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

Document Version	3.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Description
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

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

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

Table of Contents

References

[1] Methodology
AUTOSAR_TR_Methodology.pdf

[2] Glossary
AUTOSAR_TR_Glossary.pdf

[3] Requirements on ECU Configuration
AUTOSAR_RS_ECUConfiguration.pdf

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

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

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[7] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules.pdf

[8] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[9] System Template
AUTOSAR_TPS_SystemTemplate.pdf

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

[11] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate.pdf

[12] Specification of ECU Configuration Parameters (XML)
AUTOSAR_MOD_ECUConfigurationParameters.arxml

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

[14] Meta Model
AUTOSAR_MMOD_MetaModel.eap

[15] XML Schema 1.1
http://www.w3.org/XML/Schema

[16] Specification of ECU Configuration Parameters (XML)
AUTOSAR_MOD_ECUConfigurationParameters.pdf

[17] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[18] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

1 Introduction

According to AUTOSAR Methodology the configuration process contains 4 steps that will be discussed in chapter 2 in more detail:

- Configure System
- Extract ECU-Specific Information
- Configure ECU
- Generate Executable

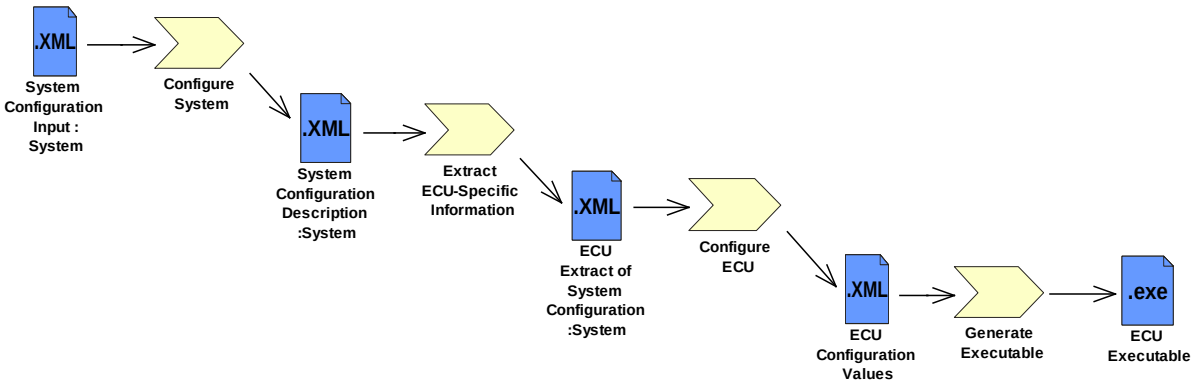

Figure 1.1: AUTOSAR Methodology Overview (from [1])

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. This ECU extract is the basis for the ECU Configuration step.

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).

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 3.3).

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 3.4). 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 [2].

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 Requirements Traceability

Following table references the requirements specified in AUTOSAR ECU Configuration Requirements [3] and in General Requirements on Basic Software Modules [4] and links to the "ecuc_sws" fulfillments of these.

Requirement	Description	Satisfied by
[BSW00344]	Reference to link time configuration	not applicable (BSW implementation issue)
[BSW00345]	Pre-compile time configuration	not applicable (BSW implementation issue)
[BSW00380]	Separate C-File for configuration parameters	not applicable (BSW implementation issue)
[BSW00381]	Separate configuration header file for pre-compile time parameters	not applicable (BSW implementation issue)
[BSW00382]	Not-used configuration elements need to be listed	not applicable (requirement on the BSW Module Description [5])
[BSW00383]	List dependencies of configuration files	not applicable (requirement on the BSW Module Description [5])
[BSW00384]	List dependencies to other modules	not applicable (requirement on the BSW Module Description [5])
[BSW00385]	List possible error notifications	not applicable (requirement on the BSW Module Description [5])
[BSW00386]	Configuration for detecting an error	not applicable (requirement on ECU Configuration Parameters and BSW Module Description [5])
[BSW00387]	Specify the configuration class of callback function	[ecuc_sws_2016]
[BSW00388]	Introduce containers	[ecuc_sws_2006]
[BSW00389]	Containers shall have names	[ecuc_sws_2043]
[BSW00390]	Parameter content shall be unique within the module	not applicable (requirement on the BSW SWS)

Requirement	Description	Satisfied by
[BSW00391]	Parameters shall have unique names	[ecuc_sws_2043] [ecuc_sws_2014]
[BSW00392]	Parameters shall have a type	[ecuc_sws_2014]
[BSW00393]	Parameters shall have a range	[ecuc_sws_2027] [ecuc_sws_2028]
[BSW00394]	Specify the scope of the parameters	not applicable (requirement on the BSW SWS) [ecuc_sws_6002]
[BSW00395]	List the required parameters (per parameter)	[ecuc_sws_2039]
[BSW00396]	Configuration classes	[ecuc_sws_2016]
[BSW00397]	Pre-compile time parameters	[ecuc_sws_2017] [ecuc_sws_1031]
[BSW00398]	Link time parameters	[ecuc_sws_2018] [ecuc_sws_1032]
[BSW00399]	Loadable Post-build time parameters	[ecuc_sws_4006] [ecuc_sws_4000] [ecuc_sws_4005]
[BSW00400]	Selectable Post-build time parameters	[ecuc_sws_4007]
[BSW00401]	Documentation of multiple instances of configuration parameters	not applicable (requirement on the BSW SWS)
[BSW00402]	Published information	not applicable (requirement on the BSW Module Description [5])
[BSW00404]	Reference to post-build time configuration	not applicable (BSW implementation issue)
[BSW00405]	Reference to multiple configuration sets	not applicable (BSW implementation issue)
[BSW00408]	Configuration parameter naming convention	requirement on ECU Configuration Parameters
[BSW00410]	Compiler switches shall have defined values	not applicable (BSW implementation issue)
[BSW00411]	Get version info keyword	not applicable (BSW implementation issue)
[BSW00412]	Separate H-File for configuration parameters	not applicable (BSW implementation issue)
[BSW00413]	Accessing instances of BSW modules	requirement on ECU Configuration Parameters
[BSW00414]	Parameter of init function	not applicable (BSW implementation issue)
[BSW00415]	User dependent include files	not applicable (BSW implementation issue)
[BSW00416]	Sequence of Initialization	requirement on the BSW SWS
[BSW00417]	Reporting of Error Events by Non-Basic Software	not applicable (non-BSW implementation issue)
[BSW00419]	Separate C-Files for pre-compile time configuration parameters	not applicable (BSW implementation issue)
[BSW00420]	Production relevant error event rate detection	not applicable (requirement on the BSW SWS)
[BSW00421]	Reporting of production relevant error events	not applicable (requirement on the BSW SWS and BSW Module Description [5])
[BSW00422]	Debouncing of production relevant error status	not applicable (requirement on the BSW SWS)
[BSW00423]	Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	not applicable (requirement on BSW Module Description [5])
[BSW00424]	BSW main processing function task allocation	not applicable (requirement on BSW scheduler module)

Requirement	Description	Satisfied by
[BSW00425]	Trigger conditions for schedulable objects	not applicable (requirement on BSW Module Description [5])
[BSW00426]	Exclusive areas in BSW modules	not applicable (requirement on BSW Module Description [5] and BSW implementation issue)
[BSW00427]	ISR description for BSW modules	not applicable (requirement on BSW Module Description [5])
[BSW00428]	Execution order dependencies of main processing functions	not applicable (requirement on BSW Module Description [5])
[BSW00429]	Restricted BSW OS functionality access	not applicable (BSW implementation issue)
[BSW00431]	The BSW Scheduler module implements task bodies	not applicable (requirement on BSW scheduler module)
[BSW00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	not applicable (BSW implementation issue)
[BSW00433]	Calling of main processing functions	not applicable (BSW implementation issue)
[BSW00434]	The Schedule Module shall provide an API for exclusive areas	not applicable (BSW implementation issue)
[BSW159]	Tool-based configuration	[ecuc_sws_1030]
[BSW167]	Static configuration checking	[ecuc_sws_6038]
[BSW170]	Data for reconfiguration of AUTOSAR SW-Components	not applicable (requirement on the Software-Component Template [6])
[BSW171]	Configurability of optional functionality	[ecuc_sws_2009] [ecuc_sws_6007]
[ECUC0002]	Support of vendor-specific ECU Configuration Parameters	[ecuc_sws_1029] [ecuc_sws_1001] [ecuc_sws_6007] [ecuc_sws_1011] [ecuc_sws_1013] [ecuc_sws_1014] [ecuc_sws_1015] [ecuc_sws_5002] [ecuc_sws_5003]
[ECUC0008]	Definition of post-build time changeable configuration of BSW	[ecuc_sws_2132] [ecuc_sws_2019] [ecuc_sws_4006] [ecuc_sws_4000]
[ECUC0012]	One description mechanism for different configuration classes	[ecuc_sws_2016]
[ECUC0015]	Configuration of multiple instances of BSW modules	[ecuc_sws_2059] [ecuc_sws_2008]
[ECUC0016]	Execution order of runnable entities	not applicable (requirement on ECU Configuration Parameters)
[ECUC0018]	Extension handling	fulfilled by the Model Persistence Rules for XML [7]
[ECUC0021]	Select Application SW Component and BSW module implementation	[ecuc_sws_1036] [ecuc_sws_1029]
[ECUC0025]	Compatible with iterative design	[ecuc_sws_1027] [ecuc_sws_4000]
[ECUC0029]	Identify mechanisms not criteria	proceeding was followed in the development of the ECU Configuration Specification

Requirement	Description	Satisfied by
[ECUC0030]	Clarify configuration terminology	terms defined in AUTOSAR Glossary [2]
[ECUC0032]	ECU Configuration Description shall be the root for the whole configuration information of an ECU	[ecuc_sws_2003] [ecuc_sws_1029]
[ECUC0039]	Support configuration of BSW	requirement on ECU Configuration Parameters[ecuc_sws_1029]
[ECUC0040]	Support configuration of RTE	requirement on ECU Configuration Parameters[ecuc_sws_1029]
[ECUC0041]	Support AUTOSAR SW Component Integration	requirement on ECU Configuration Parameters
[ECUC0043]	Duplication free description	[ecuc_sws_1027] [ecuc_sws_1028] [ecuc_sws_2124]
[ECUC0046]	Support definition of configuration class	[ecuc_sws_2016]
[ECUC0047]	Pre-compile time configuration of BSW	[ecuc_sws_2017] [ecuc_sws_1031]
[ECUC0048]	Link time configuration of BSW	[ecuc_sws_2018] [ecuc_sws_1032]
[ECUC0049]	ECU Configuration description shall be tool processable	[ecuc_sws_2001] [ecuc_sws_1030]
[ECUC0050]	Specify ECU Configuration Parameter Definition	[ecuc_sws_2065]
[ECUC0055]	Support mandatory and optional configuration parameters	[ecuc_sws_3011] [ecuc_sws_3010] [ecuc_sws_3030] [ecuc_sws_2008] [ecuc_sws_2009] [ecuc_sws_6007]
[ECUC0065]	Development according to the AUTOSAR Generic Structure Template document	[ecuc_sws_2000]
[ECUC0066]	Transformation of ECUC modeling according to the AUTOSAR Model Persistence Rules for XML	[ecuc_sws_2001]
[ECUC0070]	Support mandatory and optional containers	[ecuc_sws_2008] [ecuc_sws_2009] [ecuc_sws_6007]
[ECUC0071]	Support for Generic Configuration Editor	[ecuc_sws_2124]
[ECUC0072]	Support for referencing from dependent containers	[ecuc_sws_3027] [ecuc_sws_3033] [ecuc_sws_2039]
[ECUC0073]	Support Service Configuration of AUTOSAR SW Components	[ecuc_sws_2087]
[ECUC0074]	Support Sequential ECU Configuration	[ecuc_sws_2124]
[ECUC0076]	Support the configuration of which AUTOSAR Services are available on a specific ECU	[ecuc_sws_6014]
[ECUC0078]	Variable existence of container	[ecuc_sws_2119] [ecuc_sws_2120]
[ECUC0079]	Variable existence of value	[ecuc_sws_2121] [ecuc_sws_2122]
[ECUC0080]	Variable value	
[ECUC0082]	Variable lower and upper multiplicity in ECU Configuration Parameter definition	[ecuc_sws_6016] [ecuc_sws_2110] [ecuc_sws_6009] [ecuc_sws_6010] [ecuc_sws_6013]
[ECUC0083]	Variable default value in ECU Configuration Parameter definition	[ecuc_sws_2111] [ecuc_sws_2114] [ecuc_sws_2115] [ecuc_sws_2112]
[ECUC0084]	Variable min and max ranges in ECU Configuration Parameter definition	[ecuc_sws_2116] [ecuc_sws_2117]

2 ECU Configuration Methodology

ECU Configuration is one step in the overall AUTOSAR methodology, which is described in [1]. Figure 1.1 already introduced in chapter 1 is taken from that document and shows the most abstract view of the methodology. In this document, the activities regarding configuring an ECU and generate the configuration data will be defined in more detail than provided in [1]. To understand this chapter, the reader should be familiar with [1].

2.1 Notation used

Figure 1.1 and all other figures taken from the AUTOSAR Methodology [1] use a formal notation called SPEM (Software Process Engineering Meta-Model), explained in detail in [1]. The SPEM elements used in this document are

- *Work products* (blue document shaped elements),
- *Activities* (block arrows) and
- *Guidances* (set square) to depict tools, attached to the activity they support by a dashed line.

The flow of work products is depicted by solid lines with arrow heads, pointing in the direction of the work flow. Dependencies are depicted by dashed lines with arrow heads, pointing from the dependent element to the element it depends on. Compositions are depicted by a solid line with a solid diamond on the end of the aggregating element.

2.2 Inputs to ECU Configuration

[ecuc_sws_1036] ECU Configuration has two input sources. First of all, all configuration that must be agreed across ECUs is defined in the System Configuration, which results in a `System Configuration Description` (and the resulting `ECU Extract of the System Configuration` for the individual ECUs). Secondly, the ECU BSW is built using BSW modules. The specifics of these module implementation are defined in the `BSW Module Descriptions` (not shown in figure 1.1, see figure 2.7). The latter is described in [8] in more detail. The concept of the ECU extract is depicted below:

ECU Extract of System Description

ECU Configuration can only be started once a plausible `System Configuration Description` and the corresponding ECU extract has been generated (see figure 1.1). Details on the `System Configuration Description` can be found in [9]. The `System Configuration Description` contains all relevant system-wide configuration, such as

- ECUs present in the system
- Communication systems interconnecting those ECUs and their configuration
- Communication matrices (frames sent and received) for those communication systems
- Definition of Software Components with their ports and interfaces and connections (defined in the SWC Description and referenced in the `System Configuration Description`).
- Mapping of SWCs to ECUs

The `ECU Extract of the System Configuration` is a description in the same format as the `System Configuration Description`, but with only those elements included that are relevant for the configuration of one specific ECU.

2.3 ECU Configuration

ECU Configuration can be broken down into three activities, as shown in figure 2.1:

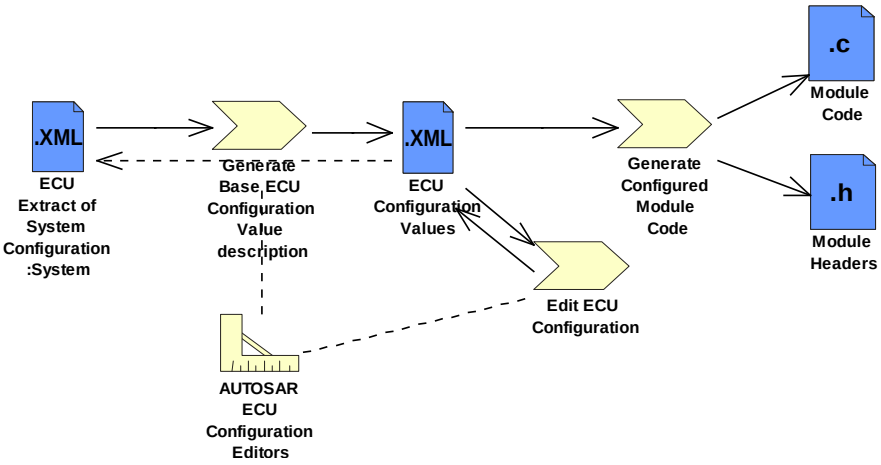

Figure 2.1: ECU Configuration broken down into three sub-activities

- `Generate Base ECU Configuration Value description`
- `Edit ECU Configuration`
- `Generate Configured Module Code`

[ecuc_sws_1027] All three activities use a single work product, the `ECU Configuration Values`, which contains (i.e. references) all the configuration information for all BSW modules on the ECU. In order to better understand the three different activities an introduction to configuration classes is essential. In a real implementation of a BSW module all configuration parameters are most likely not the same configuration class. I.e it will be a mix of parameters with different configuration classes within a BSW module.

These three activities are introduced in detail in later sections, but first is an introduction to ECU Configuration Value description and configuration classes.

2.3.1 ECU Configuration Value description

The `ECU Extract of System Configuration` only defines the configuration elements that must be agreed between ECUs. In order to generate a working executable that runs on the ECU, much more configuration information must be provided.

The remaining part of the configuration is about configuring all BSW modules within the ECU. Typical BSW modules within an ECU can be: RTE, Com, Can, OS, NVRAM etc. There are also dependencies between BSW modules to consider when configuring the ECU. When the configuration is done, the generation of configuration data takes place. I.e. there are both configuration editors and configuration generators involved in the process. In order to obtain consistency within the overall configuration of the ECU, AUTOSAR has defined a single format, the ECU Configuration Value description to be used for all BSW modules within an ECU. Both configuration editors and configuration generators are working toward ECU Configuration Value descriptions.

[ecuc_sws_1028] This one description (`ECU Configuration Value description`) collects the complete configuration of BSW modules in a single ECU. Each module generator may then extract the subset of configuration data it needs from that single format.

2.3.2 Introduction To Configuration Classes

The development of BSW modules involve the following development cycles: compiling, linking and downloading of the executable to ECU memory. Configuration of parameters can be done in any of these process-steps: pre-compile time, link time or even post-build time. According to the process-step that does the configuration of parameters, the configuration classes are categorized as below

- pre-compile time
- link time
- post-build time

The configuration in different process-steps has some consequences for the handling of ECU configuration parameters. If a configuration parameter is defined as pre-compile time, after compilation this configuration parameter can not be changed any more. Or if a configuration parameter is defined at post-build time the configuration parameter has to be stored at a known memory location. Also, the format in which the BSW module is delivered determines in what way parameters are changeable. A source code delivery or an object code delivery of a BSW module has different degrees of freedom regarding the configuration.

The configuration class of a parameter is typically not fixed in the standardized parameter definition since several variants are possible. However once the module is implemented the configuration class for each of the parameters is fixed in that implementation. Choosing the right configuration class from the available variants is depending on the type of application and the design decisions taken by the module implementer. Different configuration classes can be combined within one module. For example, for post-build time configurable BSW implementations only a subset of the parameters might be configurable post-build time. Some parameters might be configured as pre-compile time or link time.

File formats used for describing the configuration classes:

- `.arxml` (An xml file standardized by AUTOSAR.)
- `.exe` (An executable that can be downloaded to an ECU.)
- `.hex` (A binary file that can be downloaded to an ECU , but it can not execute by its own.)
- `.c` (A C-source file containing either source code or configuration data.)
- `.h` (A header file for either source code or configuration data.)
- `.obj` (A object file for either source code or configuration data.)

2.3.2.1 Configuration Class pre-compile time

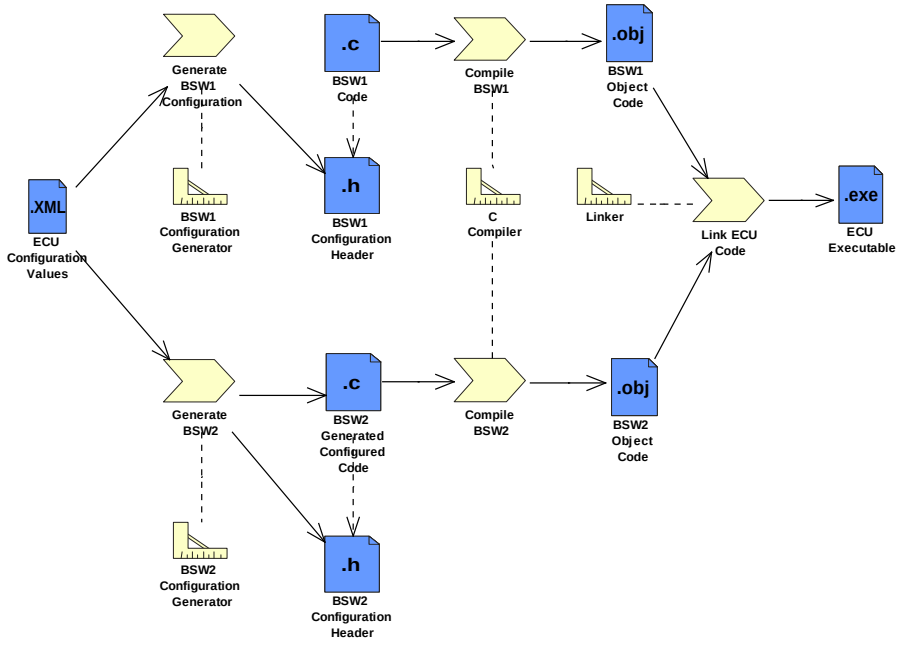

Figure 2.2: Pre-compile time configuration chain

[ecuc_sws_1031] This type of configuration is a standalone configuration done before compiling the source code. That means parameter values for those configurable elements are selected before compiling and will be effective after compilation time. The value of the configurable parameter is decided in earlier stage of software development process and any changes in the parameter value calls for a re-compilation. The contents of pre-compile time parameters can not be changed at the subsequent development steps like link time or post-build time.

Example BSW1 in figure 2.2 shows one possible approach to implement pre-compile time parameters. Configurable parameter values will be kept in a configuration header file and compiled along with the module source file, which is not touched by the `BSW1 Configuration Generator`. Example BSW2 in figure 2.2 shows an alternative approach, in which the `BSW2 Configuration Generator` generates the complete, configuration-specific code. Both approaches are equally valid.

Whenever the decision of parameter value must be taken before the selection of other dependable parameters, pre-compile time configuration is the right choice. For example, the algorithm choice for CRC initial checksum parameter is based on the selection of CRC type (CRC16 or CRC32). When CRC16 is selected, there will be increase in processing time but reduction in memory usage. Whereas when CRC32 is selected, there will be decrease in processing time but increase in memory usage. The correct

choice should be made by the implementer before compilation of source code based on the requirement and resource availability. [1]

Sample cases where pre-compile time configuration can be adopted are listed below:

- Configure the number of memory tables and block descriptor table of NVRAM manager.
- Enable the macro for the development error tracing of the software modules.

[1] It would be possible to have both CRC16 and CRC32 available for link-time and post-build time, but that would require additional resources which can be spared if the decision for one type is taken during pre-compile time

2.3.2.2 Configuration Class link time

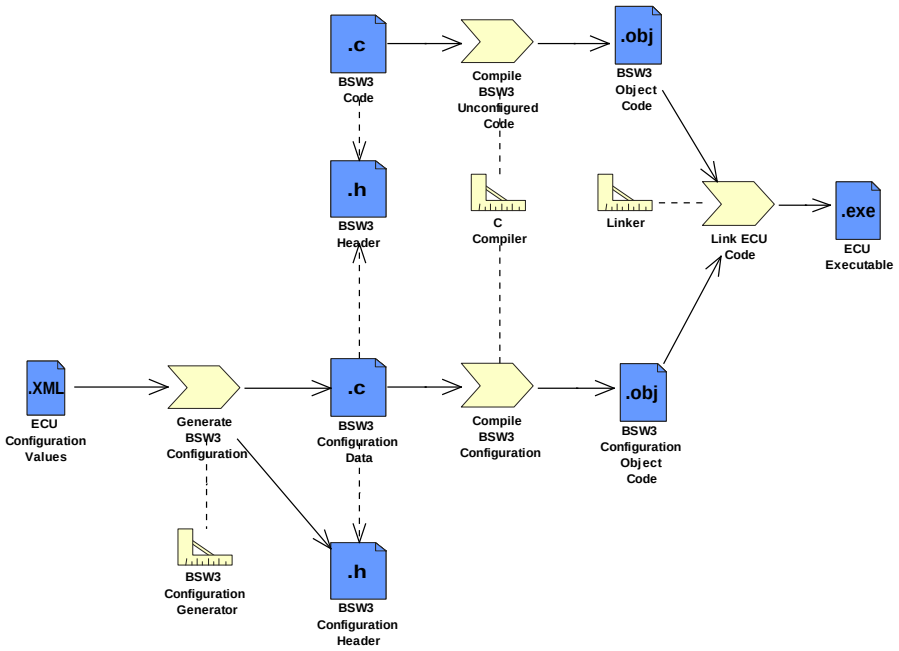

Figure 2.3: Link time configuration chain

[ecuc_sws_1032] This type of configuration is done for the BSW module during link time. That means the object code of the BSW module receives parts of its configuration from another object code file. Link time parameters are typically used when delivering object code to the integrator.

This configuration class provides a modular approach to the configuration process. A separate module will handle the configuration details and those parameter values will be made available to the other modules during the linking process.

In figure 2.3 the configuration parameter data is defined in a common header file (`BSW3 Header`) and included by both module source file (`BSW3 Code`) and module configuration source file (`BSW3 Configuration Data`). The module source file needs this header file to resolve the references and module configuration source file will need it in order to cross check the declaration of data type against the definition. Both module source file and module configuration source file are compiled separately and generates module object file and module configuration object file respectively. During the linking process, the configuration data will be available to module object file by resolving the external references. When the value of configuration parameters is to be changed the module configuration object file needs to be replaced by the one containing the new parameters.

