Configuration Parameter Definition XML file. ECUC Parameters can be defined by any other method as long as an AUTOSAR conforming ECUC Parameter Definition XML file is created.

- Boolean: Some parameters have been defined as `enumeration` or `#define` where the actual information stored is of type boolean. In those cases they have been modeled as `boolean`.
- Float: Some parameters store a time value as `integer` where it is stated that this is a time in e.g. micro-seconds. If the time specified is an absolute time it has been formalized as a `float` in seconds. If the time is a factor of some given time-base the `integer` is preserved.

3.1.1 ECU Configuration Parameter Definition table

The configuration parameters are structured into containers which can hold parameters, references and other containers. Beside the graphical visualization in UML diagrams, tables are used to specify the structure of the parameters.

In the following table one container is specified which holds two parameters and also two additional containers as an example.

SWS Item	[SWS requirement IDs]
Container Name	ContainerName {original name from SWS}
Description	Container description.
Configuration Parameters	

Name	ParameterName {original name from SWS} [SWS requiremenets IDs]		
Description	Parameter description.		
Multiplicity	Parameter multiplicity		
Type	Parameter type		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	–	
Scope / Dependency			

Name	EnumerationTest {ENUMERATION_TEST} [SWS0815]		
Description	description.		
Multiplicity	0..1 (optional)		
Type	EnumerationParamDef		
Range	ReceiveUnqueuedExternal		
	ReceiveUnqueuedInternal		
	SendStaticExternal		
	SendStaticInternal		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers

Container Name	Multiplicity	Scope / Depedency
Container_1	0..1	Optional sub-container.
Container_2	0..*	Optional sub-container which can be present several times.

For a detailed description of the elements in the tables please refer to chapter 2.

3.2 AUTOSAR Stack Overview

The software architecture of an AUTOSAR ECU has been divided into several parts to allow independent modules with clean definitions of the interfaces between the different modules. This architecture is depicted in figure 3.1.

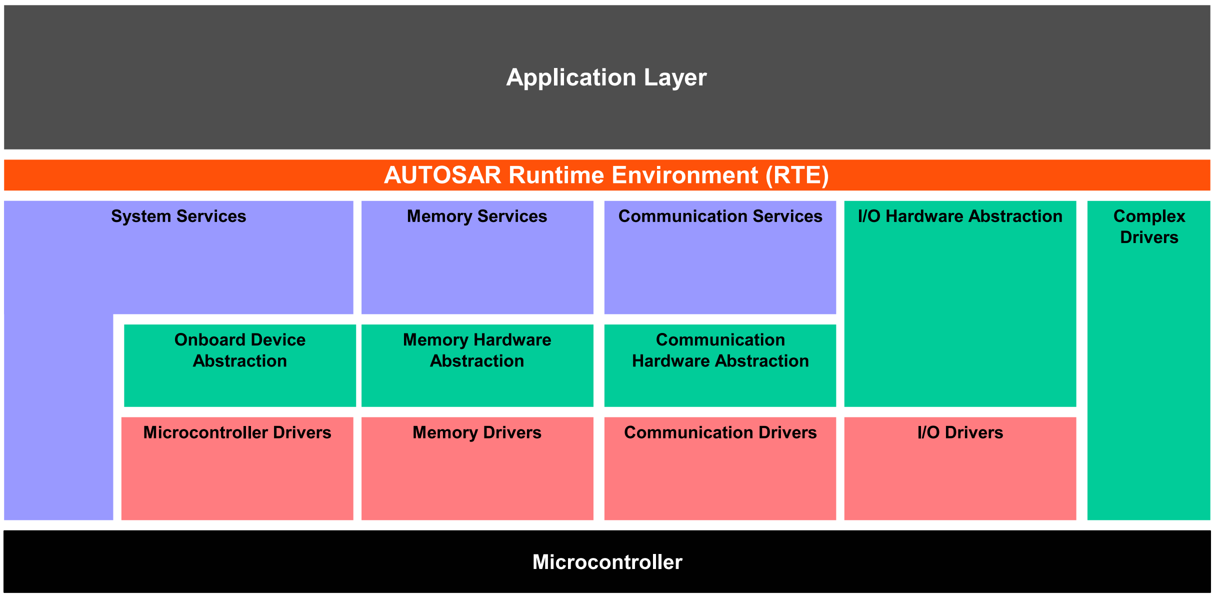

Figure 3.1: ECU Architecture Overview [15]

The Application SW-Components are located at the top and can gain access to the rest of the ECU and also to other ECUs only through the RTE.

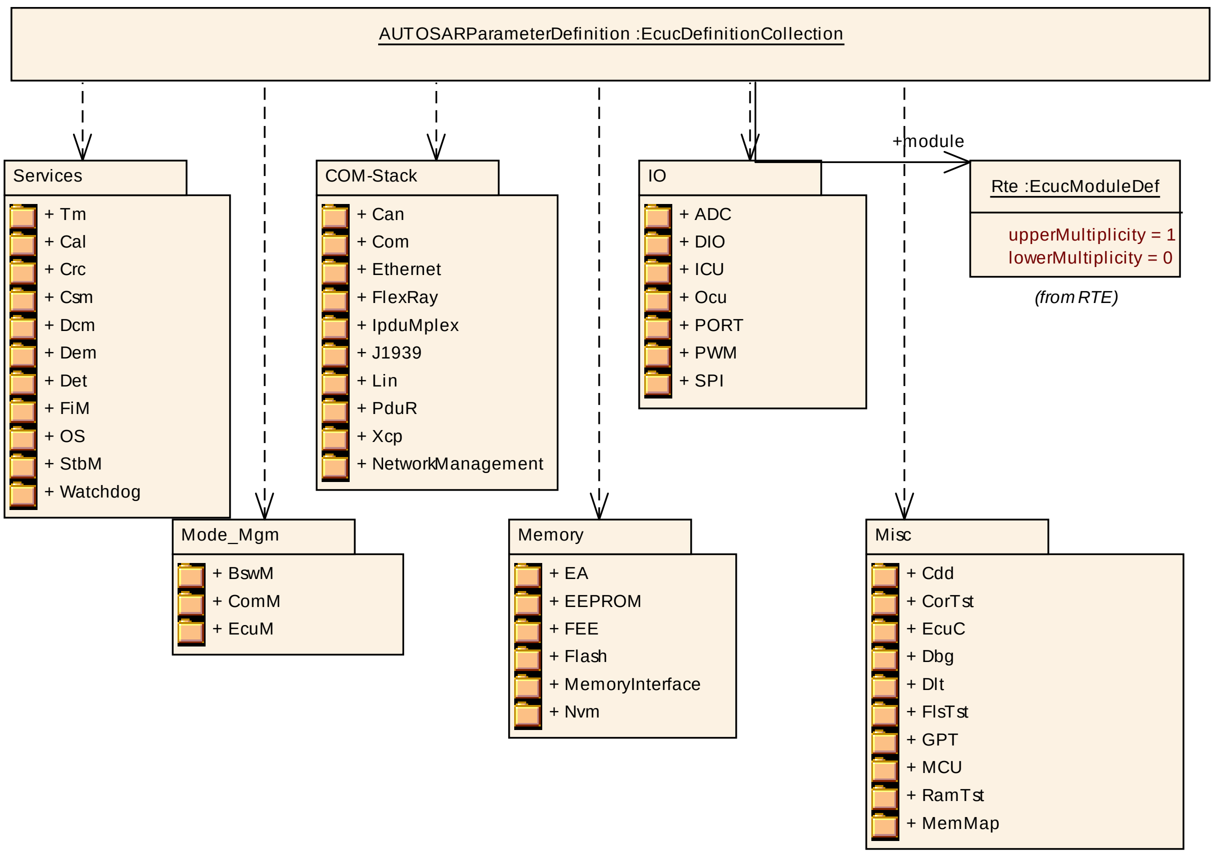

Figure 3.2: AUTOSAR Parameter Definition Overview

The RTE provides the encapsulation of communication and basic services to the Application SW-Components, so it is possible to map the Application SW-Components between different ECUs.

The Basic Software Modules are located below the RTE. The Basic Software itself is divided into the subgroups: System Services, Memory, Communication and IO HW-Abstraction. The Complex Drivers are also located below the RTE.

Among other, the Operating System (OS), the Watchdog manager and the Diagnostic services are located in the System Services subgroup.

The Memory subgroup contains modules to provide access to the non-volatile memories, namely Flash and EEPROM.

In the Communication subgroup the whole AUTOSAR communication stack (COM-Stack) is specified including the COM, Network Management and the communication drivers.

The top-level structure of the `AUTOSARParameterDefinition` is shown in figure 3.2.

The container `AUTOSARParameterDefinition` is the top-level element of the AUTOSAR ECU Configuration Parameter Definition structure. Inside this container references to the diverse configuration container definitions for the different SW modules are defined.

The upper multiplicities defined in the context of each `EcucModuleDef` directly impact the instantiation of the specific modules. If `EcucModuleDef.upperMultiplicity` is set to 1 this means that the respective module can only appear once in an AUTOSAR BSW stack. If the value of `EcucModuleDef.upperMultiplicity` is greater than 1 (i.e. 0..*) the module can be multiply instantiated.

ECU Conf. Name	AUTOSARParameterDefinition	
ECU Conf. Description	Top level container for the definition of AUTOSAR configuration parameters. All of the parameter definitions for the different modules are contained in this container.	
Included Modules		
Module Name	**Multiplicity**	**Scope / Dependency**
Adc	0..1	Configuration of the Adc (Analog Digital Conversion) module.
BswM	0..1	Configuration of the BswM (Basic SW Mode Manager) module.
Cal	0..1	Configuration of the Cal (CryptoAbstractionLibrary) module.
Can	0..*	This container holds the configuration of a single CAN Driver.
CanIf	0..1	This container includes all necessary configuration sub-containers according the CAN Interface configuration structure.
CanNm	0..1	Configuration Parameters for the Can Nm module.
CanSM	0..1	Configuration of the CanSM module
CanTp	0..1	Configuration of the CanTp (CAN Transport Protocol) module.
CanTrcv	0..*	Configuration of the CanTrcv (CAN Transceiver driver) module.
Cdd	0..*	The CDD module describes the minimal requirements that are necessary for the configuration of a CDD with respect to the surrounding standardized BSW modules.
Com	0..1	Configuration of the AUTOSAR COM module.
ComM	0..1	Configuration of the ComM (Communications Manager) module.
CorTst	0..1	Configuration of the CorTst module.
Crc	0..1	Configuration of the Crc (Crc routines) module.
Csm	0..1	Configuration of the Csm (CryptoServiceManager) module.
Dbg	0..1	Configuration of the debugging module.
Dcm	0..1	Configuration of the Dcm (Diagnostic Communications Manager) module.
Dem	0..1	Configuration of the Dem (Diagnostic Event Manager) module.
Det	0..1	Det configuration includes the functions to be called at notification. On one side the application functions are specified and in general it can be decided whether Dlt shall be called at each call of Det.
Dio	0..1	Configuration of the Dio (Digital IO) module.
Dlt	0..1	
DoIP	0..1	Configuration of the DoIP (Diagnostic over IP) module.

Module Name	Multiplicity	Scope / Dependency
Ea	0..1	Configuration of the Ea (EEPROM Abstraction) module. The module shall abstract from the device specific addressing scheme and segmentation and provide the upper layers with a virtual addressing scheme and segmentation as well as a 'virtually' unlimited number of erase cycles.
EcuC	0..1	Virtual module to collect ECU Configuration specific / global configuration information.
EcuM	0..1	Configuration of the EcuM (ECU State Manager) module.
Eep	0..*	Configuration of the Eep (internal or external EEPROM driver) module. Its multiplicity describes the number of EEPROM drivers present, so there will be one container for each EEPROM driver in the ECUC template. When no EEPROM driver is present then the multiplicity is 0.
Eth	0..*	Configuration of the Eth (Ethernet Driver) module.
EthIf	0..1	Configuration of the EthIf (Ethernet Interface) module.
EthSM	0..1	Configuration of the Ethernet State Manager
EthTrcv	0..*	Configuration of Ethernet Transceiver Driver module
Fee	0..1	Configuration of the Fee (Flash EEPROM Emulation) module.
FiM	0..1	Configuration of the FiM (Function Inhibition Manager) module.
Fls	0..*	Configuration of the Fls (internal or external flash driver) module. Its multiplicity describes the number of flash drivers present, so there will be one container for each flash driver in the ECUC template. When no flash driver is present then the multiplicity is 0.
FlsTst	0..1	
Fr	0..*	Configuration of the Fr (FlexRay driver) module.
FrArTp	0..1	Configuration of the FrArTp (FlexRay Transport Protocol) module.
FrIf	0..1	Configuration of the FrIf (FlexRay Interface) module.
FrNm	0..1	The Flexray Nm module
FrSM	0..1	Configuration of the FlexRay State Manager
FrTp	0..1	Configuration of the FlexRay Transport Protocol module according to ISO 10681-2.
FrTrcv	0..*	Configuration of the FrTrcv (FlexRay Transceiver driver) module.
Gpt	0..1	Configuration of the Gpt (General Purpose Timer) module.
Icu	0..*	Configuration of the Icu (Input Capture Unit) module.
IpduM	0..1	Configuration of the IpduM (Ipdu Multiplexer) module.
J1939Dcm	0..1	The SAE J1939 Dcm module
J1939Nm	0..1	Configuration of the J1939 Network Management module.
J1939Rm	0..1	Configuration of the J1939 Request Manager.
J1939Tp	0..1	Configuration of the J1939Tp (J1939 Transport Protocol) module.
Lin	0..*	Configuration of the Lin (LIN driver) module.
LinIf	0..1	Configuration of the LinIf (LIN Interface) module.
LinNm	0..1	Configuration Parameters for the Lin Nm module.

Module Name	Multiplicity	Scope / Dependency
LinSM	0..1	Configuration of the Lin State Manager module.
LinTp	0..1	Configuration of the LIN Transport Protocol.
LinTrcv	0..*	Configuration of LIN Transceiver Driver module
Mcu	0..1	Configuration of the Mcu (Microcontroller Unit) module.
MemIf	0..1	Configuration of the MemIf (Memory Abstraction Interface) module.
MemMap	0..1	Configuration of the Memory Mapping and Compiler Abstraction module.
Nm	0..1	The Generic Network Management Interface module
NvM	0..1	Configuration of the NvM (NvRam Manager) module.
Ocu	0..1	Configuration of Ocu (Output Compare Unit) module.
Os	0..1	Configuration of the Os (Operating System) module.
PduR	0..1	Configuration of the PduR (PDU Router) module.
Port	0..1	Configuration of the Port module.
Pwm	0..*	Configuration of Pwm (Pulse Width Modulation) module.
RamTst	0..1	Configuration of the RamTst module.
Rte	0..1	Configuration of the Rte (Runtime Environment) module.
Sd	0..1	Configuration of the Service Discovery module.
SoAd	0..1	Configuration of the SoAd (Socket Adaptor) module.
Spi	0..1	Configuration of the Spi (Serial Peripheral Interface) module.
StbM	0..1	Configuration of the Synchronized Time-base Manager (StbM) module.
TcpIp	0..1	Configuration of the TcpIp (TCP/IP stack) module.
Tm	0..1	Configuration of the Time Service module.
UdpNm	0..1	
Wdg	0..*	Configuration of the Wdg (Watchdog driver) module.
WdgIf	0..1	Configuration of the WdgIf (Watchdog Interface) module.
WdgM	0..1	Configuration of the WdgM (Watchdog Manager) module.
Xcp	0..1	Configuration of the XCP module

3.3 Virtual Module EcuC

In the configuration of an ECU there is information which needs to be shared between multiple BSW Modules. Since it can not be defined who owns this shared information the *virtual* module EcuC has been introduced to the AUTOSAR ECU Configuration Parameter Definition.

Module Name	EcuC	
Module Description	Virtual module to collect ECU Configuration specific / global configuration information.	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EcucConfigSet	0..1	This container contains the configuration parameters and sub containers of the global PduCollection. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
EcucHardware	0..1	Hardware definition of this Ecu.
EcucPartitionCollection	0..1	Collection of Partitions defined for this ECU.
EcucUnitGroupAssignment	0..1	Collection of UnitGroup references to support the generation of ASAM MCD file.
EcucVariationResolver	0..1	Collection of PredefinedVariant elements containing definition of values for SwSystemconst which shall be applied when resolving the variability during ECU Configuration.

EcucConfigSet

SWS Item	[ECUC_EcuC_00061]
Container Name	EcucConfigSet[Multi Config Container]
Description	This container contains the configuration parameters and sub containers of the global PduCollection. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
EcucPduCollection	0..1	Collection of all Pdu objects flowing through the Com-Stack.

3.3.1 Hardware description

In order to allow the unique description and access to hardware resources the EcucHardware has been introduced.

One section of the EcucHardware is concerned with the definition of computation cores and the assignment of unique EcucCoreIds to these cores. Additionally it is possible to refer to the Ecu Resource Template `HwElement` which represents the core in hardware.

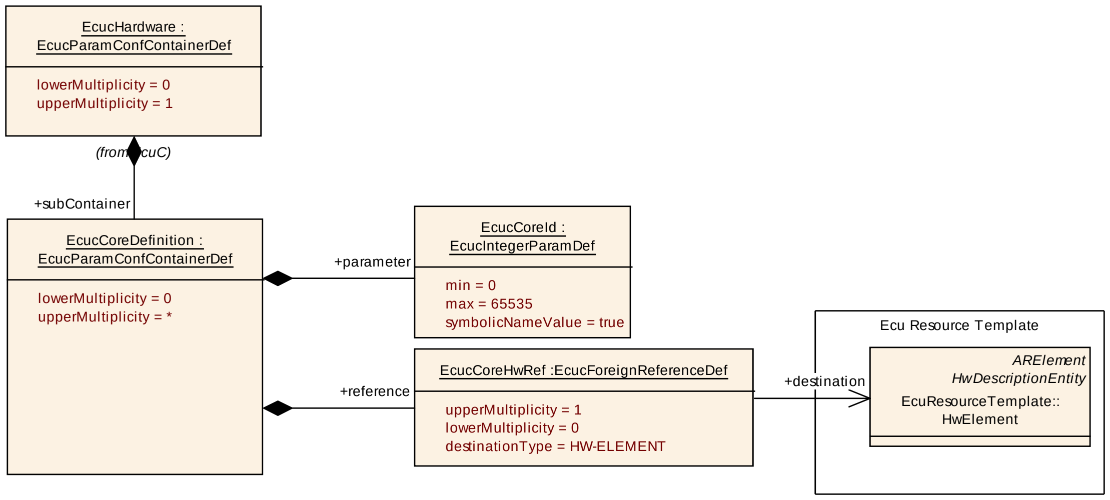

Figure 3.3: Description of ECU Hardware

EcucHardware

SWS Item	[ECUC_EcuC_00056]
Container Name	EcucHardware
Description	Hardware definition of this Ecu.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
EcucCoreDefinition	0..*	Definition of one Core on this Ecu.

EcucCoreDefinition

SWS Item	[ECUC_EcuC_00057]
Container Name	EcucCoreDefinition
Description	Definition of one Core on this Ecu.
Configuration Parameters	

Name	EcucCoreHwRef [ECUC_EcuC_00059]		
Description	Optional reference to the HwElement of HwCategory ProcessingUnit that represents this Core in the ECU Resource Template.		
Multiplicity	0..1		
Type	Foreign reference to HW-ELEMENT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucCoreId [ECUC_EcuC_00058]		
Description	ID of the core.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.3.2 Definition of Partitions

In order to support *memory-partitioning* and *multi-core* the notion of a *EcucPartition* has been introduced into the EcuC virtual Module.

The EcuC Module can have one `EcucPartitionCollection` which can hold an arbitrary number of `EcucPartition` elements. The *memory-partitioning* enables to create protection boundaries around groups of SWCs. The allocation of SWCs to `EcucPartition`s is possible via the `EcucPartitionSoftwareComponentInstanceRef` reference to SW Component instances. An `EcucPartition` is implemented by an OS-Application within the OS. Therefore the mapping of SWCs to partitions restricts the runnable to task mapping as shown in figure 3.4.

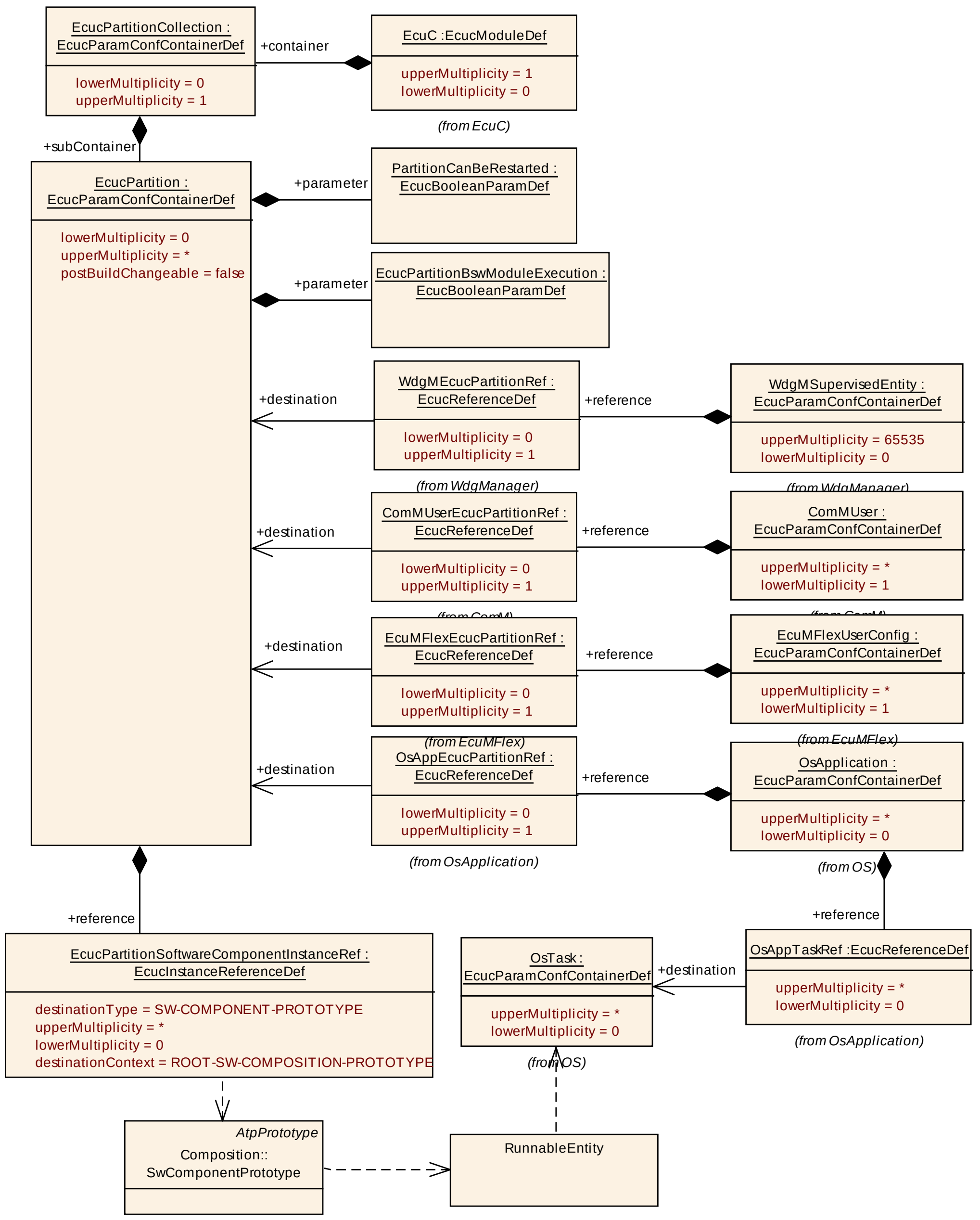

Figure 3.4: Definition of Partitions on one ECU

EcucPartitionCollection

SWS Item	[ECUC_EcuC_00007]
Container Name	EcucPartitionCollection
Description	Collection of Partitions defined for this ECU.
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Depedency
EcucPartition	0..*	Definition of one Partition on this ECU. One Partition will be implemented using one Os-Application.

EcucPartition

SWS Item	[ECUC_EcuC_00005]
Container Name	EcucPartition
Description	Definition of one Partition on this ECU. One Partition will be implemented using one Os-Application. **Attributes:** postBuildChangeable=false
Configuration Parameters	

Name	EcucPartitionBswModuleDistinguishedPartition [ECUC_EcuC_00068]		
Description	This maps the abstract partition of the Bsw Module to a concrete Partition existing in the ECU.		
Multiplicity	0..*		
Type	Foreign reference to BSW-DISTINGUISHED-PARTITION		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucPartitionBswModuleExecution [ECUC_EcuC_00037]		
Description	Denotes that this partition will execute BSW Modules. BSW Modules can only be executed in such partitions.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucPartitionSoftwareComponentInstanceRef [ECUC_EcuC_00036]		
Description	References the SW Component instances from the Ecu Extract that shall be executed in this partition.		
Multiplicity	0..*		
Type	Instance reference to SW-COMPONENT-PROTOTYPE context: ROOT-SW-COMPOSITION-PROTOTYPE		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	PartitionCanBeRestarted [ECUC_EcuC_00006]		
Description	Specifies the requirement whether the Partition can be restarted. If set to true all software executing in this partition shall be capable of handling a restart.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

The design principle is that after the creation of a partition the software (SWC) is mapped to this partition. In the second step the BSW is configured and every member of a partition (BSW) defines a reference to the `EcucPartition` element.

One example is the Os module: The Os-Application is used to implement one Partition, therefore there shall be a reference from each Os-Application to one Partition which specifies which partition this Os-Application is implementing.

Another example is the interaction of a SWC with the ComM: A SWC running in a partition other than the BSW modules is requesting full communication at the ComM. If now the partition which the SWC is running in will be stopped due to an partition violation there is now an outstanding full communication request at the ComM which will prohibit a network to be sent to sleep. With the provided configuration means it is possible to implement counter measures for such use-cases.

The interaction between `EcucPartition` and `EcucCoreDefinition` is done via the `OsApplicationCoreRef` of `OsApplication`.

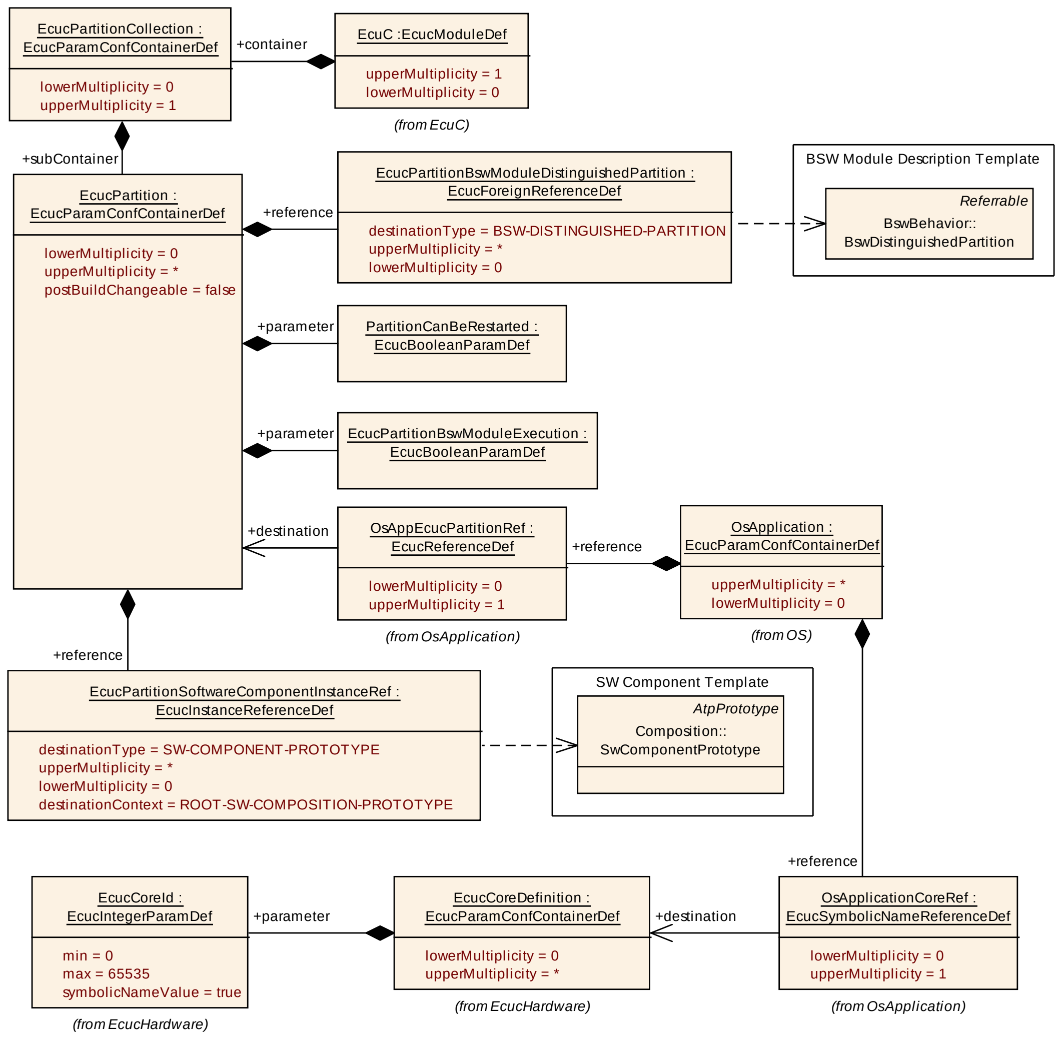
EcucPartitionCollection : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
EcuC :EcucModuleDef
upperMultiplicity = 1
lowerMultiplicity = 0
(from EcuC)
+subContainer
EcucPartition : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
postBuildChangeable = false
+reference
EcucPartitionBswModuleDistinguishedPartition : EcucForeignReferenceDef
destinationType = BSW-DISTINGUISHED-PARTITION
upperMultiplicity = *
lowerMultiplicity = 0
BSW Module Description Template
Referrable
BswBehavior:: BswDistinguishedPartition
+parameter
PartitionCanBeRestarted : EcucBooleanParamDef
+parameter
EcucPartitionBswModuleExecution : EcucBooleanParamDef
+destination
OsAppEcucPartitionRef : EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
(from OsApplication)
+reference
OsApplication : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
(from OS)
+reference
EcucPartitionSoftwareComponentInstanceRef : EcucInstanceReferenceDef
destinationType = SW-COMPONENT-PROTOTYPE
upperMultiplicity = *
lowerMultiplicity = 0
destinationContext = ROOT-SW-COMPOSITION-PROTOTYPE
SW Component Template
AtpPrototype
Composition:: SwComponentPrototype
+reference
EcucCoreId : EcucIntegerParamDef
min = 0
max = 65535
symbolicNameValue = true
(from EcucHardware)
+parameter
EcucCoreDefinition : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
(from EcucHardware)
+destination
OsApplicationCoreRef : EcucSymbolicNameReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
(from OsApplication)

Figure 3.5: Interaction between `EcucPartition` and `EcucCoreDefinition`

3.3.3 Variation Resolver Description

In order to support the variant handling approach (see Generic Structure Template [7]) the already given values of system constants are specified in using the collection `SwSystemconstantValueSet`. In the `EcuC` the applicable `SwSystemconstant-ValueSet` elements are referenced indirectly via the `PredefinedVariant` collection.

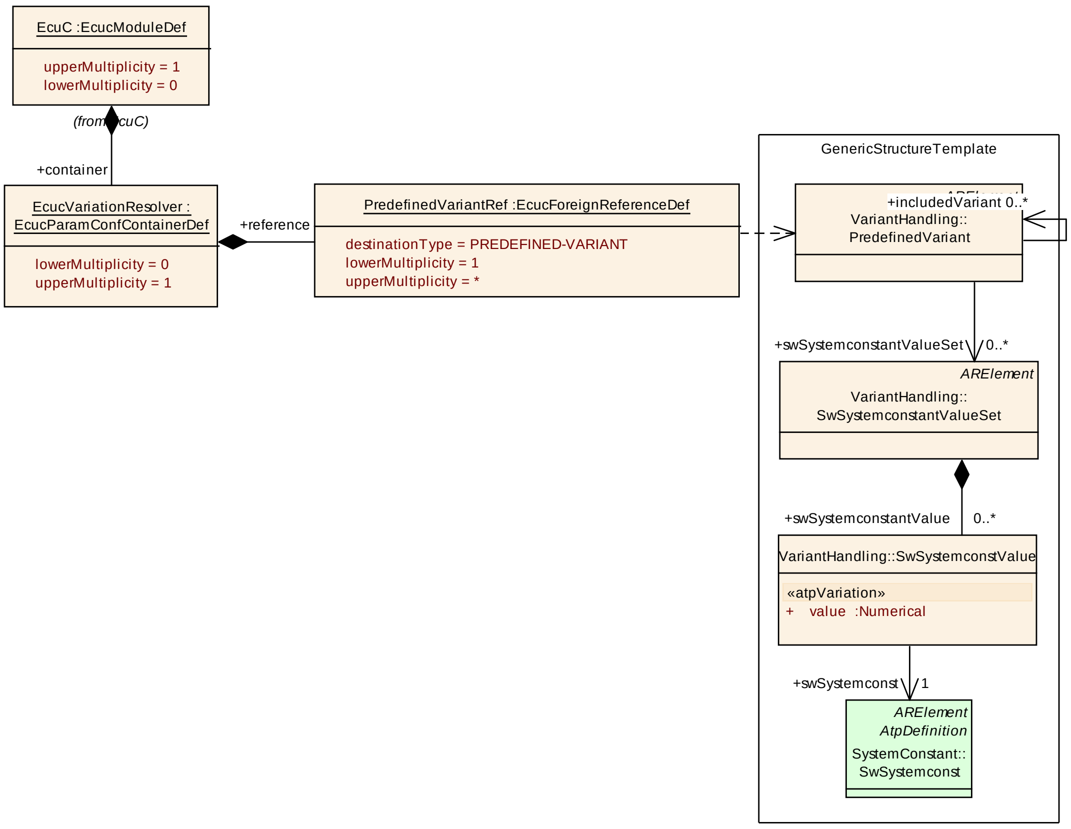

Figure 3.6: Description of Variation Resolver

EcucVariationResolver

SWS Item	[ECUC_EcuC_00009]
Container Name	EcucVariationResolver
Description	Collection of PredefinedVariant elements containing definition of values for SwSystemconst which shall be applied when resolving the variability during ECU Configuration.
Configuration Parameters	

Name	PredefinedVariantRef [ECUC_EcuC_00010]		
Description			
Multiplicity	1..*		
Type	Foreign reference to PREDEFINED-VARIANT		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.3.4 UnitGroup Assignment

To support the generation of ASAM MCD files UnitGroups may be selected in the EcuC that are relevant for the MCD system. Please note that the EcucUnitGroupAssignment can be used to control the generation of the A2L file in a way that the units used for calculation are replaced by application domain specific units.

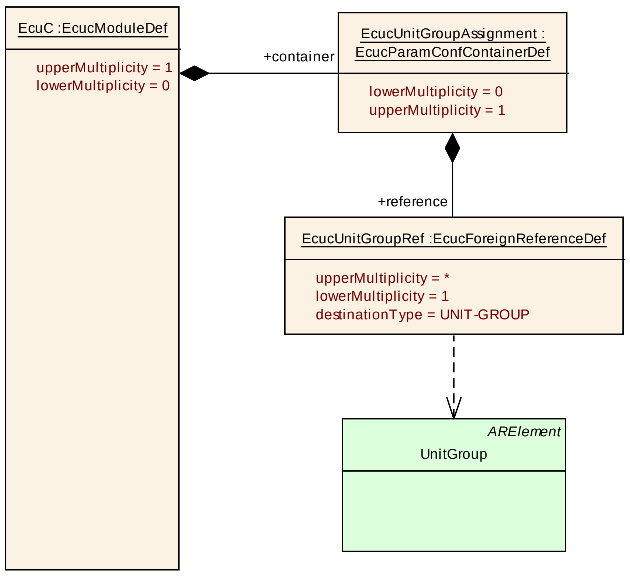

Figure 3.7: Assignment of UnitGroups

EcucUnitGroupAssignment

SWS Item	[ECUC_EcuC_00063]
Container Name	EcucUnitGroupAssignment
Description	Collection of UnitGroup references to support the generation of ASAM MCD file.
Configuration Parameters	

Name	EcucUnitGroupRef [ECUC_EcuC_00062]		
Description	Optional reference to the UnitGroup to support the generation of ASAM MCD file. These UnitGroups are selecting a set of units for a specific country.		
Multiplicity	1..*		
Type	Foreign reference to UNIT-GROUP		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.3.5 Definition of Pdus

In order to support the synchronization of Handle IDs (see section 3.4.1) two modules need to be able to refer to the same Pdu object[2]. Therefore a generic Pdu container has been defined which does not belong to any module but is defined in the EcuC module.

Since the Pdu flowing through the COM-Stack does not belong to an individual module, the "virtual" module EcuC has been introduced in the ECU Configuration. This module is used to collect configuration information not associated with any specific standardized module.

The EcucPduCollection may contain several "global" Pdu objects as shown in figure 3.8. Each Pdu may either represent a FrameTriggering (for Pdus not going through the Pdu Router: UserDefinedPdus, NmPdus and NPdus) or PduTriggering (for all other Pdus) belonging to the specific ECU from the AUTOSAR System Description[2] (ECU Extract). Therefore there is an optional reference to either FrameTriggering (SysTPduToFrameTriggeringRef) or PduTriggering (SysTPduToPduTriggeringRef) element in the System Template. Either SysTPduToFrameTriggeringRef or SysTPduToPduTriggeringRef shall be used.

[2]For the aspect of the configuration it does not matter what kind of Pdu it is, i.e. I-PDU, L-PDU or N-PDU.

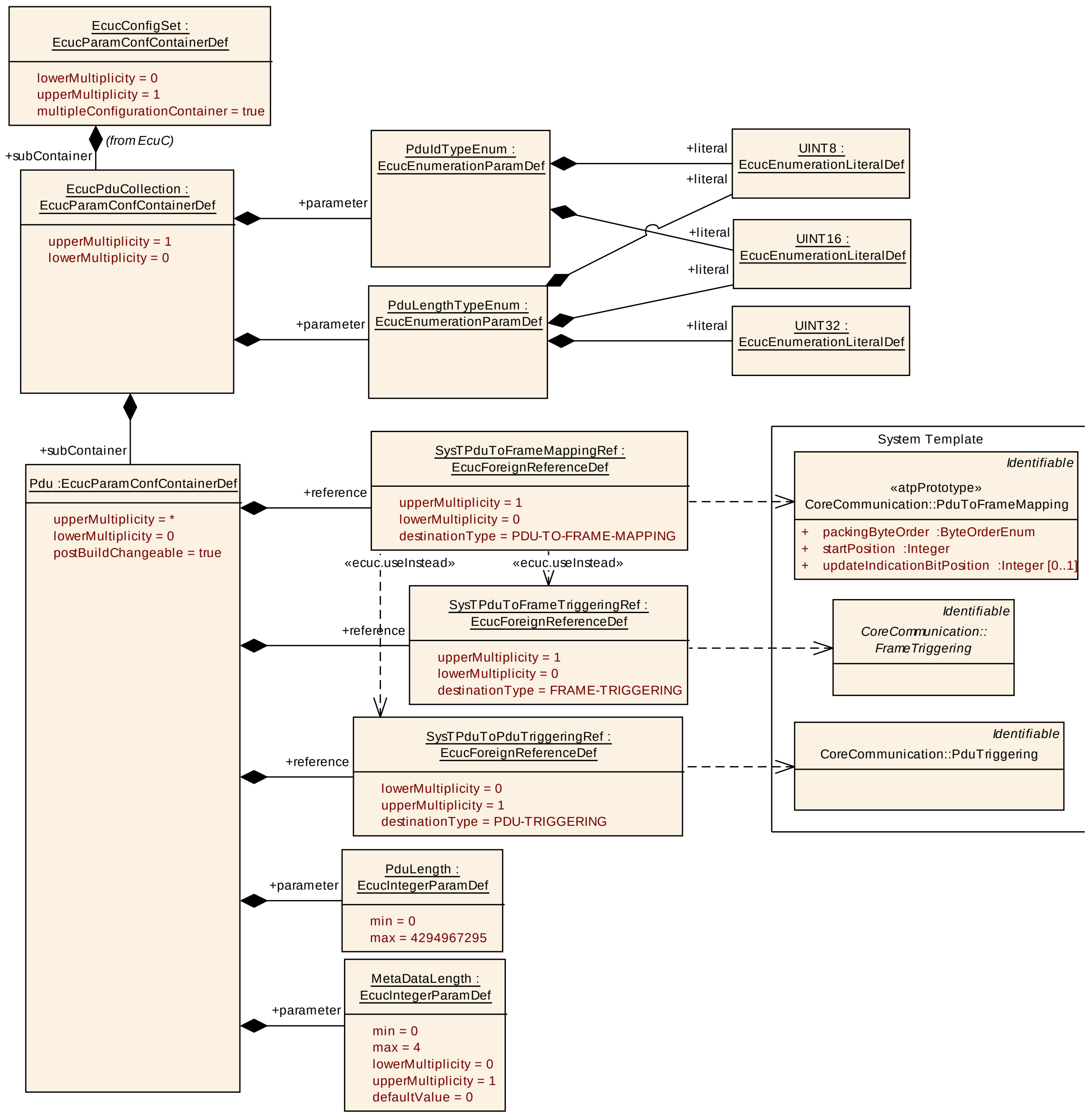
EcucConfigSet : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
multipleConfigurationContainer = true
(from EcuC)
+subContainer
EcucPduCollection : EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 0
+parameter
PduIdTypeEnum : EcucEnumerationParamDef
+literal
UINT8 : EcucEnumerationLiteralDef
UINT16 : EcucEnumerationLiteralDef
UINT32 : EcucEnumerationLiteralDef
PduLengthTypeEnum : EcucEnumerationParamDef
+subContainer
Pdu :EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
postBuildChangeable = true
+reference
SysTPduToFrameMappingRef : EcucForeignReferenceDef
upperMultiplicity = 1
lowerMultiplicity = 0
destinationType = PDU-TO-FRAME-MAPPING
«ecuc.useInstead»
SysTPduToFrameTriggeringRef : EcucForeignReferenceDef
upperMultiplicity = 1
lowerMultiplicity = 0
destinationType = FRAME-TRIGGERING
SysTPduToPduTriggeringRef : EcucForeignReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
destinationType = PDU-TRIGGERING
System Template
Identifiable
«atpPrototype» CoreCommunication::PduToFrameMapping
+ packingByteOrder :ByteOrderEnum
+ startPosition :Integer
+ updateIndicationBitPosition :Integer [0..1]
CoreCommunication:: FrameTriggering
CoreCommunication::PduTriggering
PduLength : EcucIntegerParamDef
min = 0
max = 4294967295
MetaDataLength : EcucIntegerParamDef
min = 0
max = 4
lowerMultiplicity = 0
upperMultiplicity = 1
defaultValue = 0

Figure 3.8: Generic Pdu Container

EcucPduCollection

SWS Item	[ECUC_EcuC_00002]
Container Name	EcucPduCollection
Description	Collection of all Pdu objects flowing through the Com-Stack.
Configuration Parameters	

Name	PduIdTypeEnum [ECUC_EcuC_00041]	
Description	The PduIdType is used within the entire AUTOSAR Com Stack except for bus drivers. The size of this global type depends on the maximum number of PDUs used within one software module. If no software module deals with more PDUs that 256, this type can be set to uint8. If at least one software module handles more than 256 PDUs, this type must be set to uint16. See AUTOSAR_SWS_CommunicationStackTypes for more details.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	UINT16	
	UINT8	
Configuration Class	**Pre-compile time**	–
	Link time	–
	Post-build time	–
Scope / Dependency		

Name	PduLengthTypeEnum [ECUC_EcuC_00042]	
Description	The PduLengthType is used within the entire AUTOSAR Com Stack except for bus drivers. The size of this global type depends on the maximum length of PDUs to be sent by an ECU. If no segmentation is used the length depends on the maximum payload size of a frame of the underlying communication system (for FlexRay maximum size is 255 bytes, therefore uint8). If segmentation is used it depends on the maximum length of a segmented N-PDU (in general uint16 is used). See AUTOSAR_SWS_CommunicationStackTypes for more details.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	UINT16	
	UINT32	
	UINT8	
Configuration Class	**Pre-compile time**	–
	Link time	–
	Post-build time	–
Scope / Dependency		

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
Pdu	0..*	One Pdu flowing through the COM-Stack. This Pdu is used by all Com-Stack modules to agree on referencing the same Pdu.

Pdu

SWS Item	[ECUC_EcuC_00001]
Container Name	Pdu
Description	One Pdu flowing through the COM-Stack. This Pdu is used by all Com-Stack modules to agree on referencing the same Pdu. **Attributes:** postBuildChangeable=true
Configuration Parameters	

Name	MetaDataLength [ECUC_EcuC_00055]		
Description	Number of additional bytes in PDU data that contain auxiliary information (MetaData) for the PDU, e.g. the CAN ID. These bytes are handled as part of the PDU data and increase the PDU data length accordingly.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 4		
Default Value	0		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Name	PduLength [ECUC_EcuC_00003]		
Description	Length of the Pdu in bytes. It should be noted that in former AUTOSAR releases (Rel 2.1, Rel 3.0, Rel 3.1, Rel 4.0 Rev. 1) this parameter was defined in bits.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Name	SysTPduToFrameMappingRef [ECUC_EcuC_00004] (Obsolete. Use SysTPduToFrameTriggeringRef, SysTPduToPduTriggeringRef instead.)		

Description	Optional reference to the PduToFrameMapping from the SystemTemplate which this Pdu represents.
	
	This parameter is obsolete and will be removed in future.
	
	Tags:
	atp.Status=obsolete
	atp.StatusComment=replaced by the SysTPduToFrameTriggeringRef and SysTPduToPduTriggeringRef
	atp.StatusRevisionBegin=4.1.1		

Multiplicity	0..1		
Type	Foreign reference to PDU-TO-FRAME-MAPPING		
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	SysTPduToFrameTriggeringRef [ECUC_EcuC_00052]		
Description	Reference to the FrameTriggering from the SystemTemplate which this Pdu belongs to. SysTPduToFrameTriggeringRef shall be used for UserDefinedPdus, NmPdus and NPdus which are not going through the Pdu Router. This reference shall not be used if SysTPduToPduTriggeringRef exists.		
Multiplicity	0..1		
Type	Foreign reference to FRAME-TRIGGERING		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: SysTPduToFrameTriggeringRef shall be used for UserDefinedPdus, NmPdus and NPdus which are not going through the Pdu Router. This reference shall not be used if SysTPduToPduTriggeringRef exists.		

Name	SysTPduToPduTriggeringRef [ECUC_EcuC_00054]		
Description	Reference to the PduTriggering from the SystemTemplate which this Pdu represents. SysTPduToPduTriggeringRef shall be used for all Pdus except UserDefinedPdus, NmPdus and NPdus which are not going through the Pdu Router. For these Pdus, SysTPduToFrameTriggeringRef shall be used.		
Multiplicity	0..1		
Type	Foreign reference to PDU-TRIGGERING		
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	dependency: SysTPduToPduTriggeringRef shall be used for all Pdus except UserDefinedPdus, NmPdus and NPdus which are not going through the Pdu Router. This reference shall not be used if SysTPduToFrameTriggeringRef exists.		

No Included Containers

[TPS_ECUC_06030] Invalid `PduLength` parameter value configuration ⌈ Configuring the `PduLength` larger than the underlying layer supports results in an invalid configuration. ⌋

3.4 COM-Stack configuration

To cope with the complexity of the COM-Stack configuration, reoccurring patterns have been applied which will be described in this section. Only the patterns, together with some examples, are shown. To get detailed specification of the configuration for each individual module please refer to the actual BSW SWS documents of these modules.

3.4.1 Handle IDs

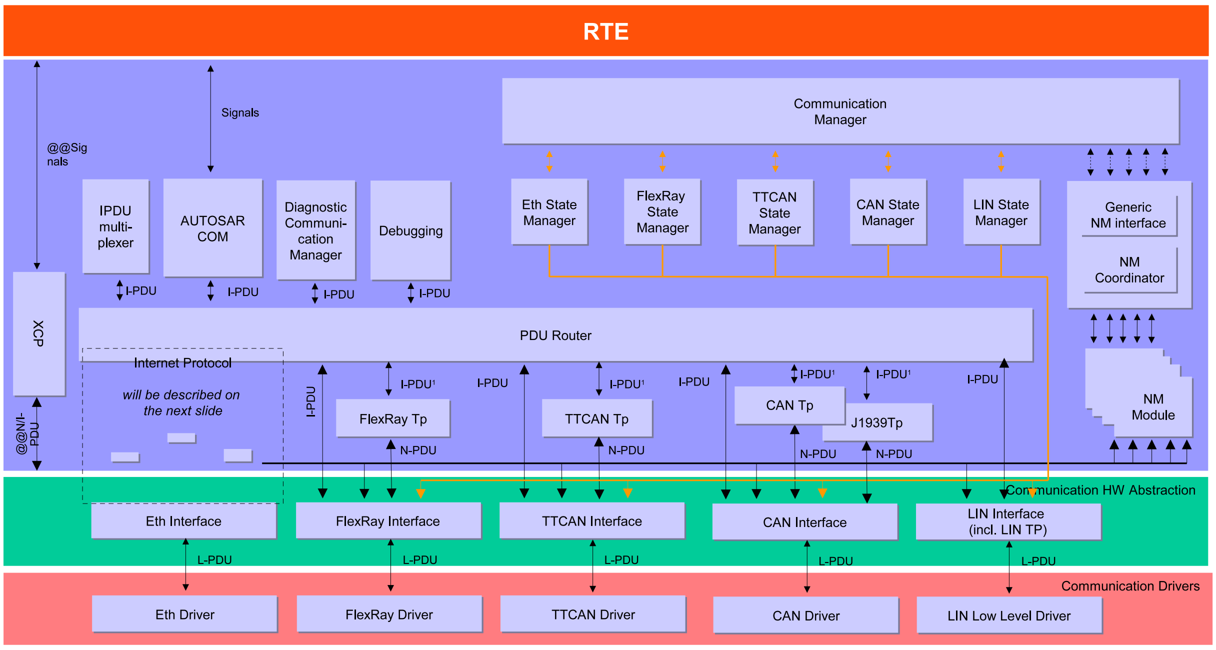

Figure 3.9: Interfaces in the COM-Stack [15]

In figure 3.9 a detailed view of the COM-Stack modules and their interaction is shown. There are several kinds of interactions between adjacent[3] modules.

3.4.1.1 Handle ID concept

The API definitions in the COM-Stack utilize two concepts to achieve the interaction between adjacent modules:

- Pointers to Pdu data buffer (the Pdu data buffer contains the actual communicated information, depending on the actual layer the interaction happens)
- Handle IDs to identify to what Pdu the pointer is referring to.

A typical API call is for instance:

`PduR_ComTransmit(PduIdType ComTxPduId, PduInfoType *PduInfoPtr)`

Which BSW Module is actually providing the value of the Handle ID is specified in the ECU Configuration Parameter Definition of the corresponding BSW Module (see section 3.4.1.2 for details on the specification).

The choice of the value for a Handle ID is open to the implementation of the providing module. There might be different strategies to optimize the Handle ID values and

[3]Modules are called adjacent if they share an interface, so PduR and Com are adjacent, while PduR and Can driver are not.

therefore the internal structures of the implementation may have an influence on the choice of the values.

Also the Handle IDs can be chosen freely per module, so a Pdu might be sent from Com to the PduR with the `ID=5` and then the PduR transmits it further to the CanIf with `ID=19`. In the configuration information of the PduR it has to be possible to conclude that if a Pdu arrives from Com with `ID=5` it has to be forwarded to the CanIf with `ID=19`.

It has to be guaranteed that each Pdu does have a unique handle ID within the scope of the corresponding API. For example: The PduR gets transmission requests from both, the Com and the Dcm modules. But there are also two distinct APIs defined for those requests:

- `PduR_ComTransmit(...)`
- `PduR_DcmTransmit(...)`

Therefore the PduR can distinguish two Pdus, even when they have the same handle ID but are requested via different APIs.

Another use-case in the COM-Stack only provides one API for all the callers: the interface layer (CanIf, FrIf, LinIf).

- `CanIf_Transmit(...)`

Here it has to be guaranteed that each transmit request for a distinct Pdu does have a unique handle ID.

The actual values of the handle IDs can only be assigned properly when the configuration of one module is completed, since only then the internal data structures can be defined.

In the next sections the patterns used to define and utilize Handle IDs are described.

3.4.1.2 Definition of Handle IDs

Handle IDs are defined by the module providing the API and used by the module calling the API. Handle IDs that are used in callback functions (e.g. Tx Confirmation functions or Trigger Transmit functions) shall be defined by the upper layer module. In the upper layer module the same HandleId shall be used for the Tx Confirmation and for the Trigger Transmit callback functions. I.e. the module that receives a transmission request can call the Tx confirmation callback with a different Handle Id than the transmission request Handle Id. This is a difference to previous releases of AUTOSAR where the Tx confirmation was called with the same Handle Id.

The ECU Configuration Value description (which holds the actual values of configuration parameters) is structured according to the individual BSW Module instances. Therefore the ECU Configuration Parameter Definition is also structured in this way.

In figure 3.10 an exemplary definition of a partial Can Interface transmit configuration is shown.

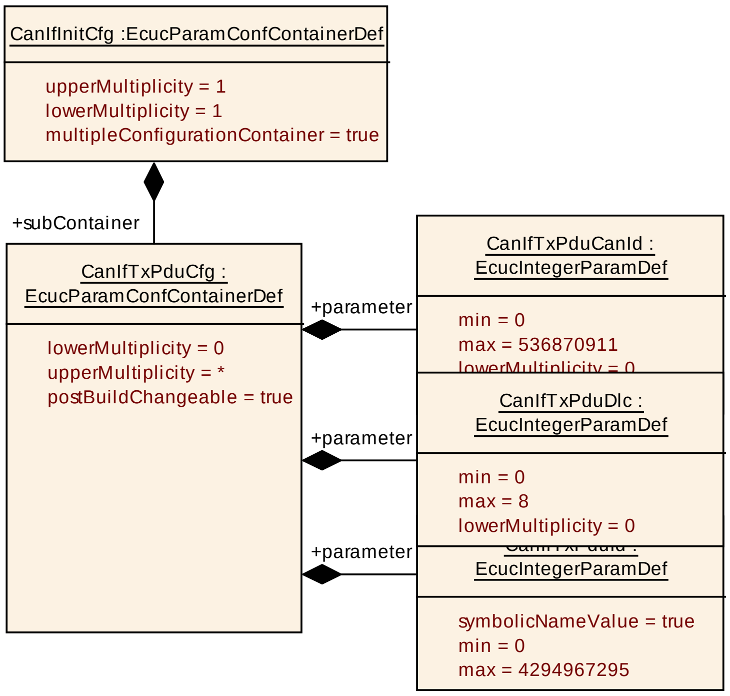

Figure 3.10: Example of Can Interface Tx configuration

The configuration of the module `CanIf` may contain several `CanIfTxPduConfig` objects.

Each `CanIfTxPduConfig` object contains information on one Pdu which is coming from an upper layer (e.g. PduR or Nm) and is going to some Can driver. In this example the `CanIfCanTxPduCanId` and `CanIfCanTxPduDlc` are specified for each to be transmitted Pdu. There is a similar structure needed for the receive use-case as well.

Additionally the parameter `CanIfCanTxPduId` is specified. This integer parameter will later hold the actual value for the handle ID. So the handle ID value is stored inside the structure of the defining module.

Since the handle ID `CanIfCanTxPduId` is part of the container `CanIfTxPduConfig` the semantics of the symbolic names can be applied.

The described example only applies for the communication between CanIf and Upper Layer modules. CanDrv does not support the handle ID concept and indicates TxConfirmation using the PduId passed during Can_Write().

[TPS_ECUC_02106] Handle Id which needs to be shared between several modules ⌈ If a configuration parameter holds a handle Id which needs to be shared between several modules it shall have the `symbolicNameValue` = true set. ⌋

Thus it is required that all handle Id values are accessible via a symbolic name reference (see section 3.4.1.4).

3.4.1.3 Agreement on Handle IDs

During the configuration of a module, information for each `Pdu` flowing through this module is created (see again figure 3.10: `CanIfTxPduConfig`) which hold module-specific configuration information. Now each of these "local" `Pdu` configurations needs to be related to a "global" `Pdu` element (see section 3.3.5) representing information flowing through the COM-Stack. This is done by introducing a `EcucReferenceDef` from the "local" `Pdu` to the "global" `Pdu`.

In figure 3.11 this relationship is shown for the `PduRDestPdu` and the `CanIfTxPduConfig`.