Sample cases where Link time configuration can be adopted are listed below:

- Initial value and invalid value of signal.

- Unique channel identifier configured for the respective instance of the Network Management.
- Logical handle of CAN network.
- Identifier and type of Hardware Reception Handle and Hardware Transmission Handle for CAN interface.
- Definition of ComFilterAlgorithm.
- COM callback function to indicate RTE about the reception of an invalidated signal.

2.3.2.3 Post-build Configuration

There are two kinds of post-build configuration defined. They are:

- Post-build time loadable.
- Post-build time selectable.

For the methodology and the ECU build it is essential to distinguish between `PostBuildLoadable` and `PostBuildSelectable`. That influences the allocation of configuration data structures to the memory of ECU and it has an effect on the ECU startup behavior.

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

2.3.2.3.1 Configuration Class post-build time loadable

[ecuc_sws_4006] This type of configuration is possible after building the BSW module or the ECU software. The BSW module gets the parameters of its configuration by downloading a separate file to the ECU memory separately, avoiding a re-compilation and re-build of the BSW module.

In figure 2.4 one approach of post-build time loadable is described.

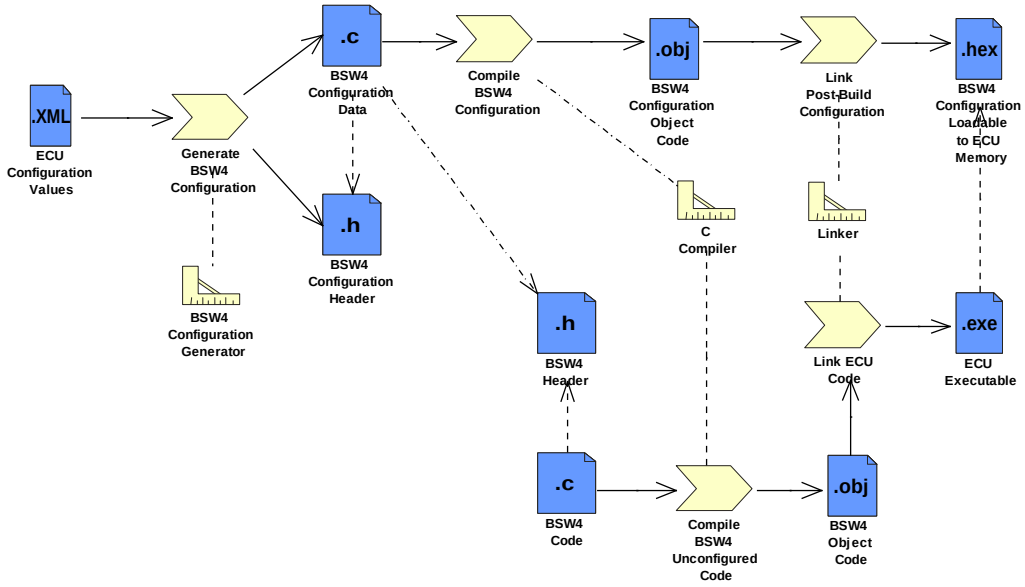

Figure 2.4: Post-build time loadable configuration chain

In order to make the post-build time loadable re-configuration possible, the re-configurable parameters shall be stored at a known memory location of the ECU memory. An example is shown in figure 2.4. The BSW4 source code (`BSW4 Code`) is compiled and linked independently of its configuration data. The `BSW4 Configuration Generator` generates the configuration data as normal C source code (`BSW4 Configuration Data`) that is compiled and linked independently of the source code. The configuration data, `BSW4 Configuration Loadable to ECU Memory`, is stored at a known memory location and it is possible to exchange the configuration data without replacing the `ECU Executable`.

Another approach of post-build time loadable is shown in figure 2.5.

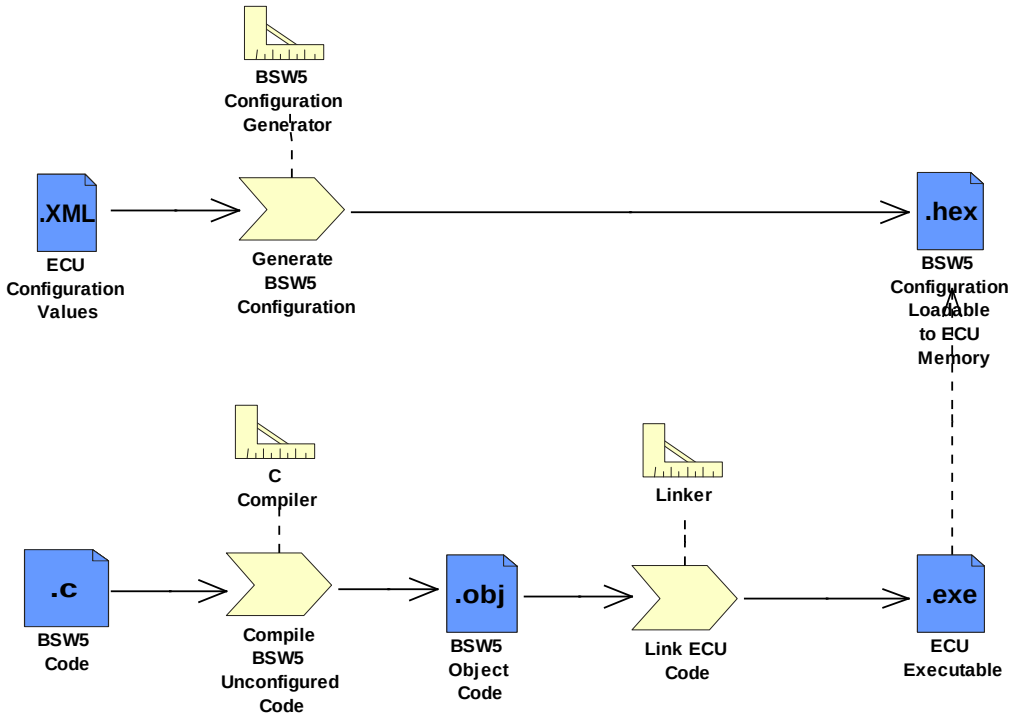

Figure 2.5: Post-build time loadable configuration chain

The difference compared to the other approach is that the `BSW5 Configuration Generator` does perform the tasks performed by the compiler and linker in the prior approach. I.e the `BSW5 Configuration Loadable to ECU Memory` is generated directly from the generator. The configuration data and the executable is still independently exchangeable.

Sample cases where post-build time loadable configuration can be adopted are listed below.

- Identifiers of the CAN frames
- CAN driver baudrate and propagation delay
- COM transmission mode, transmission mode time offset and time period

2.3.2.3.2 Configuration Class post-build time selectable

[ecuc_sws_4007]Post-build time selectable makes it possible to define multiple configuration sets. Which set that will become active is choosen during boot-time. A description of post-build time selectable is shown in figure 2.6.

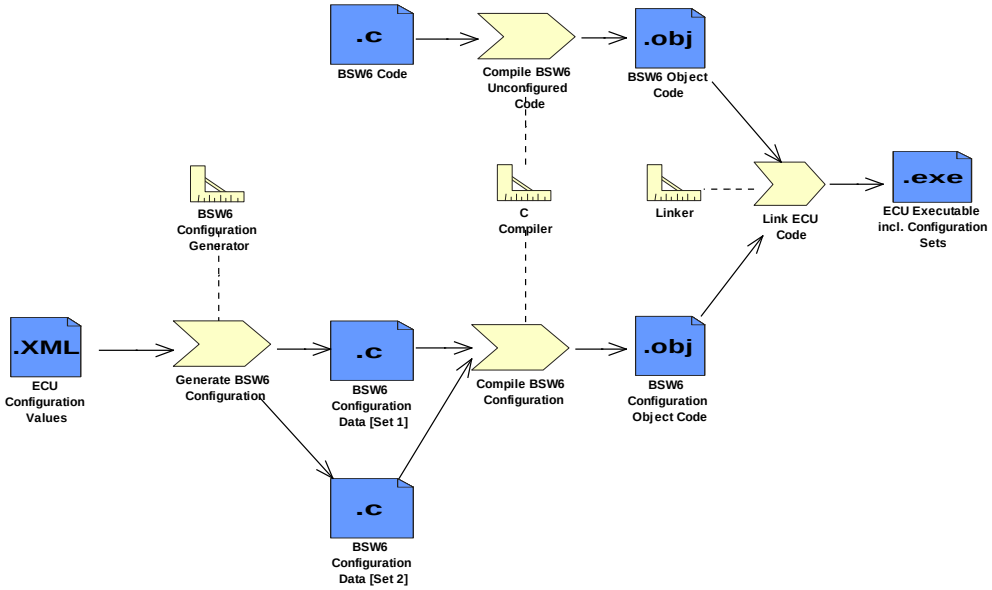

Figure 2.6: Post-build time selectable configuration chain

In the example the `BSW6 Configuration Generator` generates two sets of configuration parameters. The configuration data is compiled and linked together with the source code of the BSW module (`BSW6 Code`). The resulting executable, `ECU Executable incl.Configuration Sets`, includes all configuration sets as well as the source code of the BSW module. I.e. it is not possible to exchange the configuration data without re-building the entire executable. It is only possible to choose between the provided parameter sets without individually changing the values of configuration parameters. This choice is done during module initialization.

2.3.3 Generate Base ECU Configuration Value description

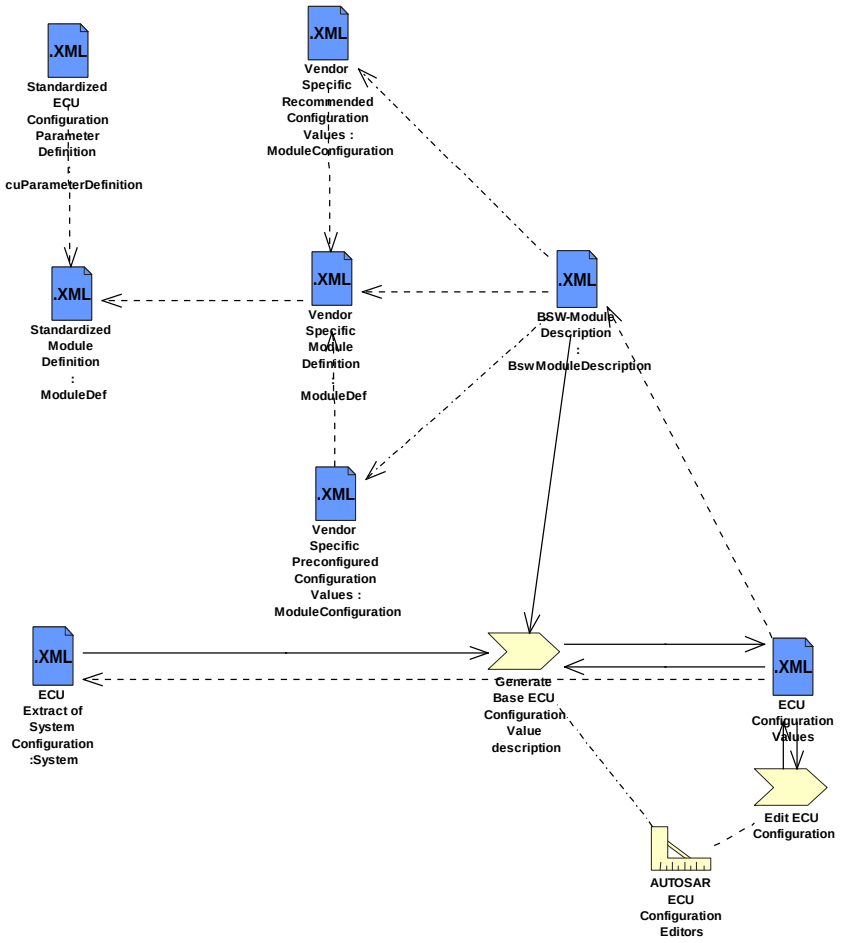

Figure 2.7: Generation of the base ECU Configuration Value description

The first step in the process of ECU configuration is to generate the base `ECU Configuration Value description`.

The `ECU Configuration Value description` contains (i.e. references) the configuration of all BSW modules present on the ECU. The configuration of the different BSW modules is done in different sections of the overall description. The section for a specific BSW module in the base `ECU Configuration Value description` can be generated using the `Vendor Specific ECU Configuration Parameter Definition` (referenced via the `BSW Module Description` BSWMD for that module) and the `ECU Extract of the System Configuration`, as input, see figure 2.7. This generation is a semi-automatic process.

[ecuc_sws_1029] For each BSW module that shall be present in the ECU, the implementation must be chosen. This is done by referencing the BSWMD delivered with the BSW module. The BSWMD defines all configuration parameters, and their structuring in containers, relevant for this specific implementation of the module. This is done in the `Vendor Specific Module Definition`. The rules that must be followed

when building the base ECU Configuration Value description are available in chapter 5.2.

2.3.3.1 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 2.8 *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.

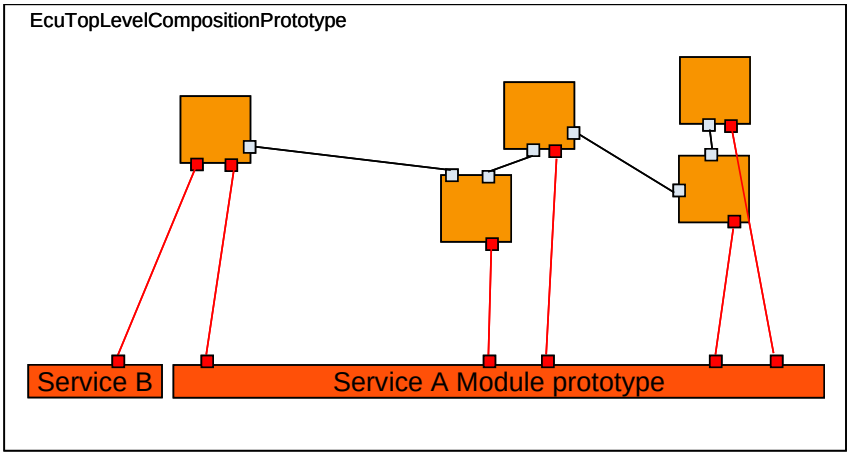

Figure 2.8: Structure of the EcuTopLevelCompositionPrototype introduced in the ECU Configuration

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 2.9. 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.

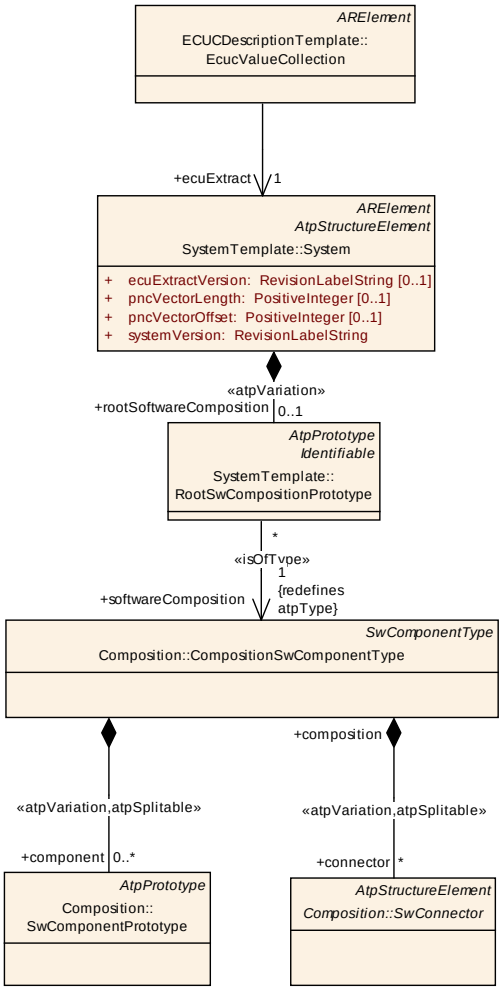

Figure 2.9: Splittable aggregations of `SwComponentPrototype` and `SwConnector`

[ecuc_sws_2087] When generating the AUTOSAR Service SW-Components the actual *service needs*[2] 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.

[ecuc_sws_6014] 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, `SwComponentPrototypes` for all required AUTOSAR Service modules and `AssemblySwConnectors` for the required connections between the Application SW-Component ports and the AUTOSAR Service module's ports.

[2]The *needs* of the Application SW-Components are defined in the SW-Component description in the `ServiceNeeds` section.

2.3.4 Edit ECU Configuration

The second step in the process of ECU configuration is to edit the configuration parameters for all BSW modules.

[ecuc_sws_1030] Once the section for a specific BSW module has been generated in the base `ECU Configuration Value description`, it can be edited with AUTOSAR `ECU Configuration Editors`. Those editors may operate with user interaction, semi automatically or automatically, depending on BSW module and implementation. A straightforward approach editing the `ECU Configuration Value description` is described in figure 2.1.

2.3.4.1 Details in Edit ECU Configuration

Editing the ECU Configuration is a process that has some aspects which put specific requirements on tools and workprocedures. One aspect is the iterative process when editing ECU configuration parameters and another aspect is support for configuration management.

2.3.4.1.1 Iterations within ECU Configuration

What appears clear is that there are likely to be both optimizations and trade-offs to be made between parameters, both within and between BSW modules. The configuration deals with, for example, detailed scheduling information or the configuration data for the needed BSW modules. Hence this is a non-trivial design step and requires complex design algorithms and/or engineering knowledge. ECU Configuration is thus likely to be an iterative process. This iteration will initially be between editors and then, when a plausible ECU Configuration is achieved, code generation may highlight additional changes that require further iteration. It is hoped that the majority of generator-editor iterations will be limited by ensuring that the editor tools are capable of spotting/predicting potential generator errors and ensuring that the engineer corrects them prior to entering generation.

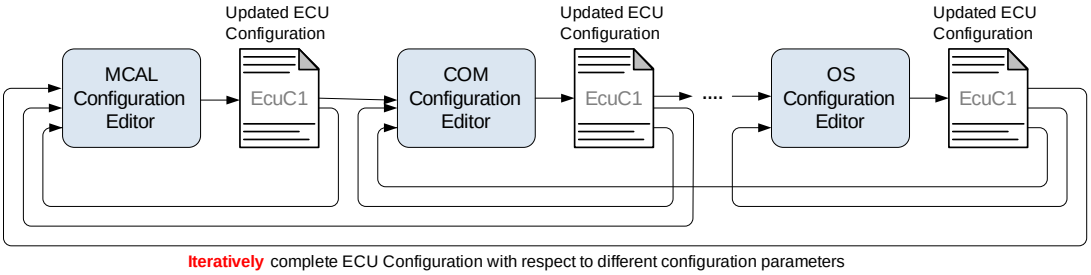

Figure 2.10: Sequential Application of tools

Figure 2.10 shows how a set of custom tools might be used in a chain with iteration in order to achieve a successful ECU Configuration. Tools are sequentially called within a tool chain to create an `ECU Configuration Value description`. Iteration cycles must be implemented by repeated activation of different configuration tools for specific aspects of the BSW. Dependencies between tools, as well as the configuration work flow, might need to be expressed explicitly. Configuration tools are required only to support a single standardized interface, the `ECU Configuration Value Template`.

Tools supporting the methodology and the iterations needed for ECU configuration can be designed based on different strategies. Chapter A.1.1 gives som information about this topic.

Iterations *can* be divided between several organizations due to the fact that parameters within a BSW module are either configured pre-compile time, link time, post-build-select time or post-build-load time.

Typically the following cases apply:

- values of preconfigured pre-compile-time-parameters are set by a Tier2 supplier.
- values of all other pre-compile-time-parameters are set by a Tier 1 supplier, typically according to the requirements of the OEM.
- values of link-time-parameters are set by a Tier 1 supplier, typically according the requirements of the OEM.
- values of postbuild-selectable parameters are set by a Tier 1 supplier, typically according to the requirements of the OEM and delivered as part of the regular ECU-Executable.
- values of postbuild-loadable parameters are set by either a Tier 1 supplier or by the OEM depending on the business model respectively liability issues. However they are delivered as part of a separated Hex-File (called ECU-Executable in Methdololgy).

A description of editing ECU configuration parameters in an iterative manner with several organisations involved is described in figure 2.11.

Configuring of pre-compile time and post-build time parameters occur at different organisations. The methodology supports parallel and iterative configuration activities. A

pre-compile time parameter can affect e.g. a post-build time parameter. The methodology supports description of dependencies between parameters.

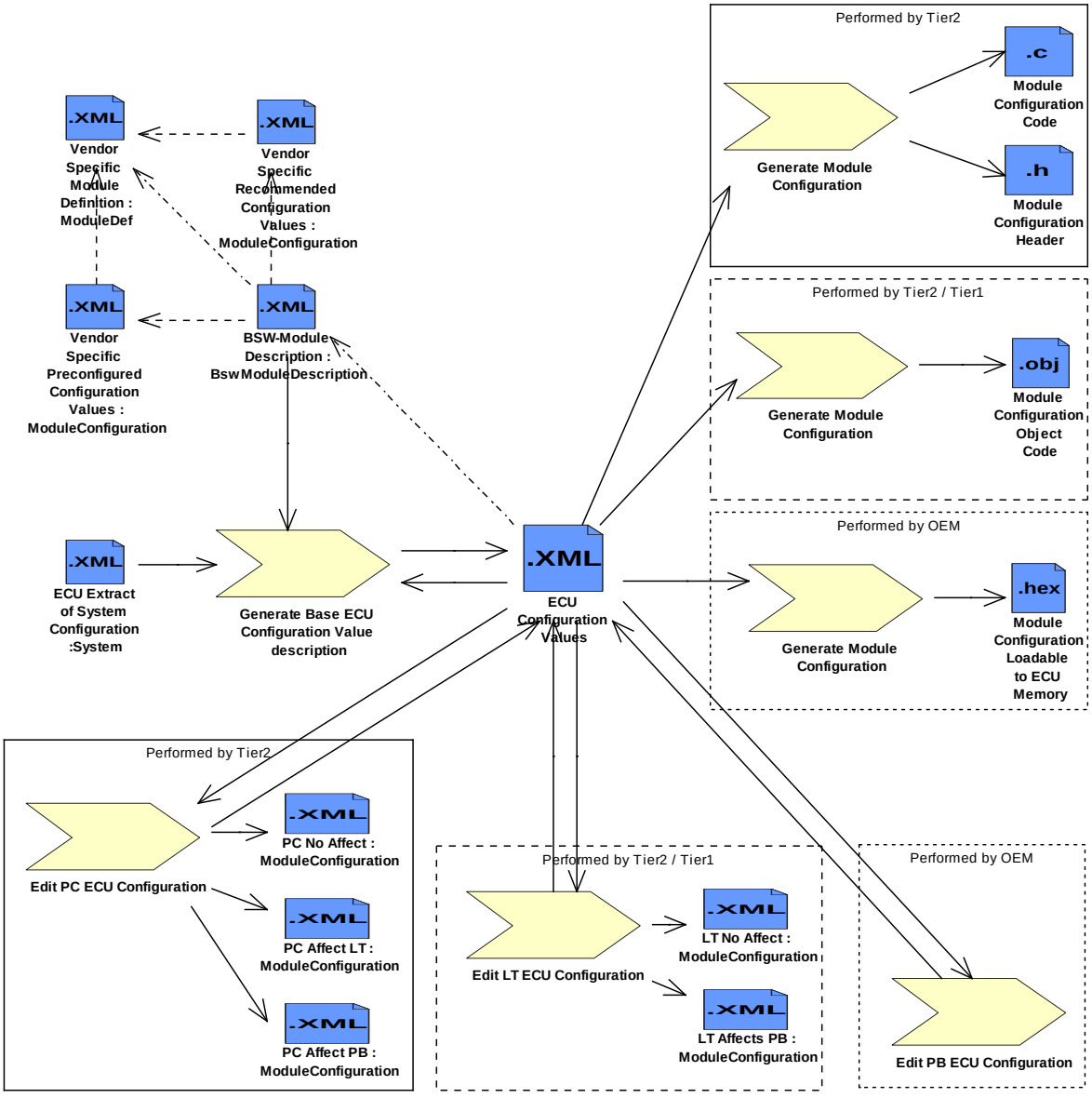

Figure 2.11: Detailed description of ECU configuration

2.3.4.1.2 Affection between parameters

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

The BSWMD (delivered by a Tier2 supplier) contains a description of the entire BSW module, including dependencies between parameters. Each parameter has an attribute that can be assigned one of these values:

- NO-AFFECT (The parameter has no affect on any other parameter)
- PC-AFFECTS-PC (A pre-compile time parameter affecting one or several other pre-compile parameter(s)).
- PC-AFFECTS-LT (A pre-compile time parameter affecting one or several link time parameter(s))
- PC-AFFECTS-PB (A pre-compile time parameter affecting one or several post-build time parameter(s))
- PC-AFFECTS-LT-AND-PB (A pre-compile time parameter affecting one or several link time and post-build time parameter(s))
- LT-AFFECTS-PB (A link time parameter affecting one or several post-build time parameter(s))

In addition it is also possible to list the affected parameters in the BSWMD. The description of dependencies makes it possible to inform about changes that will affect other organisations taking part in the ECU configuration.

[ecuc_sws_4000] In the figure 2.11 there are three activities defined: "'Edit PC ECU Configuration"', "'Edit LT ECU Configuration"' and "'Edit PB ECU Configuration"'. These activities are performed by different organisations. In order to transfer information from one step of the configuration to another, it is possible to generate a file containing parameters that affect another step of the configuration. This is done by i.e. generating the xml-file: "'PC Affect PB"'. The xml-file is a `EcucModuleConfigurationValues` and contains the pre-compile time parameters that has an affect on post-build time parameter(s) for a specific BSW module or cluster of BSW modules. Since the editor is the same in all steps of the configuration of a BSW module or a cluster the "'PC Affect PB"' and all other files can be generated by any person or organisation using the editor.