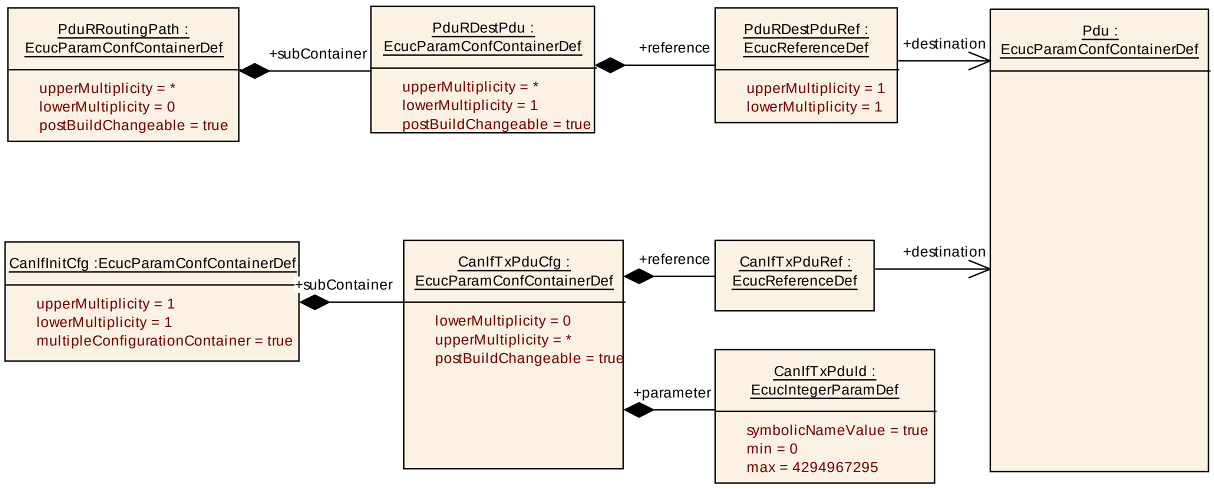
PduRRoutingPath : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
postBuildChangeable = true
+subContainer
PduRDestPdu : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
postBuildChangeable = true
+reference
PduRDestPduRef : EcucReferenceDef
upperMultiplicity = 1
lowerMultiplicity = 1
+destination
Pdu : EcucParamConfContainerDef
CanIfInitCfg :EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 1
multipleConfigurationContainer = true
+subContainer
CanIfTxPduCfg : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
postBuildChangeable = true
+reference
CanIfTxPduRef : EcucReferenceDef
+destination
+parameter
CanIfTxPduId : EcucIntegerParamDef
symbolicNameValue = true
min = 0
max = 4294967295

Figure 3.11: Transmission from PduR to CanIf

There are two reasons why the "global" `Pdu` has been introduced and why all "local" Pdus have to point to the "global" `Pdu` only.

- When doing the configuration of module PduR only the "global" `Pdu` needs to be present, there is no need for the "local" Pdu in the CanIf to be present yet.
- The References are stored in the "local" `Pdu` structure, so changes applied do only influence the structure of the changed module.

Taking the structure shown in figure 3.11 it is now possible to generate both modules.

The CanIf (automatic) configuration editor collects all "local" `CanIfTxPduConfig`s and generates/stores the values for their handle ID in `CanIfCanTxPduId`. If the CanIf needs to know where the Pdu transmit request is coming from it can follow the `PduIdRef` to the "global" `Pdu` and then "query" all references pointing to that `Pdu`. By following those references in reversed direction the transmitting module can be found.

The PduR generator has to know which handle ID to use for each Pdu that has to be sent to the CanIf. To get the actual handle ID value the mechanism is the same in the CanIf use-case: follow the "global" `Pdu` reference and "query" the modules pointing to that "global" `Pdu`. Then find the module(s) type this `Pdu` is going to be transmitted to. In case of a multicast there might be several modules to send the same Pdu to.

With this approach a high degree of decoupling has been achieved between the configuration information of the involved modules. Even when modules are adjacent and need to share information like handle ID, the references between the modules are always indirect using the "global" `Pdu` elements.

3.4.1.4 Handle IDs with symbolic names

The usage of handle Ids together with symbolic names is targeting several use-cases for the methodology of configuring adjacent modules. For the definition of possible configuration approaches please refer to section A.1.1.

For the discussion of the Handle Id use-cases two basic approaches can be distinguished when dividing the methodology into the steps configuration editing and module generation:

- Handle Ids assigned by the configuration editor
- Handle Ids assigned by the module generator

It is assumed that the configuration and generation of the whole stack is done using different tools (possibly from different vendors) which might implement one of the two approaches mentioned above.

In order to support the definition whether a parameter value shall be provided by the user or whether it will be calculated by the editor / generator tooling the attribute `withAuto` has been introduced to the `EcucParameterDef` (see section 2.3.5).

In requirement [TPS_ECUC_02106] it is required that all handle Ids are represented as `symbolicNameValue` = true configuration parameters thus decoupling the value from its usage.

In requirement [TPS_ECUC_02107] it is required that the assigned values are stored in the XML (latest after module generation) so the assigned values are documented. In case the assignment of values has to be performed at a later point in time again (with updated input information) the non affected values can be preserved. It is also needed to support debugging.

In requirement [TPS_ECUC_02108] it is required that the handle Id values are always generated into the module's header file. With this approach it is possible to freely choose the configuration approach of the adjacent modules.

This approach has significant effect on the methodology due to the circular dependencies between the adjacent modules(Com sends to the PduR using PduR handle Ids, PduR indicates to Com using Com handle Ids). Therefore the configuration of all adjacent modules has to be re-visited in case some handle Id changes happen. This contributes to the approach that FIRST the *configuration* of the stack is performed and SECOND the *generation* is triggered.

An example of this approach is provided below: By adding the attribute `symbolicNameValue` = true to the parameter holding the handle ID (in figure 3.11 this is the parameter `CanIfTxPduId`) the code generator doing the `CanIf` will generate a `#define` in the `CanIf_cfg.h` file.

According to [TPS_ECUC_02108] the name of the symbol is composed of the module abbreviation <MA> of the declaring BSW Module followed by the literal "Conf_" followed by the shortName of the EcucParamConfContainerDef of the declaring module followed by the shortName of the EcucContainerValue container which holds the symbolicNameValue configuration parameter value. The value is the actual number assigned to that handle ID.

For example in `CanIf_cfg.h`:

```
#define CanIfConf_CanIfTxPduCfg_Pdu_2345634_985   17
```

The benefit is that the generator of the `PduR` does not need to wait for the `CanIf` to be configured completely and handle IDs are generated. If the `CanIf` publishes the symbolic names for the handle IDs, the `PduR` can expect those symbolic names and generate the `PduR` code using those symbolic names.

For example in `PduR.c`:

```
CanIf_Transmit( CanIfConf_CanIfTxPduCfg_Pdu_2345634_985, PduPtr )
```

Therefore the `PduR` can be generated as soon as its own configuration is finished and there is no need to wait for the `CanIf` to be finished completely. However, at least the "local" `Pdu` in the `CanIf` has to be already created to allow this, because the name of the symbol has to be fetched from this configuration.

Of course the `PduR` can only be compiled after the `CanIf` has been generated as well, but with the utilization of the symbolic names together with handle IDs an even higher degree of decoupling in the configuration process is achieved.

3.4.2 Configuration examples for the Pdu Router

In this section several use-cases of the PduR are described from the configuration point of view. The focus is on the interaction of the PduR configuration with the configuration of the other COM-Stack modules. Therefore only some configuration parameters are actually shown in these examples.

3.4.2.1 Tx from Com to CanIf

In the example in figure 3.12 a Pdu is sent from the Com module – via the Pdu Router – to the Can Interface. Since this one Pdu is handed over through these layers there is only need for one global Pdu object `System_Pdu`.

The Com module's configuration points to the `System_Pdu` to indicate which Pdu shall be sent. The actual Handle Id which has to be used in the API call will however be defined by the PduR in the parameter `PduRSrcPdu::HandleId`. In this example the Com module has to use the Hanlde Id `23` to transmit this Pdu to the PduR.

Then, since the CanIf is pointing to the same `System_Pdu` the PduR can be configured to send this Pdu to the CanIf. The Handle Id is defined in the CanIf configuration in the parameter value of `CanIfCanTxPduId`.

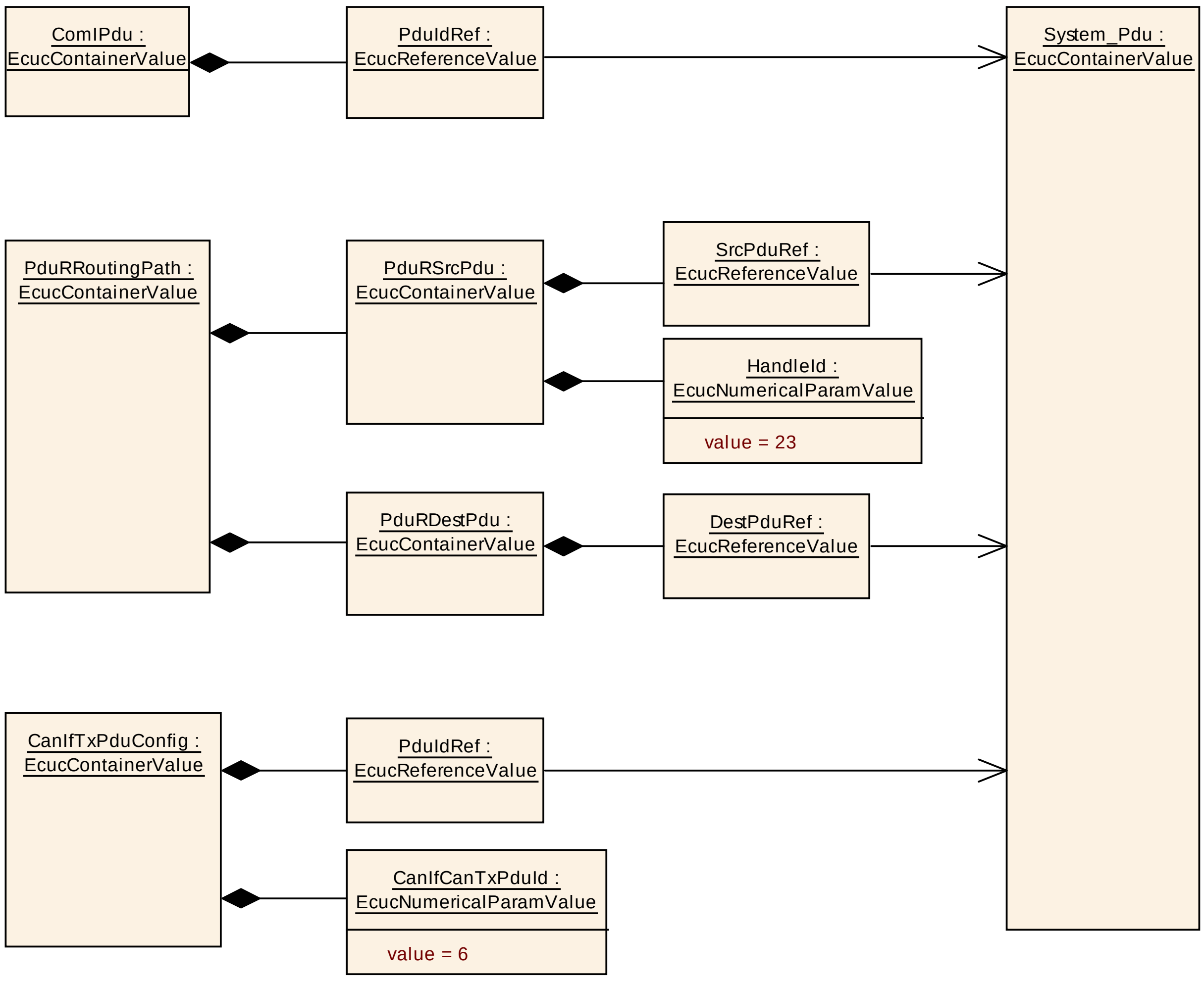
ComIPdu : EcucContainerValue
PduIdRef : EcucReferenceValue
System_Pdu : EcucContainerValue
PduRRoutingPath : EcucContainerValue
PduRSrcPdu : EcucContainerValue
SrcPduRef : EcucReferenceValue
HandleId : EcucNumericalParamValue
value = 23
PduRDestPdu : EcucContainerValue
DestPduRef : EcucReferenceValue
CanIfTxPduConfig : EcucContainerValue
PduIdRef : EcucReferenceValue
CanIfCanTxPduId : EcucNumericalParamValue
value = 6

Figure 3.12: Tx from Com to CanIf example

3.4.2.2 Rx from CanIf to Com

In the example in figure 3.13 the reception use-case from the CanIf to the Com module is configured. Here the Handle Ids are defined in the PduR and the Com module's configuration.

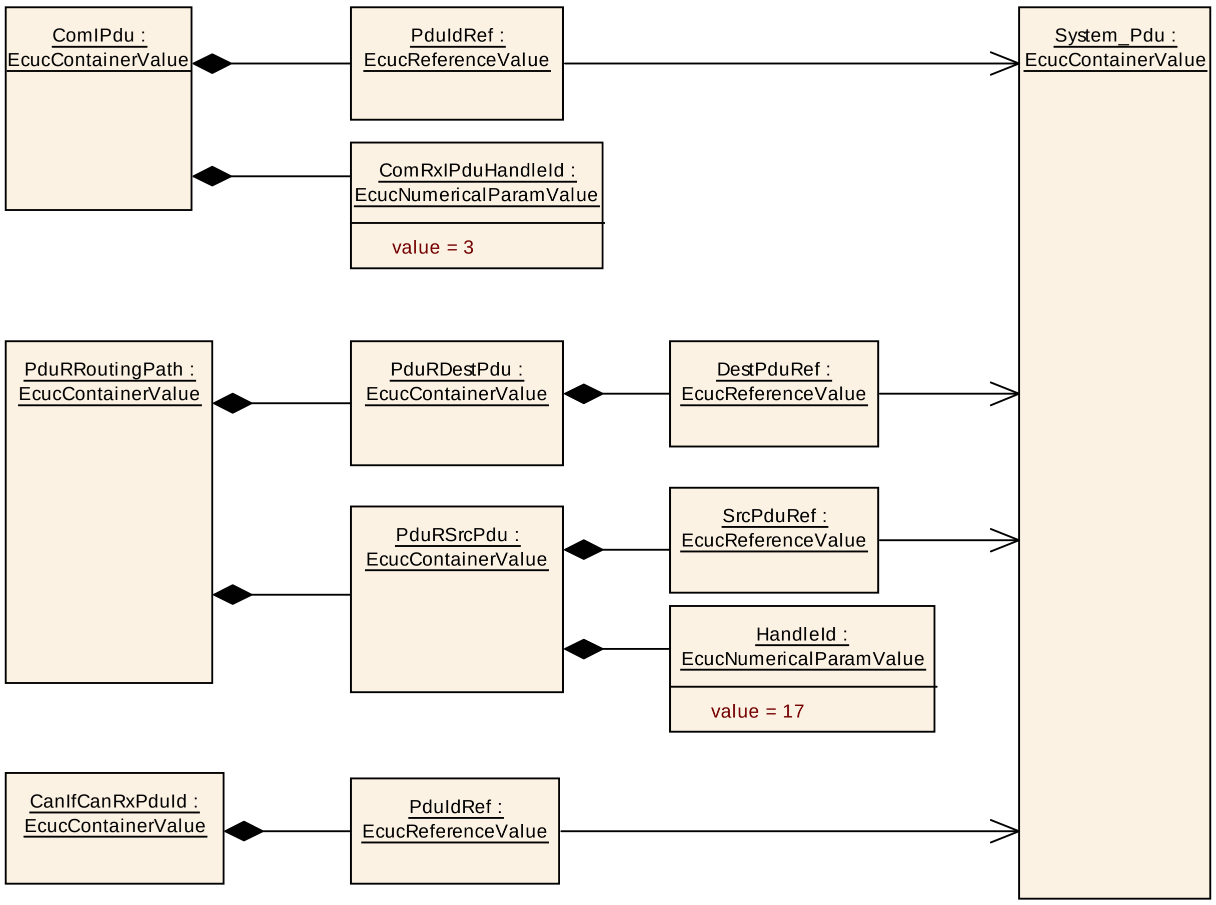
ComIPdu : EcucContainerValue
PduIdRef : EcucReferenceValue
System_Pdu : EcucContainerValue
ComRxIPduHandleId : EcucNumericalParamValue
value = 3
PduRRoutingPath : EcucContainerValue
PduRDestPdu : EcucContainerValue
DestPduRef : EcucReferenceValue
PduRSrcPdu : EcucContainerValue
SrcPduRef : EcucReferenceValue
HandleId : EcucNumericalParamValue
value = 17
CanIfCanRxPduId : EcucContainerValue
PduIdRef : EcucReferenceValue

Figure 3.13: Rx from CanIf to Com example

3.4.2.3 Gateway from CanIf to FrIf

In the example in figure 3.14 the gateway use-case is shown. Since there are two Pdus involved there are two `System_Pdu` objects defined: one which is representing the Can Pdu and one which represents the Fr Pdu. Via the references to these two `System_Pdu` objects the gateway is configured.

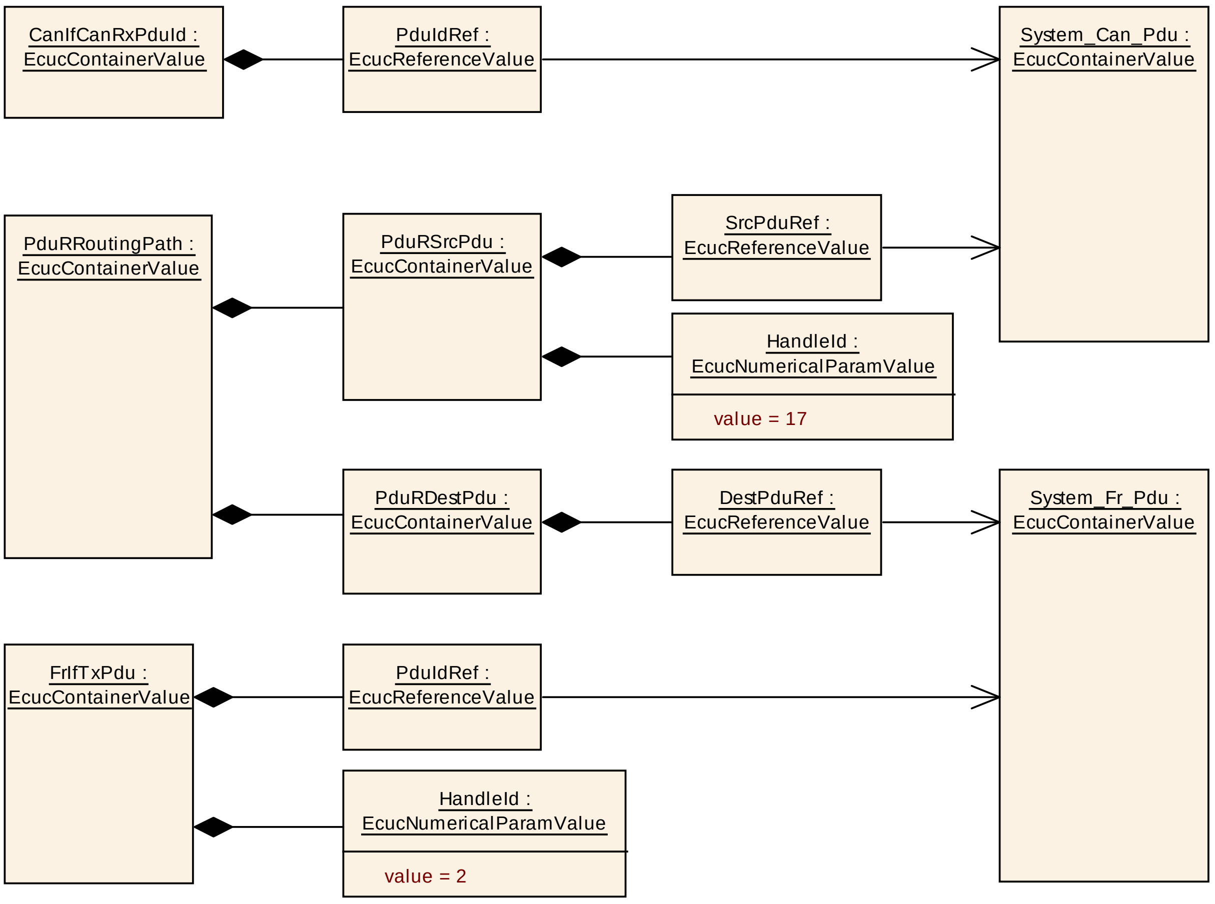
CanIfCanRxPduId : EcucContainerValue
PduIdRef : EcucReferenceValue
System_Can_Pdu : EcucContainerValue
PduRRoutingPath : EcucContainerValue
PduRSrcPdu : EcucContainerValue
SrcPduRef : EcucReferenceValue
HandleId : EcucNumericalParamValue
value = 17
PduRDestPdu : EcucContainerValue
DestPduRef : EcucReferenceValue
System_Fr_Pdu : EcucContainerValue
FrIfTxPdu : EcucContainerValue
PduIdRef : EcucReferenceValue
HandleId : EcucNumericalParamValue
value = 2

Figure 3.14: Gateway from CanIf to FrIf example

3.4.3 Communication Channel IDs

For the configuration of the control path modules (e.g. Communication manager, state managers, network managers) the respective channels are identified using a unique *Communication Channel ID* approach. This is different than the configuration of the *Pdu Handle ID*s of the COM-Stack (see section 3.4.1) where individual *Pdu Handle ID*s are configured per module.

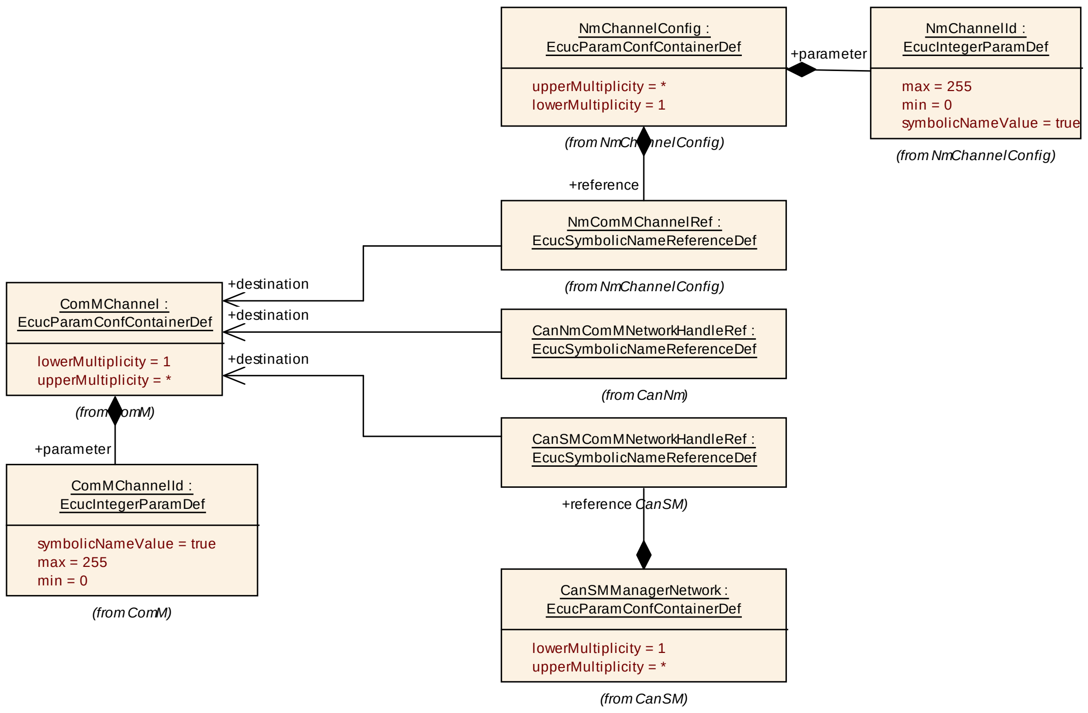

Figure 3.15: Example of Channel interaction between ComM, Can and Nm

In figure 3.15 the `ComMChannel` defines a global communication channel and provides the Communication Channel ID of this channel in the parameter value `ComMChannelId`. Other modules using communication channels (e.g. Nm, CanSM, CanNm, ...) refer to the `ComMChannel` and can utilize the Communication Channel ID in two ways:

- the module does not store the value of the Communication Channel ID itself but always relies on the value provided by the ComM module (like shown for CanNm).
- the module replicates the value of the Communication Channel ID and requires that the replicated id value is equal to the one provided by ComM module (like shown for Nm and CanSM).

Both approaches are currently used in the COM-Stack configuration.

3.5 Cdd module

The CDD module describes the minimal requirements that are necessary for the configuration of a Complex Driver with respect to the surrounding standardized BSW modules.

[TPS_ECUC_06031] Interaction of Complex Driver with standardized AUTOSAR BSW modules ⌈ If a Complex Driver wants to interact with a surrounding standardized BSW module it has to define a Vendor Specific Module Definition from the Standardized CDD Module Definition. The rules that must be followed when generating the Vendor Specific Module Definition are described in chapter 4.1. ⌋

As defined in [TPS_ECUC_06001] the `shortName` of a VSMD module shall be the same as the `shortName` of the StMD. According to this requirement the `shortName` of the module definition of a Complex Driver is always "CDD".

[TPS_ECUC_06036] Distinction of module definitions of Complex Drivers ⌈ To distinguish module definitions of Complex Drivers from each other the package structure shall be used. ⌋

[TPS_ECUC_06037] `apiServicePrefix` attribute for Complex Driver modules ⌈ The `apiServicePrefix` attribute of a Complex Driver shall contain the module abbreviation. ⌋

[constr_3023] Usage of `apiServicePrefix` ⌈The attribute `apiServicePrefix` is mandatory for VSMDs derived from the CDD StMD. The attribute shall not be provided for VSMDs derived from any other StMDs. ⌋

Consider a Complex Driver named "MyCdd". The VSMD of this Complex Driver has to be derived from the CDD StMD. The `shortName` of the module definition of this Complex Driver has to be equal to "CDD". The `apiServicePrefix` attribute is mandatory for the VSMD of this Complex Driver and has to be equal to "MyCdd".

Note that the configuration parameters for the VSMD of CDD do not specify any configuration class. It is up to the implementor of the specific CDD to define the configuration class for all configuration parameters - standardized and vendor specific ones (see [TPS_ECUC_02139]).

If a CDD implementation does utilize the post build configuration it shall specify the `CddConfigSet` container (see [TPS_ECUC_02140]).

Module Name	Cdd	
Module Description	The CDD module describes the minimal requirements that are necessary for the configuration of a CDD with respect to the surrounding standardized BSW modules.	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CddComStackContribution	0..1	Contribution of COM Stack modules.
CddConfigSet	0..1	Configuration set container of the Cdd.
CddEcucPartitionInteraction	0..1	This optional container holds the partition interaction configuration.

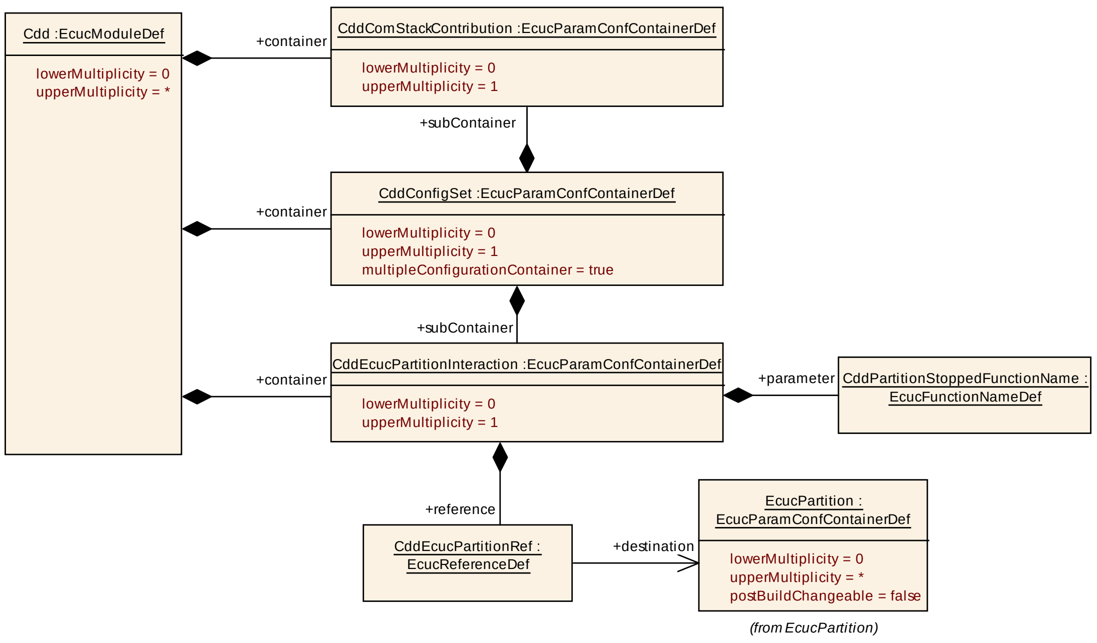
Cdd :EcucModuleDef
lowerMultiplicity = 0
upperMultiplicity = *
+container
CddComStackContribution :EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
+container
CddConfigSet :EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
multipleConfigurationContainer = true
+subContainer
+container
CddEcucPartitionInteraction :EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
CddPartitionStoppedFunctionName : EcucFunctionNameDef
+reference
CddEcucPartitionRef : EcucReferenceDef
+destination
EcucPartition : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
postBuildChangeable = false
(from EcucPartition)

Figure 3.16: Cdd Module

CddConfigSet

SWS Item	[ECUC_Cdd_00067]
Container Name	CddConfigSet[Multi Config Container]
Description	Configuration set container of the Cdd.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddComStack Contribution	0..1	Contribution of COM Stack modules.
CddEcucPartition Interaction	0..1	This optional container holds the partition interaction configuration.

CddEcucPartitionInteraction

SWS Item	[ECUC_Cdd_00038]
Container Name	CddEcucPartitionInteraction
Description	This optional container holds the partition interaction configuration.
Configuration Parameters	

Name	CddEcucPartitionRef [ECUC_Cdd_00039]		
Description	Reference to the "EcucPartition" which executes the software which triggers the CDD.		
Multiplicity	1		
Type	Reference to EcucPartition		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPartitionStoppedFunctionName [ECUC_Cdd_00040]		
Description	Function name to be called when the partition which is triggering the Complex Driver is stopped.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default Value			
Regular Expression			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddComStackContribution

SWS Item	[ECUC_Cdd_00017]
Container Name	CddComStackContribution
Description	Contribution of COM Stack modules.
Configuration Parameters	

Included Containers

Container Name	**Multiplicity**	**Scope / Depedency**
CddComIfUpperLayer Contribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the Com Interface module.
CddComMLowerLayer Contribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Communication Manager module.
CddGenericNmLower LayerContribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Generic NM module.
CddPduRLowerLayer Contribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Pdu Router module.
CddPduRUpperLayer Contribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the Pdu Router module.

CddSoAdUpperLayer Contribution	0..1	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the SoAd module.

The following sections describe particular COM stack modules and the interaction with Complex Drivers.

3.5.1 Pdu Router

In the AUTOSAR COM Stack upper and lower layer Complex Drivers are allowed to access the Pdu Router. In both cases the Pdus that are exchanged between the CDD and the Pdu Router shall be configured. The contribution of the Complex Driver implies a reference to the global Pdu and the definition of a HandleId. Figure 3.17 shows an example of a Complex Driver between the CanIf and the PduR and one Complex Driver above the PduR.

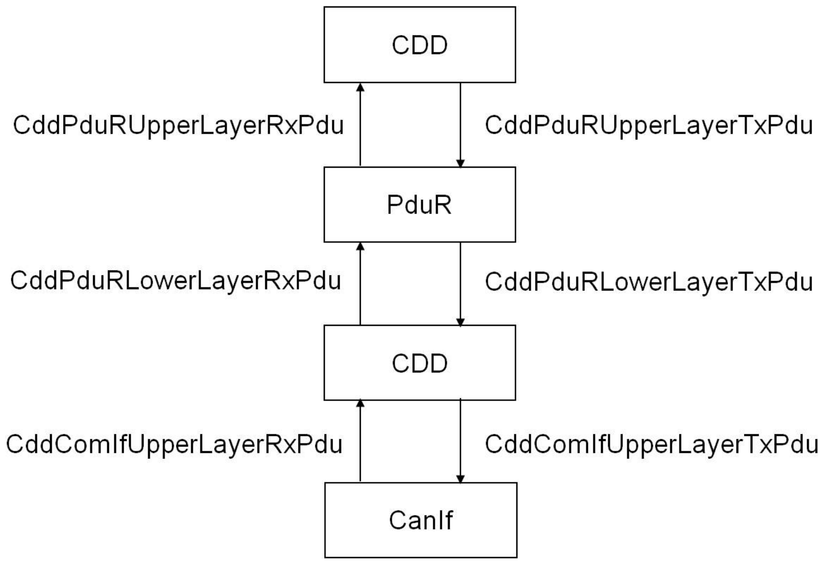

Figure 3.17: CDD Example

Figure 3.18 shows the CDD contribution in the configuration model.

Note that the optional presence of the *TxPdu* and *RxPdu* does not influence the existence of the respective APIs in the *Cdd*.

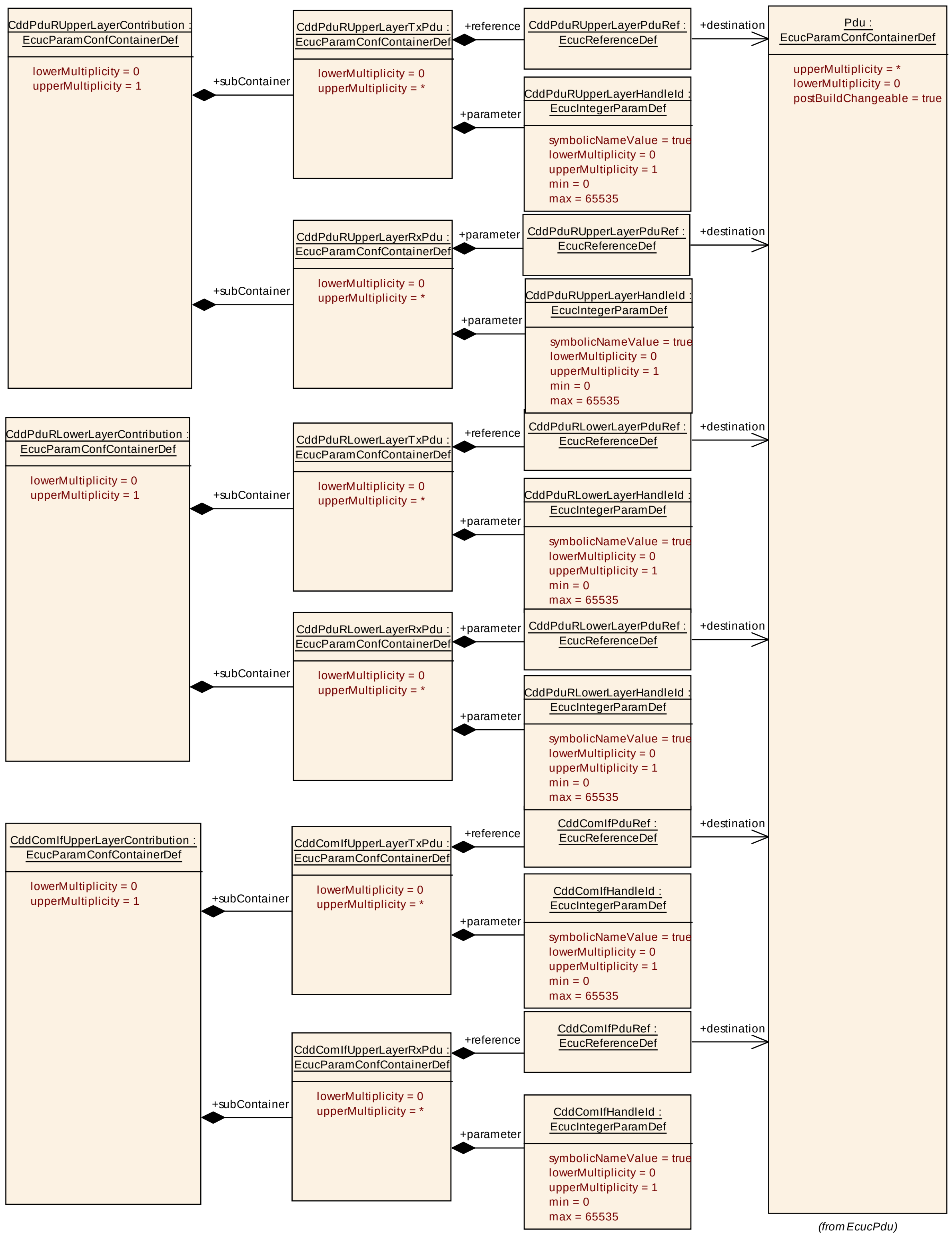

Figure 3.18: PduR and Com Interface contribution

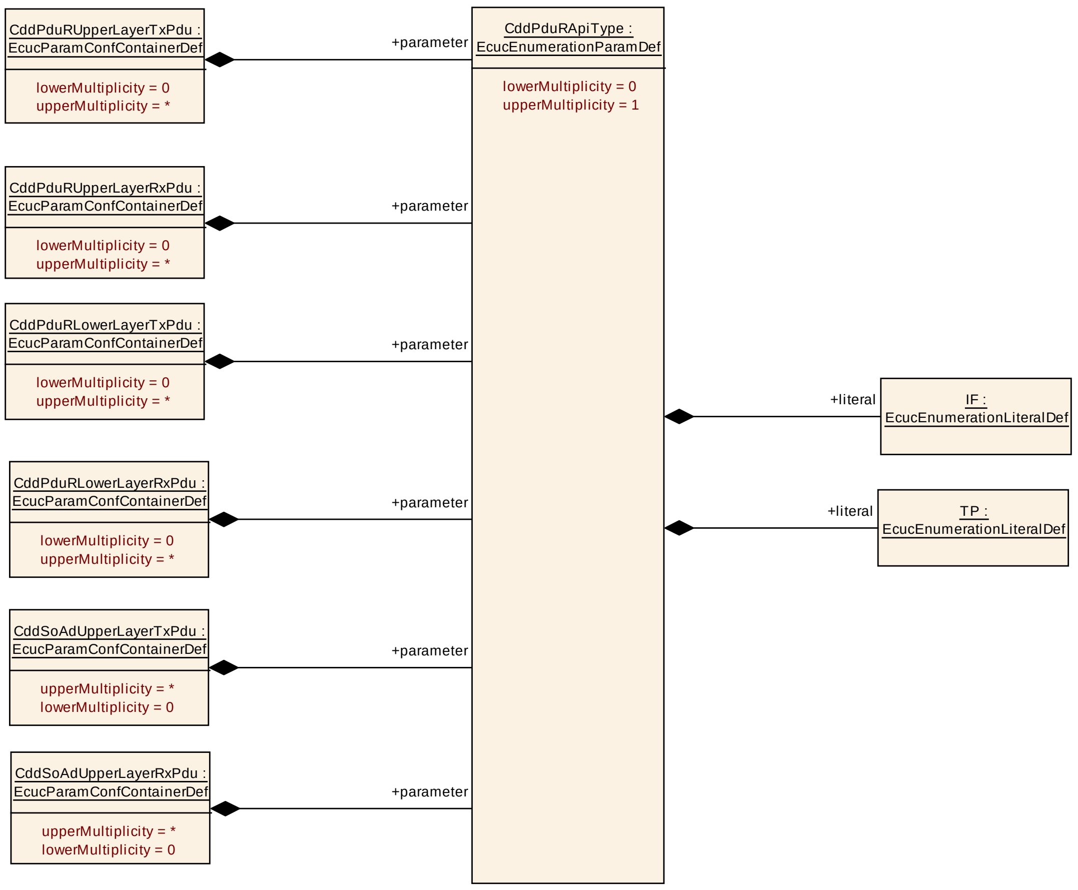
CddPduRUpperLayerTxPdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CddPduRUpperLayerRxPdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CddPduRLowerLayerTxPdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CddPduRLowerLayerRxPdu : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
CddSoAdUpperLayerTxPdu : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
CddSoAdUpperLayerRxPdu : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 0
+parameter
CddPduRApiType : EcucEnumerationParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+literal
IF : EcucEnumerationLiteralDef
+literal
TP : EcucEnumerationLiteralDef

Figure 3.19: Configuration of the CDD interface API type

CddPduRUpperLayerContribution

SWS Item	[ECUC_Cdd_00026]
Container Name	CddPduRUpperLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the Pdu Router module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddPduRUpperLayerRxPdu	0..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddPduRUpperLayerTxPdu	0..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddPduRLowerLayerContribution

SWS Item	[ECUC_Cdd_00022]
Container Name	CddPduRLowerLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Pdu Router module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddPduRLowerLayerRxPdu	0..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddPduRLowerLayerTxPdu	0..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddPduRUpperLayerTxPdu

SWS Item	[ECUC_Cdd_00027]
Container Name	CddPduRUpperLayerTxPdu
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRUpperLayerHandleId [ECUC_Cdd_00029]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRUpperLayerPduRef [ECUC_Cdd_00028]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddPduRUpperLayerRxPdu

SWS Item	[ECUC_Cdd_00043]
Container Name	CddPduRUpperLayerRxPdu
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRUpperLayerHandleId [ECUC_Cdd_00045]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRUpperLayerPduRef [ECUC_Cdd_00044]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddPduRLowerLayerTxPdu

SWS Item	[ECUC_Cdd_00023]
Container Name	CddPduRLowerLayerTxPdu
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRLowerLayerHandleId [ECUC_Cdd_00025]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRLowerLayerPduRef [ECUC_Cdd_00024]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddPduRLowerLayerRxPdu

SWS Item	[ECUC_Cdd_00046]
Container Name	CddPduRLowerLayerRxPdu
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRLowerLayerHandleId [ECUC_Cdd_00048]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddPduRLowerLayerPduRef [ECUC_Cdd_00047]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.5.2 COM Interface modules

A Complex Driver is not allowed to access the COM Stack modules FrDrv, CanDrv and LinDrv. For these modules there is no more than one user. Therefore the lower layer of the COM Stack Bus Interface modules (FrIf, LinIf, CanIf) is not regarded in the CDD module. Upper layer Complex Drivers are allowed to access the interface of these modules. Equal to the `PduRContribution` the `CddComIfUpperLayerContribution` of the Complex Driver implies a reference to the global Pdu and the definition of a HandleId. Figure 3.18 shows the CDD contribution in the configuration model.

Note that the optional presence of the *TxPdu* and *RxPdu* does not influence the existence of the respective APIs in the *Cdd*.

CddComIfUpperLayerContribution

SWS Item	[ECUC_Cdd_00018]
Container Name	CddComIfUpperLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the Com Interface module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddComIfUpperLayerRx Pdu	0..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddComIfUpperLayerTx Pdu	0..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddComIfUpperLayerTxPdu

SWS Item	[ECUC_Cdd_00019]
Container Name	CddComIfUpperLayerTxPdu
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddComIfHandleId [ECUC_Cdd_00021]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddComIfPduRef [ECUC_Cdd_00020]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddComIfUpperLayerRxPdu

SWS Item	[ECUC_Cdd_00049]
Container Name	CddComIfUpperLayerRxPdu
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddComIfHandleId [ECUC_Cdd_00051]	
Description	ECU wide unique, symbolic handle for the Pdu.	
Multiplicity	0..1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default Value		

Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddComIfPduRef [ECUC_Cdd_00050]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.5.3 Communication Manager

Complex Drivers are allowed to access the Communication Manager on the upper layer. The contribution of the lower layer Complex Driver implies for each channel a reference to to the unique handle to identify one certain network handle in the ComM configuration.

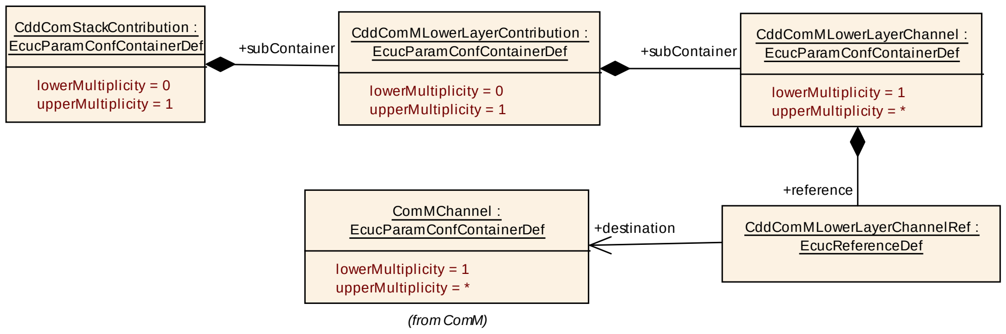

Figure 3.20: ComM lower layer contribution

CddComMLowerLayerContribution

SWS Item	[ECUC_Cdd_00030]
Container Name	CddComMLowerLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Communication Manager module.
Configuration Parameters	

Included Containers

Container Name	Multiplicity	Scope / Depedency
CddComMLowerLayer Channel	1..*	This container contains the network specific parameters.

CddComMLowerLayerChannel

SWS Item	[ECUC_Cdd_00031]
Container Name	CddComMLowerLayerChannel
Description	This container contains the network specific parameters.
Configuration Parameters	

Name	CddComMLowerLayerChannelRef [ECUC_Cdd_00032]		
Description	Unique handle to identify one certain network. Reference to one of the network handles configured for the ComM.		
Multiplicity	1		
Type	Reference to ComMChannel		
Configuration Class	Pre-compile time	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.5.4 Generic Network Management

Complex Drivers are allowed to access the GenericNm module on the upper layer. The contribution of the lower layer Complex Driver implies in each `NmChannel` configuration a reference to the respective NM channel handle.

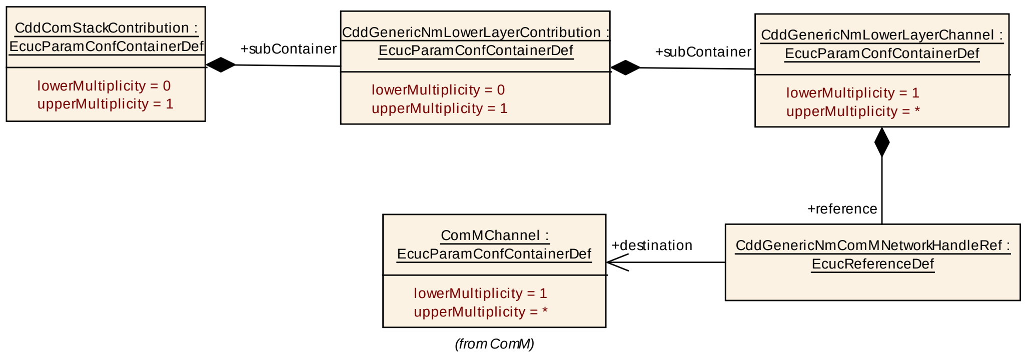
CddComStackContribution : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
CddGenericNmLowerLayerContribution : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
CddGenericNmLowerLayerChannel : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
+reference
CddGenericNmComMNetworkHandleRef : EcucReferenceDef
+destination
ComMChannel : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
(from ComM)

Figure 3.21: GenericNm lower layer contribution

CddGenericNmLowerLayerContribution

SWS Item	[ECUC_Cdd_00033]
Container Name	CddGenericNmLowerLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the LowerLayer of the Generic NM module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddGenericNmLower LayerChannel	1..*	NM Channel specific configuration parameters.

CddGenericNmLowerLayerChannel

SWS Item	[ECUC_Cdd_00034]
Container Name	CddGenericNmLowerLayerChannel
Description	NM Channel specific configuration parameters.
Configuration Parameters	

Name	CddGenericNmComMNetworkHandleRef [ECUC_Cdd_00035]		
Description	This reference points to the unique channel defined by the ComMChannel and provides access to the unique channel index value in ComMChannelId.		
Multiplicity	1		
Type	Reference to ComMChannel		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.5.5 Socket Adaptor

Complex Drivers are allowed to access the SoAd module on the upper layer. The Pdus that are exchanged between the CDD and the SoAd shall be configured. The contribution of the Complex Driver implies a reference to the global Pdu and the definition of a HandleId. Figure 3.22 shows the CDD contribution in the configuration model.

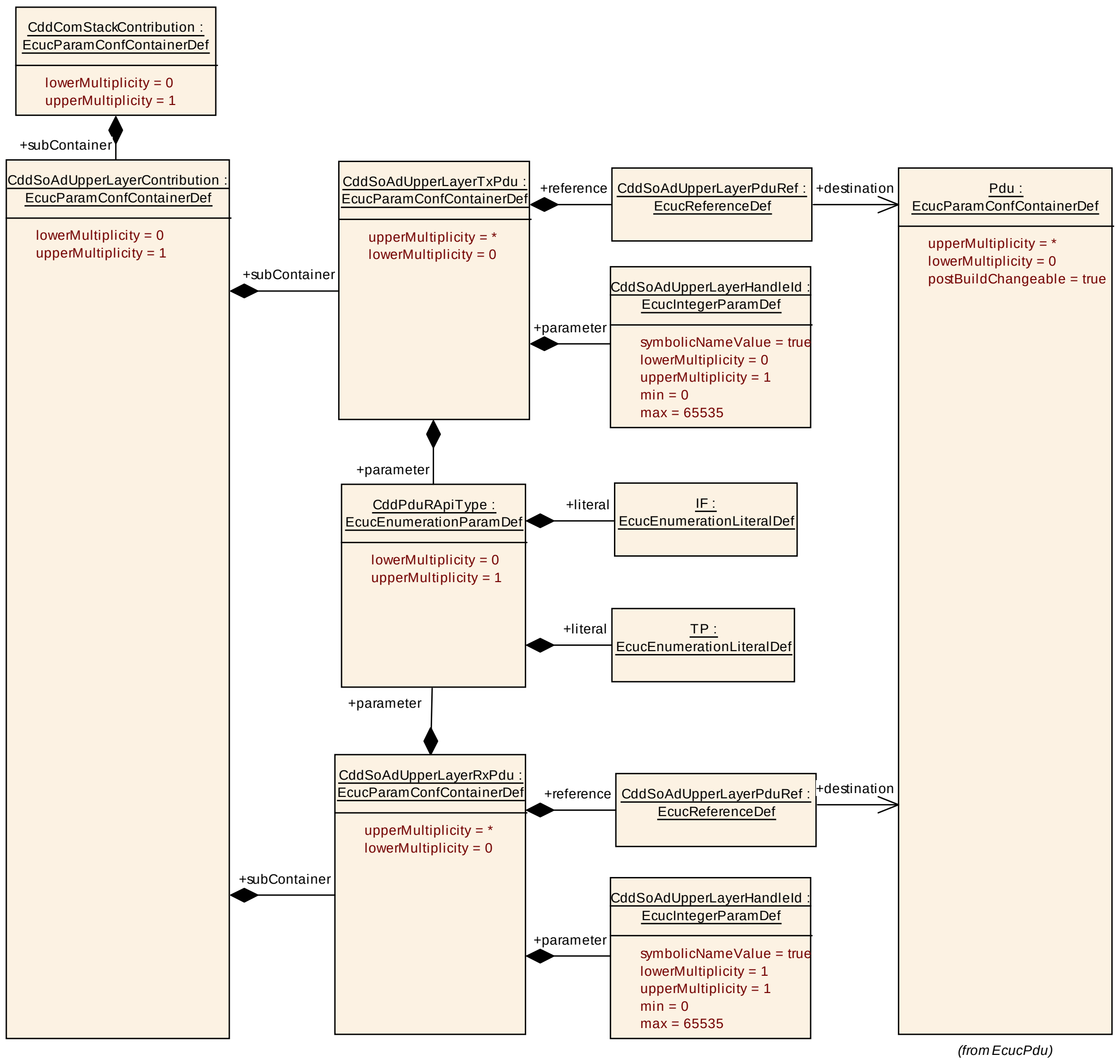

Figure 3.22: SoAd contribution

CddSoAdUpperLayerContribution

SWS Item	[ECUC_Cdd_00060]
Container Name	CddSoAdUpperLayerContribution
Description	Parameters that are necessary for the configuration of a Complex Driver that serves as the UpperLayer of the SoAd module.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddSoAdUpperLayerRxPdu	0..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddSoAdUpperLayerTxPdu	0..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddSoAdUpperLayerTxPdu

SWS Item	[ECUC_Cdd_00061]
Container Name	CddSoAdUpperLayerTxPdu
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddSoAdUpperLayerHandleId [ECUC_Cdd_00064]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddSoAdUpperLayerPduRef [ECUC_Cdd_00063]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

CddSoAdUpperLayerRxPdu

SWS Item	[ECUC_Cdd_00062]		
Container Name	CddSoAdUpperLayerRxPdu		
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.		
Configuration Parameters			

Name	CddPduRApiType [ECUC_Cdd_00052]		
Description	This parameter configures the type of the CDD interface (IF/TP)		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	IF		
	TP		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddSoAdUpperLayerHandleId [ECUC_Cdd_00066]		
Description	ECU wide unique, symbolic handle for the Pdu.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	CddSoAdUpperLayerPduRef [ECUC_Cdd_00065]		
Description	Reference to the "global" Pdu structure to allow harmonization of handle IDs in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

3.6 EcuM configuration to initialize post-build capable BSW Modules

In order to initialize a post-build capable BSW module, either the reference to the init structure of the corresponding BSW module shall be used in case of `multipleConfigurationContainer` approach (see section 2.4.7), or the reference to the `PostBuildVariantCriterion` with the right `PostBuildVariantCriterionValue` shall be used in case of `Variant Handling` approach (see section 2.4.8). An example of how the initialization is done using EcuMFlex is provided below:

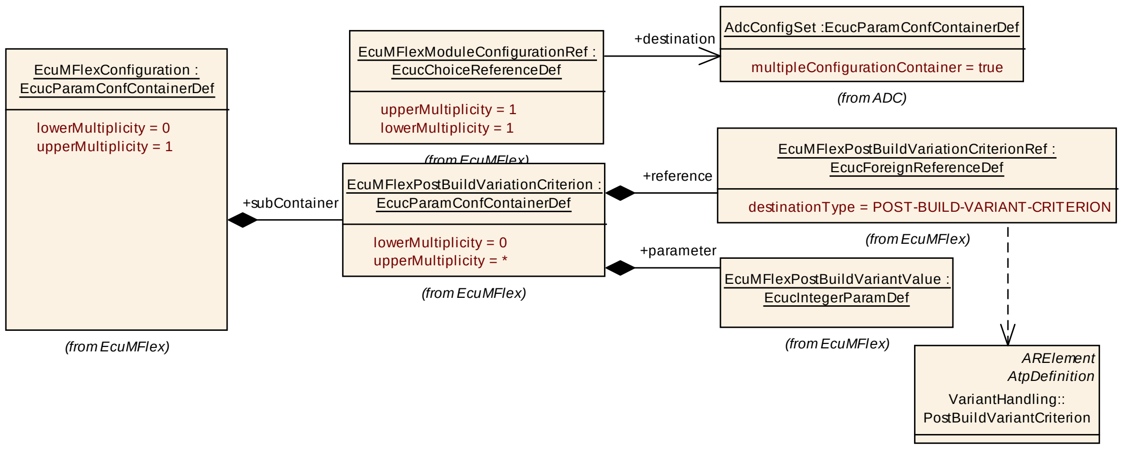
EcuMFlexConfiguration : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
(from EcuMFlex)
+subContainer
EcuMFlexModuleConfigurationRef : EcucChoiceReferenceDef
upperMultiplicity = 1
lowerMultiplicity = 1
(from EcuMFlex)
+destination
AdcConfigSet :EcucParamConfContainerDef
multipleConfigurationContainer = true
(from ADC)
EcuMFlexPostBuildVariationCriterion : EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
(from EcuMFlex)
+reference
EcuMFlexPostBuildVariationCriterionRef : EcucForeignReferenceDef
destinationType = POST-BUILD-VARIANT-CRITERION
(from EcuMFlex)
+parameter
EcuMFlexPostBuildVariantValue : EcucIntegerParamDef
(from EcuMFlex)
ARElement
AtpDefinition
VariantHandling::
PostBuildVariantCriterion

Figure 3.23: Example of EcuM configuration to initialize post-build modules

In case `multipleConfigurationContainer` approach is taken, `EcuMFlexModuleConfigurationRef` references to the init structure of the corresponding BSW module shall be used.

In case `Variant Handling` approach is taken, `EcuMFlexPostBuildVariationCriterion` container shall be used. It contains a reference to the `PostBuildVariantCriterion` which is used as a common selecting element for the post-build variants, and `EcuMFlexPostBuildVariantValue` which contains a value that identifies the selected post-build variant.

Either `EcuMFlexModuleConfigurationRef` or `EcuMFlexPostBuildVariationCriterion` shall be used depending on the approach.

3.7 Optional reporting of Production Errors and Extended Production Errors

The reporting of Production errors from any BSW Module to the Dem is configurable (see figure 2.17 for an example). The respective `EcucSymbolicNameReferenceDef`s from the reporting module to the `DemEventParameter` are optional.

[TPS_ECUC_02143] Optional configuration of Production Error and Extended Production Error reporting ⌈ The configuration of Production Error and Extended

Production Error reporting is optional for the reporting BSW module. Due to further functional requirements in the reporting BSW Module it may still be required to detect the Production Error or Extended Production Error and behave accordingly, even when the reporting to the Dem is not configured. ⌋

Another possibility is to configure and report the Production Error or Extended Production Error to the Dem and then filter inside the Dem configuration the behavior for this `DemEventParameter` such that it will not have an effect.

3.8 Converting time parameters of main functions to ticks

Typically the time related parameters in AUTOSAR are given as float values. Nevertheless for some parameters the unit [ticks] is required. The advantage of having ticks in the ECU configuration is that the final value is already known before the code generator is called. Otherwise it depends on the implementer of the code generator what final value is calculated.

[TPS_ECUC_08010] Ticks in the Ecuc Parameter Value description ⌈ An error shall be generated if the generated number of ticks with the current main cycle does not match the desired timing. ⌋

3.9 Clock Tree Configuration

In the standardized ECU Configuration Parameter Definition only HW independent parameters can be specified. Since the clock tree is highly HW dependent the MCU clock reference point has been introduced which allows an abstract description of clock properties independent of the hardware.