After the first generation of the base `ECU Configuration Value description` the different organisations can start their part of the parameter configuration by editing the `ECU Configuration Value description`. The different organisations are most likely working with a local copy of the `ECU Configuration Value description`. Eventually there is a need to combine the local copies into one `ECU Configuration Value description`. This is done with a simple merge tool, that is a part of the `ECU Configuration Editor`. The merging is easy since there should be no redundant sections. To ensure editing only pre-compile time, link time

or post-build time parameters, the ECU Configuration Editor shall be able to define a subset of parameters that are allowed to be edited. The subset can be any combination of pre-compile time, link time and post-build time parameters.

Requirements for ECU Configuration Editors supporting the methodology can be found in chapter 5.3.

2.3.4.1.3 Configuration Management and Post-build Time Loadable

Post-build time loadable permits the change of configuration parameter values after building the rest of the ECU-SW (BSW modules and SW-Cs) by downloading a new configuration loadable to the ECU memory at a specific address. This implies that there are at least two SW articles with unique part numbers for an ECU if using the post-build loadable strategy. (There can be more than two since every BSW module can theoretically be configured post-build time loadable). Since there are several SW articles with unique part numbers there is a must to keep track of each SW article from a Configuration Management perspective. In order to do this for each post-build time loadable, ModuleConfigurations describing different aspects (E.g the post-build aspect) of a BSW module needs to be put under Configuration Management as a separate file.

In figure 2.12 the relationships between ModuleConfigurations describing different aspects and the SW articles with their unique part numbers are shown. Note that the relationships are Configuration Management(CM) relations. This means if a parameter in the EcucModuleConfigurationValues describing the post-build aspect is changed it is only needed to re-build the configuration data, the rest of the SW articles in the ECU remain untouched. If a parameter in a EcucModuleConfigurationValues describing the pre-compile aspect is changed the BSW module needs to be re-build. The configuration data must also be re-build if the EcucModuleConfigurationValues contains e.g. a pre-compile time parameter with the attribute "'PC affects PB"'. See chapter 2.3.4.1.1. Each EcucModuleConfigurationValues describing a certain aspect, e.g the post-build aspect, must be put under Configuration Management as a Configuration Management Artifact(CMA).

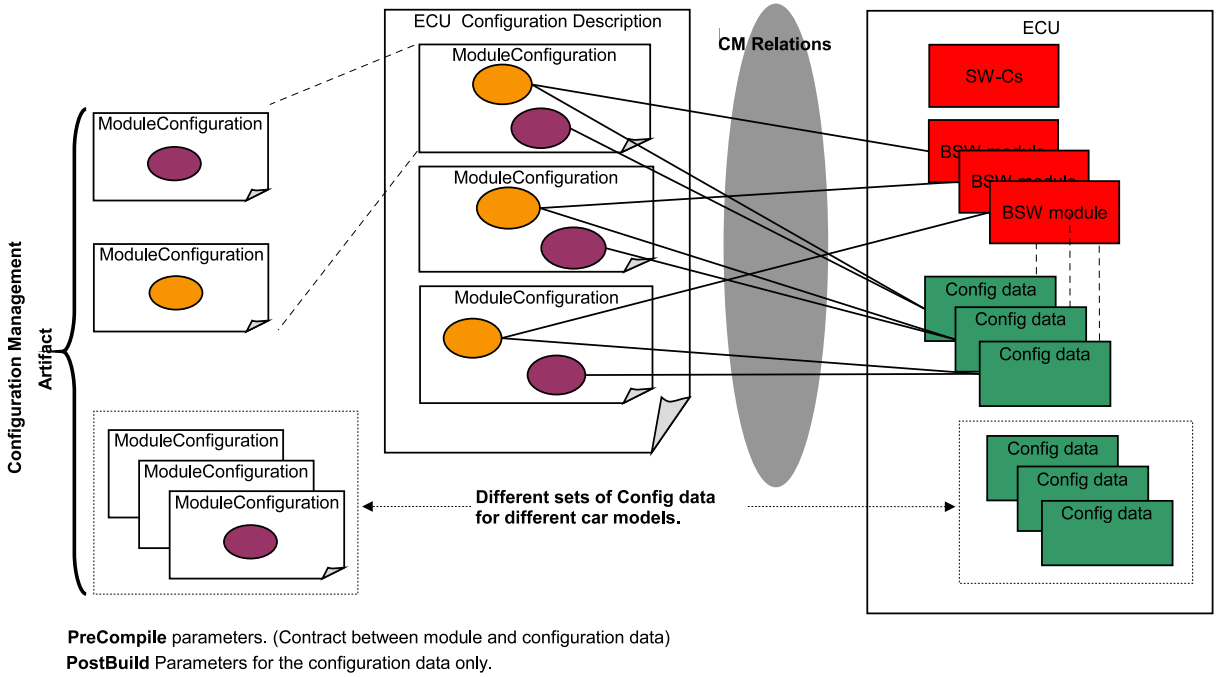

Figure 2.12: Configuration Management Artifacts(CMAs) and post-build time loadable configuration.

Another use case where `ModuleConfigurations` describing different aspects must be Configuration Items, is for different car models which can have different set of configuration data, but the rest of the SW in the ECU is the same for all car models. See figure 2.12.

Requirements for the `ECU Configuration Editors` supporting post-build time loadable strategy can be found in chapter 5.3.

2.3.5 Generate Configured Module Code

The third and last step of the AUTOSAR ECU Configuration methodology has already been referenced in the preceding sections and so comes as no surprise. Generation of configured module code for the different BSW modules. Generation is the process of applying the tailored `ECU Configuration Value description` to the software modules. This can be performed in different ways, and is dependent on the configuration classes chosen for the different modules (see chapter 2.3.2), and on implementers choices.

For each BSW module, a generator reads the relevant parameters from the `ECU Configuration Value description` and creates code that implements the specified configuration, as shown on the right hand side of figures A.1 and A.2. In this generation step, the abstract parameters of the `ECU Configuration Value description` are translated to hardware and implementation-specific data structures that fit to the implementation of the corresponding software module. This specification does not specify the generator tools in detail. It is assumed however that generators perform error, consistency and completeness checks on the part of the configuration they require for generation.

There are some alternative approaches when it comes to generation of configuration data. See chapter A.1.2 for more details.

3 Configuration Metamodel

3.1 Introduction

AUTOSAR exchange formats are specified using a metamodel based approach (see also *Specification of Interoperability of Authoring Tools* [10]). 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 are presented in the following sections.

3.2 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 3.1.

[1] The rules are defined in the Specification of Interoperability of Authoring Tools document [10].

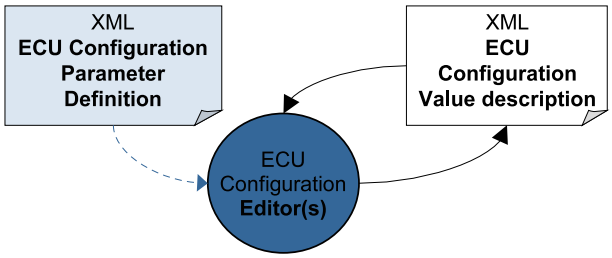

Figure 3.1: Parameter Definition and ECU Configuration Value files

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.

[ecuc_sws_2000] The modeling of the ECU Configuration Value and ECU Configuration Parameter Definition metamodels is done according to the Generic Structure Template [11].

Please note that the Generic Structure Template [11] 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

[ecuc_sws_2001] 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 [7].

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.

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

[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

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 [7] 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 3.2.

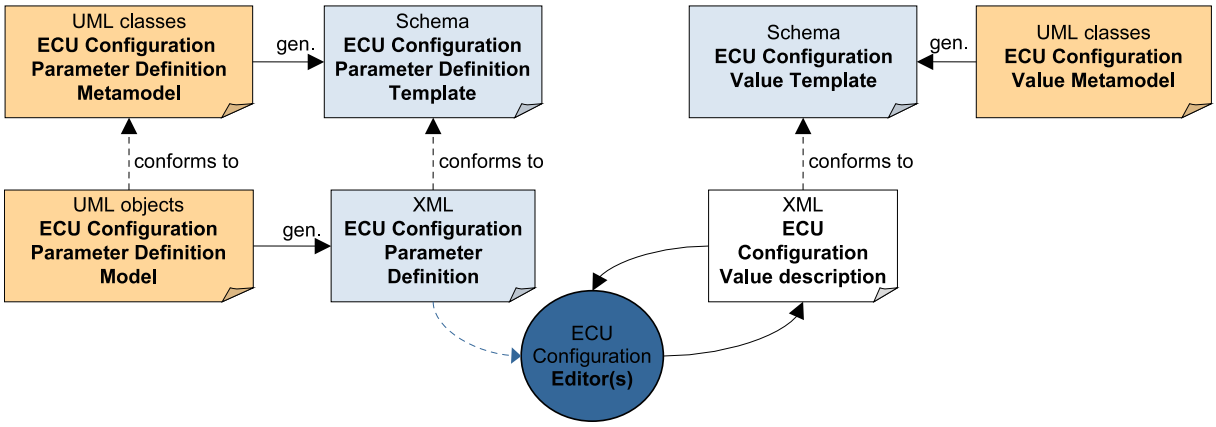

Figure 3.2: Relationship between UML models and XML files

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 3.3. 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 [11]. 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 [7]. 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 [7].

3.3 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.

3.3.1 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 3.3. For an overview of the complete ECU Configuration top level structure please refer to chapter 3.4.1.

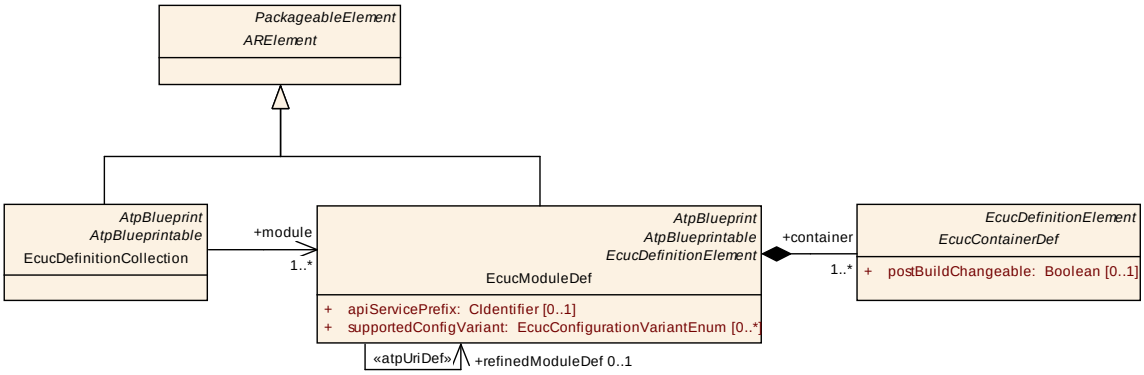

Figure 3.3: ECU Configuration Parameter Definition top-level structure

[ecuc_sws_2002] The generic structure of all AUTOSAR templates is described in detail in the AUTOSAR Generic Structure Template [11].

[ecuc_sws_2130] The Standardized Module Definition (StMD) as delivered by AUTOSAR [12] shall be provided inside the package structure */AUTOSAR/EcucDefs/*.

[ecuc_sws_6070] Ecu Configuration Parameter Definitions shall be sorted alphabetically.

[ecuc_sws_2003] 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.

[ecuc_sws_2065] The ECU Configuration Parameter Definition of one module is called `EcucModuleDef` and inherits from `ARElement`.

`ARElement` itself inherits form `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

[4] A Software Module might be Basic Software, Application Software Component or the RTE, see AUTOSAR Glossary [2]

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 [11].

[ecuc_sws_2004] 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.

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 3.1: EcucDefinitionCollection

3.3.1.1 Usage of the Admin Data

AdminData is an attribute of Identifiable [11] 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.

[ecuc_sws_6004] 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 3.1 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 3.1

```
<?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>
```

[ecuc_sws_6005] For each module definition there needs to be provided which revision the StMD is. For the VSMD the AUTOSAR release version and the vendor's own version information must be provided. The usage of `AdminData` on `EcucModuleDef` is mandatory.

Example 3.2 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 3.2

```
<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>
```

3.3.1.2 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 [11].

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 3.3 shows the split of the `DESC` and `INTRODUCTION`.

Example 3.3

```
          <SHORT-NAME>Adc</SHORT-NAME>
          <CONTAINERS>
<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>
```

Example 3.4 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 [11].

Example 3.4

```
<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>
```

3.3.2 ECU Configuration Module Definition

[ecuc_sws_2005] 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 an unique name within its enclosing `ARPackage`.

[ecuc_sws_2059] 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 3.3.4.2).

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,CollectableElement,EcucDefinitionElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***

[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*
apiService Prefix	CIdentifier	0..1	ref	For CDD modules this attribute holds the module abbreviation. The shortName of the module definition of a complex device driver is always "CDD". Therefore for CDD modules the module shortName shall be described with this apiServicePrefix attribute.
container	EcucContainerDef	1..*	aggr	Aggregates the top-level container definitions of this specific module definition. **Tags:** xml.sequenceOffset=11
refinedModuleDef	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.
supported ConfigVariant	EcucConfigurationVariantEnum	*	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 3.2: EcucModuleDef

[ecuc_sws_2094] The `EcucModuleDef` aggregates container definitions (`EcucContainerDef`) with the role name `container` which may hold other container definitions, parameter definitions and reference definitions.

[ecuc_sws_2095] The reference `refinedModuleDef` from an `EcucModuleDef` with the category *VENDOR_SPECIFIC_MODULE_DEFINITION* to an `EcucModuleDef` with the category *STANDARDIZED_MODULE_DEFINITION* is mandatory and specifies that the source `EcucModuleDef` is the *Vendor Specific Module Definition* which refines the referenced Standardized `EcucModuleDef`.

[ecuc_sws_6044] The reference `refinedModuleDef` from an `EcucModuleDef` with the category *STANDARDIZED_MODULE_DEFINITION* shall not be used.

[ecuc_sws_6043] 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 3.3: EcucModuleDef class categories

[constr_3022] **`EcucModuleDef` category restriction** ⌈The category definition shall be restricted to exactly the two defined ones: VENDOR_SPECIFIC_MODULE_DEFINITION and STANDARDIZED_MODULE_DEFINITION ⌋

[ecuc_sws_2096] The `EcucModuleDef` specifies which configuration variants are supported by this software modules configuration using the element `supportedConfigVariant`. For a detailed description how the configuration variants are related to the configuration classes please refer to section 3.3.4.3.2. For each configuration variant that is supported one entry shall be provided.

In figure 3.4 an example of the top-level structure is provided and in the example 3.5 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 3.28.

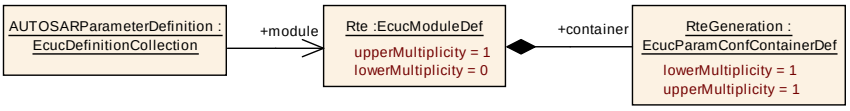

Figure 3.4: ECU Configuration Definition example

Example 3.5

```
<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.

3.3.3 Container Definition

[ecuc_sws_2006] The container definition is used to group other parameter container definitions, parameter definitions and reference definitions.

There are two specializations of a container definition. The abstract class `EcucContainerDef` is used to gather the common features (see figure 3.5).

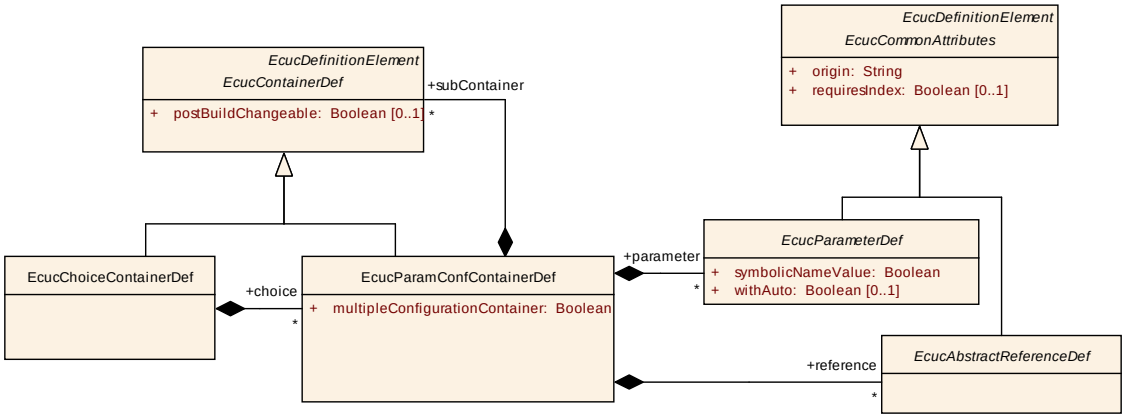

Figure 3.5: Class diagram for parameter container definition

Class	**EcucContainerDef (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Base class used to gather common attributes of configuration container definitions.			
Base	ARObject,EcucDefinitionElement,Identifiable,MultilanguageReferrable,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 post-build time. This parameter may only be set to true if all of the following conditions hold: • the container's upperMultiplicity > lowerMultiplicity • all parameters within the container and subContainers are post-build time changeable. If any of the aggregated parameters is either pre-compile time or link time this attribute is ignored and may be omitted. This attribute 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 attribute is mandatory.

Table 3.4: EcucContainerDef

[ecuc_sws_2043] Each `EcucContainerDef` is an `Identifiable`.

[ecuc_sws_2044] 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 3.3.4.2).

[ecuc_sws_2064] The attribute `postBuildChangeable` specifies if the number of containers can be changed `PostBuild time` in the ECU Configuration Value description.

[ecuc_sws_2132] The attribute `postBuildChangeable` shall be only set to true if the corresponding container is a descendant subcontainer of the container that has the `multipleConfigurationContainer` attribute set to `true`.

[ecuc_sws_2007] 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,EcucContainerDef,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	**Datatype**	**Mul.**	**Kind**	**Note**

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 3.5: EcucParamConfContainerDef

For a detailed description of Multiple Configuration sets please refer to chapters 3.3.3.2 and sec:MultipleConfigSets.

One example of a `EcucContainerDef` and its embedding in the ECU Configuration Parameter Definition is shown in figure 3.6. One `EcucModuleDef Rte` is specified being part of the `EcucDefinitionCollection`. Two containers of type `ParameterConfParamDef` 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.

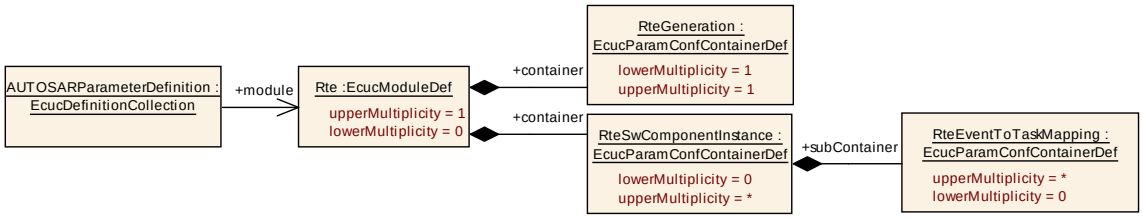

Figure 3.6: Example of an object diagram for container definition

In the XML outtake in example 3.6 only the relevant part from figure 3.6 is shown, not including the `EcucDefinitionCollection`[6]. The corresponding ECU Configuration Value description XML file extract is shown in example 3.32.

Example 3.6

```
<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>
```

3.3.3.1 Choice Container Definition

[ecuc_sws_2011] The `EcucChoiceContainerDef` can be used to specify that certain containers might occur exclusively in 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.

[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*>

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,EcucContainerDef,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
choice	EcucParamConfContainerDef	*	aggr	The choices available in a EcucChoiceContainerDef.

Table 3.6: EcucChoiceContainerDef

[ecuc_sws_2067] 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.

[ecuc_sws_2012] 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 3.7 and the XML definition is shown in example 3.7. The corresponding ECU Configuration Value description is shown in example 3.33.

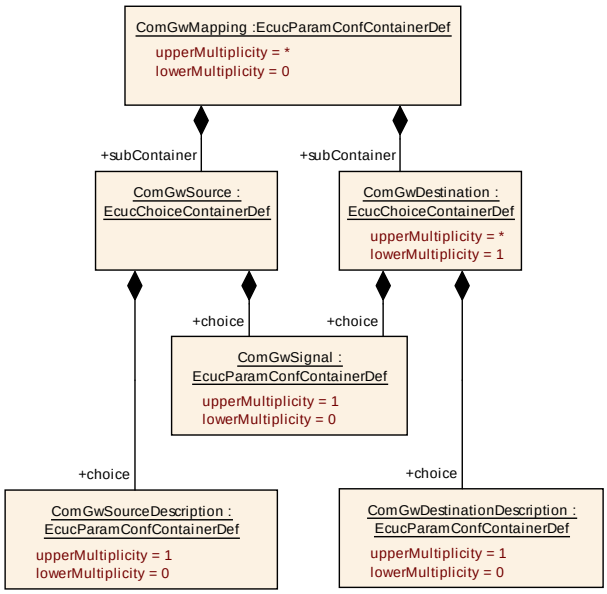

Figure 3.7: Example of an object diagram for two choice container definitions

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 3.7

```
<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.

3.3.3.2 Multiple Configuration Set Definition

To allow the description of several Ecu Configuration Sets a EcucModuleDef may contain *one* EcucParamConfContainerDef which is specified to be the multipleConfigurationContainer.

[ecuc_sws_2091] A EcucParamConfContainerDef does specify whether the defined container is the multipleConfigurationContainer. Each EcucModuleDef shall contain exactly one such container if it supports multiple configuration sets.

[ecuc_sws_2133] The upperMultiplicity of a container that has the multipleConfigurationContainer attribute set to true shall always be 1.

If a EcucParamConfContainerDef is specified to be the multipleConfigurationSetContainer 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 3.4.7 for details on the Ecu Configuration Value description). The corresponding ECUC Value description XML file extract is shown in example 3.48.[8]

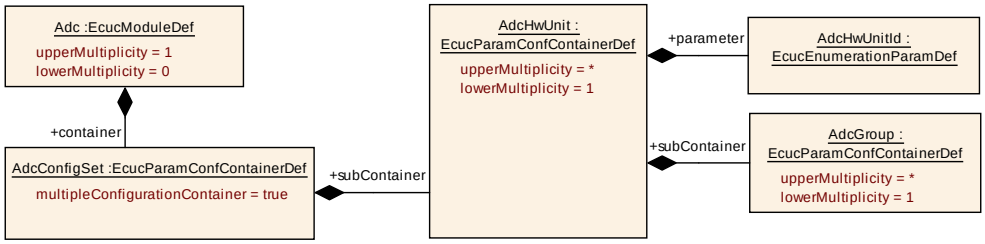

Figure 3.8: Example of an object diagram for multiple configuration container definition

Example 3.8

```
<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>
```

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

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

```
          <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>
```

For the ECU Configuration Value description of this example please refer to section 3.4.7.

3.3.4 Common Configuration Elements

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

3.3.4.1 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 [11]. 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.

3.3.4.2 Configuration Multiplicity

[ecuc_sws_2008] 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.

[ecuc_sws_6016] To express a countable infinite number of occurrences of this element the upperMultiplicityInfinite element shall exist and shall be set to true. [9]

[ecuc_sws_6017] The existence of the elements upperMultiplicityInfinite and upperMultiplicity shall be mutually exclusive.

[ecuc_sws_2110] The attributes lowerMultiplicity, upperMultiplicity and upperMultiplicityInfinite are subject to variant handling (see section 3.3.4.1). The values can be computed using the variant handling mechanism.

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.

[ecuc_sws_2009] 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'.

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.

[ecuc_sws_2010] In the specification object diagrams 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,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.

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

Attribute	*Datatype*	*Mul.*	*Kind*	*Note*
lowerMultiplicity	PositiveInteger	1	attr	The lower multiplicity of the specified element. 0: optional 1: at least one occurence n: at least n occurrences atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime
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: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime
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. **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime

Table 3.7: EcucDefinitionElement

For examples please refer to figure 3.6 and example 3.6

3.3.4.3 Common Configuration Attributes

Several attributes are available on both, parameters and references. These common attributes are shown in figure 3.9.

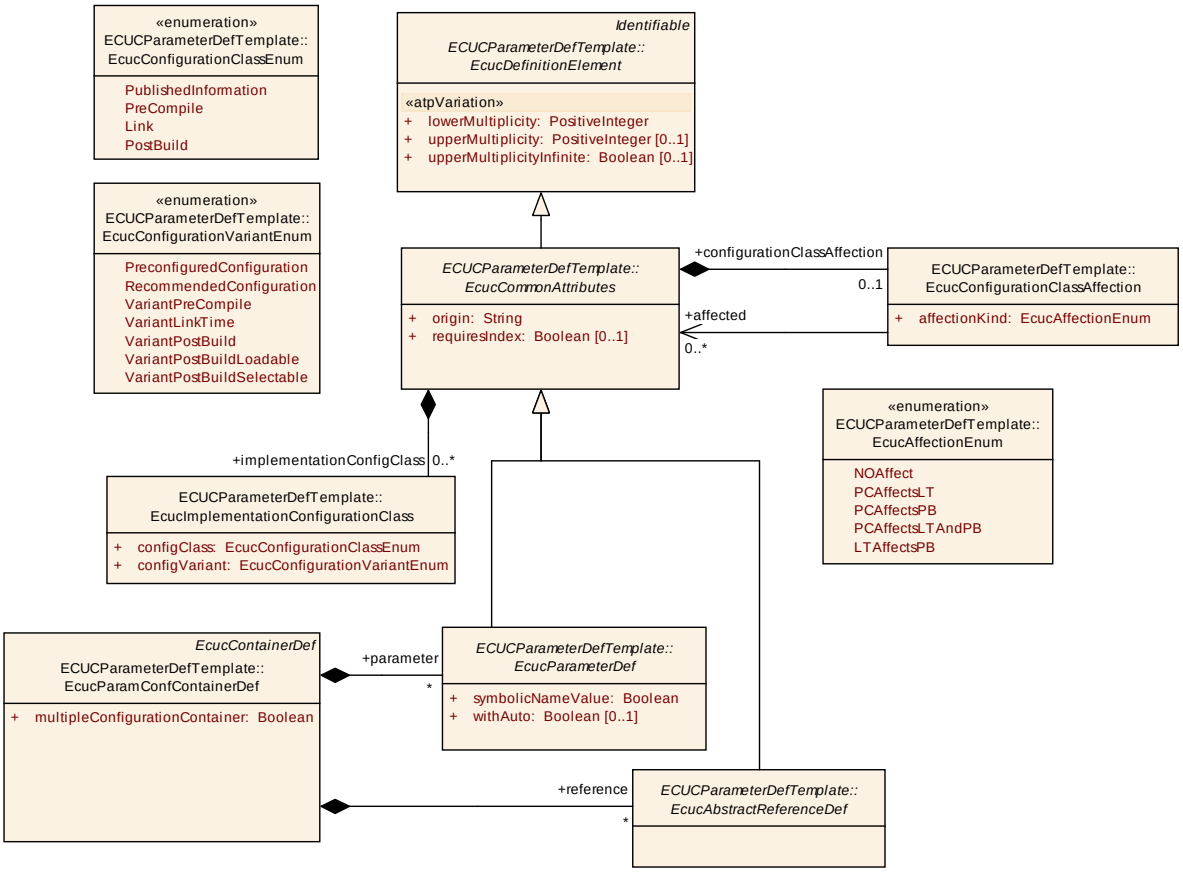

Figure 3.9: Common Attributes for parameters and references

Class	**EcucCommonAttributes (abstract)**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Attributes used by Configuration Parameters as well as References.			
Base	ARObject,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
configurationClassAffection	EcucConfigurationClassAffection	0..1	aggr	Specifes 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
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.

Attribute	Datatype	Mul.	Kind	Note
requiresIndex	Boolean	0..1	attr	Used to define whether the value element for this definition shall be provided with an index.

Table 3.8: EcucCommonAttributes

3.3.4.3.1 Parameter Origin

[ecuc_sws_2015] 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 3.9

```
<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 3.9 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.

3.3.4.3.2 Implementation Configuration Classes

[ecuc_sws_2016] 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]:

- **[ecuc_sws_2070]** `PublishedInformation`
- **[ecuc_sws_2017]** `PreCompile`
- **[ecuc_sws_2018]** `Link`
- **[ecuc_sws_2019]** `PostBuild`[11]

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

[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.

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

[ecuc_sws_2071] If PublishedInformation is selected as configuration class it has to be the for all configuration variants.

[ecuc_sws_2022] The configuration parameter definition of the BSW has the possibility to define up to three configuration variants how actual configuration parameters can be implemented. So the implementor 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.

[ecuc_sws_2097] The supported configuration variants in the StMD are[12]:

- **[ecuc_sws_2098]** VariantPreCompile
- **[ecuc_sws_2099]** VariantLinkTime
- **[ecuc_sws_2100]** VariantPostBuild

[ecuc_sws_6052] The supported configuration variants in the VSMD are:

- **[ecuc_sws_6053]** VariantPreCompile
- **[ecuc_sws_6054]** VariantLinkTime
- **[ecuc_sws_6055]** VariantPostBuildLoadable
- **[ecuc_sws_6056]** 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 3.9: EcucImplementationConfigurationClass

Enumeration	**EcucConfigurationClassEnum**
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate
Note	Possible configuration classes for the AUTOSAR configuration parameters.
Literal	***Description***

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

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 3.10: 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 3.11: EcucConfigurationVariantEnum

[ecuc_sws_2101] For each `EcucConfigurationVariantEnum` the `EcucModuleDef` supports there shall be one `EcucImplementationConfigurationClass` element.

The supported configuration variants of the module are described in section 3.3.2.

[ecuc_sws_2102] Every `EcucImplementationConfigurationClass` specifies which `EcucConfigurationClassEnum` this parameter or reference shall be implemented for this `EcucConfigurationVariantEnum`.

The example 3.10 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 3.10

```
<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 [12].

3.3.4.3.3 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]:

- **[ecuc_sws_2076]** `NOAffect`
- **[ecuc_sws_2077]** `PCAffectsLT`
- **[ecuc_sws_2078]** `PCAffectsPB`
- **[ecuc_sws_2079]** `PCAffectsLTAndPB`
- **[ecuc_sws_2080]** `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`.

[ecuc_sws_2081] The reference `affected` from the `EcucConfigurationClassAffection` to any subclass of `EcucCommonAttributes` is used to define which actual parameters and references are affected.

For a detailed description of the affection mechanism refer to section 2.3.4.1.

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 3.12: EcucConfigurationClassAffection

3.3.5 Parameter Definition

[ecuc_sws_2013] Parameters are defined within a `EcucParamConfContainerDef` using an aggregation with the role name `parameter` at the parameter side.

[ecuc_sws_2014] The possible parameter types are specified using one of the specialized classes derived from `EcucParameterDef`. The `EcucParameterDef` does inherit from `Identifiable`, `EcucCommonAttributes` and `EcucDefinitionElement`.

The available parameter types are shown in figure 3.10.

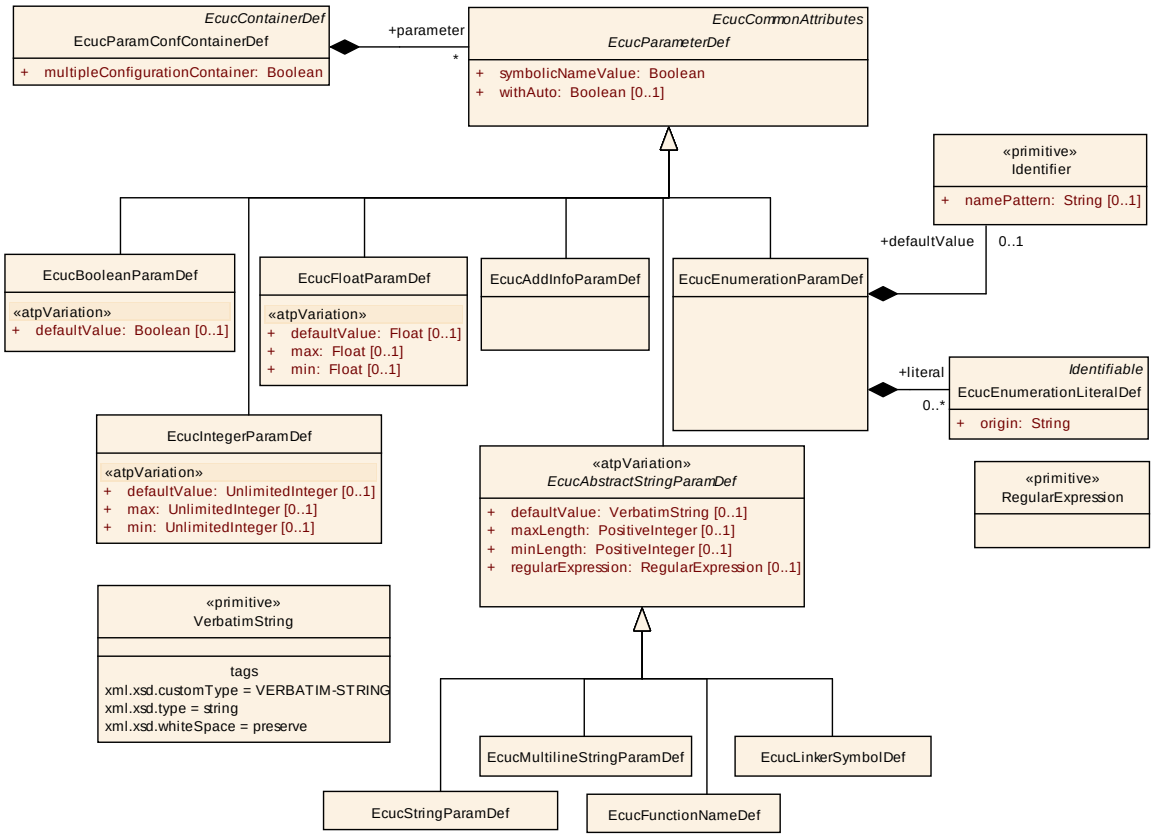

Figure 3.10: Class diagram for parameter definition

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,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 3.13: EcucParameterDef

The use-case for the attribute `symbolicNameValue` is described in section 3.3.6.5.

The use-case for the attribute `withAuto` is described in section 4.4.1.

In the next sections the different parameter types will be described in detail. The examples for the individual parameters are taken from figure 3.11.

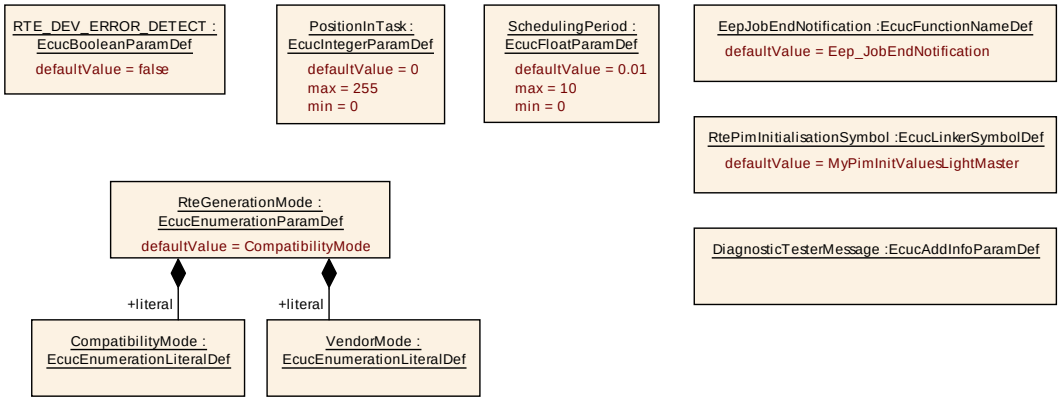

Figure 3.11: Example of parameter definitions using different types

3.3.5.1 Boolean Type

[ecuc_sws_2026] 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.

[ecuc_sws_2127] 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 3.4.4.2). 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 [11] for more details).

[ecuc_sws_2111] The attribute defaultValue of EcucBooleanParamDef is subject to variant handling (see section 3.3.4.1). The value can be computed using the variant handling mechanism.

Class	**EcucBooleanParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Boolean. Allowed values are true and false. **Tags:** xml.sequenceOffset=0			
Base	ARObject,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	Boolean	0..1	attr	Default value of the boolean configuration parameter. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime

Table 3.14: EcucBooleanParamDef

Example 3.11 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.37.

Example 3.11

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

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

3.3.5.2 Integer Type

[ecuc_sws_2027] 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.

[ecuc_sws_2114] The attribute `defaultValue` of `EcucIntegerParamDef` is subject to variant handling (see section 3.3.4.1). The value can be computed using the variant handling mechanism.

[ecuc_sws_2116] The attributes `min` and `max` of `EcucIntegerParamDef` are subject to variant handling (see section 3.3.4.1). The values can be computed using the variant handling mechanism.

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

[ecuc_sws_2072] 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`).

[ecuc_sws_2073] 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`).

[ecuc_sws_6032] 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.

[ecuc_sws_2074] `IntegerValue` has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits. If the sign is omitted, "+" is assumed.

[ecuc_sws_3040] The value of an `IntegerValue` shall to be specified in signed decimal notation without decimal point.

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

Class	**EcucIntegerParamDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Configuration parameter type for Integer. **Tags:** xml.sequenceOffset=0			
Base	ARObject,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	UnlimitedInteger	0..1	attr	Default value of the integer configuration parameter. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime
max	UnlimitedInteger	0..1	attr	Max value allowed for the parameter defined. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime
min	UnlimitedInteger	0..1	attr	Min value allowed for the parameter defined. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGenerationTime

Table 3.15: EcucIntegerParamDef

Example 3.12 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.38.

Example 3.12

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

3.3.5.3 Float Type

[ecuc_sws_2028] 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].

[ecuc_sws_2115] The attribute `defaultValue` of `EcucFloatParamDef` is subject to variant handling (see section 3.3.4.1). The value can be computed using the variant handling mechanism.

[ecuc_sws_2117] The attributes `min` and `max` of `EcucFloatParamDef` are subject to variant handling (see section 3.3.4.1). The values can be computed using the variant handling mechanism.

[ecuc_sws_6033] 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.

[ecuc_sws_6034] The notation of the special float values "Not a Number" and positive/negative "infinity" shall be:

- NaN
- INF
- -INF

[ecuc_sws_2075] For the representation the `IEEE double-precision 64-bit floating point` of the *IEEE 754-1985* standard [13] 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,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	Float	0..1	attr	Default value of the float configuration parameter. atpVariation: **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	Float	0..1	attr	Max value allowed for the parameter defined. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGeneration Time
min	Float	0..1	attr	Min value allowed for the parameter defined. atpVariation: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=CodeGeneration Time

Table 3.16: EcucFloatParamDef

Example 3.13 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.39.

Example 3.13

```
<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>
```

3.3.5.4 String Parameter

[ecuc_sws_2029] The subclasses of the class `EcucAbstractStringParamDef` provide means to specify strings in the ECUC Value description. Additionally an optional `defaultValue` can be provided.

[ecuc_sws_2112] The attribute `defaultValue` of `EcucAbstractStringParamDef` and its subclasses is subject to variant handling (see section 3.3.4.1). The value can be computed using the variant handling mechanism.

[ecuc_sws_6035] The regular expression is provided according to the Generic Structure Template [11].

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: ECUC0082, ECUC0083 **Tags:** Vh.latestBindingTime=CodeGenerationTime			
Base	ARObject,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
defaultValue	VerbatimString	0..1	attr	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 3.17: EcucAbstractStringParamDef

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

Table 3.18: EcucStringParamDef

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

Table 3.19: EcucMultilineStringParamDef

3.3.5.5 Linker Symbol Parameter

[ecuc_sws_6006] 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.

[ecuc_sws_2030] The restriction on the default value 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].

[ecuc_sws_2031] 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,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 3.20: EcucLinkerSymbolDef

Example 3.14 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.34.

Example 3.14

```
<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>
```

3.3.5.6 Function Name Parameter

[ecuc_sws_2033] 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,EcucAbstractStringParamDef,EcucCommonAttributes,EcucDefinitionElement,EcucParameterDef,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 3.21: EcucFunctionNameDef

Example 3.15 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.35.

Example 3.15

```
<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>
```

3.3.5.7 Enumeration Parameter

[ecuc_sws_2034] 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,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 3.22: EcucEnumerationParamDef

3.3.5.8 Enumeration Literal Definition

[ecuc_sws_2035] To provide the available choices for the `EcucEnumerationParamDef` the `EcucEnumerationLiteralDef` is used. For each available choice there needs to be one `EcucEnumerationLiteralDef` defined.

[ecuc_sws_2036] For the text used to define the `EcucEnumerationLiteralDef` no additional attribute is needed because the `shortName` inherited from identifiable is used to define the literals.

[ecuc_sws_2054] For the allowed string in `shortName` the restrictions apply as defined in the Generic Structure Template [11], 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 32 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 [ecuc_sws_2088].

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

[ecuc_sws_2131] 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 3.23: EcucEnumerationLiteralDef

Example 3.16 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.36.

Example 3.16

```
<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>
```

3.3.5.9 AddInfo

[ecuc_sws_2118] 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 [11].

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

Table 3.24: EcucAddInfoParamDef

Example 3.17 shows the ECUC Parameter definition XML file. The corresponding ECUC Value description XML file extract is shown in example 3.40.

Example 3.17

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

3.3.6 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 3.3.6.1).
- 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 3.3.6.2).
- Entities outside the ECU Configuration Value description can be referenced when they have been specified in a different AUTOSAR Template (see section 3.3.6.3).
- Entities outside the ECU Configuration Value description can be referenced using the `instanceRef` semantics defined in the Generic Structure Template [11] (see section 3.3.6.4).
- A container can be referenced to achieve a symbolic name semantics (see section 3.3.6.5).

The metamodel of those references is shown in figure 3.12.

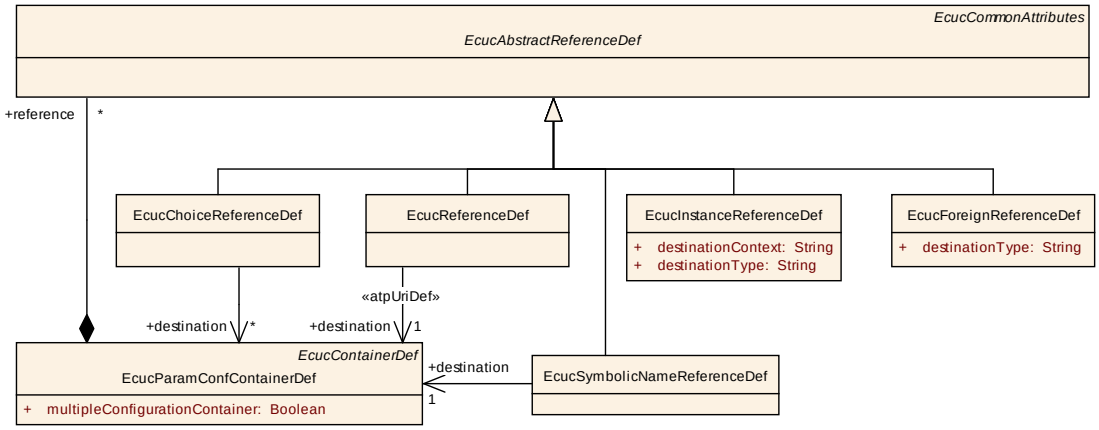

Figure 3.12: Class diagram for parameter references

[ecuc_sws_2037] 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 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,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
–	–	–	–	–

Table 3.25: EcucAbstractReferenceDef

3.3.6.1 Reference

[ecuc_sws_2039] The `EcucReferenceDef` is used to establish references from one `EcucParamConfContainerDef` to one other specific `EcucParamConfContainerDef` within the same ECU Configuration Value description. For this purpose an object representing the reference has to be used.

[ecuc_sws_2038] The destination for the `EcucReferenceDef` and the `EcucChoiceReferenceDef` is both the `EcucParamConfContainerDef`. So it is not possible to reference to a specific `EcucParameterDef` directly but only to its container.

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,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.

Table 3.26: 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 3.13 for an example).

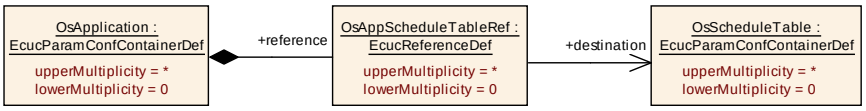

Figure 3.13: Example of an object diagram for a reference

In the example in figure 3.13 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 3.18. 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 3.41.

Example 3.18

```
<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>
```

3.3.6.2 Choice Reference

[ecuc_sws_2040] 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.

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,EcucAbstractReferenceDef,EcucCommonAttributes,EcucDefinitionElement,Identifiable,MultilanguageReferrable,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
destination	EcucParamConfContainerDef	*	ref	All the possible parameter containers for the reference are specified.

Table 3.27: EcucChoiceReferenceDef

All the available choices are connected via associations with the role name `destination` at the referenced object (see example in figure 3.14).

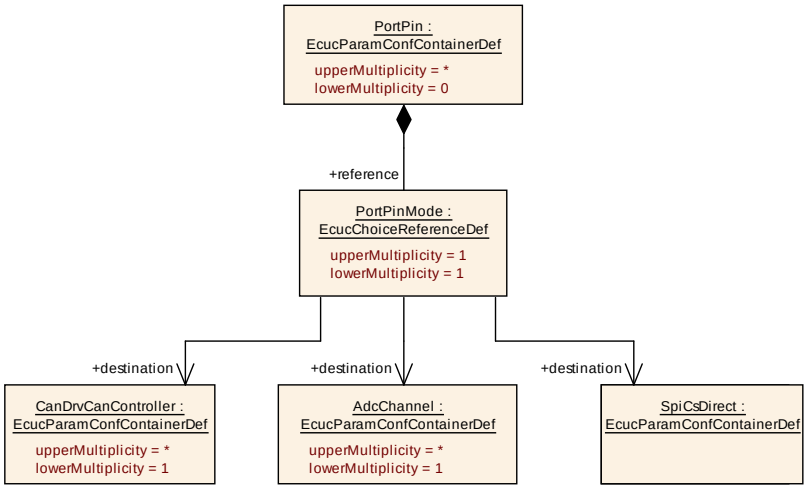

Figure 3.14: Example of an object diagram for a choice reference

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 `ChoiceReferenceParamDef` (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 3.19. The corresponding ECUC Value description XML file extract is shown in example 3.42.

Example 3.19

```
<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].

3.3.6.3 Foreign Reference

[ecuc_sws_2041] 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. The string entered as `destinationContext` shall be an ordered list of M2 class names defined in the metamodel [14] under 'M2::AUTOSAR Templates' as these are represented in the XML-Schema [15], separated by the SPACE character.

Class	**EcucForeignReferenceDef**			
Package	M2::AUTOSARTemplates::ECUCParameterDefTemplate			
Note	Specify a reference to an XML description of an entity described in another AUTOSAR template.			
Base	ARObject,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 3.28: EcucForeignReferenceDef

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

[ecuc_sws_2042] Since the AUTOSAR Generic Structure Template [11] 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, as shown in example 3.20.

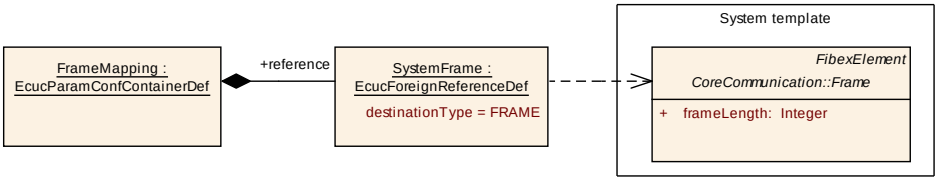

Figure 3.15: Example of an object diagram for a foreign reference

In the example in figure 3.15 the reference is defined to be pointing to a description of a `Frame`. The `Frame` is defined in the System Template metamodel [9] and is derived from `Identifiable`. The corresponding ECUC Value description XML file extract is shown in example 3.43.

Example 3.20

```
<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>
```

3.3.6.4 Instance Reference

[ecuc_sws_2060] 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.

[ecuc_sws_2082] The string entered as `destinationType` shall have the name of a M2 class defined in the metamodel [14] under 'M2::AUTOSAR Templates' as it is represented in the XML-Schema [15] and the referenced class needs to be derived (directly or indirectly) from `Identifiable`. In the generated Parameter Definition XML file [16] the XML-Schema name shall be used.

[ecuc_sws_2083] The string entered as `destinationContext` shall be an ordered list of M2 class names defined in the metamodel [14] under 'M2::AUTOSAR Templates'

[17] For a detailed description of the `instanceRef` concept please refer to the Generic Structure Template [11]

as it is represented in the XML schema [15] 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*
```

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

Table 3.29: EcucInstanceReferenceDef

[ecuc_sws_2061] Since the AUTOSAR Generic Structure Template [11] 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 names of the referenced class, as shown in example 3.21.

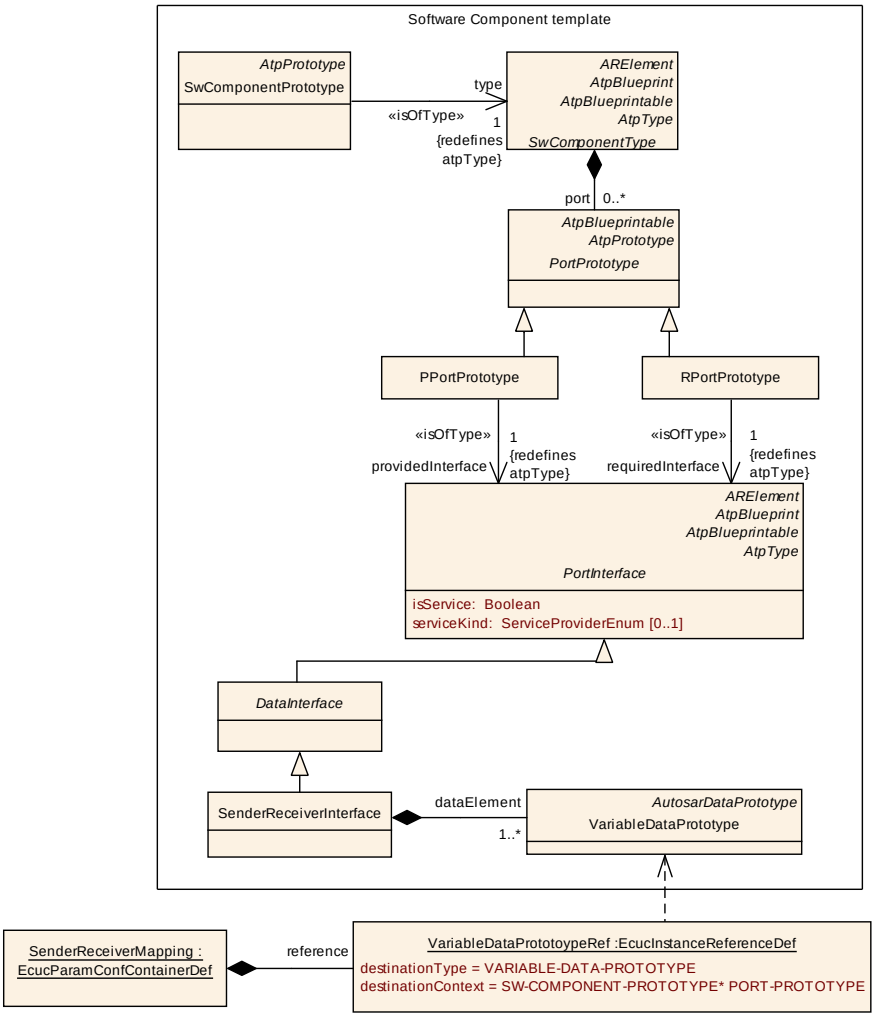

Figure 3.16: Example of an object diagram for an instance reference

In the example in figure 3.16 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 [6] 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 3.44.