Thus the details of the clock tree configuration must be hardware/vendor specific additions to the MCU Driver Configuration added by the implementor of the MCU Driver. This means, that other drivers (possibly vendor specific), such as CAN Driver, need a mechanism to derive the correct settings for their timing registers, since they do not know the actual hardware specific parameters.

The MCU module defines a container `McuClockReferencePoint` (multiplicity 1..*). In this container a parameter `McuClockReferencePointFrequency` (type float, in Hz) is provided.

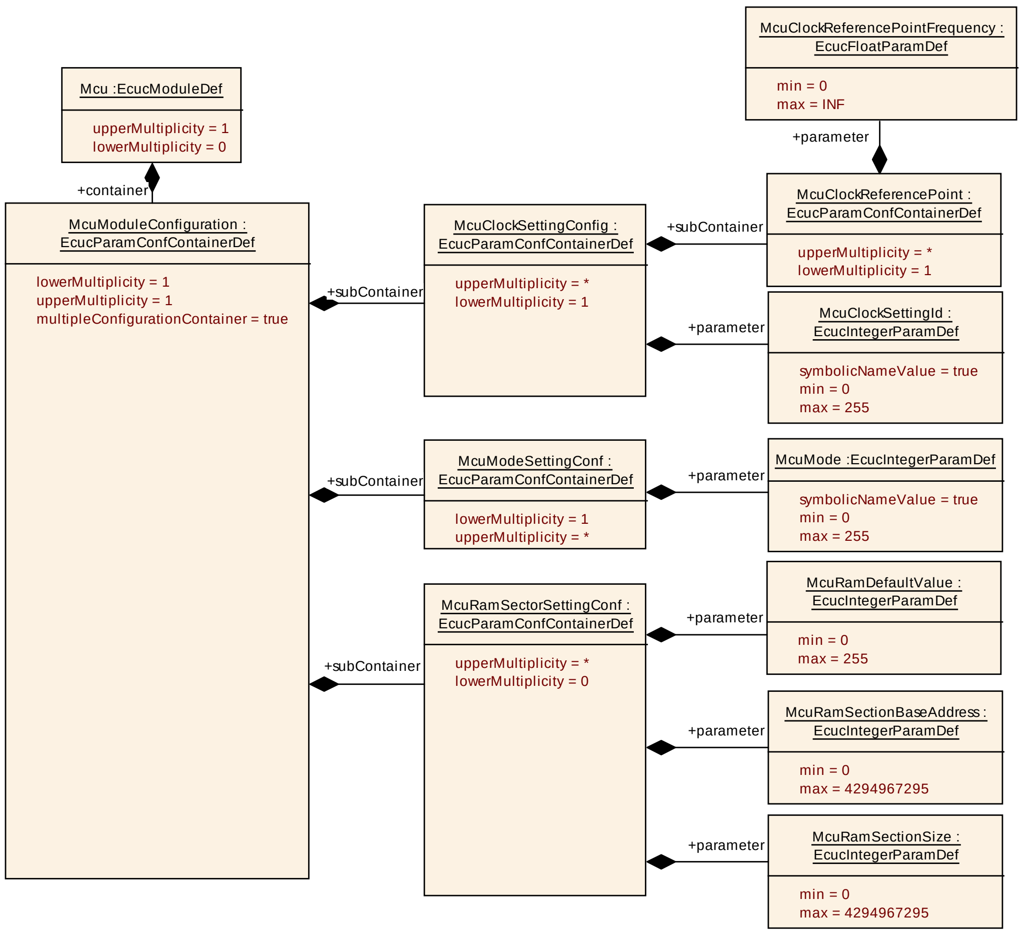

Figure 3.24: MCU Setting

McuClockReferencePoint

SWS Item	[ECUC_Mcu_00174]
Container Name	McuClockReferencePoint
Description	This container defines a reference point in the Mcu Clock tree. It defines the frequency which then can be used by other modules as an input value. Lower multiplicity is 1, as even in the simplest case (only one frequency is used), there is one frequency to be defined.
Configuration Parameters	

Name	McuClockReferencePointFrequency [ECUC_Mcu_00175]		
Description	This is the frequency for the specific instance of the McuClockReferencePoint container. It shall be given in Hz.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	0 .. INF		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

The ECU integrator and/or MCU configuration/generation tool need to derive from those required output frequencies - together with other parameters such as input clock frequency - how its internal settings for prescalers, muxes, etc. need to be configured.

The users of clock frequencies (e.g. CanDrv, LinDrv, PWM) define in their configuration a reference to the container `McuClockReferencePoint` that allows them to select which input clock they choose. In that container the modules generator will find the frequency to use as input frequency (value of parameter `McuClockReferencePointFrequency`). The users of clock frequencies might need to adjust the clock further by setting local prescalers and dividers.

The configuration editor for the peripheral module (i.e. CanDrv configuration editor) can support the integrator by only allowing a selection of those clock reference points that can be connected physically to that peripheral.

The design guideline is that all settings until the MCU clock reference point are under the responsibility of the MCU Driver (see figure 3.25). Further adjustments on the clock frequency are under the responsibility of the specific user peripheral's driver.

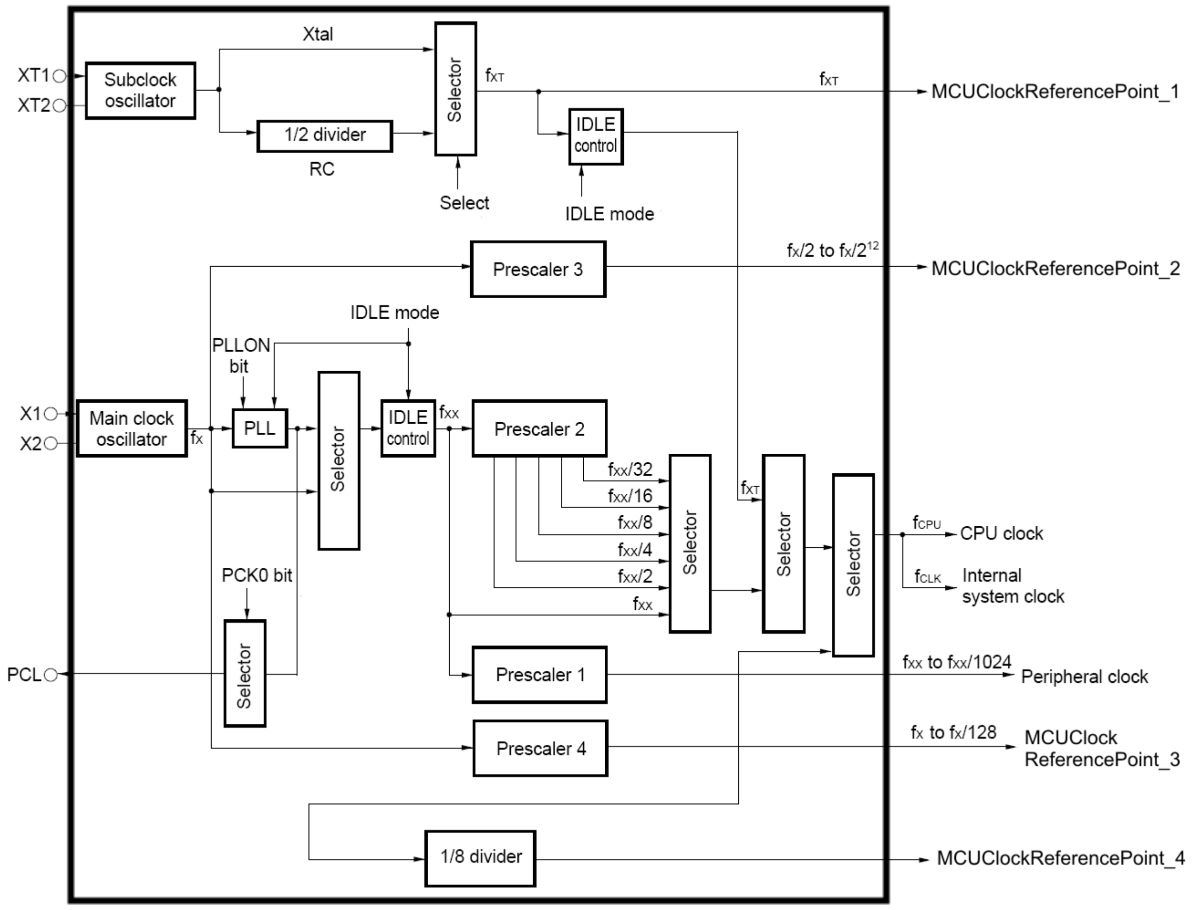

Figure 3.25: Clocktree example

4 Rules to follow in different configuration activities

This chapter defines rules relevant for the relation between standardized module definitions and vendor specific module definitions, rules for building the base `ECU configuration Value description` and rules for configuration editors. The generation of the base `ECU configuration Value description` as a part of the ECU configuration process is explained in the AUTOSAR Methodology ([1], chapter 2.7.3 and chapter 3.6.1.3).

4.1 Deriving vendor specific module definitions from standardized module definitions

The basic relationship between the `Vendor Specific Module Definition` (abbreviated with VSMD in this chapter) and `Standardized Module Definition` (abbreviated StMD in this chapter) is depicted in figure 4.1.

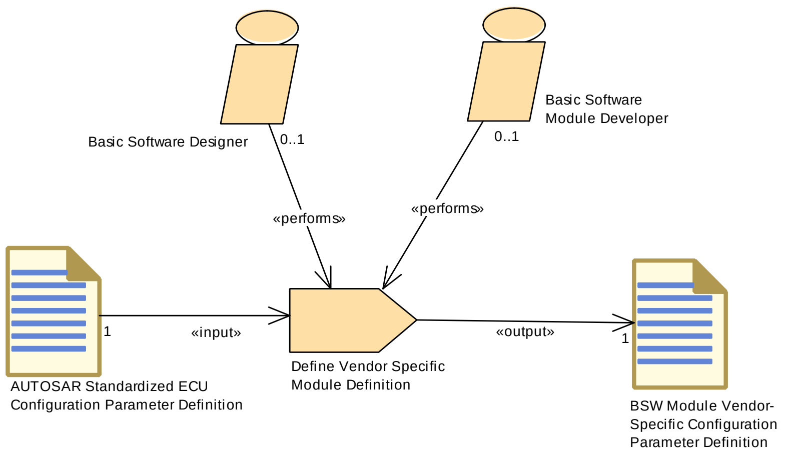
Basic Software Designer
0..1
Basic Software Module Developer
0..1
«performs»
«performs»
1
«input»
«output»
1
AUTOSAR Standardized ECU Configuration Parameter Definition
Define Vendor Specific Module Definition
BSW Module Vendor-Specific Configuration Parameter Definition

Figure 4.1: Generating Vendor Specific Module Definitions (per module)

Please note that also a pure VSMD which has no counterpart in the StMD is allowed to exist. Vendor specific parameters/containers/references with no relationship to StMD may also be available in a VSMD. Figure 4.2 shows an example with pure vendor specific containers and references (marked with red color).

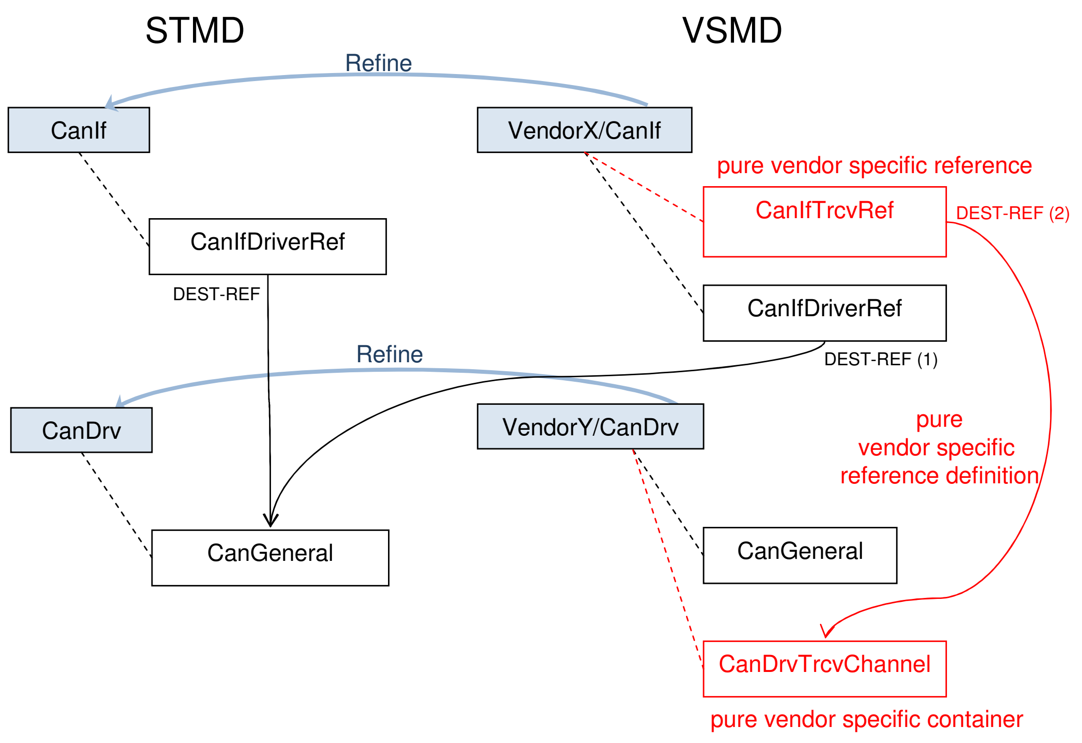

Figure 4.2: Relation between STMD and VSMD

In example 4.1 the StMD of the two modules of figure 4.2 is defined.

Example 4.1

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>EcucDefs</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-DEF>
          <SHORT-NAME>CanIf</SHORT-NAME>
          <CONTAINERS>
            <ECUC-PARAM-CONF-CONTAINER-DEF>
              <SHORT-NAME>CanIfDriver</SHORT-NAME>
              <REFERENCES>
                <ECUC-REFERENCE-DEF>
                  <SHORT-NAME>CanIfDriverRef</SHORT-NAME>
                  <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
                     AUTOSAR/EcucDefs/CanDrv/CanGeneral</DESTINATION-REF>
                </ECUC-REFERENCE-DEF>
              </REFERENCES>
            </ECUC-PARAM-CONF-CONTAINER-DEF>
          </CONTAINERS>
        </ECUC-MODULE-DEF>
        <ECUC-MODULE-DEF>
          <SHORT-NAME>CanDrv</SHORT-NAME>
          <CONTAINERS>
            <ECUC-PARAM-CONF-CONTAINER-DEF>
              <SHORT-NAME>CanGeneral</SHORT-NAME>
            </ECUC-PARAM-CONF-CONTAINER-DEF>
          </CONTAINERS>
        </ECUC-MODULE-DEF>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

In Example 4.2 the VSMD of a `CanDrv` implementation is shown. Here a vendor specific container `CanDrvTrcvContainer` has been introduced.

Example 4.2

```
<AR-PACKAGE>
  <SHORT-NAME>VendorY</SHORT-NAME>
  <ELEMENTS>
    <ECUC-MODULE-DEF>
      <SHORT-NAME>CanDrv</SHORT-NAME>
      <REFINED-MODULE-DEF-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/
         CanDrv</REFINED-MODULE-DEF-REF>
      <CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>CanGeneral</SHORT-NAME>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>CanTrcvChannel</SHORT-NAME>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </CONTAINERS>
    </ECUC-MODULE-DEF>
  </ELEMENTS>
</AR-PACKAGE>
```

In Example 4.3 the VSMD of a `CanIf` implementation is shown. The implicitly refined reference `CanIfDriverRef` still has the DESTINATION-REF in the VSMD pointing to the standardized AUTOSAR short-name path.

Additionally the pure vendor specific reference `CanIfTrcvRef` has been introduced which points to the vendor specific container `CanDrvTrcvContainer` using the DESTINATION-REF with a fully qualified vendor specific short-name path.

Example 4.3

```
<AR-PACKAGE>
  <SHORT-NAME>VendorX</SHORT-NAME>
  <ELEMENTS>
    <ECUC-MODULE-DEF>
      <SHORT-NAME>CanIf</SHORT-NAME>
      <REFINED-MODULE-DEF-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/
         CanIf</REFINED-MODULE-DEF-REF>
      <CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>CanIfDriver</SHORT-NAME>
          <REFERENCES>
```

```
            <ECUC-REFERENCE-DEF>
              <SHORT-NAME>CanIfDriverRef</SHORT-NAME>
              <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
                 AUTOSAR/EcucDefs/CanDrv/CanGeneral</DESTINATION-REF>
            </ECUC-REFERENCE-DEF>
            <ECUC-REFERENCE-DEF>
              <SHORT-NAME>CanIfDrvTrcvRef</SHORT-NAME>
              <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
                 VendorY/CanDrv/CanDrvTrcvChannel</DESTINATION-REF>
            </ECUC-REFERENCE-DEF>
          </REFERENCES>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </CONTAINERS>
    </ECUC-MODULE-DEF>
  </ELEMENTS>
</AR-PACKAGE>
```

[TPS_ECUC_06038] Rules to validate a BSW module implementation ⌈ The following rules shall be checked by tools that validate whether a SW module implementation conforms to its AUTOSAR specification. ⌋*(SRS_BSW_00167)*

- **[TPS_ECUC_01001] `lowerMultiplicity` and `upperMultiplicity` of modules in the VSMD** ⌈ The `lowerMultiplicity` of the module in the VSMD must be equal or bigger to what is defined in the StMD. The `upperMultiplicity` of that module must be equal or less to what is defined in the StMD. StMD lowerMult $\leq$ VSMD lowerMult $\leq$ VSMD upperMult $\leq$ StMD upperMult. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_06001] `shortName` of a VSMD module** ⌈ The `shortName` of a VSMD module shall be the same as the `shortName` of the StMD. ⌋

- **[TPS_ECUC_06049] Restriction of `supportedConfigVariant`s in the VSMD** ⌈ The supported `EcucModuleDef.supportedConfigVariant` shall be restricted in the VSMD to the actually supported configuration variants of this implementation. This can be a subset of the `EcucModuleDef.supportedConfigVariant` in the StMD. ⌋

- **[TPS_ECUC_06050] `supportedConfigVariant`s in the VSMD in case `VariantPostBuild` is supported in the StMD** ⌈ If the supported `EcucModuleDef.supportedConfigVariant` in the StMD is `VariantPostBuild` and it is supported by the implementation the value in the VSMD shall be one or both of

  - `VariantPostBuildLoadable`

  - `VariantPostBuildSelectable`

  replacing `VariantPostBuild`. I.e. `EcucModuleDef.supportedConfigVariant` of a VSMD should not contain `VariantPostBuild`. For compatibility reasons, if it contains `VariantPostBuild`, it is considered as `VariantPostBuildLoadable`. ⌋

- **[TPS_ECUC_06051] ImplementationConfigClass of an `EcucParameterDef` or `EcucAbstractReferenceDef` in VSMD** ⌈ The `implementationConfigClass` of an `EcucParameterDef` or `EcucAbstractReferenceDef` in VSMD shall be the same or higher (where `PreCompile` configuration class is considered to be the lowest and `PostBuild` the highest) as in StMD with respect to the selected subset defined by the actually implemented `supportedConfigVariant`. ⌋

- **[TPS_ECUC_06003] Package structure of the VSMD** ⌈ The package structure of the VSMD has to be different than "/AUTOSAR/EcucDefs/" so that it is possible to distinguish the standardized from the vendor specific module definitions. Example 4.4 shows the difference between the VSMD and StMD. The package structure of the vendor specific CanIf module definition begins with "/VendorX/-CanIf" and the package structure of the vendor specific CanDrv module definition begins with "/VendorY/Can". ⌋

- **[TPS_ECUC_06015] DESTINATION-REF in the VSMD** ⌈ The DESTINATION-REF in the VSMD shall point to the standardized AUTOSAR short-name path (e.g. /AUTOSAR/EcucDefs/Can/CanController) if the reference definition has an STMD counterpart. In this case the vendor specific short-name path (e.g. /VendorX/Can) shall not be used. Example 4.4 shows a DESTINATION-REF from the CanIf module provided from VendorX to the CanDrv module provided by VendorY. The DESTINATION-REF content is not changed from "/AUTOSAR/EcucDefs/..." in the VSMD. ⌋

- **[TPS_ECUC_06046] Vendor specific reference definition with no counterpart in the STMD** ⌈ A pure vendor specific reference definition (which has no counterpart in the STMD) can refer either to a standardized container (has a counterpart in the STMD) or to a vendor specific container. If the reference points to a standardized container the standardized AUTOSAR short-name path shall be used. If the reference points to the vendor specific container the fully qualified vendor specific short-name path shall be used. ⌋

Example 4.4

CanIf and CanDrv AUTOSAR standardized XML:

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>EcucDefs</SHORT-NAME>
        <ELEMENTS>
          <ECUC-MODULE-DEF>
            <SHORT-NAME>CanIf</SHORT-NAME>
            <CONTAINERS>
              <ECUC-PARAM-CONF-CONTAINER-DEF>
                <SHORT-NAME>CanIfDriverConfig</SHORT-NAME>
                <REFERENCES>
                  <!--Reference Definition:CanIfDriverRef-->
                  <ECUC-REFERENCE-DEF>
```

```
                        <SHORT-NAME>CanIfDriverRef</SHORT-NAME>
                        <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
                           AUTOSAR/EcucDefs/Can/CanGeneral</DESTINATION-REF>
                      </ECUC-REFERENCE-DEF>
                    </REFERENCES>
                  </ECUC-PARAM-CONF-CONTAINER-DEF>
                </CONTAINERS>
              </ECUC-MODULE-DEF>
              <ECUC-MODULE-DEF>
                <SHORT-NAME>Can</SHORT-NAME>
                <CONTAINERS>
                  <ECUC-PARAM-CONF-CONTAINER-DEF>
                    <SHORT-NAME>CanGeneral</SHORT-NAME>
                    <PARAMETERS>
                      <!-- ... -->
                    </PARAMETERS>
                  </ECUC-PARAM-CONF-CONTAINER-DEF>
                </CONTAINERS>
              </ECUC-MODULE-DEF>
            </ELEMENTS>
          </AR-PACKAGE>
        </AR-PACKAGES>
</AR-PACKAGE>
```

CanIf VendorX XML:

```
<AR-PACKAGE>
  <SHORT-NAME>VendorX</SHORT-NAME>
  <ELEMENTS>
    <ECUC-MODULE-DEF>
      <SHORT-NAME>CanIf</SHORT-NAME>
      <REFINED-MODULE-DEF-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/
         CanIf</REFINED-MODULE-DEF-REF>
      <CONTAINERS>
        <ECUC-PARAM-CONF-CONTAINER-DEF>
          <SHORT-NAME>CanIfDriverConfig</SHORT-NAME>
          <REFERENCES>
            <!--Reference Definition:CanIfDriverRef-->
            <ECUC-REFERENCE-DEF>
              <SHORT-NAME>CanIfDriverRef</SHORT-NAME>
              <DESTINATION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
                 AUTOSAR/EcucDefs/Can/CanGeneral</DESTINATION-REF>
            </ECUC-REFERENCE-DEF>
          </REFERENCES>
        </ECUC-PARAM-CONF-CONTAINER-DEF>
      </CONTAINERS>
    </ECUC-MODULE-DEF>
  </ELEMENTS>
</AR-PACKAGE>
```

CanDrv VendorY XML:

```
<AR-PACKAGE>
  <SHORT-NAME>VendorY</SHORT-NAME>
  <ELEMENTS>
    <ECUC-MODULE-DEF>
```

```
        <SHORT-NAME>Can</SHORT-NAME>
        <REFINED-MODULE-DEF-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Can<
           /REFINED-MODULE-DEF-REF>
        <CONTAINERS>
          <ECUC-PARAM-CONF-CONTAINER-DEF>
            <SHORT-NAME>CanGeneral</SHORT-NAME>
            <PARAMETERS>
              <!-- ... -->
            </PARAMETERS>
          </ECUC-PARAM-CONF-CONTAINER-DEF>
        </CONTAINERS>
      </ECUC-MODULE-DEF>
    </ELEMENTS>
</AR-PACKAGE>
```

For all EcucContainerDefs and EcucParameterDefs and EcucAbstractReferenceDefs defined within the EcucModuleDef in the StMD, it holds:

- **[TPS_ECUC_06007] Elements defined in the StMD shall be present in the VSMD** ⌈ Elements defined in the StMD shall be present in the VSMD and shall not be omitted, even if the upperMultiplicity of an element in the VSMD is set to 0. ⌋*(RS_ECUC_00002, SRS_BSW_00171, RS_ECUC_00055, RS_ECUC_00070)*

- **[TPS_ECUC_06008] lowerMultiplicity and upperMultiplicity of elements in the VSMD** ⌈ The lowerMultiplicity of an element in the VSMD shall be bigger or equal and the upperMultiplicity shall be equal or less than in the StMD:
⌋ StMD lowerMult $\leq$ VSMD lowerMult $\leq$ VSMD upperMult $\leq$ StMD upperMult.

- **[TPS_ECUC_08005] postBuildChangeable attribute in the VSMD in case it is not defined in the StMD** ⌈ In case supportedConfigVariant of a EcucModuleDef equals VariantPostBuild and the attribute postBuildChangeable of an EcucContainerDef is not defined in the StMD, it shall be defined in the VSMD for all EcucContainerDefs that have upperMultiplicity greater than lowerMultiplicity and upperMultiplicity is greater than 1 (the attribute can have either true or false value). This includes vendor specific EcucContainerDefs.

  In case the value of the attribute postBuildChangeable equals FALSE it is not mandatory to set it in the VSMD. ⌋

- **[TPS_ECUC_08006] postBuildChangeable attribute in the VSMD in case it is set to false in the StMD** ⌈ In case supportedConfigVariant of a EcucModuleDef equals VariantPostBuild and an EcucContainerDef in the StMD has the attribute postBuildChangeable set to false, the corresponding VSMD may change it to true if its upperMultiplicity is greater than lowerMultiplicity and upperMultiplicity is greater than 1. ⌋

- **[TPS_ECUC_08007] `postBuildChangeable` attribute in the VSMD in case it is set to true in the StMD** ⌈ In case `supportedConfigVariant` of an `EcucModuleDef` equals `VariantPostBuild` and an `EcucContainerDef` in the StMD has the attribute `postBuildChangeable` set to `true`, it shall be set to `true` in the corresponding VSMD as well. ⌋

- **[TPS_ECUC_01034] ShortName of elements in the VSMD that are taken over from the StMD** ⌈ Elements taken over from the StMD to the VSMD shall use exactly the same `shortName`, since the short name identifies the element. This holds for container definitions and individual parameters. ⌋

- **[TPS_ECUC_01035] UUID of elements in the VSMD that are taken over from the StMD** ⌈ Elements taken over from the StMD to the VSMD must have unique `uuid` in each Value description. Thus a new `uuid` might be generated when taking over an element. ⌋

- **[TPS_ECUC_01005] Origin attribute of parameters in the VSMD that are taken over from the StMD** ⌈ The `origin` attribute must not be changed for any parameter taken over from the StMD, even when attributes of the parameter are modified in the VSMD. ⌋

- **[TPS_ECUC_01006] `DefaultValues` of parameters in the VSMD** ⌈ The `defaultValue` attribute may be changed (or added, if missing). ⌋

- **[TPS_ECUC_01007] `min`, `max` values of parameters in the VSMD** ⌈ The `min` values specified in the VSMD must be bigger or equal, the `max` value must be less or equal than the corresponding value specified in the StMD:
⌋ StMD minValue $\leq$ VSMD minValue $\leq$ VSMD maxValue $\leq$ StMD maxValue.

- **[TPS_ECUC_06045] `min`, `max` values of parameters in the VSMD in case that the min or max value in the StMD is set to infinite** ⌈ If the min value equals *-inf* or the max value equals *inf* in the StMD the min/max values in the VSMD shall be replaced with the actually supported min/max values for this implementation.
⌋

- **[TPS_ECUC_01009] Calculation of derived parameters in the StMD may change in the VSMD** ⌈ For derived parameters defined in the StMD, the values of the `calculationFormula` and `calculationLanguage` may change in the VSMD. ⌋

- **[TPS_ECUC_01011] Vendor specific `choice`s in `EcucChoiceContainerDef`s** ⌈ Additional vendor specific `choice`s (i.e. aggregated `EcucParamConfContainerDef`s) may be added to `EcucChoiceContainerDef`s in the VSMD. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_01013] Vendor specific `destination`s in `EcucChoiceReferenceDef`s** ⌈ Additional vendor specific references may be added for `EcucChoiceReferenceDef`s in the VSMD. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_01014] Addition of vendor specific parameter definitions, container definitions and references** ⌈ Additional vendor specific parameter definitions, container definitions and references shall be added to the VSMD according to the alphabetical order. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_01015] Origin attribute in vendor specific elements** ⌈ The `origin` attribute of vendor specific additional elements shall contain the name of the vendor that defines the element. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_02084] Addition of vendor specific `EcucEnumerationLiteralDef`s to an `EcucEnumerationParamDef` from the StMD** ⌈ For an `EcucEnumerationParamDef` from the StMD there can be additional `EcucEnumerationLiteralDef`s added in the VSMD if the `scope` of the `EcucEnumerationParamDef` is `local`. ⌋

- **[TPS_ECUC_05002] Creation of VSMD from the StMD** ⌈ Induce VSMD into the StMD in a simplified manner, so that the configuration can be carried out without any disarray. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_05003] `desc` field of parameters in VSMD** ⌈ The `desc` in VSMD can be used to specify detailed information about the respective parameter. ⌋*(RS_ECUC_00002)*

- **[TPS_ECUC_02134] `requiresIndex` setting in the VSMD** ⌈ The `requiresIndex` setting may be changed in the VSMD. ⌋

- **[TPS_ECUC_02137] `EcucValidationCondition`s from the StMD shall be taken over to the VSMD.** ⌈ If the StMD defines any `ecucValidationCond`s they shall be taken to the VSMD. ⌋

- **[TPS_ECUC_02138] Addition of vendor specific `EcucValidationConditions`** ⌈ Additional `ecucValidationCond`s may be added to the VSMD. Semantically they shall provide more restrictive validation conditions than the ones defined in the StMD. ⌋

Figure 4.3 shows an overview about rules, which shall be checked by tools that validate whether a SW module implementation conforms to its AUTOSAR specification. In this example three parameters are defined in the StMD.

The multiplicity in each of these parameter definitions specifies how often a parameter is allowed to occur in the ECU Configuration Value description. In the VSMD optional elements (with `lowerMultiplicity` = 0) must be present, but the `lowerMultiplicity` and `upperMultiplicity` may be set to 0, as it happens with parameter A (1). The `lowerMultiplicity` of parameters in the VSMD must be bigger or equal than in the StMD (1, 2, 3). The `upperMultiplicity` of parameters in the VSMD must be equal (2) or less (1, 3) than in the StMD. New vendor specific parameters may also be added in the VSMD (4).

The VSMD defines which parameters are available in which container and what kind of restrictions are to be respected.

[TPS_ECUC_06009] Existence of a parameter in the Ecuc Parameter Value description in case the upperMultiplicity of the parameter definition is zero ⌈ If the upperMultiplicity of a parameter definition in the VSMD is 0, the parameter may be omitted in the parameter Value description. If such a parameter exists in the parameter Value description it shall be ignored by the tool (5). ⌋*(RS_ECUC_00082)*

[TPS_ECUC_06010] Existence of a parameter in the Ecuc Parameter Value description in case the lowerMultiplicity of the parameter definition is bigger than zero ⌈ If the lowerMultiplicity of a parameter definition is bigger than 0, the parameter must exist in the parameter Value description (6). ⌋*(RS_ECUC_00082)*

[TPS_ECUC_06011] Missing parameters in the Ecuc Parameter Value description ⌈ Missing parameters shall be detected by tools (8). ⌋

[TPS_ECUC_06012] Parameters without parameter definitions in the Ecuc Parameter Value description ⌈ Parameters without parameter definitions shall be ignored by tools (9). ⌋

[TPS_ECUC_06013] Number of parameters in the Ecuc Parameter Value description ⌈ The number of parameters in the ECUC Value description shall not exceed the upperMultiplicity of the parameter definition in the VSMD (7). ⌋*(RS_ECUC_00082)*

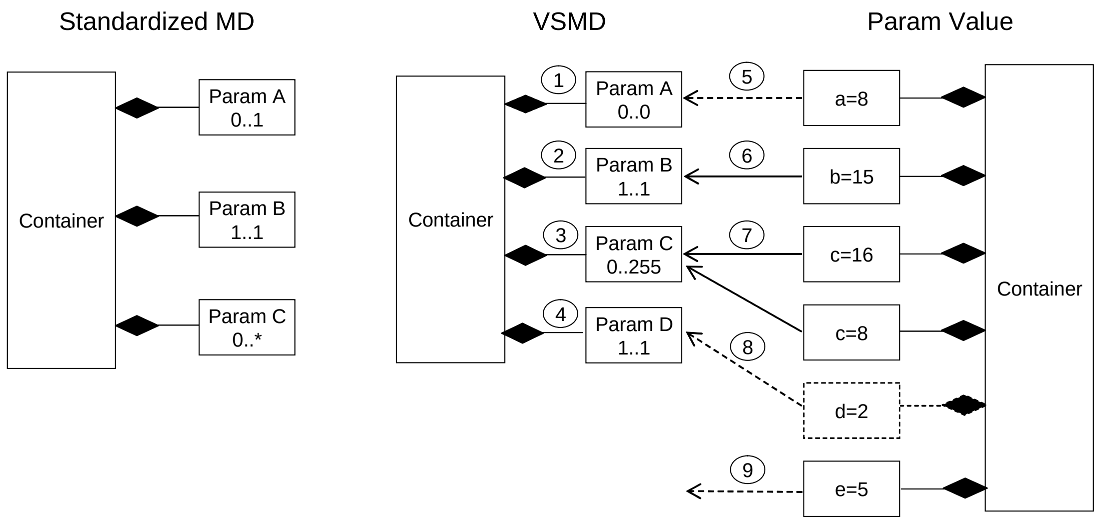

Figure 4.3: Relation between standardized module definition and vendor specific module definition

Example 4.5 depicts the usage of VSMD in case of parameter definition.

Example 4.5