Example 3.21

```
<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 3.4.5).

3.3.6.5 Symbolic Name Reference

[ecuc_sws_2032] 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 specialization of a EcucContainerReferenceDef specifies that the implementation of the reference is done using a symbolic name defined by the referenced Container's shortName.			
Base	ARObject,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.

Table 3.30: EcucSymbolicNameReferenceDef

[ecuc_sws_2063] If the attribute `symbolicNameValue` of a configuration parameter (see section 3.3.5) 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 is not present it shall be assumed to be set to `false`.

In the example definition shown in figure 3.17 the IoHwAb module can contain several IoHwAbDemErrors. 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 DemErrorId parameter which belongs to each DemError.

For an example how this is used in the ECU Configuration Value description refer to section 3.4.5.2. The corresponding ECUC Value description XML file extract is shown in example 3.45.

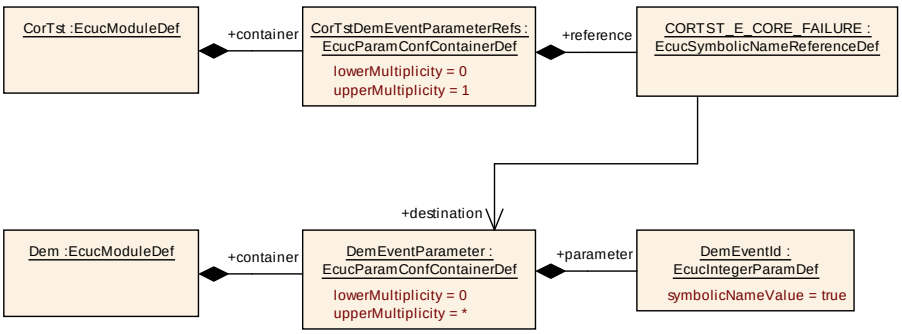

Figure 3.17: Example of an object diagram for a Symbolic Name Reference

Example 3.22

```
<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>
```

3.3.7 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.

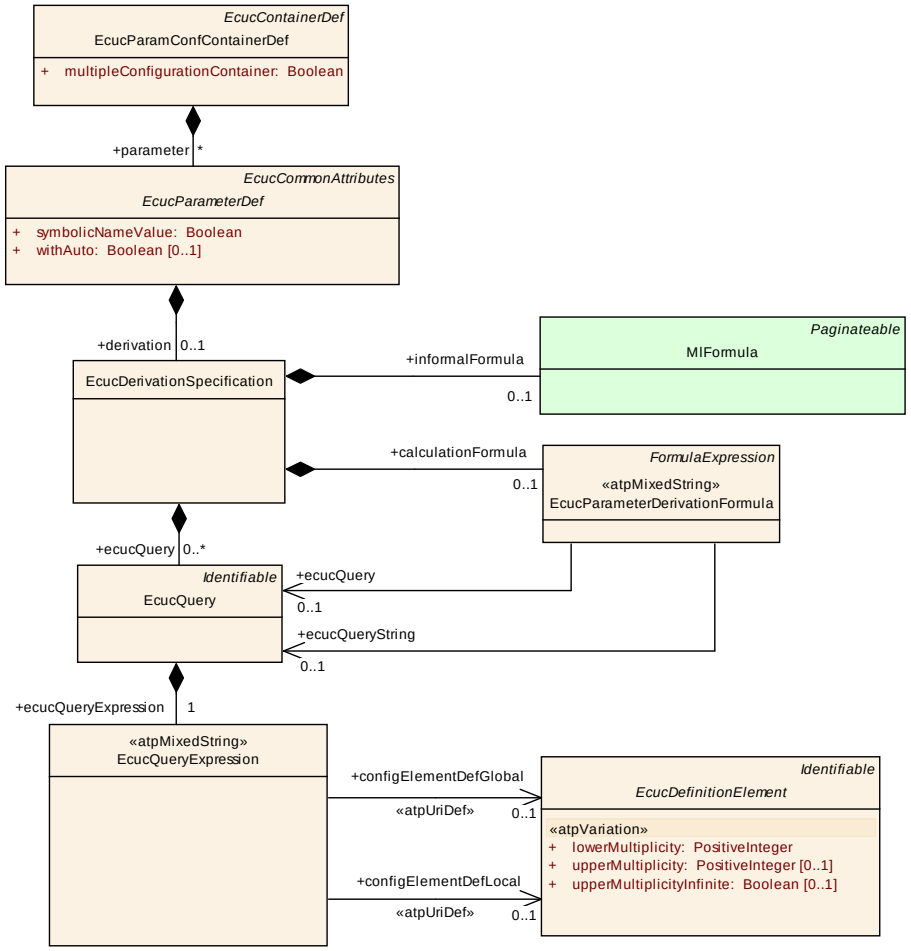

Figure 3.18: Definition of Derived Parameters

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 3.31: EcucDerivationSpecification

[ecuc_sws_2047] For each `EcucParameterDef` it can be specified how the parameter value will be computed. This is captured in the element `EcucDerivationSpecification`.

[ecuc_sws_2129] For all `EcucParameterDef` types an informal description of the derivation can be specified in the element `informalFormula`.

[ecuc_sws_2128] For the `EcucParameterDef` types

- `EcucBooleanParamDef`
- `EcucIntegerParamDef`
- `EcucFloatParamDef`

a formal `calculationFormula` can be specified in the element `EcucParameterDerivationFormula`.

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.

3.3.7.1 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 3.18). The formal language is defined in the Generic Structure Template [11]. 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.
ecucQuery String	EcucQuery	0..1	ref	This indicates that the referenced query shall return a string.

Attribute	Datatype	Mul.	Kind	Note

Table 3.32: 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 `EcucQuerys` 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 3.33: 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**
configElem entDefGlo bal	EcucDefinitionEl ement	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.

[18] `configElementDefLocal`, `configElementDefGlobal`

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

Table 3.34: EcucQueryExpression

[ecuc_sws_6018] 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`.

[ecuc_sws_6019] The *refvalue* function shall result in an error if the `EcucDefinitionElement` points to a not existing element in the ECU Configuration Parameter Definition.

[ecuc_sws_6020] 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.

[ecuc_sws_6021] 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.

[ecuc_sws_6022] In case the member of the first parameter is a reference the *deref* function returns the referenced element as a set.

[ecuc_sws_6023] 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.

[ecuc_sws_6024] The *value* function takes the result of a *deref* function or *refvalue* function, which is an element set.

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

[ecuc_sws_6025] The *value* function returns the parameter's value as numerical value.

[ecuc_sws_6026] 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)

[ecuc_sws_6057] The *strValue* function takes the result of a *deref* function or *refvalue* function, which is an element set.

[ecuc_sws_6058] The *strValue* function returns the parameter's value as string.

[ecuc_sws_6059] 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)

[ecuc_sws_6060] The *valueAt* function takes the result of a *deref* function or *refvalue* function, which is an element set and a zero-based position argument.

[ecuc_sws_6061] The *valueAt* function returns the value of the parameter as numerical value at the position according to the sorting criteria defined in section x.x.x.

[ecuc_sws_6062] 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

[ecuc_sws_6063] The *strValueAt* function takes the result of a *deref* function or *refvalue* function, which is an element set and a zero-based position argument.

[ecuc_sws_6064] 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.

[ecuc_sws_6065] 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)

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

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

- the position is larger than the count-1

[ecuc_sws_6027] The *count* function gets the result of the *deref* or *refvalue* function as input parameter.

[ecuc_sws_6028] The *count* function returns the number of elements in the input parameter set.

[ecuc_sws_6029] 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 3.19 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 3.24.

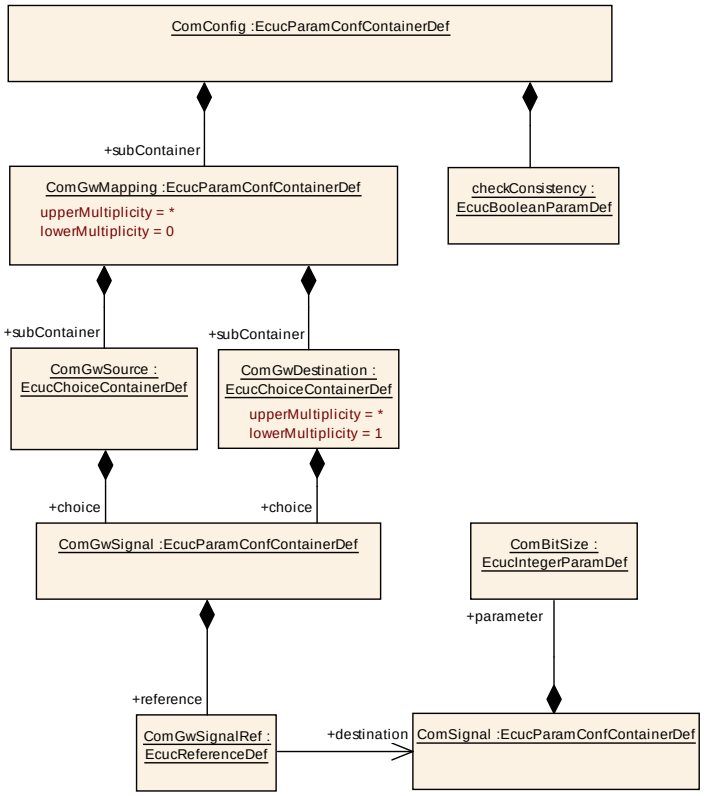

Figure 3.19: Calculation Formula Example

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 3.47.

Example 3.23

```
<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 3.24

```
<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 3.25 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 3.25

```
<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 3.20 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.

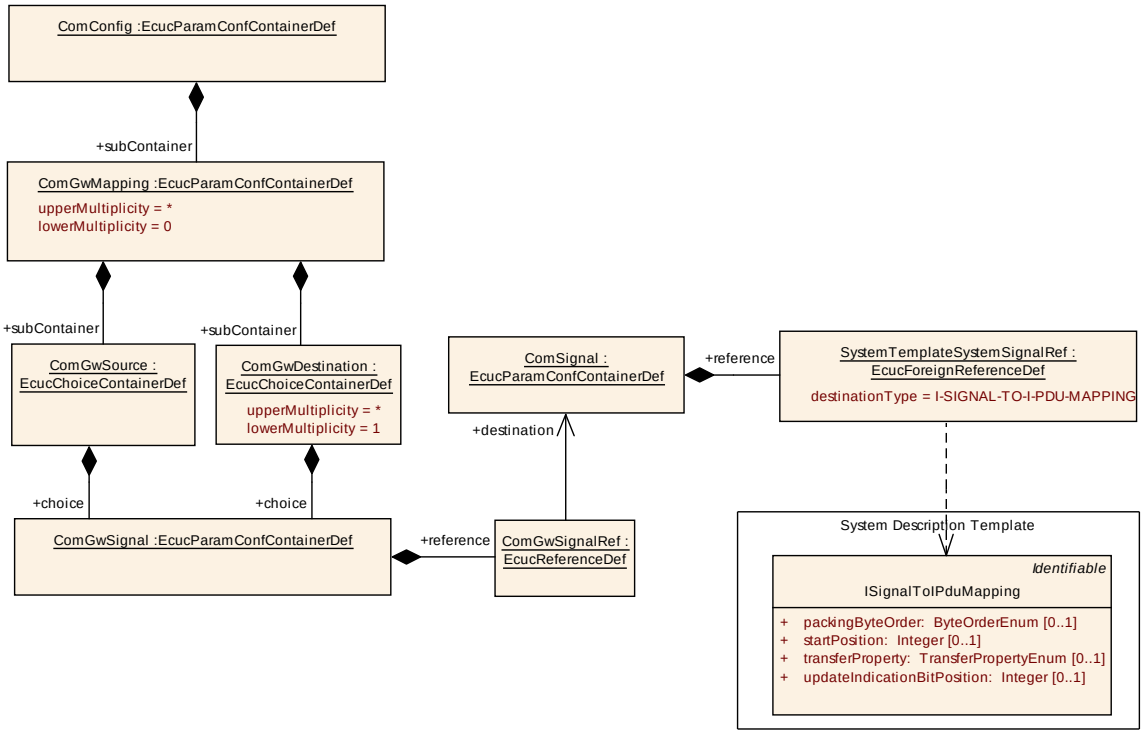

Figure 3.20: Calculation Formula Example

Example 3.26

```
<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>
```

3.3.7.2 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.

[ecuc_sws_2055] 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:

- **[ecuc_sws_2058]** If the derived parameter uses information from parameters defined as `PreCompile` then the derived parameter can be any configuration class.
- **[ecuc_sws_2056]** If the derived parameter uses information from parameters defined as `Link` then the derived parameter needs to be `Link` or `PostBuild` configurable.
- **[ecuc_sws_2057]** If the derived parameter uses information from parameters defined as `PostBuild` then the derived parameter needs to be `PostBuild` configurable as well.

3.3.8 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 3.3.7. 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.

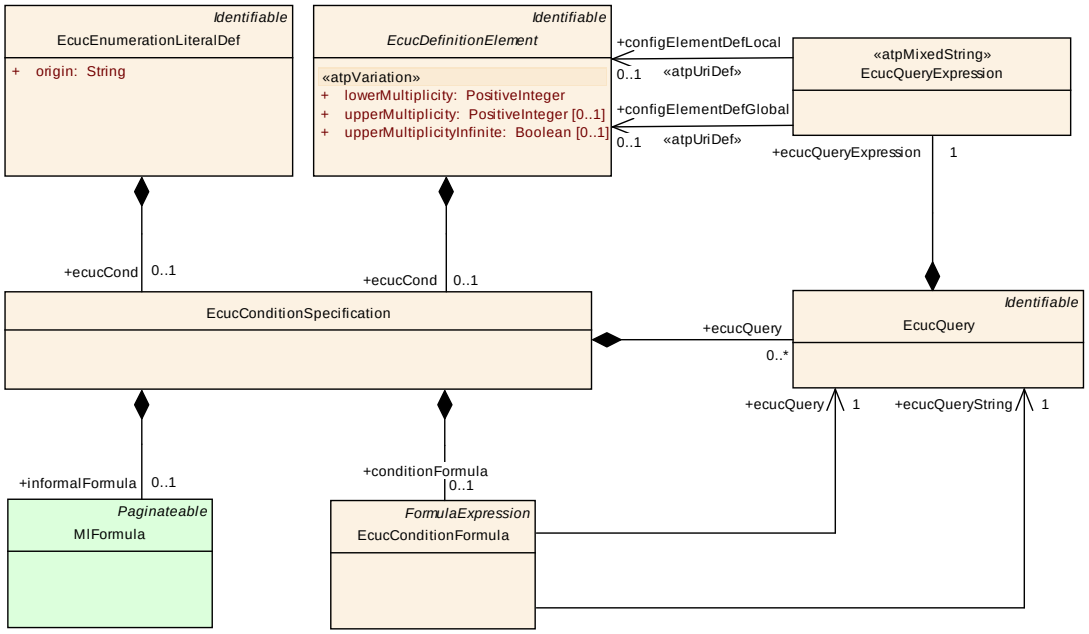

Figure 3.21: Existence dependence of parameter definitions and literal definitions

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 3.3.7.1.

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 3.35: 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. Therfore the properties have the uppermultiplicity 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 3.36: EcucConditionFormula

In the following example in figure 3.22 – taken from the Can Interface module – a possible usage of the condition formula is shown.

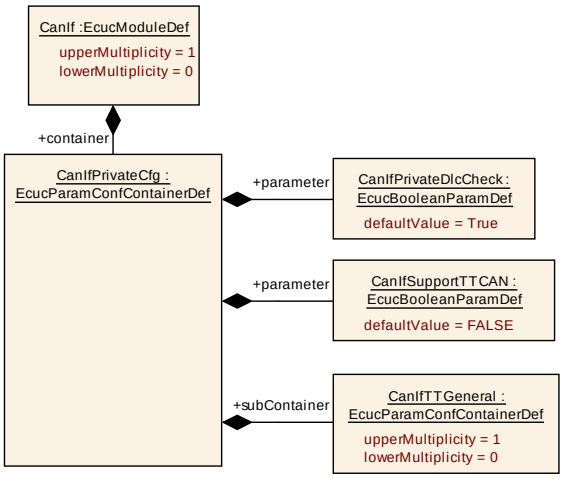

Figure 3.22: Example for condition formula

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 3.27

```
<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 3.27). 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.

3.4 ECU Configuration Value Metamodel

As mentioned in section 3.2 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 3.4.1 and 3.4.2 will introduce the top-level structure of a configuration Value description and the module configurations, whereas the sections 3.4.3, 3.4.4 and 3.4.5 will describe the means to file and structure the actual configuration values.

3.4.1 ECU Configuration Value Top-Level Structure

The top-level entry point to an AUTOSAR ECU Configuration Value description is the `EcucValueCollection` (see figure 3.23). Because of the inheritance from `ARElement` the `EcucValueCollection` can be part of an AUTOSAR description like its counterpart the `EcucDefinitionCollection` does. A valid `EcucValueCollection` needs to reference the System description (provided as an `ecuExtract`) [9] that specifies the environment in which the configured ECU operates. Additionally it references all Software Module configurations (see section 3.4.2) that are part of this ECU Configuration.

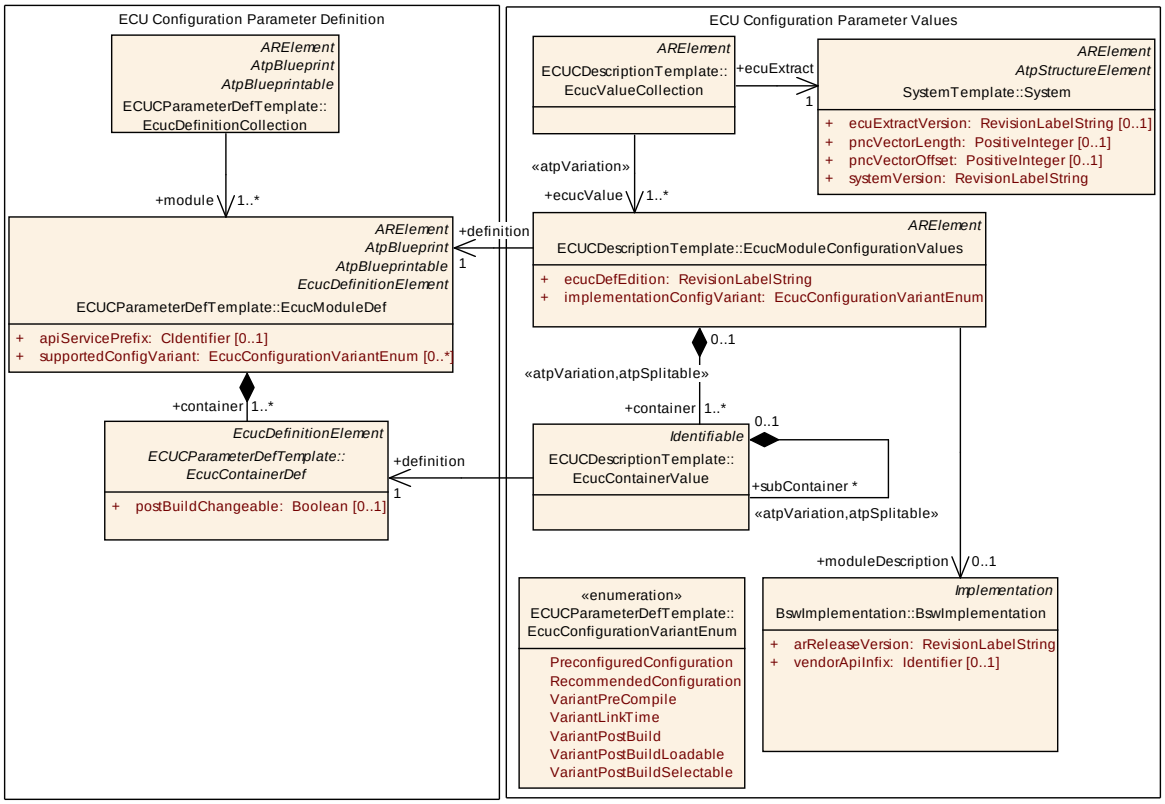

Figure 3.23: ECU Configuration Value Top-Level Structure

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: ECUC0078 **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 3.37: EcucValueCollection

3.4.2 Module Configurations

[ecuc_sws_3016] 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).

[ecuc_sws_2089] 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 3.4.2.1).

[ecuc_sws_2119] The aggregated `container` is subject to variant handling (see section 3.3.4.1). The existence can be evaluated using the variant handling mechanism.

[ecuc_sws_3017] 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.

[ecuc_sws_3035] The reference `definition` assigns the `EcucModuleConfigurationValues` to the according `EcucModuleDef` it is depending on.

[ecuc_sws_6066] Container-, Parameter- and Reference-Values shall be ordered according to the `shortName` of the parameter definition (which is the last chunk of DEFINITION-REF).

[ecuc_sws_6067] Containers on the Values side which have the same parameter definition shall be sorted according to the `shortName` of the `EcucContainerValue`.

[ecuc_sws_6068] 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).

[ecuc_sws_6069] 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 allow specifying ordering of parameter values.			
Base	ARObject			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
index	PositiveInteger	0..1	attr	Used to allow specifying ordering of parameter values. **Tags:** xml.sequenceOffset=-5

Table 3.38: EcucIndexableValue

[ecuc_sws_6072] Parameter- and Reference-Values which have the `requiredIndex` set to true in their definition shall provide an `index`.

[ecuc_sws_3031] 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 tthe BSW module description, the EcucModuleConfigurationValues element has two different roles: The recommendedConfiguration contains parameter values recommended by the BSW module vendor. The preconfiguredConfiguration contains values for those parameters which are fixed by the implementation and cannot be changed. These two EcucModuleConfigurationValues are used when the base EcucModuleConfigurationValues (as part of the base ECU configuration) is created to fill parameters with initial values. **Tags:** atp.recommendedPackage=EcucModuleConfigurationValuess			
Base	ARElement,ARObject,CollectableElement,Identifiable,MultilanguageReferrable,PackageableElement,Referrable			
Attribute	***Datatype***	***Mul.***	***Kind***	***Note***
container	EcucContainerValue	1..*	aggr	Aggregates all containers that belong to this module configuration. atpVariation: **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=definition, shortName, variationPoint.shortLabel 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
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 ADMIN-DATA shall be used to express the semantical 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.

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

Table 3.39: EcucModuleConfigurationValues

Figure 3.24 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.

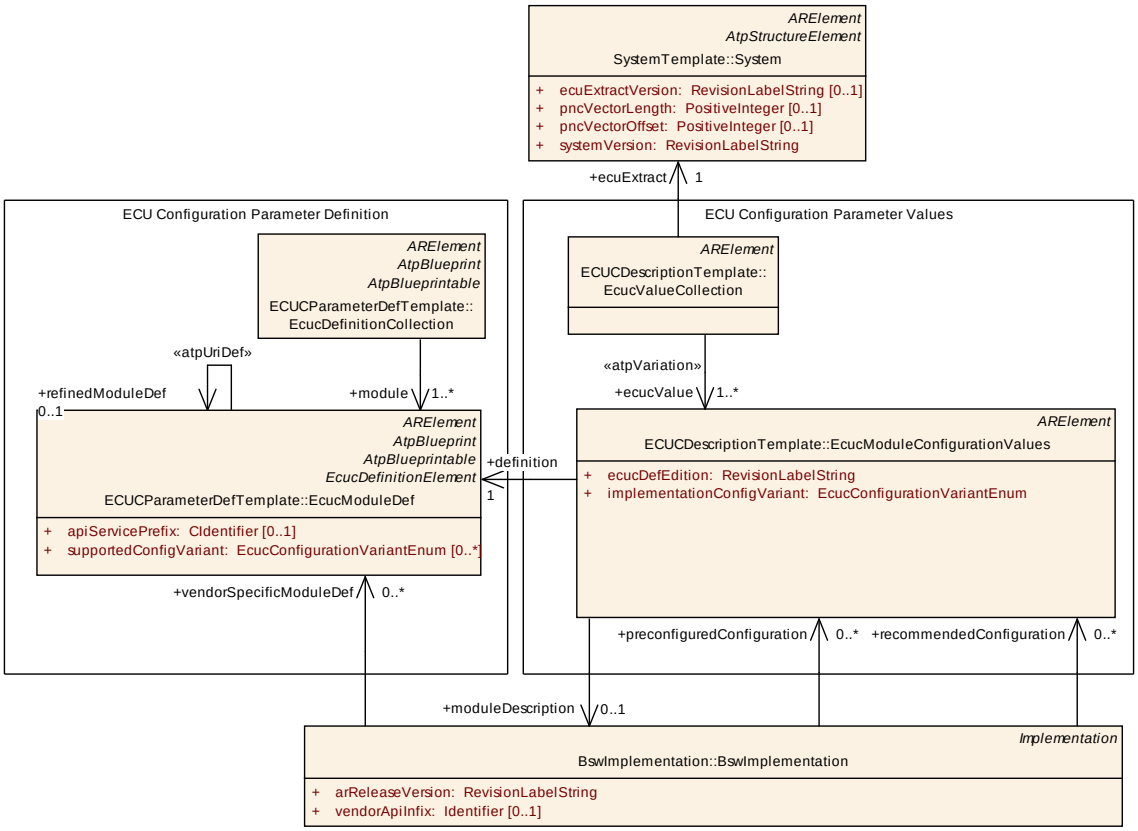

Figure 3.24: Dependencies of ModuleConfigurations

[ecuc_sws_2103] 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 3.3.2 and section 3.3.4.3.2.

To illustrate the structure of an ECU Configuration Value description example 3.28 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 3.5.

The only supportedConfigVariant of example 3.5 is taken for the implementationConfigVariant element.

Example 3.28

```
<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>
```

3.4.2.1 Splitable ModuleConfiguration

In the document *Generic Structure Template* [11] 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 section 2.3.2.3 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 [11].

In Example 3.29 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 3.30 the value for the PRE-COMPILE time parameter ComSignalLength is specified, while in example 3.31 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 3.29

```
<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 3.30

```
<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 3.31

```
<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/ComSignalInitValue</DEFINITION-REF>
          <VALUE>0</VALUE>
        </ECUC-NUMERICAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

3.4.3 Parameter Container Description

Symmetrically to the parameter container definition (see section 3.3.3) the parameter container description is specified to group other containers, parameter values and references. Figure 3.25 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`.