```
<ECUC-INTEGER-PARAM-DEF>
  <SHORT-NAME>ClockRate</SHORT-NAME>
  <ORIGIN>AUTOSAR_ECUC</ORIGIN>
</ECUC-INTEGER-PARAM-DEF>
<ECUC-BOOLEAN-PARAM-DEF>
  <SHORT-NAME>VendorExtensionEnabled</SHORT-NAME>
  <ORIGIN>VendorXYZ_v1.3</ORIGIN>
</ECUC-BOOLEAN-PARAM-DEF>
```

Example 4.6 depicts the usage of VSMD in case of parameter description.

Example 4.6

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-INTEGER-PARAM-DEF">/VendorXYZ/Mcu/McuGeneral/
      ClockRate</DEFINITION-REF>
  <VALUE>123</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
<ECUC-NUMERICAL-PARAM-VALUE>
  <DEFINITION-REF DEST="ECUC-BOOLEAN-PARAM-DEF">/VendorXYZ/Mcu/McuGeneral/
      VendorExtensionEnabled</DEFINITION-REF>
  <VALUE>1</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>
```

In case of a CDD module the configuration parameters for the VSMD of CDD do not specify any configuration class. It is up to the implementor of the specific CDD to define the configuration class for all configuration parameters - standardized and vendor specific ones.

[TPS_ECUC_02139] Definition of configuration classes for all CDD configuration parameters and references ⌈ For the CDD module the standardized configuration parameters do not specify any configuration class. It is up to the implementor of the specific CDD module to define the configuration class for all configuration parameters - standardized and vendor specific ones in the VSMD (therefore [TPS_ECUC_06051] does not apply). ⌋

[TPS_ECUC_02144] Definition of supported config variants for CDD ⌈ For the CDD module the standardized configuration does not specify any supported configuration variants. It is up to the implementor of the specific CDD module to define the configuration variant in the VSMD (therefore [TPS_ECUC_06049] does not apply). ⌋

[TPS_ECUC_02140] Mandatory configuration of `CddConfigSet` for post build configured CDD ⌈ For the CDD module the standardized configuration parameters do specify the container `CddConfigSet` as optional. In case of a post build implementation of the CDD and the usage of the `multipleConfigurationContainer` approach (see section 2.4.7) the container `CddConfigSet` shall become mandatory in the VSMD. ⌋

4.2 Rules for building the Base ECU configuration

The AUTOSAR Methodology ([1], chapter 2.7.3 and chapter 3.6.1.3) defines the activity how to generate the base ECU configuration Value description. The following rules apply during generation of the base ECU configuration for a module:

- **[TPS_ECUC_01016] Generation of instances for mandatory definitions** ⌈ For mandatory containers, parameters and references (i.e. with `lowerMultiplic-`

ity > 0 in their definition) at least the number of instances defined by the lowerMultiplicity shall be generated. ⌋

E.g. the configuration of a CAN controller may contain the configuration of one or more hardware objects, depending on the hardware. The configuration of hardware objects is done in a subcontainer. Since at least one hardware object is always present, one instance of this subcontainer always has to be present and must be generated together with the enclosing container for the CAN controller.

- **[TPS_ECUC_01017] Generation of instances for optional definitions** ⌈ For optional containers, parameters and references (i.e. with lowerMultiplicity $= 0$ in their definition), no instances may be generated. ⌋

  E.g. the configuration may contain the definition of RX PDUs in a subcontainer. One subcontainer instance is defined for each PDU received. Since there may be no RX PDUs, it is well possible that no container instance needs to be generated.

- **[TPS_ECUC_01018] Generation of instances for container definitions with variable multiplicity** ⌈ For containers with variable multiplicity (i.e. lowerMultiplicity $<$ lupperMultiplicity), any number of instances between lower and upper multiplicity may be generated. (additional instances may be added during editing of the configuration Value description). ⌋

  E.g., continuing the previous example, several instances may be generated if the definition of RX PDUs can be derived from the ECU extract of System description. If the ECU receives several frames on a CAN bus, at least one RX PDU is normally present per received frame.

- **[TPS_ECUC_01019] Setting of the initial values for configuration parameters** ⌈ For the setting of the initial values for configuration parameters, the following sources shall be used (in decreasing priority) ⌋

  – **[TPS_ECUC_01020] Values fixed by the implementation as defined in the Vendor Specific Pre-configured Configuration Value description** ⌈ Since the module implementation fixes those configuration parameters, those values shall be included in the base ECU configuration Value description and shall not be changed in later editing. ⌋

  – **[TPS_ECUC_01021] Values derived from the ECU extract of the system configuration.** ⌈ The ECU extract may define the basis for the Ecu configuration Value description, e.g. for COM stack configuration, the system description provides configuration information for bus speed, frame definitions etc, which can be taken over into the ECU configuration Value description. ⌋
  E.g. The signal definitions relevant for the COM stack can be derived from the ECU extract of system configuration. One container instance with all relevant parameter values taken from the system configuration will be generated for each signal.

- **[TPS_ECUC_01022] Values provided by the implementor in the BSWMD in the Vendor Specific Recommended Configuration Value description.** ⌈ Implementors may provide configuration settings in the BSWMD provided with their implementation. This allows the implementor to provide the integrator with his hints which values might be most useful for his implementation of the module on a specific ECU. ⌋
- **[TPS_ECUC_01023] Default values provided as part of the parameter definition.** ⌈ Since each configuration parameter is defined only once, all instances of the parameter will have the same initial value when the default values is taken as input to the base configuration. ⌋

[TPS_ECUC_01024] Generation of parameters without an initial value ⌈ If no initial value can be derived from any of these sources for a parameter, the parameter will be generated without an initial value. ⌋

[TPS_ECUC_04004] Iterative development of the `ECU Configuration Value description` ⌈ If an existing `ECU Configuration Value description` exist and an updated `ECU Extract of System Configuration` or `BSW Module Description` is released the existing `ECU Configuration Value Description` must be taken into consideration when updating to a new version of `ECU Configuration Value description`, i.e, the `Generate Base ECU Configuration Value description` activity shall consist of a merge functionality. This functionality is optional since the first time an `ECU Configuration Value description` is generated there is no existing `ECU Configuration Value description`. ⌋

4.3 Rules for Configuration Editors

The Autosar Methodology ([1] chapter 2.7.4) describes the process for editing configuration parameters. The following rules apply for a configuration editor supporting the methodology:

- **[TPS_ECUC_04002] ECU Configuration Editor shall be able to merge `ECU Configuration Value descriptions`** ⌈ The ECU Configuration Editor shall be able to perform a simple merge of `ECU Configuration Value descriptions`. ⌋
- **[TPS_ECUC_04003] ECU Configuration Editor shall be able to work with subsets of parameters** ⌈ The ECU Configuration Editor shall be able to work with subsets of parameters. The subset shall be any combination of pre-compile time, link-time and post-build time parameters. This feature is to avoid editing wrong kind of parameters. ⌋
- **[TPS_ECUC_04005] ECU Configuration Editor shall be able to generate and import `EcucModuleConfigurationValues`** ⌈ The ECU Configuration Editor shall be able to generate and import files describing a specific aspect of the con-

figuration of a module. The files that shall be generated and imported are EcucModuleConfigurationValues. The rationale for this is to support post-build time loadable configuration from a Configuration Management perspective. See Autosar Methodology [1] chapter 2.7.8.3. ⌋*(SRS_BSW_00399)*

- **[TPS_ECUC_06071] ECU Configuration Editor shall be able to read parameter values in any order** ⌈ The ECU Configuration Editor shall be able to read parameter values in any ordering according to the input. ⌋
- **[TPS_ECUC_06073] The ECU Configuration Editor shall be able to work with arbitrary package structures** ⌈ The ECU Configuration Editor shall be able to work with Ecu Configuration value descriptions in arbitrary package structure. This structure does not need to correlate in any way with the Ecu configuration definition package structure. ⌋

Following is a list (not complete) of additional requirements which a Configuration Editor shall support:

[TPS_ECUC_02088] Configuration Editor shall display the content of the longName to users ⌈ When a longName (LONG-NAME in XML) is provided for a configuration element the Configuration Editor shall display the content of the longName to it's users. ⌋

For referencing the following requirements apply:

[TPS_ECUC_06047] References in the ECUC Parameter Value description with reference definitions that refer to container definitions in the same module definition ⌈ For reference definitions that refer to container definitions in the same module definition the references on the value side shall only refer to container instances of this module instance. ⌋

The example in figure 4.4 defines a reference inside the CanDrv module. Thus the values can only refer to container instances within the respective CanDrv configuration instance.

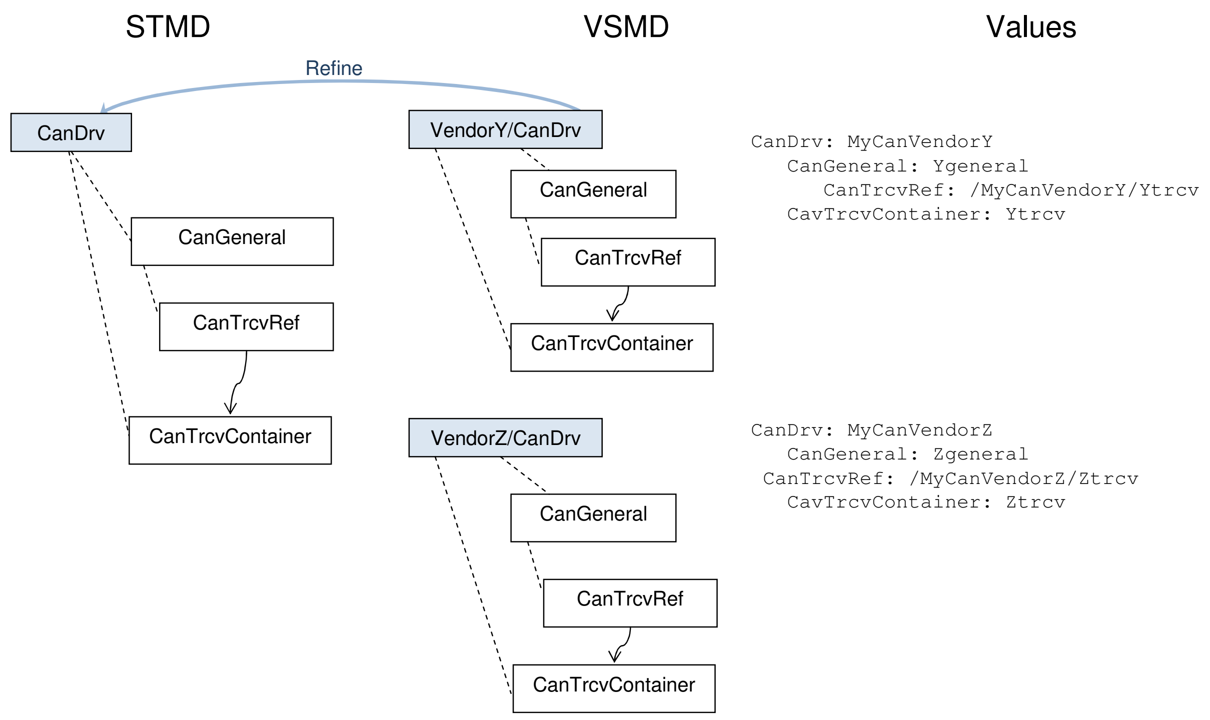

Figure 4.4: Reference inside a module

[TPS_ECUC_06048] References in the ECUC Parameter Value description with reference definitions that refer to container definitions in different module definitions ⌈ For reference definitions that refer to container definitions in a different module definition the references on the value side may refer to container instances of different module instances according to the same module definition. ⌋

The example in figure 4.5 defines a reference between the CanIf and the CanDrv module. Thus the values can refer to container instances of different CanDrv configuration instances.

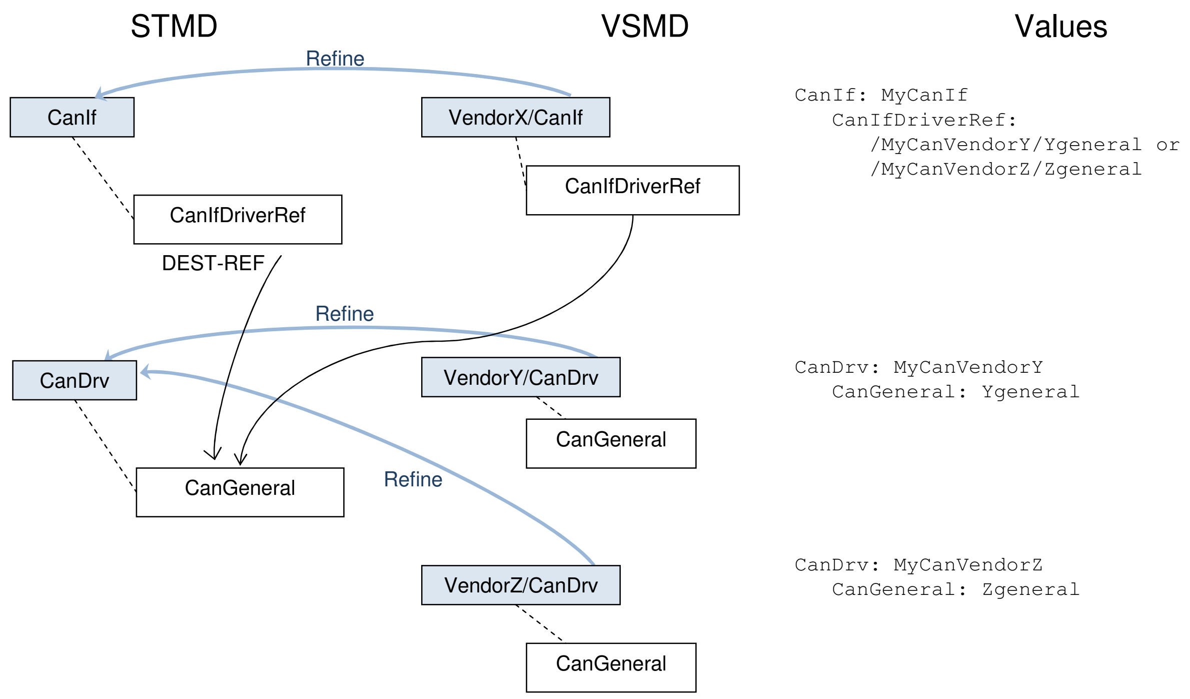

Figure 4.5: Reference between modules

4.4 Rules for navigating in Ecu Configuration Artifacts

The following rules apply for tools that are navigating in Ecu Configuration Artifacts:

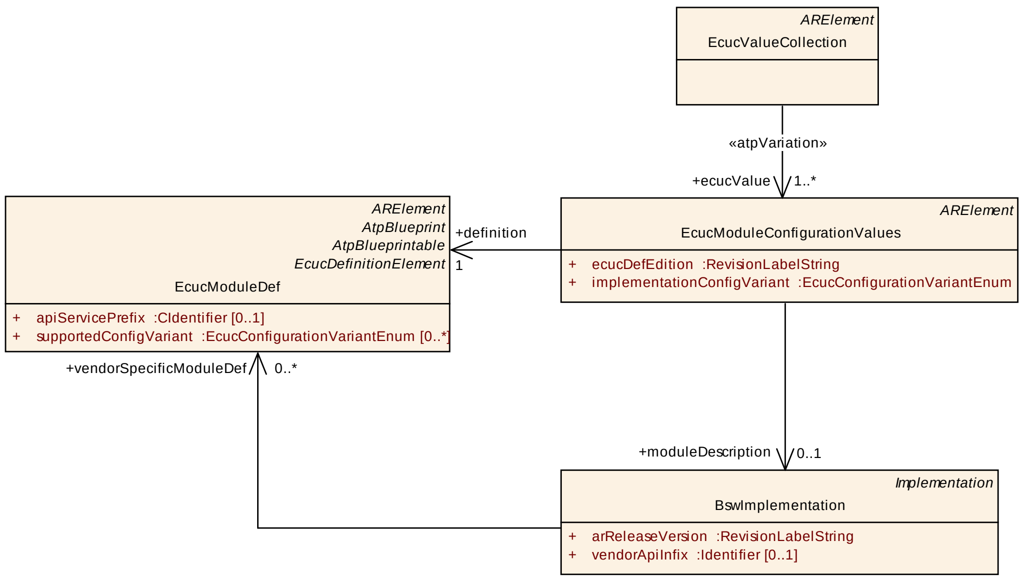
ARElement
EcucValueCollection
«atpVariation»
+ecucValue
1..*
ARElement
AtpBlueprint
AtpBlueprintable
EcucDefinitionElement
EcucModuleDef
+ apiServicePrefix :CIdentifier [0..1]
+ supportedConfigVariant :EcucConfigurationVariantEnum [0..*]
+definition
1
ARElement
EcucModuleConfigurationValues
+ ecucDefEdition :RevisionLabelString
+ implementationConfigVariant :EcucConfigurationVariantEnum
+vendorSpecificModuleDef
0..*
+moduleDescription
0..1
Implementation
BswImplementation
+ arReleaseVersion :RevisionLabelString
+ vendorApiInfix :Identifier [0..1]

Figure 4.6: Ecu Configuration Artifacts

- **[TPS_ECUC_06039] `BswImplementation` and `vendorSpecificModuleDef` shall be known by tools** ⌈ The tool knows his `BswImplementation` element and subsequently the `vendorSpecificModuleDef`. ⌋

- **[TPS_ECUC_06040] `EcucValueCollection` is the input for tools** ⌈ The tool shall get the `EcucValueCollection` as input information. Please note that according to the IOAT [6] the input can be provided as multiple files and can be structured in an arbitrary package structure. ⌋
- **[TPS_ECUC_06041] Tools shall respect the `EcucModuleConfigurationValues` elements that are referenced by the `EcucValueCollection`** ⌈. The tool shall respect only those `EcucModuleConfigurationValues` elements which are referenced by the `EcucValueCollection`. ⌋
- **[TPS_ECUC_06042] Tools interaction with `EcucModuleConfigurationValues`** ⌈ The tool shall directly interact only with those `EcucModuleConfigurationValues` elements whose definition reference is equal to the `vendorSpecificModuleDef` reference from the `BswImplementation`. ⌋ Example: If two CAN drivers from different vendors are to be configured with respective tools each tool can find the `EcucModuleConfigurationValues` to directly interact with using the definition references.

4.5 Post-build Time Loadable Consistency

When generating a post-build configuration, it shall be assured that the correct pre-compile and link-time configuration is used.

During initialization, it shall be possible to determine if the pre-compile and link-time configurations matches the post-build configuration (e.g. this can be done by placing a checksum based on the pre-compile and link-time configuration parameters in the pre-compile and link-time configuration and also placing the same checksum in the post-build configuration which are then compared).

In addition to this, additional check may be applied in order to assure that the compatible version of the configuration generator is used.

A Possible Implementations for the Configuration Steps

A.1 Alternative Approaches

This chapter contains description of alternative approaches and information that is not part of the AUTOSAR, but can be helpful and give some guidance.

A.1.1 Alternative Configuration Editor Approaches

[TPS_ECUC_02124] Tooling approaches that are supported by the ECUC parameter definition and ECUC Value description ⌈ The ECUC parameter definitions and ECUC Value descriptions are designed to support a variety of different tooling approaches.

In the following, the different approaches that have been considered during the development of the specification are introduced. These tooling approaches are supported by ECUC parameter definition and ECUC Value description. Other approaches might be consistent with this specification, but have not been considered explicitly. ⌋*(RS_ECUC_00043, RS_ECUC_00071, RS_ECUC_00074)*

Tool suppliers have a high degree of freedom in the approach their tools may take to ECU Configuration.

ECU Configuration tools might consist of a single monolithic editor capable of manipulating all aspects of ECU Configuration, it could be a core tool framework that takes plug-in components to manipulate particular aspects of ECU Configuration, it might be a set of specialized tools each capable of configuring a particular type or subset of software modules or, probably more likely, software vendors could supply individual custom tools to configure only the code blocks that they supply (similar to microprocessor vendors providing specialized debuggers for their own micros).

Common to the different tool approaches is that each configuration editor must be capable of reading an (possibly incomplete) `ECU Configuration Value description` and writing back its modified configuration results in the same format.

The modification may include changed values of ECU Configuration values and added instances of containers with all included ECU Configuration Values (only for containers/parameters with variable multiplicity).

In every case, the `ECU Configuration Value description` is expected to be the point of reference, the backbone of the process.

The sections below look at some possible tool forms and identify some of their strengths and weaknesses.

A.1.1.1 Custom Editors (Informative)

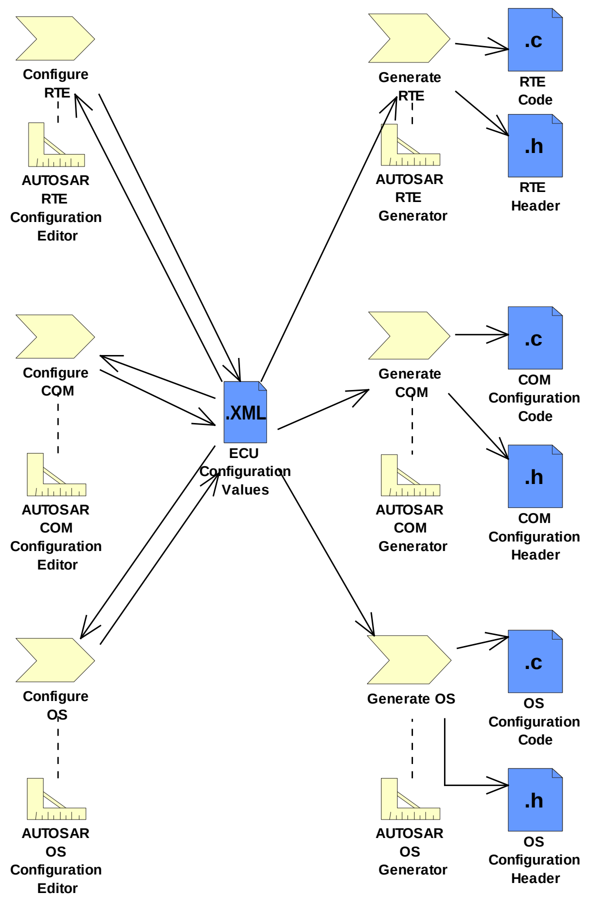

Figure A.1: Custom Editors and Generators

In the custom editors approach as shown in figure A.1, each BSW module is delivered bundled with a custom configuration editor and a generator (E.g. in figure A.1 the `AUTOSAR RTE Configuration Editor` and `AUTOSAR RTE Generator`).

These tools can be optimized to the particular task of configuring one BSW module and would likely be quite powerful. The complex dependencies between the BSW module configuration and other configuration items in the `ECU Configuration Value description` could be expressed and supported in the tool.

Each vendor of a BSW module would need to provide a tool. System and ECU engineers would require a large number of tools to deal with the range of BSW modules. Each tool would probably have an individual look and feel and this could increase the training and experience required to become proficient.