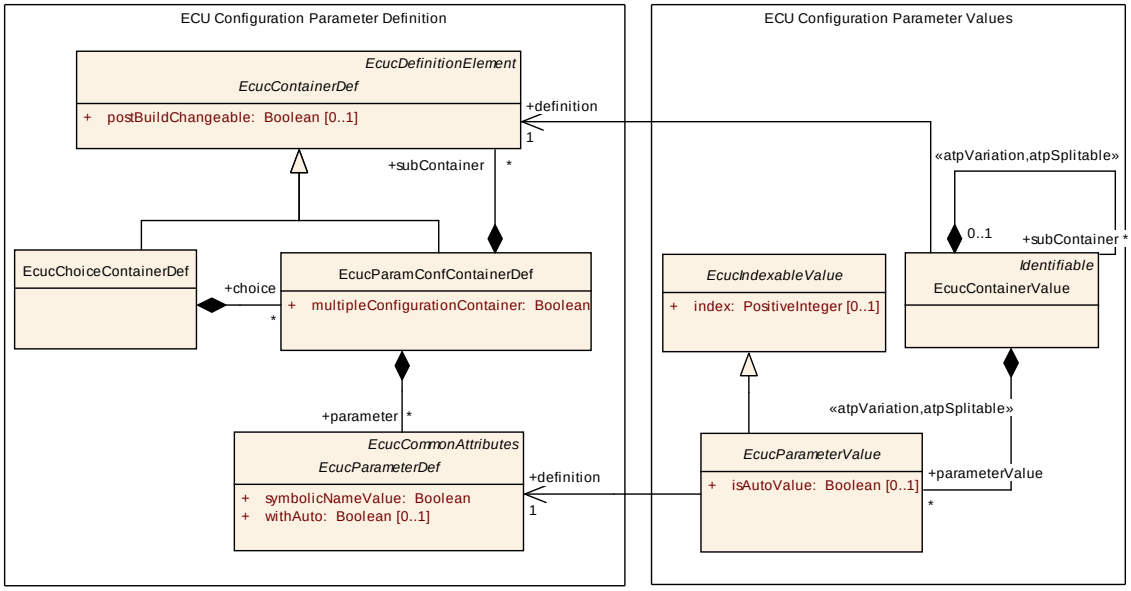

Figure 3.25: Parameter container Value description

[ecuc_sws_3012] The `EcucContainerValue` inherits from `Identifiable` defining a namespace for all `EcucContainerValue`, `EcucParameterValue` and `EcucReferenceValue` that belong to that `EcucContainerValue`.

[ecuc_sws_3019] 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.

[ecuc_sws_3011] 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.

[22] including all `EcucContainerDef`'s decendants

Class	**EcucContainerValue**			
Package	M2::AUTOSARTemplates::ECUCDescriptionTemplate			
Note	Represents a Container definition in the ECU Configuration Description.			
Base	ARObject,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: **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=definition, variationPoint.shortLabel
referenceValue	EcucAbstractReferenceValue	*	aggr	Aggregates all References with this container. atpVariation: **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=definition, variationPoint.shortLabel
subContainer	EcucContainerValue	*	aggr	Aggregates all sub-containers within this container. atpVariation: **Stereotypes:** atpSplitable; atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime atp.Splitkey=definition, shortName, variationPoint.shortLabel

Table 3.40: EcucContainerValue

[ecuc_sws_2120] The aggregated `subContainer` is subject to variant handling (see section 3.3.4.1). The existence can be evaluated using the variant handling mechanism.

[ecuc_sws_2121] The aggregated `parameterValue` is subject to variant handling (see section 3.3.4.1). The existence can be evaluated using the variant handling mechanism.

[ecuc_sws_2122] The aggregated `referenceValue` is subject to variant handling (see section 3.3.4.1). The existence can be evaluated using the variant handling mechanism.

[ecuc_sws_2092] If a `EcucParamConfContainerDef` 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 3.4.7.

In example 3.32 a snippet of an ECU Configuration Value description XML file is shown that conforms to the ECU Configuration Parameter Definition described in example 3.6. 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 3.32

```
<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>
```

3.4.3.1 Choice Containers

[ecuc_sws_3020] 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 3.33 depicts the notation of a filled out EcucChoiceContainerDef as described in example 3.7.

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 3.33

```
<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>
```

3.4.4 Parameter Values

In the ECU Configuration Parameter Definition exist individual elements for the different types of parameters (e.g. Boolean, Integer, String, see section 3.3.5). 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 3.3.4.1) and the documentation support (see section 3.3.5.9).

[ecuc_sws_3006] All metamodel classes specifying parameter values are derived from `EcucParameterValue` (see figure 3.26).

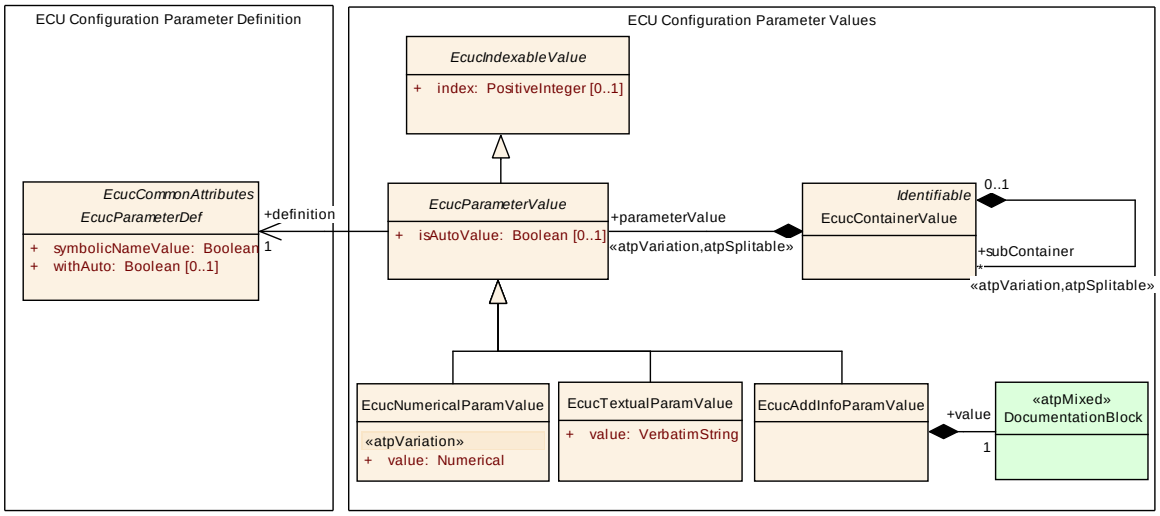

Figure 3.26: Parameter description

[ecuc_sws_3007] All inheriting metamodel classes representing an ECU Configuration Value specify an attribute `value` that stores the configuration value in XML-based description.

[ecuc_sws_3038] The reference `definition` assigns the `EcucParameterValue`[23] to the according `EcucParameterDef` it is providing the value for.

[ecuc_sws_3009] 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 5.2.

[ecuc_sws_3034] 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.

[23] and all its sub-classes

For further rules how a `value` can be provided if no `defaultValue` is specified in the ECU Configuration Definition see section 5.2.

[ecuc_sws_3010] 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.

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	When 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 3.41: EcucParameterValue

3.4.4.1 Textual Parameter Value

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

[ecuc_sws_2126] 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 3.3.5.4, section 3.3.5.5, section 3.3.5.6 and section 3.3.5.7 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	attr	Value of the parameter, not subject to variant handling.

Table 3.42: EcucTextualParamValue

3.4.4.1.1 Examples of `EcucTextualParamValue`

Example 3.34 depicts the configuration description of definition type `EcucLinkerSymbolDef` for example 3.14.

Example 3.34

```
<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 3.35 depicts the configuration description of definition type `EcucFunctionNameDef` for example 3.15.

Example 3.35

```
<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 3.36 depicts the configuration description of definition type EcucEnumerationParamDef for example 3.16.

Example 3.36

```
<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>
```

3.4.4.2 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 3.3.4.1).

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: **Stereotypes:** atpVariation **Tags:** Vh.latestBindingTime=PreCompileTime

Table 3.43: EcucNumericalParamValue

3.4.4.2.1 Examples of EcucNumericalParamValue

Example 3.37 depicts the configuration description of definition type BooleanParamDef for example 3.11.

Example 3.37

```
<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 3.38 depicts the configuration description of definition type `EcucIntegerParamDef` for example 3.12.

Example 3.38

```
<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 3.39 depicts the configuration description of definition type `EcucFloatParamDef` for example 3.13.

Example 3.39

```
<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>
```

3.4.4.3 AddInfo Parameter Value

The only type-specific distinction for the values is the ECU Configuration Parameter Type `EcucAddInfoParamDef` (see section 3.3.5.9).

[ecuc_sws_2123] 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 [11] 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 3.44: EcucAddInfoParamValue

Example 3.40 depicts the configuration description of definition type `EcucAddInfoParamDef` for example 3.17.

Example 3.40

```
<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>
```

3.4.5 References in the ECU Configuration Metamodel

Figure 3.27 depicts the ECU Configuration Metamodel to reference other description elements.

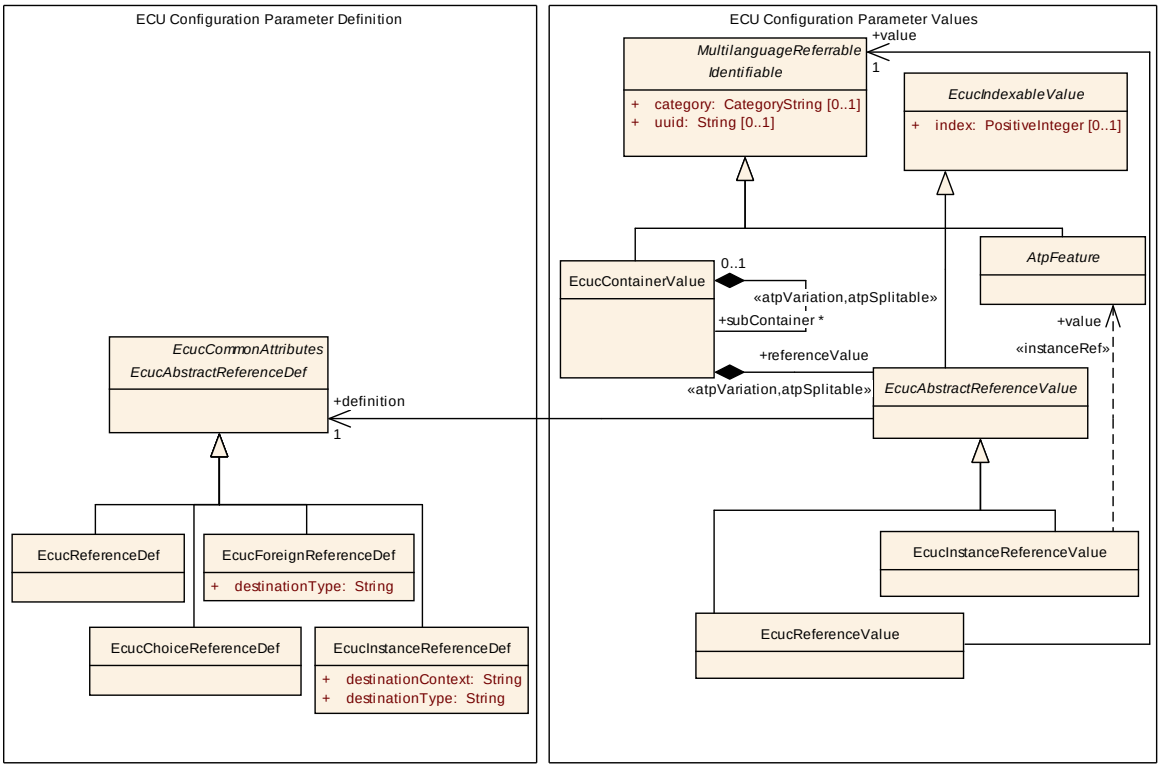

Figure 3.27: Parameter references

[ecuc_sws_3032] The metamodel class `EcucAbstractReferenceValue` acts as the generalization of all reference types in the ECU Configuration Value description.

[ecuc_sws_3039] The reference `destination` assigns the `EcucAbstractReferenceValue`[24] to the according `EcucAbstractReferenceDef` it is depending on.

[ecuc_sws_3030] 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.

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***
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.

[24] and all its descendants

Attribute	Datatype	Mul.	Kind	Note
definition	EcucAbstractReferenceDef	1	ref	Reference to the definition of this EcuAbstractReferenceValue subclasses in the ECU Configuration Parameter Definition. **Tags:** xml.sequenceOffset=-10

Table 3.45: EcucAbstractReferenceValue

[ecuc_sws_3027] The metamodel class `EcucReferenceValue` provides the mechanism to reference to any model element of type `Identifiable`.

[ecuc_sws_3028] Therefore this class provides the means to describe all kinds of reference definitions except an `EcucInstanceReferenceDef`, which is described in section 3.4.5.1 in more detail.

[ecuc_sws_3029] A `ChoiceReferenceParamDef` 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 3.46: EcucReferenceValue

[ecuc_sws_2093] 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 3.23 a `EcucModuleConfigurationValues` is part of the `EcucValueCollection` if it is referenced with the `module` 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 3.41 depicts the configuration description of definition type `EcucReferenceDef` for example 3.18.

Example 3.41

```
<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 3.42 depicts the configuration description of definition type `ChoiceReferenceParamDef` for example 3.19. 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 3.42

```
<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 3.43 depicts the configuration description of definition type `EcucForeignReferenceDef` for example 3.20.

Example 3.43

```
<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>
```

3.4.5.1 Instance Reference Values

Due to the formalization of prototypes in the AUTOSAR Templates (see [11]) the reference to the instance of a prototype needs to declare the complete context in which the instance is residing.

[ecuc_sws_3033] 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>>`.

In figure 3.28 the detailed modeling of the `EcucInstanceReferenceValue` `<<instanceRef>>` is specified.

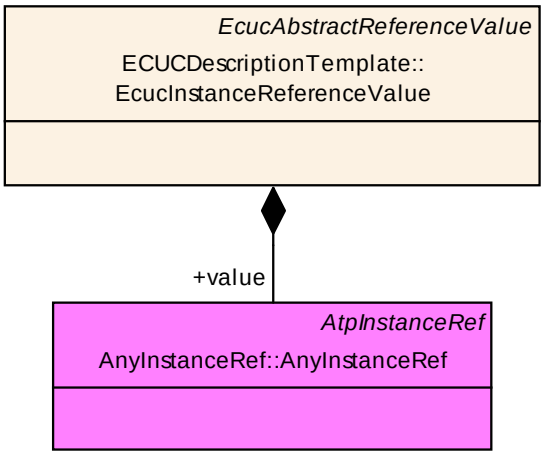

Figure 3.28: Instance Reference Value details

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 3.47: EcucInstanceReferenceValue

Example 3.44 depicts the configuration description of definition type `EcucInstanceReferenceDef` for example 3.21. 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 3.44

```
<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>
```

3.4.5.2 Representation of Symbolic Names

[ecuc_sws_3036] A `EcucSymbolicNameReferenceDef` is represented by an usual `EcucReferenceValue` in the ECU Configuration Value description.

[ecuc_sws_3037] 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.

[ecuc_sws_2107] 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 3.45 depicts the configuration description of definition type `EcucSymbolicNameReferenceDef` for example 3.22. 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 3.45

```
<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>
  <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>
```

[25] The one that is referenced to

```
        <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>
```

[ecuc_sws_2108] The values of configuration parameters which are defined as symblicNameValue = true shall be generated into the header file of the declaring module as #define. The symbol shall be composed of

- the module abbreviation <MA> of the declaring BSW Module (according to BswModuleList [17]) 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 `symblicNameValue` 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 3.29 another example of a symbolic name value configuration setup is shown. The example 3.46 shows a possible value description for this definition.

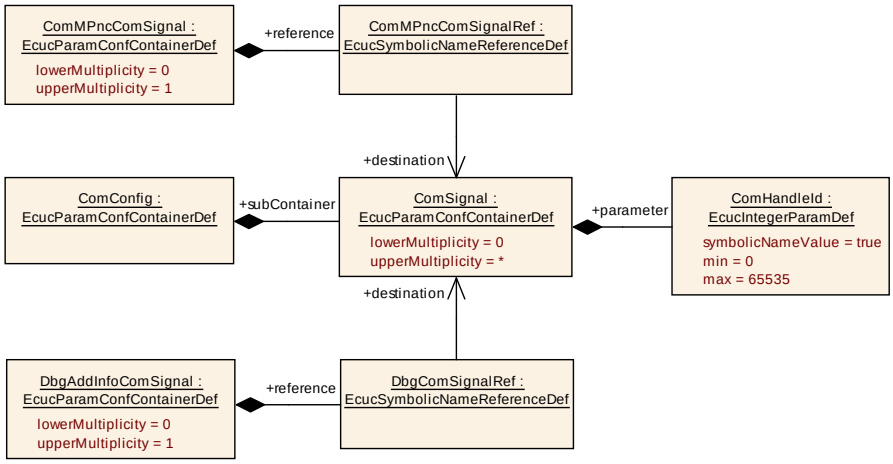

Figure 3.29: Example of ComSignal symbolic name definition

Example 3.46

```
<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

[ecuc_sws_6074] Due to the hierarchical structure of the parameter values it is possible that the same `shortName` (in independent container structures) is the base for multiple symbolic name value definitions. If the respective value is equal in all occurrences of the `shortName` the generation of the #define shall only be done once. If the respective value is different in any of the occurrences of the `shortName` the configuration is invalid.

The following example shows a valid and an invalid configuration.

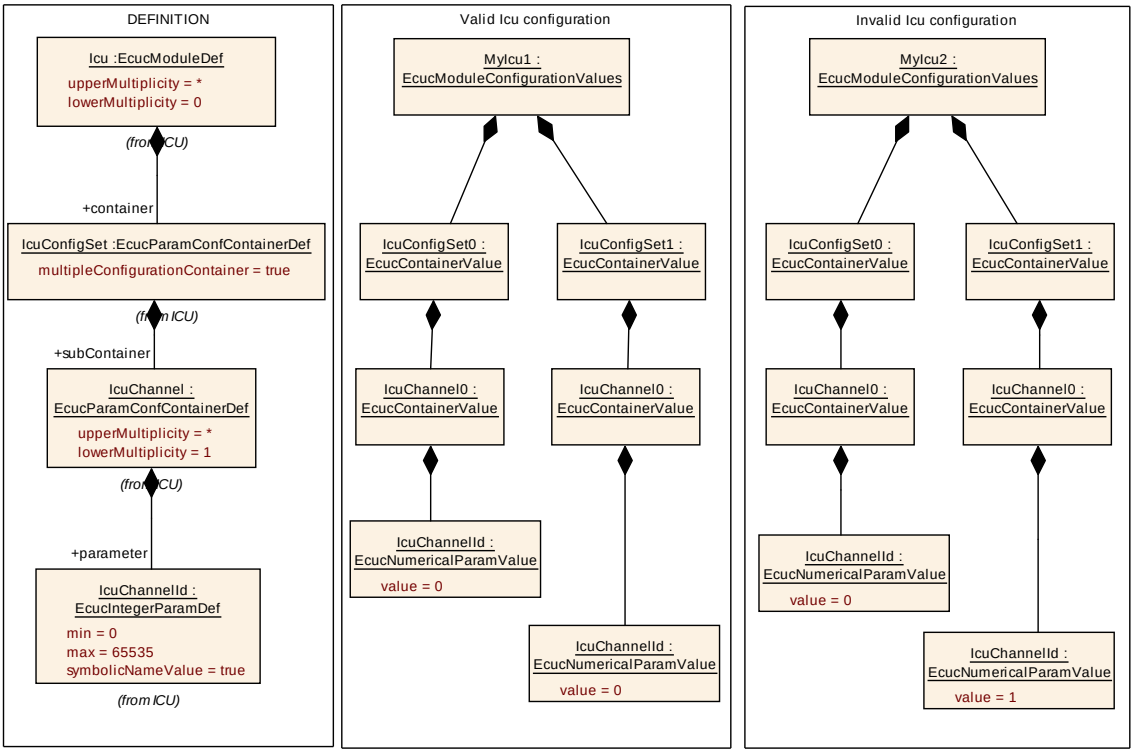

Figure 3.30: SymbolicNameValues and the generation of #defines: valid and invalid configurations

The *valid* example in figure 3.30 does lead to the following definition:

```
#define IcuConf_IcuChannel_IcuChannel0 0
```

The *invalid* example in figure 3.30 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.

3.4.6 Derived Parameters in an ECU Configuration Description

[ecuc_sws_3021] Providing the configuration value for an instance of a EcucParameterDef which has as EcucDerivationSpecification results in a NumericalParamValue.

[ecuc_sws_2125] 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 3.47 depicts the configuration description of derived parameters for example 3.24.

Example 3.47

```
<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>
```

3.4.7 Multiple Configuration Sets

As mentioned in section 3.3.3 each EcucModuleDef may specify exactly one EcucContainerValue which represents the root container of the multiple configuration set.

[ecuc_sws_3042] The definition of this EcucContainerValue has to reference the EcucParamConfContainerDef which has the attribute multipleConfigurationContainer set to true.

[ecuc_sws_3043] 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.

[ecuc_sws_3044] The shortName of the EcucContainerValue has to be the name of the configuration set, i.e. the configuration set is part of the namespace path.

[ecuc_sws_2104] 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.

[ecuc_sws_2105] 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.

[ecuc_sws_3045] 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.

[ecuc_sws_3046] Configuration tools have to check that pre-compile time and link time parameters have the same values throughout all configuration sets.

[ecuc_sws_3047] 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 3.48 depicts a EcucContainerValue with according EcucParamConfContainerDef AdcConfigSet (see example 3.8) is defined as a multiple configuration set container:

Example 3.48

```
<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 3.48 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.

4 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 4.1.

How the AUTOSAR COM-Stack is configured from an inter-module perspective is described in section 4.4.

4.1 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.

4.1.1 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 3.

4.2 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 4.1.

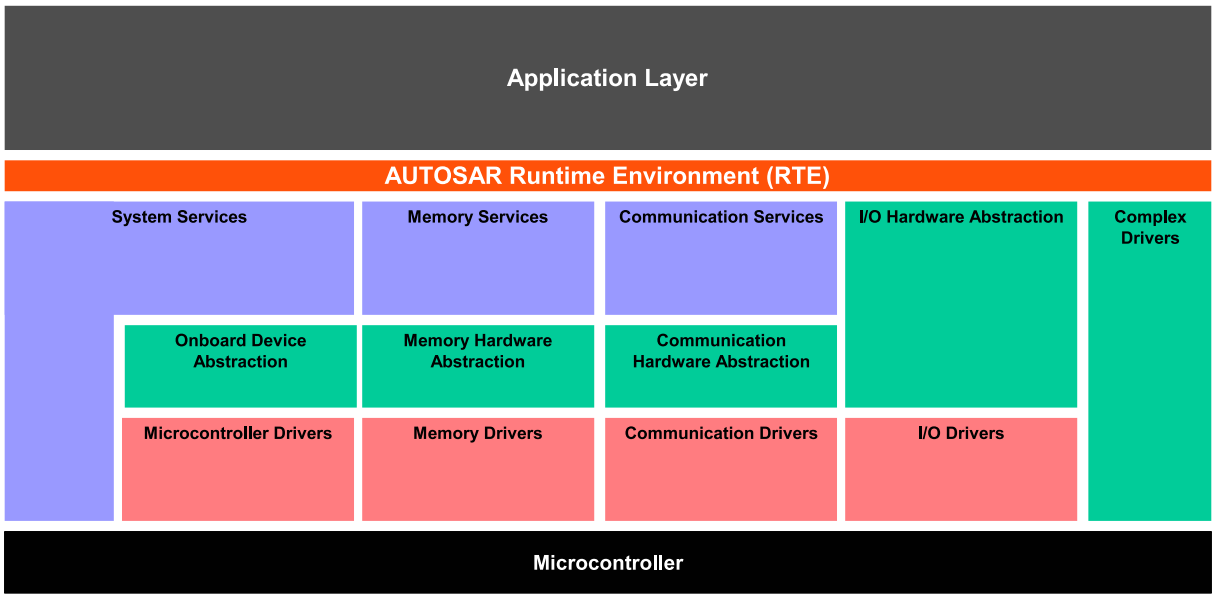

Figure 4.1: ECU Architecture Overview [18]

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.