A.1.1.2 Generic Tools (Informative)

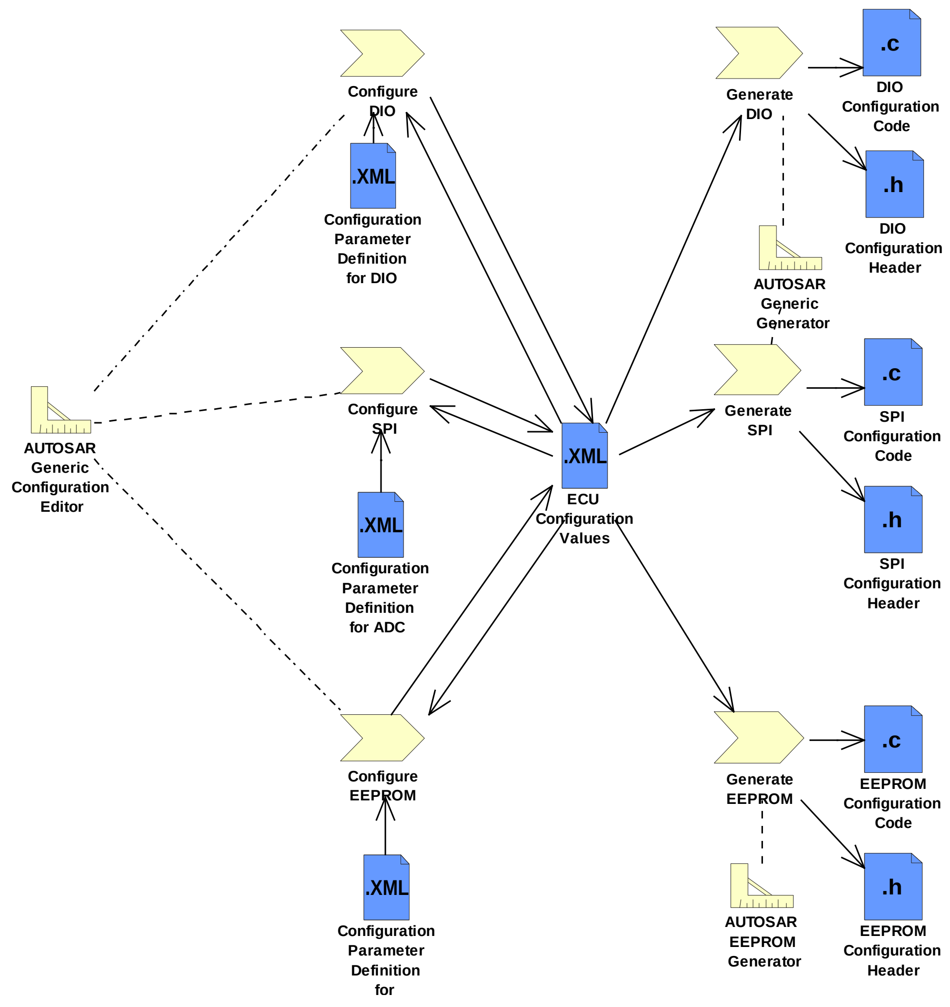

Figure A.2: Generic Configuration Editor

An `AUTOSAR Generic Configuration Editor` as shown in figure A.2 would be able to configure any parameter defined in `Configuration Parameter Definitions`. It would read those definitions and provide a generic interface to edit values for all parameters in the `ECU Configuration Value description`.

It would only be able to resolve the relatively simple dependencies explicitly defined in the `Configuration Parameter Definitions`. Only a limited number of editors would be required, maybe only one, and the look and feel is less likely to vary greatly between generic tools.

Training and tooling costs are therefore likely to be lower. Examples of such tools that already exist are tresos, GENy, DAvE and MICROSAR. On the generation side, either a generic generator may be used, or custom generators for the individual modules.

A.1.1.3 Tools Framework (Informative)

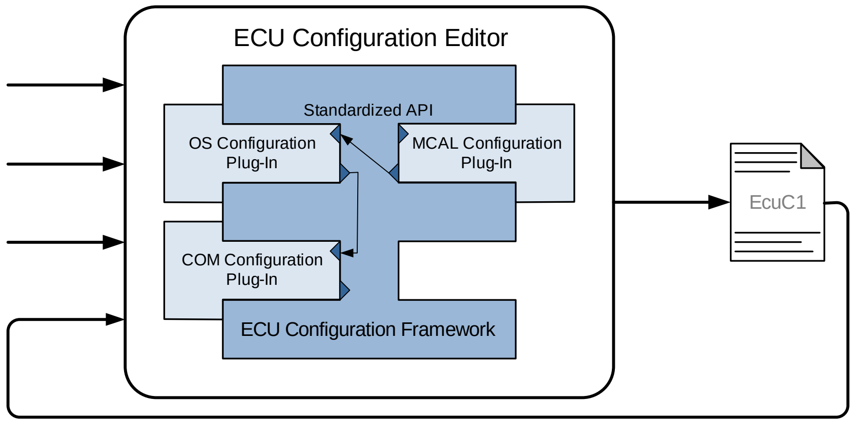
ECU Configuration Editor
Standardized API
OS Configuration Plug-In
MCAL Configuration Plug-In
COM Configuration Plug-In
ECU Configuration Framework
EcuC1

Figure A.3: Framework Tools

The tool framework as shown in figure A.3 is a cross between custom tools and generic tools where dedicated configuration steps (OS, COM, MCAL, etc.) are integrated as plug-ins into the common ECU Configuration framework.

The heart of the tool would be a framework that provides certain core services such as importing and exporting data from standard file formats, maintaining standard internal data structures and providing an HMI to the user. This ensures that the `ECU Configuration Value description` is read, interpreted and written in a defined way.

The frame could also monitor and control the user / project work flow. It provides a low initial tooling and training investment. The power of the approach would be the ability to add plug-in modules that extend the core functionality.

These would allow very sophisticated features, potentially dedicated to one BSW module, to be added by individual suppliers. Additionally, it would be possible to add generic plug-ins that addressed a specific aspect of configuration across all BSW modules. This approach relies upon a standard framework: multiple framework standards could lead to high tool costs.

An example of this kind of tool is the LabVIEW test and measurement tool from National Instruments and the Eclipse framework.

A.1.2 Alternative Generation Approaches

As stated before, the `ECU Configuration Value description` is the only configuration source that stores the configuration parameters for all modules of an AUTOSAR based ECU.

However, for several modules such as OS, existing configuration languages have already been established. Those languages probably will in future still be used for non-AUTOSAR systems. Thus, modules that are used both for AUTOSAR and non-AUTOSAR systems must support different configuration languages in parallel.

This can be implemented in different ways, as shown in figure A.4.

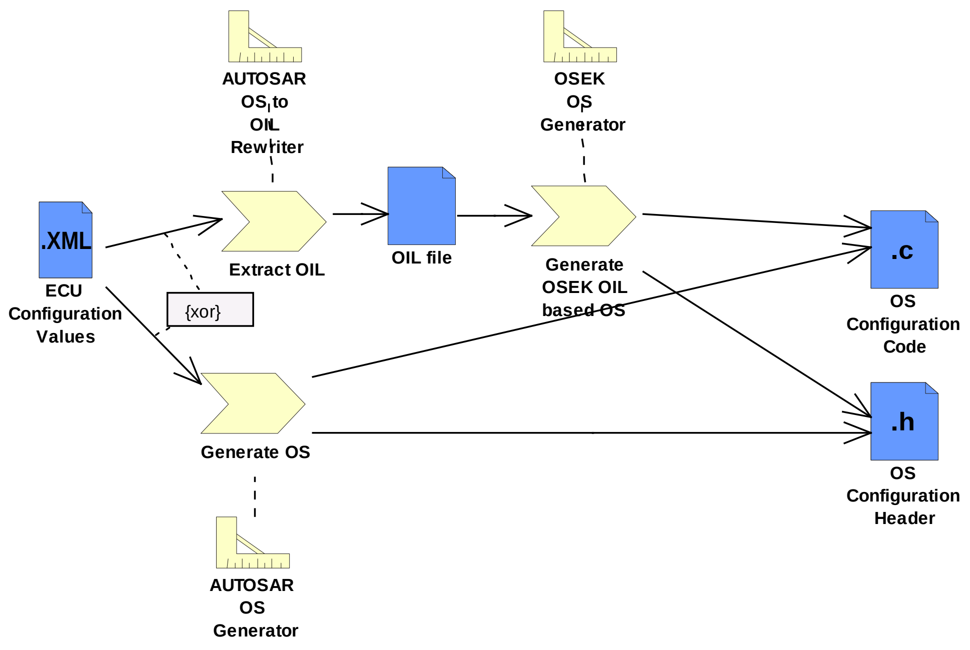

Figure A.4: Module generator implementation alternatives, example for OS

In a fully AUTOSAR based approach, the generator reads in the `ECU Configuration Value Description` and output the relevant source code directly in one step, supported by a `AUTOSAR OS Generator` in the example given.

To support reuse of existing generator tools, this single step can be split into two steps. Then the first activity is to extract all OS configuration information from the `ECU Configuration Value description` using an `AUTOSAR OS to OIL Rewriter` and to store it in the format used by the legacy tools (`OIL file` in the example chosen).

The existing `OSEK OS Generator` may then read the intermediate format to generate the code. However, the intermediate format must **not** be changed by any legacy configuration tools, since those changes would possibly make the overall configuration inconsistent, and since changes would be lost with the next generation of the intermediate format.

[TPS_ECUC_01025] Generate and extract activities are fully automatic ⌈ Thus, none of the activities (extract, generate) shown in figure A.4 must include any engineering step with decisions taken by an engineer. They are fully automatic, since all input driving these steps is included in the `ECU Configuration Value Description`. ⌋

B AUTOSAR Service Components

In the ECU Extract of the System Configuration only application Software Components are considered, while RTE and all BSW modules are not taken into account. In contrast, the ECU Configuration needs to consider all aspects of the ECU software, therefore means to support the addition of the BSW and RTE need to be provided.

To support this, the ECU Configuration Description allows the ECU extract to be extended by adding AUTOSAR Service prototypes and assembly connectors establishing the connections between applications and AUTOSAR service components (see figure B.1 *EcuTopLevelCompositionPrototype*).

AUTOSAR Services are modules like the NvRam Manager, the Watchdog Manager, the ECU State Manager, etc., which possess the characteristic trait that they interact with application software components using standardized AUTOSAR interfaces.

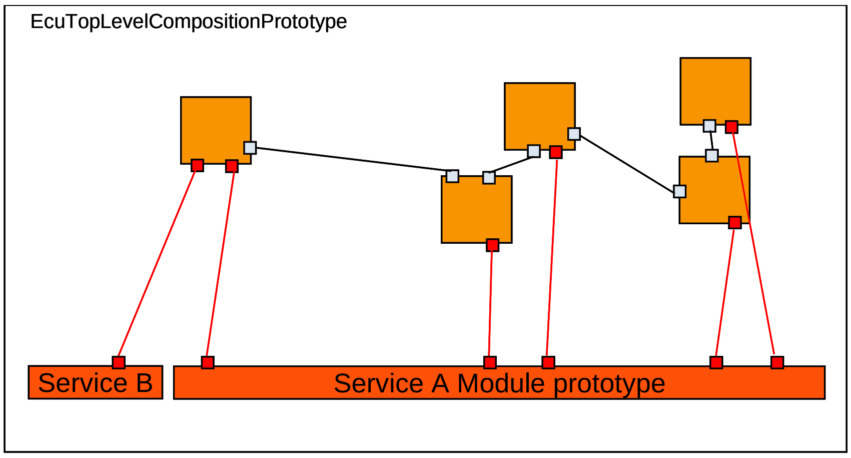

Figure B.1: Structure of the EcuTopLevelCompositionPrototype introduced in the ECU Configuration

To enable the extension of the existing ECU Extract towards a complete software system in the ECU Configuration, the aggregation of SwComponentPrototype and SwConnector by CompositionSwComponentType is stereotyped ≪atpSplitable≫.

This is shown in figure B.2. Making these aggregations ≪atpSplitable≫ allows the addition of AUTOSAR service component prototypes and connector prototypes to the CompositionSwComponentType contained in the ECU extract during the ECU integration without changing the artifacts which had been delivered as ECU extract.

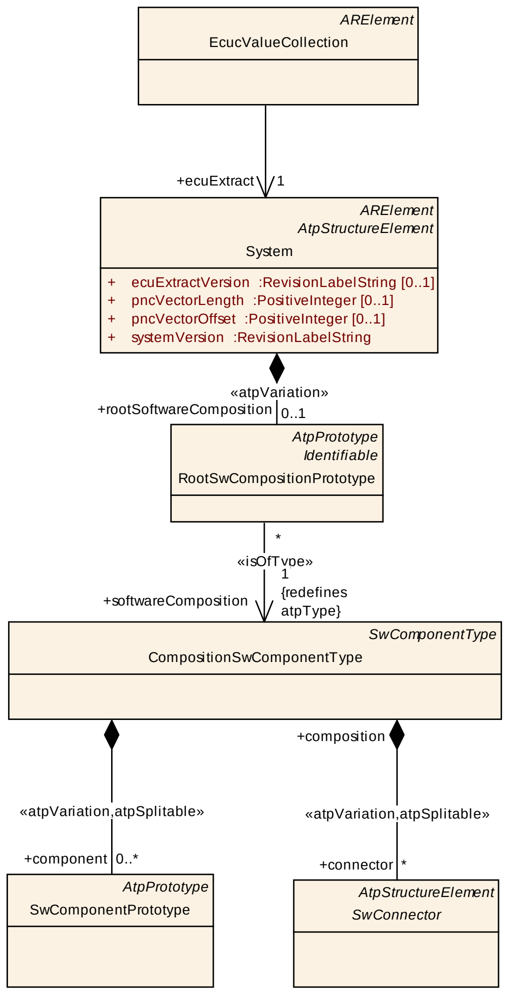

Figure B.2: Splittable aggregations of `SwComponentPrototype` and `SwConnector`

[TPS_ECUC_02087] Creation of `ServiceSwComponentType`s ⌈ When generating the AUTOSAR Service SW-Components the actual *service needs*[1] expressed by the Application SW-Components are collected.

For each AUTOSAR service required, a `ServiceSwComponentType` shall be created complete with an appropriate number of ports to enable the connection of all application component ports expressing the needs for the AUTOSAR service. ⌋*(RS_ECUC_00073)*

[TPS_ECUC_06014] Content of `CompositionSwComponentType` in the ECU Configuration ⌈ The `CompositionSwComponentType` in the ECU Configuration shall contain, additionally to prototypes of all application SW-Components running on the ECU as contained in the ECU Extract, `SwComponentPrototype`s for all required AUTOSAR Service modules and `AssemblySwConnector`s for the required connections between the Application SW-Component ports and the AUTOSAR Service module's ports. ⌋*(RS_ECUC_00076)*

[1]The *needs* of the Application SW-Components are defined in the SW-Component description in the `ServiceNeeds` section.

C 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 ([16]).

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.

Life Cycle Life Cycle is the course of development/evolutionary stages of a model element during its life time.

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 creates 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 implemented 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`.

D Change History

D.1 Change History between AUTOSAR R4.0.1 against R3.1.5

D.1.1 Renamed Meta-Model Elements for AUTOSAR Release 4.0

In the course of preparing AUTOSAR Release 4.0 some of the existing meta-model elements 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 former specifications to understand changes made by mere renaming.

Old Name	*New Name*
`AddInfoParamDef`	`EcucAddInfoParamDef`
`AddInfoParameterValue`	`EcucAddInfoParamValue`
`BooleanParamDef`	`EcucBooleanParamDef`
`ChoiceContainerDef`	`EcucChoiceContainerDef`
`ChoiceContainerReferenceParamDef`	`EcucChoiceReferenceDef`
`CommonConfigurationAttributes`	`EcucCommonAttributes`
`ConfigParameter`	`EcucParameterDef`
`ConfigReference`	`EcucAbstractReferenceDef`
`ConfigReferenceValue`	`EcucAbstractReferenceValue`
`ConfigurationAffection`	`EcucAffectionEnum`
`ConfigurationClass`	`EcucConfigurationClassEnum`
`ConfigurationClassAffection`	`EcucConfigurationClassAffection`
`ConfigurationVariant`	`EcucConfigurationVariantEnum`
`Container`	`EcucContainerValue`
`ContainerDef`	`EcucContainerDef`
`DerivationSpecification`	`EcucDerivationSpecification`
`EcuConfiguration`	`EcucValueCollection`
`EcuParameterDefinition`	`EcucDefinitionCollection`
`EcuParameterDerivationFormula`	`EcucParameterDerivationFormula`
`EnumerationLiteralDef`	`EcucEnumerationLiteralDef`
`EnumerationParamDef`	`EcucEnumerationParamDef`
`FloatParamDef`	`EcucFloatParamDef`
`ForeignReferenceDef`	`EcucForeignReferenceDef`
`FunctionNameDef`	`EcucFunctionNameDef`
`ImplementationConfigClass`	`EcucImplementationConfigurationClass`
`InstanceReferenceParamDef`	`EcucInstanceReferenceDef`
`InstanceReferenceValue`	`EcucInstanceReferenceValue`
`IntegerParamDef`	`EcucIntegerParamDef`
`LinkerSymbolDef`	`EcucLinkerSymbolDef`
`ModuleConfiguration`	`EcucModuleConfigurationValues`
`ModuleDef`	`EcucModuleDef`
`MultilineStringParamDef`	`EcucMultilineStringParamDef`
`NumericalParameterValue`	`EcucNumericalParamValue`
`ParamConfContainerDef`	`EcucParamConfContainerDef`
`ParamConfMultiplicity`	`EcucDefinitionElement`
`ParameterValue`	`EcucParameterValue`
`ReferenceParamDef`	`EcucReferenceDef`
`ReferenceValue`	`EcucReferenceValue`

StringParamDef	EcucStringParamDef
SymbolicNameReferenceParamDef	EcucSymbolicNameReferenceDef
TextualParameterValue	EcucTextualParamValue

Table D.1: Renamed meta-model elements

D.1.2 Deleted SWS Items

SWS Item	*Rationale*
[ecuc_sws_3000]	Removed type specific value definitions.
[ecuc_sws_3001]	Removed type specific value definitions.
[ecuc_sws_3002]	Removed type specific value definitions.
[ecuc_sws_3003]	Removed type specific value definitions.
[ecuc_sws_3041]	Removed type specific value definitions.
[ecuc_sws_3005]	Removed type specific value definitions.
[ecuc_sws_1031]	The requirement from chapter A.1.1 has been changed to [TPS_ECUC_02124] because there were two requirements for this Id.
[ecuc_sws_2046]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2048]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2049]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2050]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2051]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2052]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2053]	Removed due to changes on the derived parameter structure.
[ecuc_sws_3022]	Removed due to changes on the derived parameter structure.
[ecuc_sws_3023]	Removed due to changes on the derived parameter structure.
[ecuc_sws_3024]	Removed due to changes on the derived parameter structure.
[ecuc_sws_3025]	Removed due to changes on the derived parameter structure.
[ecuc_sws_3026]	Removed due to changes on the derived parameter structure.
[ecuc_sws_1008]	Removed due to changes on the derived parameter structure.
[ecuc_sws_5004]	Removed due to changes on the derived parameter structure.
[ecuc_sws_5005]	Removed due to changes on the derived parameter structure.
[ecuc_sws_5006]	Removed due to changes on the derived parameter structure.
[ecuc_sws_2085]	Removed due to changes in Ecu Extract description.
[ecuc_sws_2086]	Removed due to changes in Ecu Extract description.
[ecuc_sws_2045]	EcuC Parameter Definitions can also be manually generated.
[ecuc_sws_2113]	currently not supported by the Variant Handling concept (aggregation of Primitives).
[ecuc_sws_2068]	Replaced requirement by [TPS_ECUC_02012].
[ecuc_sws_1000]	Replaced requirement by [TPS_ECUC_06038].
[ecuc_sws_1002]	Replaced with [TPS_ECUC_06007] and [TPS_ECUC_06008] for clarification of derivation rules.
[ecuc_sws_1003]	Replaced with [TPS_ECUC_06007] and [TPS_ECUC_06008] for clarification of derivation rules.
[ecuc_sws_1010]	Removed for clarification of derivation rules.
[ecuc_sws_1012]	Removed for clarification of derivation rules.

Table D.2: Deleted SWS Items

D.1.3 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_02029]	Subclasses of EcucAbstractStringParamDef
[TPS_ECUC_02030]	Programming language identifier limitations
[TPS_ECUC_02031]	Restriction on the length of EcucLinkerSymbolDef values and defaultValue
[TPS_ECUC_02108]	Rule for the creation of #define symbols in the header file for parameters with the symbolicNameValue set to TRUE
[ecuc_sws_5001]	Refined because of unclear requirement.
[TPS_ECUC_03021]	EcucParameterDefs with EcucDerivationSpecification result in a EcucNumericalParamValue in the ECUC Value description
[TPS_ECUC_02047]	Derivation of parameter values
[TPS_ECUC_02087]	Creation of ServiceSwComponentTypes
[TPS_ECUC_02000]	Modeling of ECU Configuration Value and ECU Configuration Parameter Definition metamodels
[ecuc_sws_2084]	Incompatible inter module queries.
[ecuc_sws_6002]	Incompatible inter module queries.
[TPS_ECUC_01032]	Link time configuration
[TPS_ECUC_02030]	Programming language identifier limitations
[TPS_ECUC_02095]	VSMD refines the StMD
[TPS_ECUC_03007]	Attribute value stores the configuration value in XML-based description
[TPS_ECUC_03034]	Each parameter in an Ecuc Configuration Value description shall have a value
[TPS_ECUC_03010]	Parameters that are declared as optional in the ECU Configuration Definition may be left out in the ECU Configuration Value description
[TPS_ECUC_03040]	The value of an EcucNumericalParamValue shall be unambiguously an integer value
[TPS_ECUC_02107]	Values of parameters with the symbolicNameValue set to TRUE that are assigned by the configuration editor or module generator shall be stored in the XML file
[TPS_ECUC_02001]	Transformation of the ECU Configuration Value and ECU Configuration Parameter Definition metamodels to schema definitions
[TPS_ECUC_02002]	Generic structure of all AUTOSAR templates
[TPS_ECUC_06004]	AdminData field in ECU Configuration Parameter Definition XML file
[TPS_ECUC_02067]	Multiplicity of the to be chosen containers
[TPS_ECUC_02012]	Allowed choice of available to be chosen containers in the ECU Configuration Value description
[TPS_ECUC_02009]	Expression of optionality of containers, parameters and references
[TPS_ECUC_02029]	Subclasses of EcucAbstractStringParamDef
[TPS_ECUC_02087]	Creation of ServiceSwComponentTypes

Table D.3: Changed SWS Items

D.1.4 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_02110]	Variable lower and upper multiplicity in ECU Configuration Parameter definition
[TPS_ECUC_02111]	Variable default value in EcucBooleanParamDef
[TPS_ECUC_02112]	Variable default value in EcucAbstractStringParamDef
[ecuc_sws_2113]	Implement Variant Handling Concept.
[TPS_ECUC_02114]	Variable default value in EcucIntegerParamDef
[TPS_ECUC_02115]	Variable default value in EcucFloatParamDef
[TPS_ECUC_02116]	Variable min, max values in EcucIntegerParamDef

[TPS_ECUC_02117]	Variable min, max values in EcucFloatParamDef
[TPS_ECUC_02119]	Variable existence of container on value side
[TPS_ECUC_02120]	Variable subContainers
[TPS_ECUC_02121]	Variable parameterValues
[TPS_ECUC_02122]	Variable referenceValues
[TPS_ECUC_02118]	EcucAddInfoParamDef properties
[TPS_ECUC_02123]	The value of the parameter type EcucAddInfoParamDef
[TPS_ECUC_02124]	Tooling approaches that are supported by the ECUC parameter definition and ECUC Value description
[TPS_ECUC_02125]	Value of parameters with a defined derivation specification
[TPS_ECUC_02126]	Values for parameter types stored in the element EcucTextualParamValue
[TPS_ECUC_02127]	Possible values for EcucBooleanParamDef parameters
[TPS_ECUC_02128]	Formal description of the derivation
[TPS_ECUC_02129]	Informal description of the derivation
[TPS_ECUC_02130]	Standardized Module Definition package structure
[TPS_ECUC_06014]	Content of CompositionSwComponentType in the ECU Configuration
[TPS_ECUC_06015]	DESTINATION-REF in the VSMD
[TPS_ECUC_06016]	Countably infinite number of containers, parameters and references in the ECU Configuration Value description
[TPS_ECUC_06017]	Existence of upperMultiplicityInfinite and upperMultiplicity is mutually exclusive
[TPS_ECUC_06018]	Input and Output of the refvalue function
[TPS_ECUC_06019]	Output of the refvalue function if the EcucDefinitionElement points to a not existing element in the ECU Configuration Parameter Definition
[TPS_ECUC_06020]	Output of the refvalue function if no element in the ECU Configuration Value description is found
[TPS_ECUC_06021]	Input and Output of the deref function
[TPS_ECUC_06022]	Output of the deref function in case the first input parameter is a reference
[TPS_ECUC_06023]	Cases where the deref function reports an error
[TPS_ECUC_06024]	Input of the value function
[TPS_ECUC_06025]	Output of the value function
[TPS_ECUC_06026]	Cases where the value function reports an error
[TPS_ECUC_06027]	Input of the count function
[TPS_ECUC_06028]	Output of the count function
[TPS_ECUC_06029]	Output of the count function in case the input parameter set is empty
[TPS_ECUC_06030]	Invalid PduLength parameter value configuration
[TPS_ECUC_06031]	Interaction of Complex Driver with standardized AUTOSAR BSW modules
[TPS_ECUC_06032]	Min and max values in EcucIntegerParamDef
[TPS_ECUC_06033]	Min and max values in EcucFloatParamDef
[TPS_ECUC_06034]	Special float values
[TPS_ECUC_01032]	Link time configuration
[TPS_ECUC_06036]	Distinction of module definitions of Complex Drivers
[TPS_ECUC_06037]	apiServicePrefix attribute for Complex Driver modules
[TPS_ECUC_06038]	Rules to validate a BSW module implementation
[TPS_ECUC_06043]	EcucModuleDef categories
[TPS_ECUC_06044]	refinedModuleDef reference in the StMD
[TPS_ECUC_06035]	Regular expression
[TPS_ECUC_06006]	EcucLinkerSymbolDef properties
[TPS_ECUC_06007]	Elements defined in the StMD shall be present in the VSMD
[TPS_ECUC_06008]	lowerMultiplicity and upperMultiplicity of elements in the VSMD
[TPS_ECUC_06009]	Existence of a parameter in the Ecuc Parameter Value description in case the upperMultiplicity of the parameter definition is zero
[TPS_ECUC_06010]	Existence of a parameter in the Ecuc Parameter Value description in case the lowerMultiplicity of the parameter definition is bigger than zero

[TPS_ECUC_06011]	Missing parameters in the Ecuc Parameter Value description
[TPS_ECUC_06012]	Parameters without parameter definitions in the Ecuc Parameter Value description
[TPS_ECUC_06013]	Number of parameters in the Ecuc Parameter Value description

Table D.4: Added SWS Items

D.2 Change History between AUTOSAR R4.0.2 against R4.0.1

D.2.1 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_02072]	Signed EcucIntegerParamDef value range
[TPS_ECUC_02095]	VSMD refines the StMD

Table D.5: Changed SWS Items

D.2.2 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_02131]	Origin information in literal definitions
[TPS_ECUC_06039]	BswImplementation and vendorSpecificModuleDef shall be known by tools
[TPS_ECUC_06040]	EcucValueCollection is the input for tools
[TPS_ECUC_06041]	Tools shall respect the EcucModuleConfigurationValues elements that are referenced by the EcucValueCollection
[TPS_ECUC_06042]	Tools interaction with EcucModuleConfigurationValues
[TPS_ECUC_06043]	EcucModuleDef categories
[TPS_ECUC_06044]	refinedModuleDef reference in the StMD
[TPS_ECUC_06045]	min, max values of parameters in the VSMD in case that the min or max value in the StMD is set to infinite

Table D.6: Added SWS Items

D.3 Change History between AUTOSAR R4.0.3 against R4.0.2

D.3.1 Deleted SWS Items

SWS Item	*Rationale*
[ecuc_sws_5001]	Removed because it was unclear.

Table D.7: Deleted SWS Items

D.3.2 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_02041]	EcucForeignReferenceDef properties

[TPS_ECUC_02082]	Specification of the destinationType in a EcucInstanceReferenceDef
[TPS_ECUC_02083]	Specification of the destinationContext in a EcucInstanceReferenceDef
[TPS_ECUC_06002]	Removal of standardized EcucEnumerationLiteralDefs in the VSMD
[TPS_ECUC_06015]	DESTINATION-REF in the VSMD
[TPS_ECUC_06025]	Output of the value function
[TPS_ECUC_06037]	apiServicePrefix attribute for Complex Driver modules
[TPS_ECUC_02108]	Rule for the creation of #define symbols in the header file for parameters with the symbolicNameValue set to TRUE

Table D.8: Changed SWS Items

D.3.3 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_02132]	The postBuildChangeable attribute shall only be set to true for containers located within a multipleConfigurationContainer
[TPS_ECUC_02133]	upperMultiplicity of a multipleConfigurationContainer
[TPS_ECUC_06046]	Vendor specific reference definition with no counterpart in the STMD
[TPS_ECUC_06047]	References in the ECUC Parameter Value description with reference definitions that refer to container definitions in the same module definition
[TPS_ECUC_06048]	References in the ECUC Parameter Value description with reference definitions that refer to container definitions in different module definitions
[TPS_ECUC_06049]	Restriction of supportedConfigVariants in the VSMD
[TPS_ECUC_06050]	supportedConfigVariants in the VSMD in case VariantPostBuild is supported in the StMD
[TPS_ECUC_06051]	ImplementationConfigClass of an EcucParameterDef or EcucAbstractReferenceDef in VSMD
[TPS_ECUC_06052]	Supported configuration variants in the VSMD
[TPS_ECUC_06053]	VSMD Configuration variant “VariantPreCompile”
[TPS_ECUC_06054]	VSMD Configuration variant “VariantLinkTime”
[TPS_ECUC_06055]	VSMD Configuration variant “VariantPostBuildLoadable”
[TPS_ECUC_06056]	VSMD Configuration variant “VariantPostBuildSelectable”
[TPS_ECUC_06057]	Input of the strValue function
[TPS_ECUC_06058]	Output of the strValue function
[TPS_ECUC_06059]	Cases where the strValue function reports an error
[TPS_ECUC_06060]	Input of the valueAt function
[TPS_ECUC_06061]	Output of the valueAt function
[TPS_ECUC_06062]	Cases where the valueAt function reports an error
[TPS_ECUC_06063]	Input of the strValueAt function
[TPS_ECUC_06064]	Output of the strValueAt function
[TPS_ECUC_06065]	Cases where the strValueAt function reports an error
[TPS_ECUC_06066]	Order of Container-, Parameter- and Reference-Values
[TPS_ECUC_06067]	Sorting criteria for Containers on the Values side
[TPS_ECUC_06068]	Sorting criteria for References on the Values side
[TPS_ECUC_06069]	Sorting criteria for Parameters on the Values side
[TPS_ECUC_06070]	Sorting of Ecu Configuration Parameter Definitions
[TPS_ECUC_06071]	ECU Configuration Editor shall be able to read parameter values in any order
[TPS_ECUC_06072]	Container-, Parameter-, and Reference-Values with requiresIndex set to true
[TPS_ECUC_06073]	The ECU Configuration Editor shall be able to work with arbitrary package structures
[TPS_ECUC_06074]	Invalid configuration due to symbolic name values

Table D.9: Added SWS Items

D.3.4 Added Constraints

Number	Heading
[constr_3022]	EcucModuleDef category restriction
[constr_3023]	Usage of apiServicePrefix

Table D.10: Added Constraints in R4.0.3

D.4 Change History between AUTOSAR R4.1.1 against R4.0.3

D.4.1 Deleted SWS Items

SWS Item	*Rationale*

Table D.11: Deleted SWS Items

D.4.2 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_06067]	Sorting criteria for Containers on the Values side
[TPS_ECUC_06072]	Container-, Parameter-, and Reference-Values with requiresIndex set to true
[TPS_ECUC_02039]	References between containers are established with the EcucReferenceDef
[TPS_ECUC_02040]	EcucChoiceReferenceDef properties
[TPS_ECUC_06051]	ImplementationConfigClass of an EcucParameterDef or EcucAbstractReferenceDef in VSMD

Table D.12: Changed SWS Items

D.4.3 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_02134]	requiresIndex setting in the VSMD
[TPS_ECUC_02135]	Validation of EcucValidationCondition
[TPS_ECUC_02136]	Validation of multiple EcucValidationConditions
[TPS_ECUC_02137]	EcucValidationConditions from the StMD shall be taken over to the VSMD.
[TPS_ECUC_02138]	Addition of vendor specific EcucValidationConditions
[TPS_ECUC_02139]	Definition of configuration classes for all CDD configuration parameters and references
[TPS_ECUC_02141]	Variable reference `EcucValueCollection.ecucValue`
[TPS_ECUC_02142]	Variable value of `EcucNumericalParamValue.value`
[TPS_ECUC_08001]	Configuration class of parameters and references within postBuildChangeable containers
[TPS_ECUC_08002]	Introduction of new EcucParamConfContainerDef instances in a post-build loadable configuration set

[TPS_ECUC_08003]	Usage of postBuildChangeable attribute is independent of aggregated sub-Containers
[TPS_ECUC_08004]	Changing of values and multiplicities of EcucParameterValues at post-build time
[TPS_ECUC_08005]	postBuildChangeable attribute in the VSMD in case it is not defined in the StMD
[TPS_ECUC_08006]	postBuildChangeable attribute in the VSMD in case it is set to false in the StMD
[TPS_ECUC_08007]	postBuildChangeable attribute in the VSMD in case it is set to true in the StMD
[TPS_ECUC_08008]	Usage of the multiple configuration container in EcucModuleDefs with supportedConfigVariant of VariantPostBuild
[TPS_ECUC_08009]	Names of containers inside a multiple configuration set
[TPS_ECUC_08010]	Ticks in the Ecuc Parameter Value description
[TPS_ECUC_08011]	Pattern for creating a C symbol used by the EcuM/BswM to initialize post-build Bsw modules
[TPS_ECUC_02143]	Optional configuration of Production Error and Extended Production Error reporting
[TPS_ECUC_02144]	Definition of supported config variants for CDD

Table D.13: Added SWS Items

D.4.4 Added Constraints

[constr_3509]	Applicability of `scope` attribute
[constr_5500]	Applicability of `postBuildChangeable` attribute
[constr_5501]	`EcucParameterValue`s and `EcucAbstractReferenceValue`s in `EcucContainerValue`s that exist in multiple configuration sets
[constr_5502]	`EcucParameterValue`s of type `EcucFunctionNameDef`
[constr_5503]	`symbolicNameValue` parameters in post-build configuration sets
[constr_5504]	Removing an instance of the `EcucContainerDef` in post-build time
[constr_5505]	Configuration class of the elements of the `EcucQueryExpression`

Table D.14: Added Constraints in R4.1.1

D.5 Change History between AUTOSAR R4.1.2 against R4.1.1

D.5.1 Deleted SWS Items

SWS Item	*Rationale*
[TPS_ECUC_06002]	Removal of standardized `EcucEnumerationLiteralDef`s in the VSMD
[TPS_ECUC_01036]	Migrated to TR_METH_01114
[TPS_ECUC_01027]	Migrated to TR_METH_01115
[TPS_ECUC_01028]	Migrated to TR_METH_01116
[TPS_ECUC_01031]	Equivalent to TR_METH_01095
[TPS_ECUC_01032]	Equivalent to TR_METH_01098
[TPS_ECUC_04006]	Equivalent to TR_METH_01104
[TPS_ECUC_04007]	Equivalent to TR_METH_01107
[TPS_ECUC_01029]	Migrated to TR_METH_01117

[TPS_ECUC_01030]	Equivalent to TR_METH_01089
[TPS_ECUC_04000]	Deprecated
[TPS_ECUC_04001]	Relies on deprecated chapter
[TPS_ECUC_02132]	Contradictory to the variant handling approach described in chapter 2.4.8

Table D.15: Deleted SWS Items

D.5.2 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_06051]	ImplementationConfigClass of an `EcucParameterDef` or `EcucAbstractReferenceDef` in VSMD
[TPS_ECUC_04002]	ECU Configuration Editor shall be able to merge `ECU Configuration Value descriptions`

Table D.16: Changed SWS Items

D.5.3 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_06075]	`EcucFunctionNameDef` shall represent a valid C Identifier

Table D.17: Added SWS Items

D.6 Change History between AUTOSAR R4.1.3 against R4.1.2

D.6.1 Deleted SWS Items

SWS Item	*Heading*
[TPS_ECUC_02022]	Configuration variants of a BSW module in the ECU Configuration Parameter Definition

Table D.18: Deleted SWS Items

D.6.2 Changed SWS Items

SWS Item	*Heading*
[TPS_ECUC_02095]	VSMD refines the StMD
[TPS_ECUC_02140]	Mandatory configuration of `CddConfigSet` for post build configured CDD

Table D.19: Changed SWS Items

D.6.3 Added SWS Items

SWS Item	*Heading*
[TPS_ECUC_06076]	Use cases where the reference refinedModuleDef is mandatory
[TPS_ECUC_06077]	Use cases where the reference refinedModuleDef is optional

Table D.20: Changed SWS Items

D.6.4 Added Constraints

[constr_3091]	Multiplicity of implementationConfigClass
[constr_3092]	Usage of configVariant and configClass attributes

Table D.21: Added Constraints in R4.1.3

E Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**ARElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	An element that can be defined stand-alone, i.e. without being part of another element (except for packages of course).			
Base	ARObject,CollectableElement,Identifiable,MultilanguageReferrable,Packageable Element,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table E.1: ARElement

Class	**ARPackage**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	AUTOSAR package, allowing to create top level packages to structure the contained ARElements. ARPackages are open sets. This means that in a file based description system multiple files can be used to partially describe the contents of a package. This is an extended version of MSR's SW-SYSTEM.			
Base	ARObject,AtpBlueprint,AtpBlueprintable,Collectable Element,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arPackage	ARPackage	*	aggr	This represents a sub package within an ARPackage, thus allowing for an unlimited package hierarchy. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
element	PackageableEle ment	*	aggr	Elements that are part of this package **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=systemDesignTime xml.sequenceOffset=20

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
referenceBase	ReferenceBase	*	aggr	This denotes the reference bases for the package. This is the basis for all relative references within the package. The base needs to be selected according to the base attribute within the references. **Stereotypes:** atpSplitable **Tags:** atp.splitkey=shortLabel xml.sequenceOffset=10

Table E.2: ARPackage

Class	**AdminData**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::AdminData			
Note	AdminData represents the ability to express administrative information for an element. This administration information is to be treated as meta-data such as revision id or state of the file. There are basically four kinds of meta-data • The language and/or used languages. • Revision information covering e.g. revision number, state, release date, changes. Note that this information can be given in general as well as related to a particular company. • Document meta-data specific for a company			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
docRevision (ordered)	DocRevision	*	aggr	This allows to denote information about the current revision of the object. Note that information about previous revisions can also be logged here. The entries shall be sorted descendant by date in order to reflect the history. Therefore the most recent entry representing the current version is denoted first. **Tags:** xml.roleElement=true; xml.roleWrapperElement=true; xml.sequenceOffset=50; xml.typeElement=false; xml.typeWrapperElement=false
language	LEnum	0..1	attr	This attribute specifies the master language of the document or the document fragment. The master language is the one in which the document is maintained and from which the other languages are derived from. In particular in case of inconsistencies, the information in the master language is priority. **Tags:** xml.sequenceOffset=20

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
sdg	Sdg	*	aggr	This property allows to keep special data which is not represented by the standard model. It can be utilized to keep e.g. tool specific data. **Tags:** xml.roleElement=true; xml.roleWrapper Element=true; xml.sequenceOffset=60; xml.type Element=false; xml.typeWrapperElement=false
usedLangu ages	MultiLanguageP lainText	0..1	aggr	This property specifies the languages which are provided in the document. Therefore it should only be specified in the top level admin data. For each language provided in the document there is one entry in MultilanguagePlainText. The content of each entry can be used for illustration of the language. The used language itself depends on the language attribute in the entry. **Tags:** xml.sequenceOffset=30

Table E.3: AdminData

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	AbstractProvide dPortPrototype	0..1	iref	Instance of providing port.
requester	AbstractRequire dPortPrototype	0..1	iref	Instance of requiring port.

Table E.4: AssemblySwConnector

Class	**BswImplementation**			
Package	M2::AUTOSARTemplates::BswModuleTemplate::BswImplementation			
Note	Contains the implementation specific information in addition to the generic specification (BswModuleDescription and BswBehavior). It is possible to have several different BswImplementations referring to the same BswBehavior. **Tags:** atp.recommendedPackage=BswImplementations			
Base	ARElement,ARObject,CollectableElement,Identifiable,Implementation,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
arRelease Version	RevisionLabelSt ring	1	attr	Version of the AUTOSAR Release on which this implementation is based. The numbering contains three levels (major, minor, revision) which are defined by AUTOSAR.
behavior	BswInternalBeh avior	1	ref	The behavior of this implementation.

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
debugInfo	BswDebugInfo	0..1	aggr	Collects the debug info for this implementation. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
preconfiguredConfiguration	EcucModuleConfigurationValues	*	ref	Reference to the set of preconfigured (i.e. fixed) configuration values for this BswImplementation. If the BswImplementation represents a cluster of several modules, more than one EcucModuleConfigurationValues element can be referred (at most one per module), otherwise at most one such element can be referred. **Tags:** xml.roleWrapperElement=true
recommendedConfiguration	EcucModuleConfigurationValues	*	ref	Reference to one or more sets of recommended configuration values for this module or module cluster.
vendorApiInfix	Identifier	0..1	ref	In driver modules which can be instantiated several times on a single ECU, SRS_BSW_00347 requires that the names of files, APIs, published parameters and memory allocation keywords are extended by the vendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific API name is generated as follows: <ModuleName>_<vendorId>_ <vendorApiInfix>_<API name from SWS>. E.g. assuming that the vendorId of the implementer is 123 and the implementer chose a vendorApiInfix of "v11r456" an API name Can_Write defined in the SWS will translate to Can_123_v11r456_Write. This attribute is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1. See also SWS_BSW_00102.
vendorSpecificModuleDef	EcucModuleDef	*	ref	Reference to • the vendor specific EcucModuleDef used in this BswImplementation if it represents a single module • several EcucModuleDefs used in this BswImplementation if it represents a cluster of modules • one or no EcucModuleDefs used in this BswImplementation if it represents a library **Tags:** xml.roleWrapperElement=true

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

Table E.5: BswImplementation

Primitive	**CIdentifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
Note	This datatype represents a string, that follows the rules of C-identifiers. **Tags:** xml.xsd.customType=C-IDENTIFIER; xml.xsd.pattern=[a-zA-Z_][a-zA-Z0-9_]*; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
namePatte rn	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the CIdentifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:** xml.attribute=true

Table E.6: CIdentifier

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***

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
component	SwComponentPrototype	*	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 post-build 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:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=postBuild
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:** atp.Splitkey=shortName, variation Point.shortLabel vh.latestBindingTime=postBuild
constantValueMapping	ConstantSpecificationMappingSet	*	ref	Reference to the ConstantSpecificationMapping to be applied for initValues of PPortComSpecs and RPortComSpec.

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.
instantiationRTEEventProps	InstantiationRTEEventProps	*	aggr	This allows to define instantiation specific properties for RTE Events, in particular for instance specific scheduling. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=shortLabel, variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

Table E.7: CompositionSwComponentType

Class	**Documentation**			
Package	M2::AUTOSARTemplates::GenericStructure::DocumentationOnM1			
Note	This meta-class represents the ability to handle a so called standalone documentation. Standalone means, that such a documentation is not embedded in another ARElement or identifiable object. The standalone documentation is an entity of its own which denotes its context by reference to other objects and instances. **Tags:** atp.recommendedPackage=Documentations			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
context	DocumentationContext	*	aggr	This is the context of the particular documentation.
documentationContent	PredefinedChapter	0..1	aggr	This is the content of the documentation related to the specified contexts. **Tags:** xml.sequenceOffset=200

Table E.8: Documentation

Enumeration	**EcucAffectionEnum**

Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible affections used by the EcucConfigurationClassAffection.
Literal	***Description***
LTAffectsPB	A link time parameter affecting one or several post-build time parameter(s).
NOAffect	no affect on any other parameter.
PCAffectsLT	A pre-compile time parameter affecting one or several link time parameter(s).
PCAffectsLT AndPB	A pre-compile time parameter affecting one or several link time and post-build time parameter(s)).
PCAffectsPB	A pre-compile time parameter affecting one or several post build time parameter(s).

Table E.9: EcucAffectionEnum

Enumeration	**EcucConfigurationClassEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible configuration classes for the AUTOSAR configuration parameters.
Literal	***Description***
Link	Link Time: parts of configuration are delivered from another object code file
PostBuild	PostBuildTime: the configuration parameter has to be stored at a known memory location.
PreCompile	PreCompile Time: after compilation a configuration parameter can not be changed any more.
Published Information	PublishedInformation is used to specify the fact that certain information is fixed even before the pre-compile stage.

Table E.10: EcucConfigurationClassEnum

Enumeration	**EcucConfigurationVariantEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Specifies the possible Configuration Variants used for AUTOSAR BSW Modules.
Literal	***Description***
Preconfigured Configuration	Preconfigured (i.e. fixed) configuration which cannot be changed.
Recommended Configuration	Recommended configuration for a module.
VariantLink Time	Specifies that the BSW Module implementation may use PreCompileTime and LinkTime configuration parameters.
VariantPost Build	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild configuration parameters.
VariantPost BuildLoad-able	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild loadable configuration parameters (supported in the VSMD).
VariantPost BuildSe-lectable	Specifies that the BSW Module implementation may use PreCompileTime, LinkTime and PostBuild selectable configuration parameters (supported in the VSMD).
VariantPre Compile	Specifies that the BSW Module implementation uses only PreCompileTime configuration parameters.

Table E.11: EcucConfigurationVariantEnum

Class	**Frame (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Data frame which is sent over a communication medium. This element describes the pure Layout of a frame sent on a channel.			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
frameLength	Integer	1	attr	The used length (in bytes) of the referencing frame. Should not be confused with a static byte length reserved for each frame by some platforms (e.g. FlexRay). The frameLength of zero bytes is allowed.
pduToFrameMapping	PduToFrameMapping	*	aggr	A frames layout as a sequence of Pdus. atpVariation: The content of a frame can be variable. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

Table E.12: Frame

Class	**FrameTriggering (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The FrameTriggering describes the instance of a frame sent on a channel and defines the manner of triggering (timing information) and identification of a frame on the channel, on which it is sent. For the same frame, if FrameTriggerings exist on more than one channel of the same cluster the fan-out/in is handled by the Bus interface.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
frame	Frame	1	ref	One frame can be triggered several times, e.g. on different channels. If a frame has no frame triggering, it won't be sent at all. A frame triggering has assigned exactly one frame, which it triggers.
framePort	FramePort	*	ref	References to the FramePort on every ECU of the system which sends and/or receives the frame. References for both the sender and the receiver side shall be included when the system is completely defined.
pduTriggering	PduTriggering	*	ref	This reference provides the relationship to the PduTriggerings that are implemented by the FrameTriggering. The reference is optional since no PduTriggering can be defined for NmPdus and XCP Pdus. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

Table E.13: FrameTriggering

Class	**HwElement**			
Package	M2::AUTOSARTemplates::EcuResourceTemplate			
Note	This represents the ability to describe Hardware Elements on an instance level. The particular types of hardware are distinguished by the category. This category determines the applicable attributes. The possible categories and attributes are defined in HwCategory. **Tags:** atp.recommendedPackage=HwElements			
Base	ARElement,ARObject,CollectableElement,HwDescriptionEntity,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
hwElementConnection	HwElementConnector	*	aggr	This represents one particular connection between two hardware elements. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=110
hwPinGroup	HwPinGroup	*	aggr	This aggregation is used to describe the connection facilities of a hardware element. Note that hardware element has no pins but only pingroups. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=90
nestedElement	HwElement	*	ref	This association is used to establish hierarchies of hw elements. Note that one particular HwElement can be target of this association only once. I.e. multiple instantiation of the same HwElement is not supported (at any hierarchy level). **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=systemDesignTime xml.sequenceOffset=70

Table E.14: HwElement

Class	**Identifiable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	ARObject,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
desc	MultiLanguage OverviewParagr aph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
category	CategoryString	0..1	attr	This element assigns a category to the parent element. The category is intended to specialize the usage and/or the content identifiable object. Such a specialization may also impose particular semantic constraints on the entire substructure (not only the identifiable itself). **Tags:** xml.sequenceOffset=-50
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:** xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
introductio n	Documentation Block	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". **Tags:** xml.attribute=true

Table E.15: Identifiable

Primitive	**Identifier**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
Note	An Identifier is a string with a number of constraints on its appearance, satisfying the requirements typical programming languages define for their Identifiers. This datatype represents a string, that can be used as a c-Identifier. It needs to start with a letter, may consist of letters, digits and underscore. It shall not have two consecutive underscores (to support subsequent name mangling based on "__"). **Tags:** xml.xsd.customType=IDENTIFIER; xml.xsd.maxLength=128; xml.xsd.pattern=[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?; xml.xsd.type=string			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
namePattern	String	0..1	attr	This attribute represents a pattern which shall be used to define the value of the identifier if the identifier in question is part of a blueprint. For more details refer to TPS_StandardizationTemplate. **Tags:** xml.attribute=true

Table E.16: Identifier

Class	**MlFormula**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Documentation::BlockElements::Formula			
Note	This meta-class represents the ability to express a formula in a documentation. The formula can be expressed by various means. If more than one representation is available, they need to be consistent. The rendering system can use the representation which is most appropriate.			
Base	ARObject,DocumentViewSelectable,Paginateable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
formulaCaption	Caption	0..1	aggr	This element specifies the identification or heading of a formula. **Tags:** xml.sequenceOffset=20
genericMath	MultiLanguagePlainText	0..1	aggr	this rpresents the semantic and mathematical descriptions which are processed by a math-processor. **Tags:** xml.sequenceOffset=80
lGraphic	LGraphic	*	aggr	This represents a formula as an embedded figure. **Tags:** xml.roleWrapperElement=false; xml.sequenceOffset=30
texMath	MultiLanguagePlainText	0..1	aggr	this is the TeX representation of TeX formula. A TeX formula can be processed by a TeX or a LaTeX processor. **Tags:** xml.sequenceOffset=60
verbatim	MultiLanguageVerbatim	0..1	aggr	this represents a formula using only text and white-space. It can be used to denote the formula in a kind of pseudo code or whatever appears approprate. **Tags:** xml.sequenceOffset=50

Table E.17: MlFormula

Class	**NPdu**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	This is a Pdu of the Transport Layer. The main purpose of the TP Layer is to segment and reassemble IPdus. **Tags:** atp.recommendedPackage=Pdus			
Base	ARObject,CollectableElement,FibexElement,IPdu,Identifiable,MultilanguageReferrable,PackageableElement,Pdu,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table E.18: NPdu

Class	**NmPdu**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Network Management Pdu **Tags:** atp.recommendedPackage=Pdus			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Pdu,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
iSignalToIPduMapping	ISignalToIPduMapping	*	aggr	This optional aggregation is used to describe NmUserData that is transmitted in the NmPdu. The counting of the startPosition starts at the beginning of the NmPdu regardless whether Cbv or Nid are used.
nmDataInformation	Boolean	0..1	attr	Defines if the Pdu contains NM Data. If the NmPdu does not aggregate any ISignalToIPduMappings it still may contain UserData that is set via Nm_SetUserData(). If the ISignalToIPduMapping exists then the nmDataInformation attribute shall be ignored.
nmVoteInformation	Boolean	0..1	attr	Defines if the Pdu contains NM Vote information.
unusedBitPattern	Integer	0..1	attr	AUTOSAR COM is filling not used areas of an Pdu with this bit-pattern. This attribute can only be used if the nmDataInformation attribute is set to true.

Table E.19: NmPdu

Class	≪atpMixedString≫ **NumericalValueVariationPoint**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling::AttributeValueVariation Points			
Note	This class represents an attribute value variation point for Numerical attributes. Note that this class might be used in the extended meta-model only.			
Base	ARObject,AbstractNumericalVariationPoint,AttributeValueVariationPoint,Formula Expression,SwSystemconstDependentFormula			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table E.20: NumericalValueVariationPoint

Class	**PackageableElement (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::ARPackage			
Note	This meta-class specifies the ability to be a member of an AUTOSAR package.			
Base	ARObject,CollectableElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
–	–	–	–	–

Table E.21: PackageableElement

Class	**Pdu (abstract)**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Collection of all Pdus that can be routed through a bus interface.			
Base	ARObject,CollectableElement,FibexElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
length	Integer	0..1	attr	Pdu length in bytes. In case of dynamic length IPdus (containing a dynamical length signal), this value indicates the maximum data length. It should be noted that in former AUTOSAR releases (Rel 2.1, Rel 3.0, Rel 3.1, Rel 4.0 Rev. 1) this parameter was defined in bits. The Pdu length of zero bytes is allowed.
metaDataLength	PositiveInteger	0..1	attr	Number of additional bytes of MetaData in the PDU data field. The MetaData contains auxiliary information for the PDU, e.g. the CAN ID.

Table E.22: Pdu

Class	**PduTriggering**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The PduTriggering describes on which channel the IPdu is transmitted. The Pdu routing by the PduR is only allowed for "IPdus" and not for NmPdus and XcpPdus. Depending on its relation to entities such channels and clusters it can be unambiguously deduced whether a fan-out is handled by the Pdu router or the Bus Interface. If the fan-out is specified between different clusters it shall be handled by the Pdu Router. If the fan-out is specified between different channels of the same cluster it shall be handled by the Bus Interface.			
Base	ARObject,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
iPdu	Pdu	1	ref	Reference to the Pdu for which the PduTriggering is defined. One I-Pdu can be triggered on different channels (PduR fan-out). The Pdu routing by the PduR is only allowed for "IPdus" and not for NmPdus and XcpPdus. Nevertheless is the reference to the Pdu element necessary since the PduTriggering element is also used to specify the sending and receiving connections to EcuPorts.
iPduPort	IPduPort	*	ref	References to the IPduPort on every ECU of the system which sends and/or receives the I-PDU. References for both the sender and the receiver side shall be included when the system is completely defined.

Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
iSignalTriggering	ISignalTriggering	*	ref	This reference provides the relationship to the ISignalTriggerings that are implemented by the PduTriggering. The reference is optional since no ISignalTriggering can be defined for DCM and Multiplexed Pdus. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=postBuild

Table E.23: PduTriggering

Class	**PostBuildVariantCriterion**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies one particular PostBuildVariantSelector. **Tags:** atp.recommendedPackage=PostBuildVariantCriterions			
Base	ARElement,ARObject,AtpDefinition,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
compuMethod	CompuMethod	1	ref	The compuMethod specifies the possible values for the variant criterion serving as an enumerator.

Table E.24: PostBuildVariantCriterion

Class	**PostBuildVariantCriterionValue**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This class specifies a the value which must be assigned to a particular variant criterion in order to bind the variation point. If multiple criterion/value pairs are specified, they all must must match to bind the variation point.			
Base	ARObject			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**
annotation	Annotation	*	aggr	This provides the ability to add information why the value is set like it is. **Tags:** xml.sequenceOffset=30
value	Integer	1	attr	This is the particular value of the post-build variant criterion. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20
variantCriterion	PostBuildVariantCriterion	1	ref	This association selects the variant criterion whose value is specified. **Tags:** xml.sequenceOffset=10

Table E.25: PostBuildVariantCriterionValue

Class	**PredefinedVariant**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This specifies one predefined variant. It is characterized by the union of all system constant values and post-build variant criterion values aggregated within all referenced system constant value sets and post build variant criterion value sets plus the value sets of the included variants. **Tags:** atp.recommendedPackage=PredefinedVariants			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
includedVariant	PredefinedVariant	*	ref	The associated variants are considered part of this PredefinedVariant. This means the settings of the included variants are included in the settings of the referencing PredefinedVariant. Nevertheless the included variants might be included in several predefined variants.
postBuildVariantCriterionValueSet	PostBuildVariantCriterionValueSet	*	ref	This is the postBuildVariantCriterionValueSet contributing to the predefinded variant.
swSystemconstantValueSet	SwSystemconstantValueSet	*	ref	This ist the set of Systemconstant Values contributing to the predefined variant.

Table E.26: PredefinedVariant

Class	**Referrable (abstract)**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
shortName	Identifier	1	ref	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Tags:** xml.enforceMinMultiplicity=true; xml.sequenceOffset=-100

Table E.27: Referrable

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 E.28: ServiceNeeds

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 E.29: ServiceSwComponentType

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 E.30: SwComponentPrototype

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,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
mapping	PortInterfaceMa pping	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 E.31: SwConnector

Class	**SwSystemconstantValueSet**			
Package	M2::AUTOSARTemplates::GenericStructure::VariantHandling			
Note	This meta-class represents the ability to specify a set of system constant values. **Tags:** atp.recommendedPackage=SwSystemconstantValueSets			
Base	ARElement,ARObject,CollectableElement,Identifiable,Multilanguage Referrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
swSystem constantVa lue	SwSystemconst Value	*	aggr	This is one particular value of a system constant.

Table E.32: SwSystemconstantValueSet

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 E.33: UnitGroup

Class	**UserDefinedPdu**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	UserDefinedPdu allows to describe PDU-based communication over Complex Drivers. If a new BSW module is added above the BusIf (e.g. a new Nm module) then this Pdu element shall be used to describe the communication. **Tags:** atp.recommendedPackage=Pdus			
Base	ARObject,CollectableElement,FibexElement,Identifiable,Multilanguage Referrable,PackageableElement,Pdu,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
cddType	String	0..1	attr	This attribute defines the CDD that transmits or receives the UserDefinedIPdu. If several CDDs are defined this attribute is used to distinguish between them.

Table E.34: UserDefinedPdu

Primitive	**VerbatimString**			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Primitive Types			
Note	This primitive represents a string in which white-space needs to be preserved. **Tags:** xml.xsd.customType=VERBATIM-STRING; xml.xsd.type=string; xml.xsd.white Space=preserve			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
xmlSpace	XmlSpaceEnum	0..1	attr	This attribute is used to signal an intention that in that element, white space should be preserved by applications. It is defined according to xml:space as declared by W3C. **Tags:** atp.Status=shallBecomeMandatory xml.attribute=true; xml.attributeRef=true; xml.name=space; xml.nsPrefix=xml

Table E.35: VerbatimString