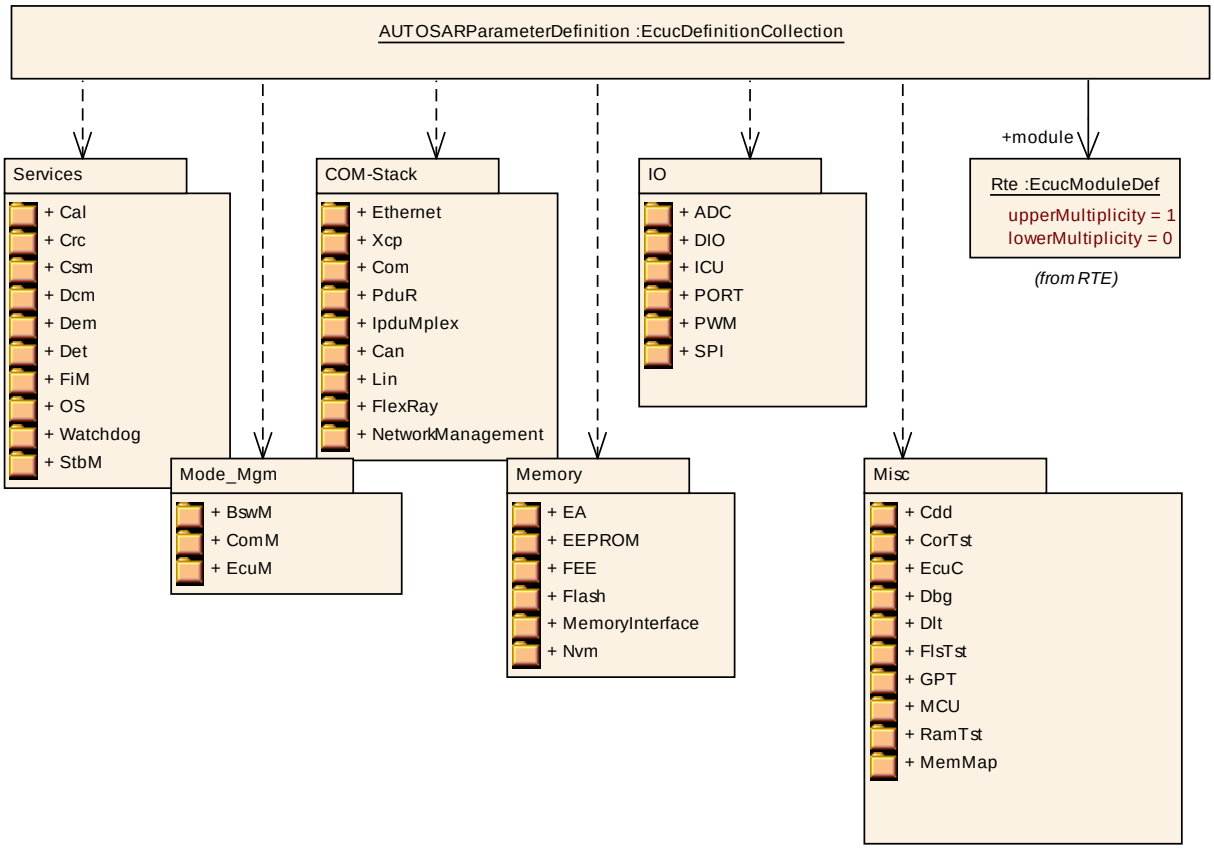

Figure 4.2: AUTOSAR Parameter Definition Overview

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 4.2.

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.

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	
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.

Module Name	Multiplicity	Scope / Dependency
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.
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.
LinSM	0..1	Configuration of the Lin State Manager module.
LinTp	0..1	Singleton descriptor for the LIN Transport Protocol.
LinTrcv	0..*	Configuration of LIN Transceiver Driver module
Mcu	0..1	Configuration of the Mcu (Microcontroler Unit) module.
MemIf	0..1	Configuration of the MemIf (Memory Abstraction Interface) module.
MemMap	0..1	Configuration of the MemMap module.
Nm	0..1	The Generic Network Management Interface module
NvM	0..1	Configuration of the NvM (NvRam Manager) 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.

Module Name	Multiplicity	Scope / Dependency
RamTst	0..1	Configuration of the RamTst module.
Rte	0..1	Configuration of the Rte (Runtime Environment) 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.
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

4.3 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**
EcucPartitionCollection	0..1	Collection of Partitions defined for this ECU.
EcucPduCollection	0..1	Collection of all Pdu objects flowing through the Com-Stack.
EcucVariationResolver	0..1	Collection of PredefinedVariant elements containing definition of values for SwSystemconst which shall be applied when resolving the variability during ECU Configuration.

4.3.1 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 `EcucPartitions` 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 4.3.

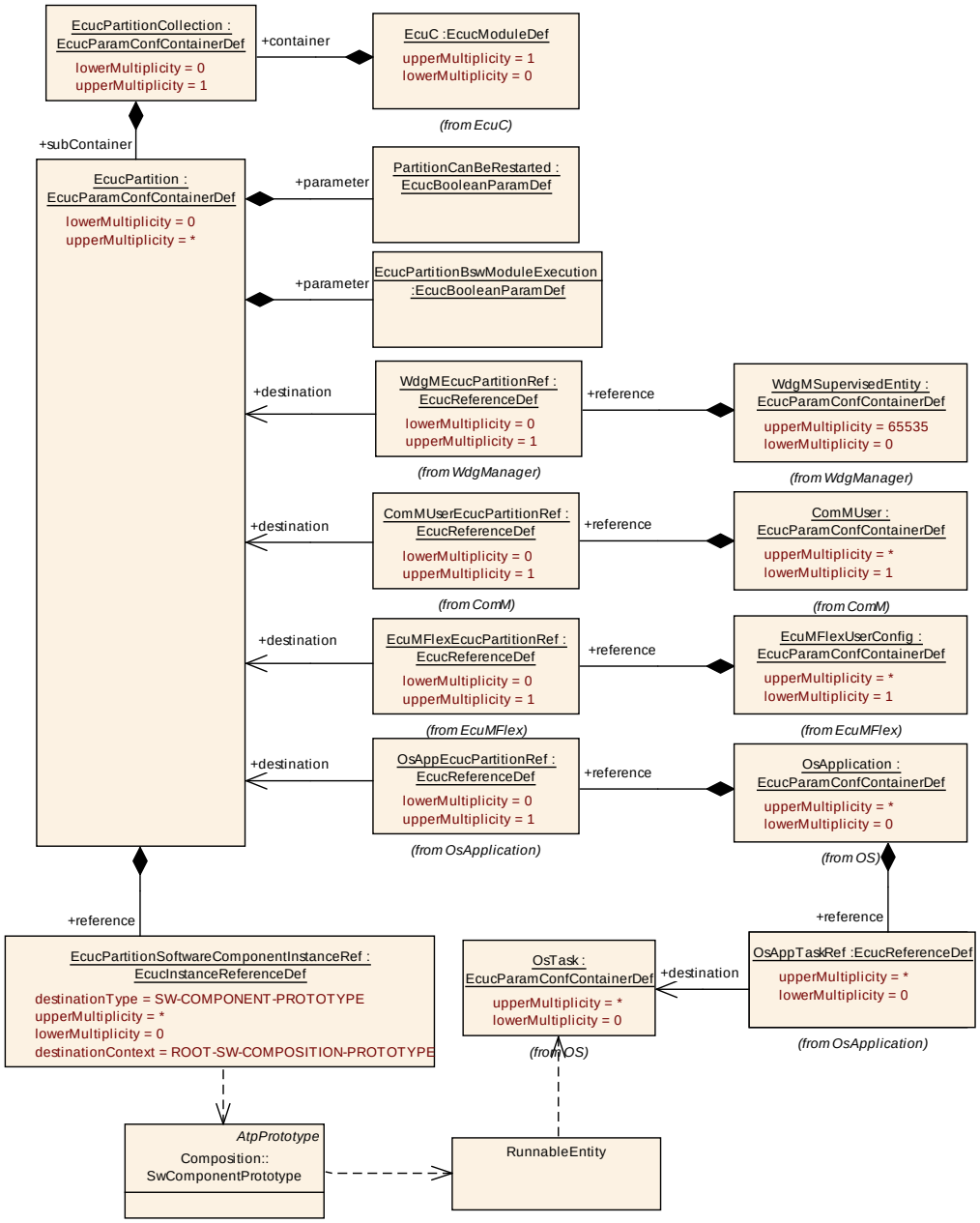

Figure 4.3: Definition of Partitions on one ECU

EcucPartitionCollection

SWS Item	[EcuC007_Conf]
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	[EcuC005_Conf]		
Container Name	EcucPartition		
Description	Definition of one Partition on this ECU. One Partition will be implemented using one Os-Application.		
Configuration Parameters			

Name	EcucPartitionBswModuleExecution [EcuC037_Conf]		
Description	Denotes that this partition will execute all BSW Modules. BSW Modules can only be executed in one partition.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

Name	EcucPartitionSoftwareComponentInstanceRef [EcuC036_Conf]		
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 [EcuC006_Conf]		
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.

4.3.2 Variant Resolver Description

In order to support the variant handling approach (see Generic Structure Template [11]) the already given values of system constants are specified in using the collection `SwSystemconstValueSet`. In the `EcuC` the applicable `SwSystemconstValueSet` elements are referenced indirectly via the `PredefinedVariant` collection.

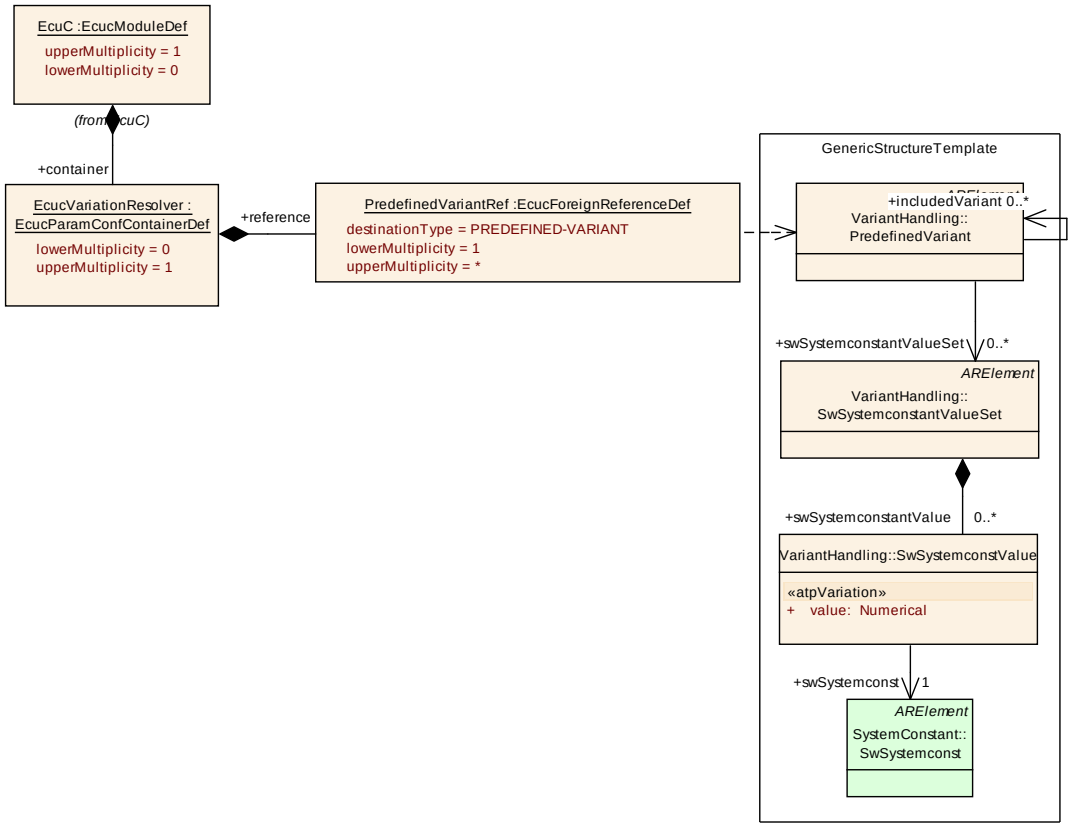

Figure 4.4: Description of Variant Resolver

EcucVariationResolver

SWS Item	[EcuC009_Conf]		
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 [EcuC010_Conf]		
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			

4.3.3 Definition of PDUs

In order to support the synchronization of Handle IDs (see section 4.4.1) 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 4.5. Each `Pdu` may be representing an actual `PduToFrameMapping`[3] from the AUTOSAR System Description[9] (the ECU Extract), therefore there is an optional reference to an element in the System Template. The reference is optional because the `PDU`s which are transported within an ECU only are not necessarily part of the ECU Extract. Especially PDUs handled by the Transport Protocol modules have no representation in the ECU Extract (There is a PDU coming over the bus which is represented by a `Pdu` object, but when the TP does the conversion and a new `Pdu` is created which then is forwarded to the upper layer. This created `Pdu` does not have a reference to a `PduToFrameMapping`).

[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.

[3]The element `PduToFrameMapping` represents the actual PDU in the specific ECU (formerly known as `PduInstance`)

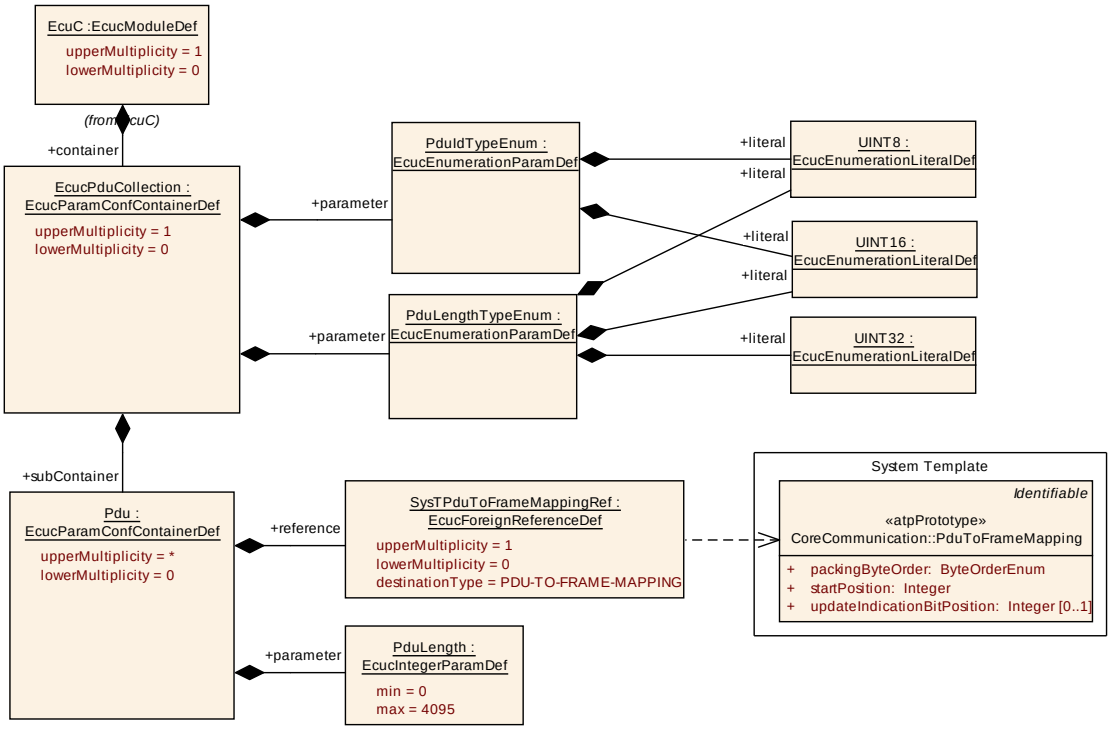

Figure 4.5: Generic Pdu Container

EcucPduCollection

SWS Item	[EcuC002_Conf]
Container Name	EcucPduCollection
Description	Collection of all Pdu objects flowing through the Com-Stack.
Configuration Parameters	

Name	PduIdTypeEnum [EcuC041_Conf]		
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 [EcuC042_Conf]		
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 segementation is used it depends on the maximum length of a segmeneted 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	[EcuC001_Conf]
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.
Configuration Parameters	

Name	PduLength [EcuC003_Conf]		
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 .. 4095		
Default Value			
Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Name	SysTPduToFrameMappingRef [EcuC004_Conf]		
Description	Optional reference to the PduToFrameMapping from the SystemTemplate which this Pdu represents.		
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			
No Included Containers			

[ecuc_sws_6030] Configuring the `PduLength` larger than the underlying layer supports results in an invalid configuration.

4.4 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.

4.4.1 Handle IDs

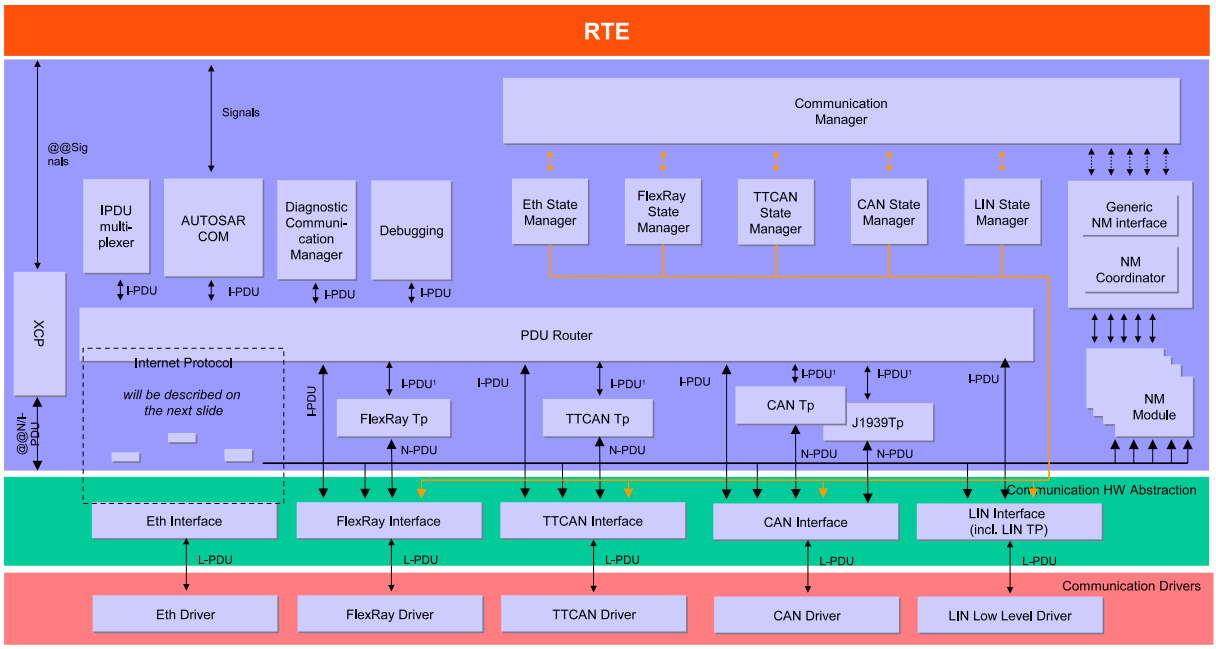

Figure 4.6: Interfaces in the COM-Stack [18]

In figure 4.6 a detailed view of the COM-Stack modules and their interaction is shown. There are several kinds of interactions between adjacent[4] modules.

4.4.1.1 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 4.4.1.2 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 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(...)`

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

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.

4.4.1.2 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 4.7 an exemplary definition of a partial Can Interface transmit configuration is shown.

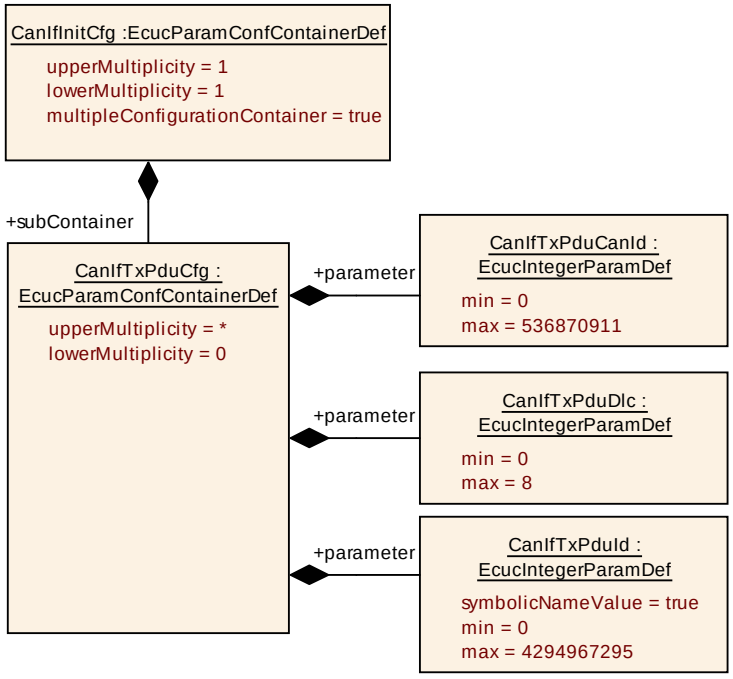

Figure 4.7: Example of Can Interface Tx configuration

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.

[ecuc_sws_2106] 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 4.4.1.4).

4.4.1.3 Agreement on Handle IDs

During the configuration of a module, information for each Pdu flowing through this module is created (see again figure 4.7: 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 4.3.3) representing information flowing through the COM-Stack. This is done by introducing a EcucReferenceDef from the "local" Pdu to the "global" Pdu.

In figure 4.8 this relationship is shown for the PduRDestPdu and the CanIfTxPduConfig.

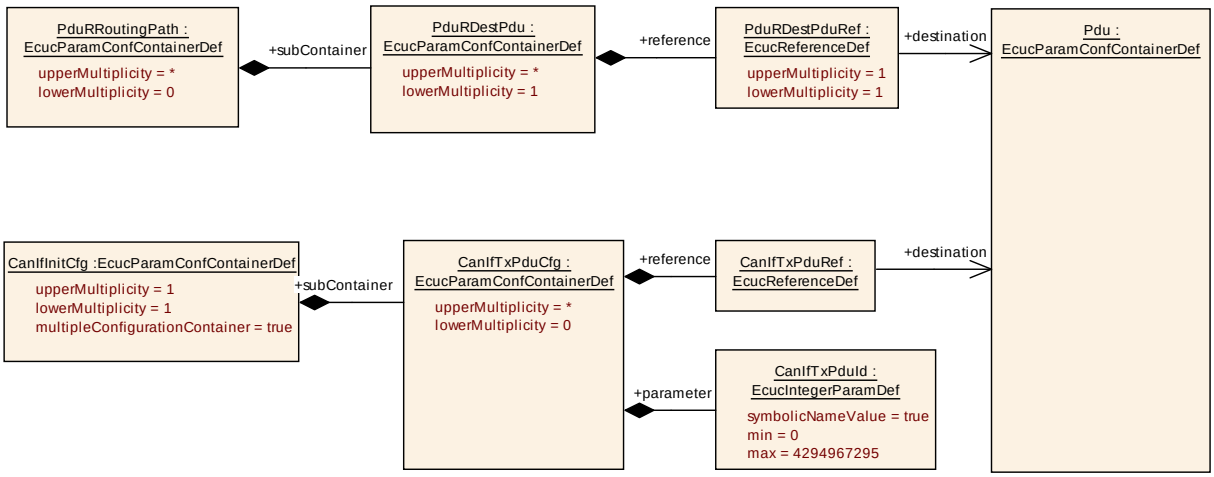

Figure 4.8: Transmission from PduR to CanIf

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 4.8 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.

4.4.1.4 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 3.3.5).

In requirement [ecuc_sws_2106] it is required that all handle Ids are represented as `symbolicNameValue = true` configuration parameters thus decoupling the value from its usage.

In requirement [ecuc_sws_2107] 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 [ecuc_sws_2108] 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 4.8 this is the parameter `CanIfTxPduId`) the code generator doing the `CanIf` will generate a `#define` in the `CanIf_cfg.h` file. The name of the define is the module short name (<MSN>) of the declaring BSW Module followed by the name of the parent container (each container does have a short name) and the value is the actual number assigned to that handle ID.

For example in `CanIf_cfg.h`:

```
#define CanIf_Tx_Pdu_2345634_985_symbol    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( CanIf_Tx_Pdu_2345634_985_symbol, 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.

4.4.2 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.

4.4.2.1 Tx from Com to CanIf

In the example in figure 4.9 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`.

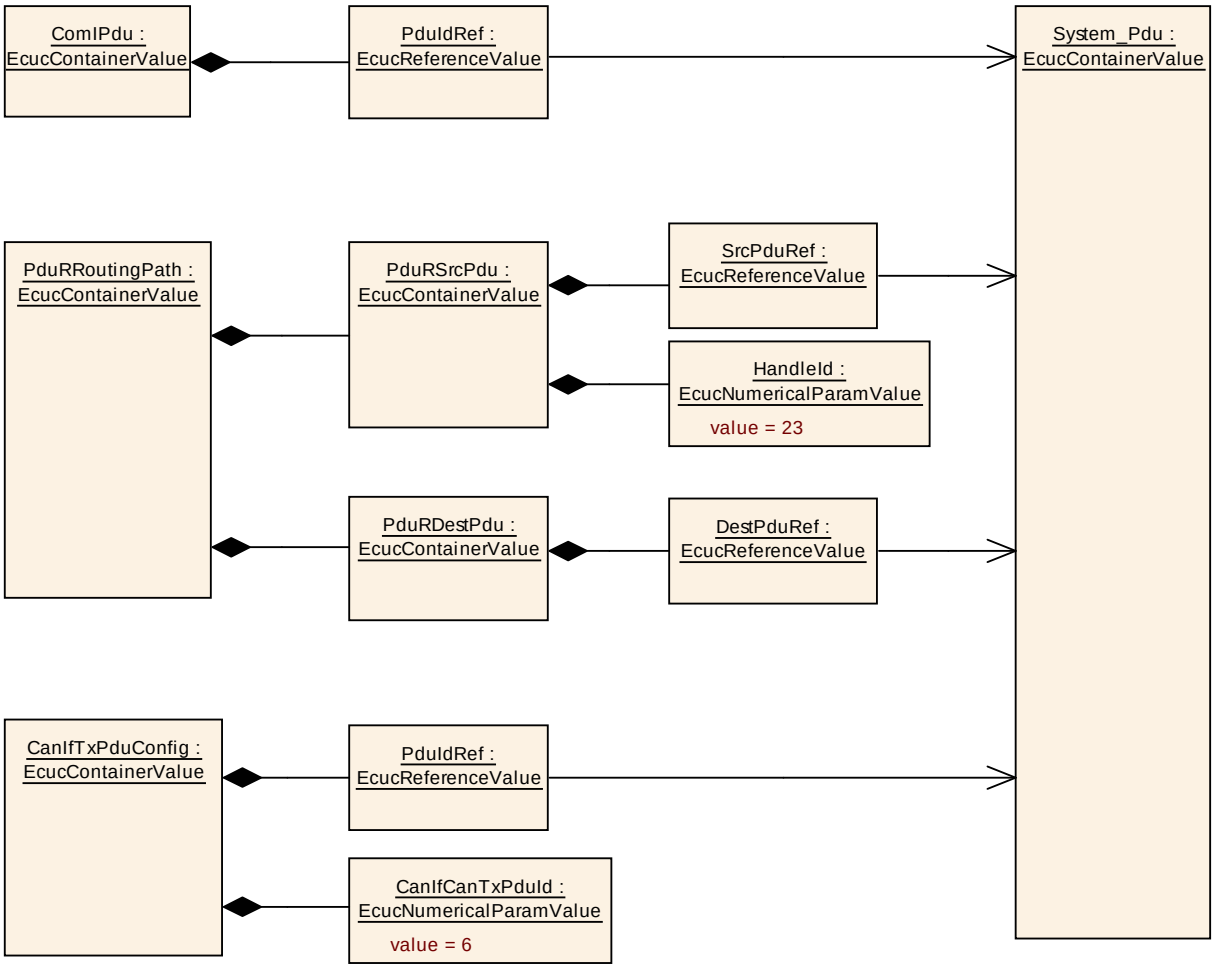

Figure 4.9: Tx from Com to CanIf example

4.4.2.2 Rx from CanIf to Com

In the example in figure 4.10 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.

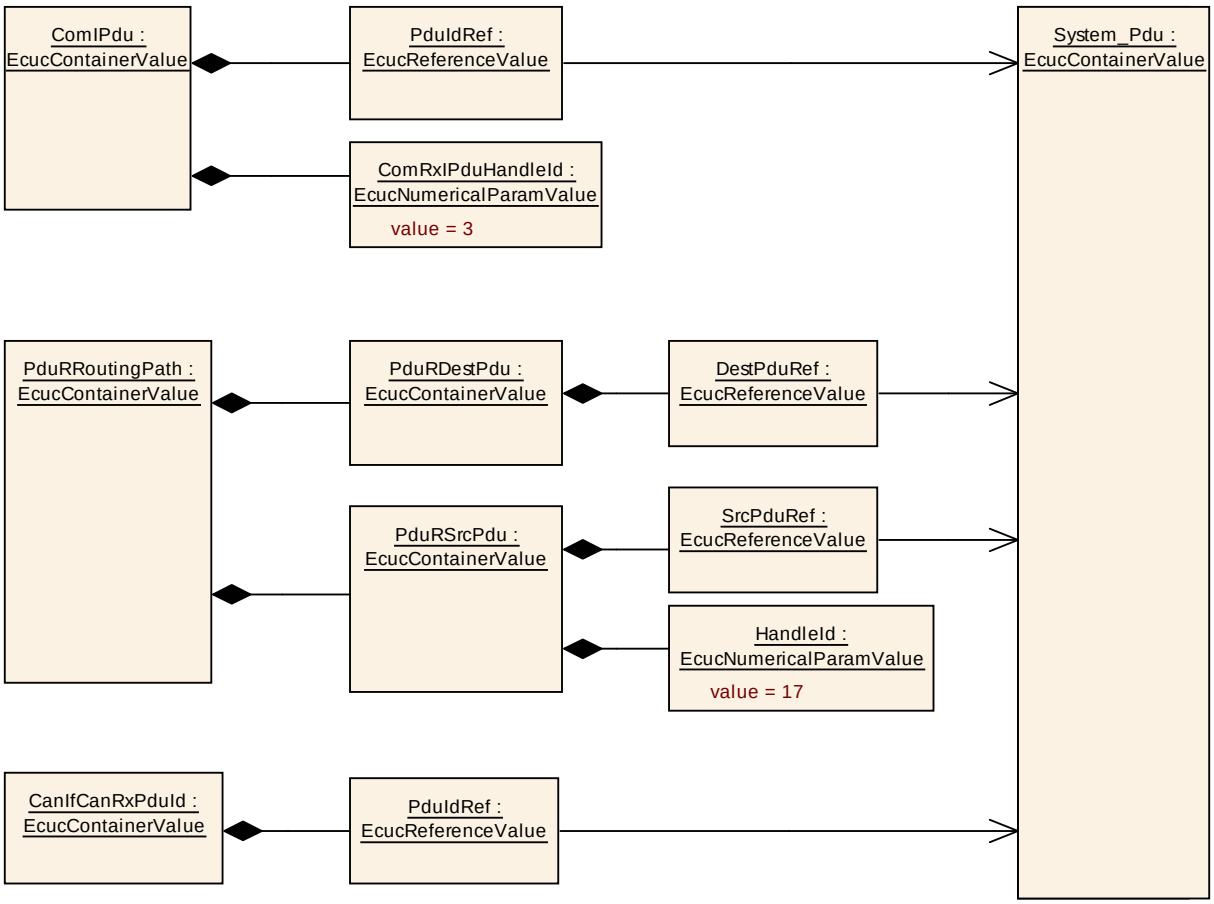

Figure 4.10: Rx from CanIf to Com example

4.4.2.3 Gateway from CanIf to FrIf

In the example in figure 4.11 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.

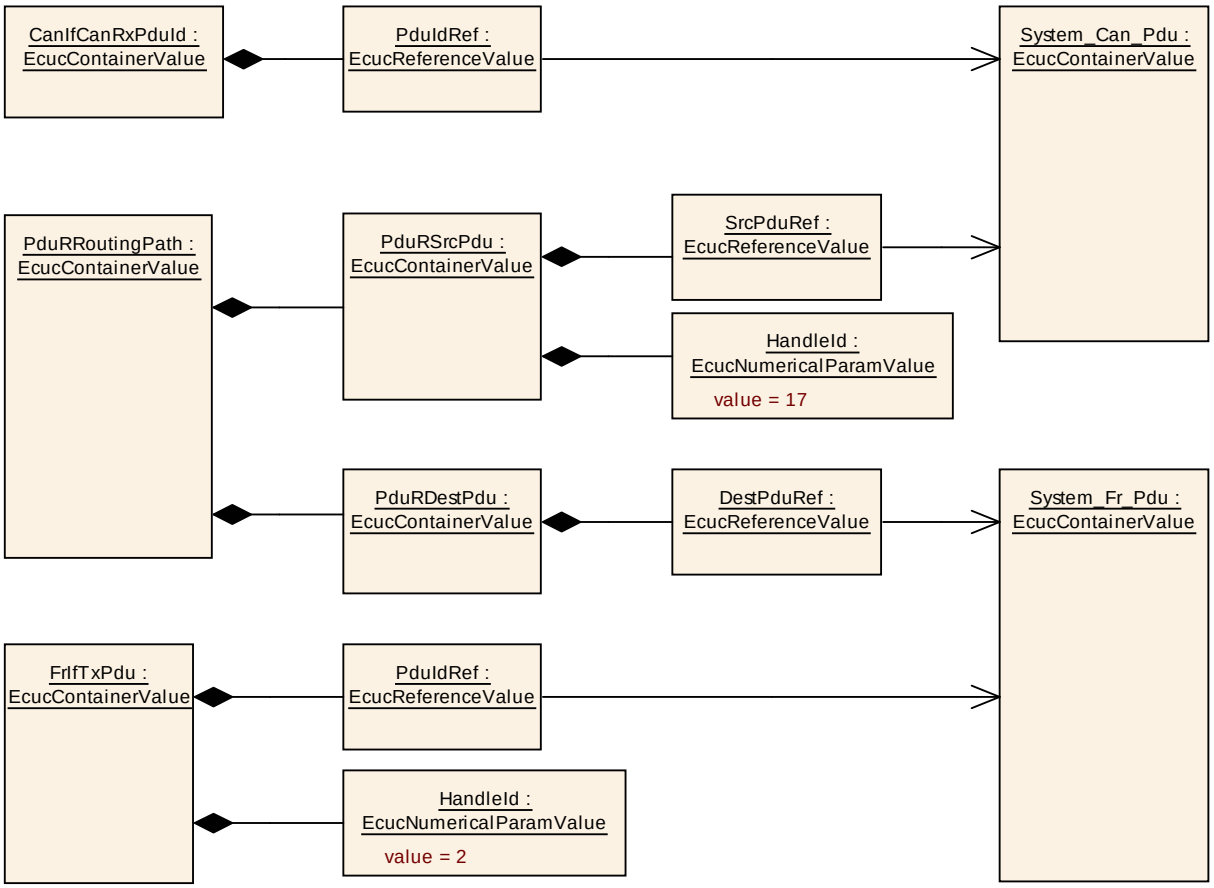

Figure 4.11: Gateway from CanIf to FrIf example

4.4.3 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 4.4.1) where individual *PDU Handle ID*s are configured per module.

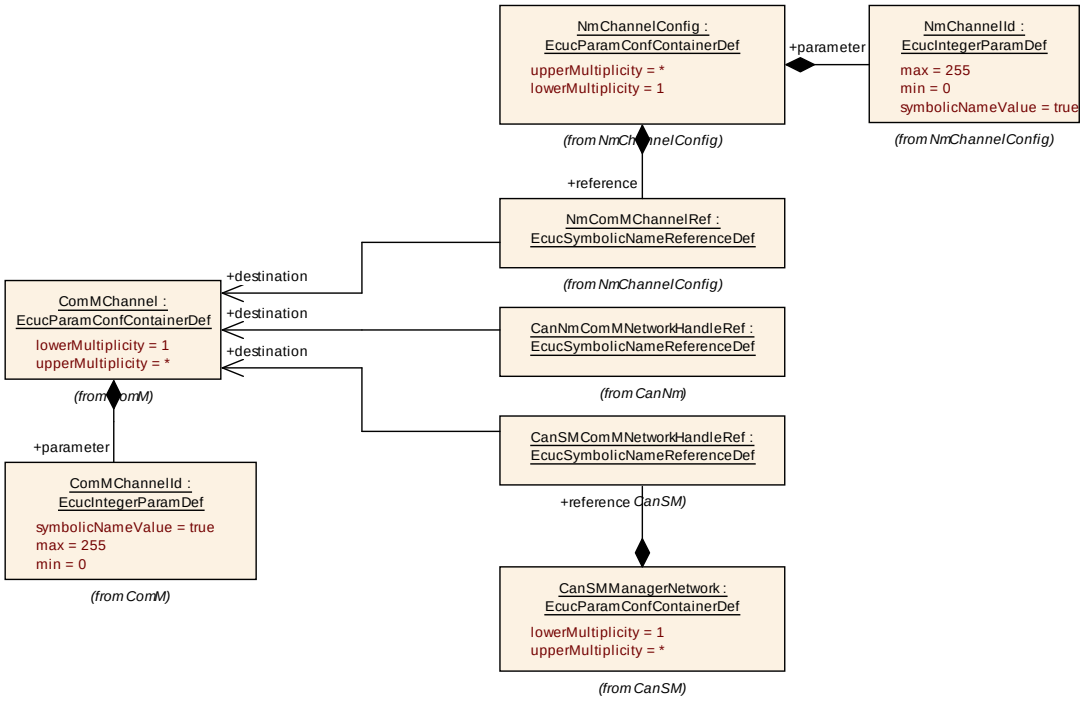

Figure 4.12: Example of Channel interaction between ComM, Can and Nm

In figure 4.12 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.

4.5 CDD module

The CDD module describes the minimal requirements that are necessary for the configuration of a Complex Device Driver with respect to the surrounding standardized BSW modules.

[ecuc_sws_6031] If a complex device 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 5.1.

As defined in [ecuc_sws_6001] 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 device driver is always "CDD".

[ecuc_sws_6036] To distinguish module definitions of complex device drivers from each other the package structure shall be used.

[ecuc_sws_6037] The `apiServicePrefix` attribute of a complex device 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 device driver named "MyCdd". The VSMD of this complex device driver has to be derived from the CDD StMD. The `shortName` of the module definition of this complex device driver has to be equal to "CDD". The `apiServicePrefix` attribute is mandatory for the VSMD of this complex device driver and has to be equal to "MyCdd".

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.
CddEcucPartitionInteraction	0..1	This optional container holds the partition interaction configuration.

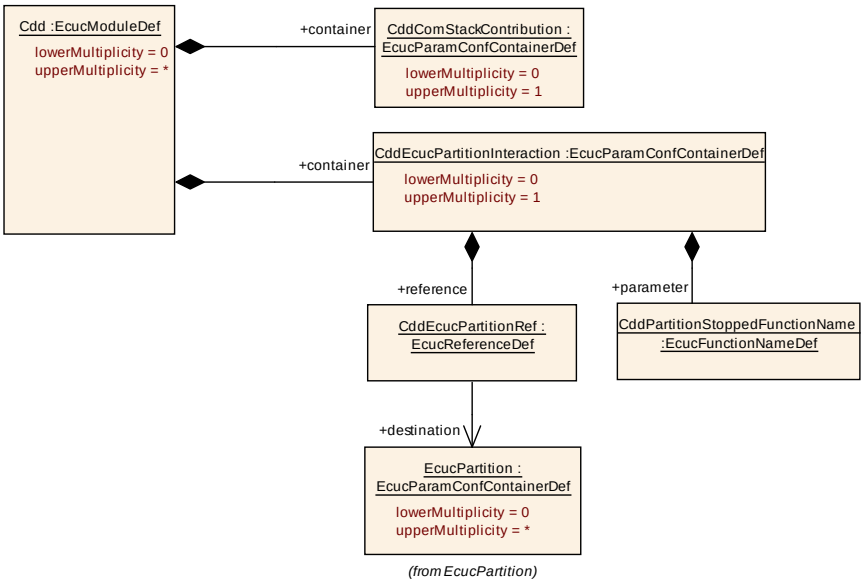

Figure 4.13: Cdd Module

CddEcucPartitionInteraction

SWS Item	[EcuC038_Conf]
Container Name	CddEcucPartitionInteraction
Description	This optional container holds the partition interaction configuration.
Configuration Parameters	

Name	CddEcucPartitionRef [EcuC039_Conf]		
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 [EcuC040_Conf]		
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	[EcuC017_Conf]
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 Device 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 Device 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 Device 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 Device 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 Device Driver that serves as the UpperLayer of the Pdu Router module.

The following sections describe particular COM stack modules and the interaction with Complex Device Drivers.

4.5.1 Pdu Router

In the AUTOSAR COM Stack upper and lower layer Complex Device 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 Device Driver implies a reference to the global Pdu and the definition of a HandleId. Figure 4.14 shows an example of a Complex Device Driver between the CanIf and the PduR and one Complex Device Driver above the PduR.

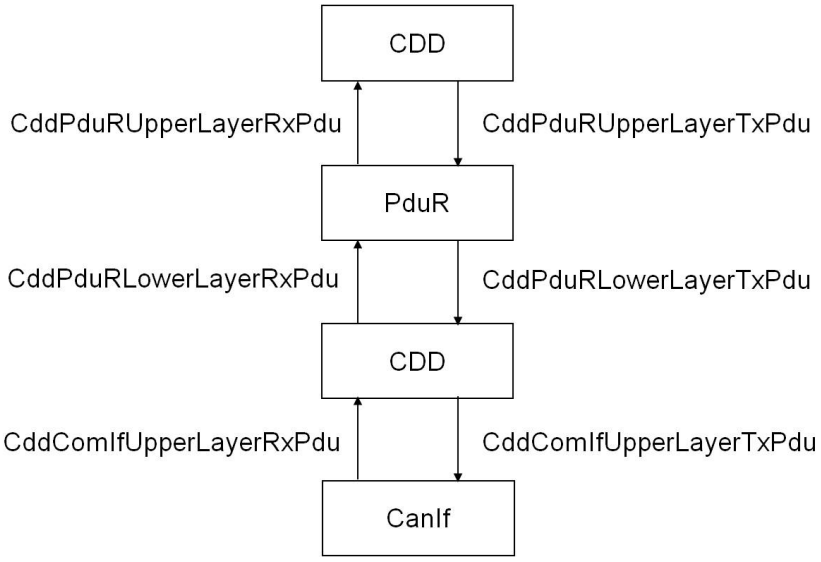

Figure 4.14: CDD Example

Figure 4.15 shows the CDD contribution in the configuration model.

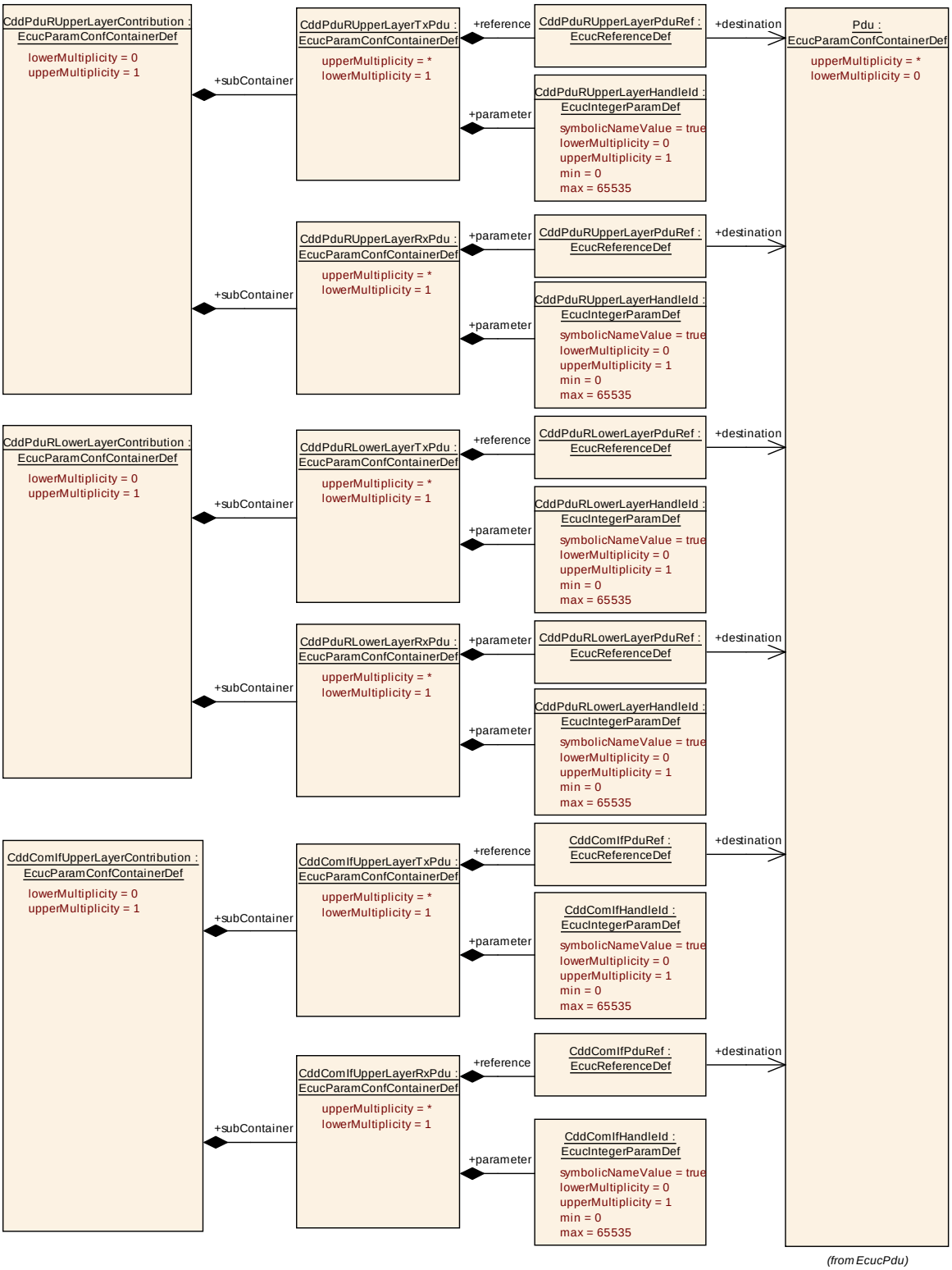

Figure 4.15: PduR and Com Interface contribution

CddPduRUpperLayerContribution

SWS Item	[EcuC026_Conf]	
Container Name	CddPduRUpperLayerContribution	
Description	Parameters that are necessary for the configuration of a Complex Device Driver that serves as the UpperLayer of the Pdu Router module.	
Configuration Parameters		
Included Containers		
Container Name	Multiplicity	Scope / Depedency
CddPduRUpperLayerRx Pdu	1..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddPduRUpperLayerTx Pdu	1..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddPduRLowerLayerContribution

SWS Item	[EcuC022_Conf]	
Container Name	CddPduRLowerLayerContribution	
Description	Parameters that are necessary for the configuration of a Complex Device Driver that serves as the LowerLayer of the Pdu Router module.	
Configuration Parameters		
Included Containers		
Container Name	Multiplicity	Scope / Depedency
CddPduRLowerLayerRx Pdu	1..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddPduRLowerLayerTx Pdu	1..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddPduRUpperLayerTxPdu

SWS Item	[EcuC027_Conf]	
Container Name	CddPduRUpperLayerTxPdu	
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.	
Configuration Parameters		

Name	CddPduRUpperLayerHandleId [EcuC029_Conf]		
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 [EcuC028_Conf]		
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	[EcuC043_Conf]		
Container Name	CddPduRUpperLayerRxPdu		
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.		
Configuration Parameters			

Name	CddPduRUpperLayerHandleId [EcuC045_Conf]		
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 [EcuC044_Conf]		
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	[EcuC023_Conf]
Container Name	CddPduRLowerLayerTxPdu
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRLowerLayerHandleId [EcuC025_Conf]		
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 [EcuC024_Conf]		
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	[EcuC046_Conf]
Container Name	CddPduRLowerLayerRxPdu
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
Configuration Parameters	

Name	CddPduRLowerLayerHandleId [EcuC048_Conf]		
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 [EcuC047_Conf]		
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			

4.5.2 COM Interface modules

A Complex Device 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 Device Drivers are allowed to access the interface of these modules. Equal to the `PduRContribution` the `CddComIfUpperLayerContribution` of the Complex Device Driver implies a reference to the global Pdu and the definition of a HandleId. Figure 4.15 shows the CDD contribution in the configuration model.

CddComIfUpperLayerContribution

SWS Item	[EcuC018_Conf]	
Container Name	CddComIfUpperLayerContribution	
Description	Parameters that are necessary for the configuration of a Complex Device Driver that serves as the UpperLayer of the Com Interface module.	
Configuration Parameters		
Included Containers		
Container Name	**Multiplicity**	**Scope / Depedency**
CddComIfUpperLayerRx Pdu	1..*	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.
CddComIfUpperLayerTx Pdu	1..*	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.

CddComIfUpperLayerTxPdu

SWS Item	[EcuC019_Conf]		
Container Name	CddComIfUpperLayerTxPdu		
Description	This container specifies Tx PDUs that are exchanged between the CDD and the standardized BSW module.		
Configuration Parameters			

Name	CddComIfHandleId [EcuC021_Conf]		
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 [EcuC020_Conf]		
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	[EcuC049_Conf]		
Container Name	CddComIfUpperLayerRxPdu		
Description	This container specifies Rx PDUs that are exchanged between the CDD and the standardized BSW module.		
Configuration Parameters			

Name	CddComIfHandleId [EcuC051_Conf]		
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 [EcuC050_Conf]		
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			

4.5.3 Communication Manager

Complex Device Drivers are allowed to access the Communication Manager on the upper layer. The contribution of the lower layer Complex Device Driver implies for each channel a reference to to the unique handle to identify one certain network handle in the ComM configuration.

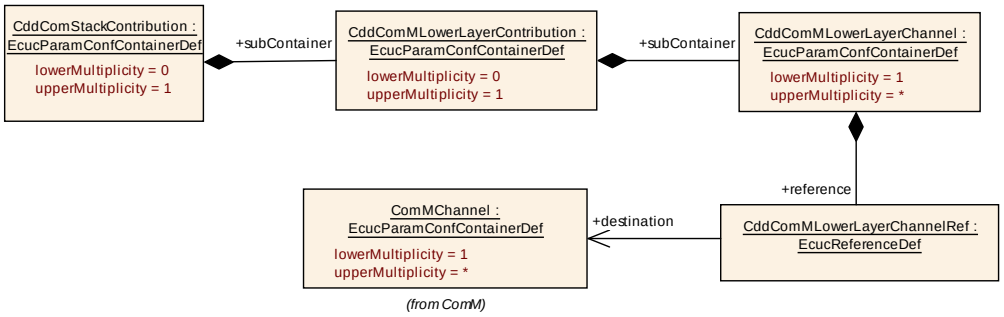

Figure 4.16: ComM lower layer contribution

CddComMLowerLayerContribution

SWS Item	[EcuC030_Conf]	
Container Name	CddComMLowerLayerContribution	
Description	Parameters that are necessary for the configuration of a Complex Device 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	[EcuC031_Conf]		
Container Name	CddComMLowerLayerChannel		
Description	This container contains the network specific parameters.		
Configuration Parameters			

Name	CddComMLowerLayerChannelRef [EcuC032_Conf]		
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			

4.5.4 Generic Network Management

Complex Device Drivers are allowed to access the GenericNm module on the upper layer. The contribution of the lower layer Complex Device Driver implies in each `NmChannel` configuration a reference to the respective NM channel handle.

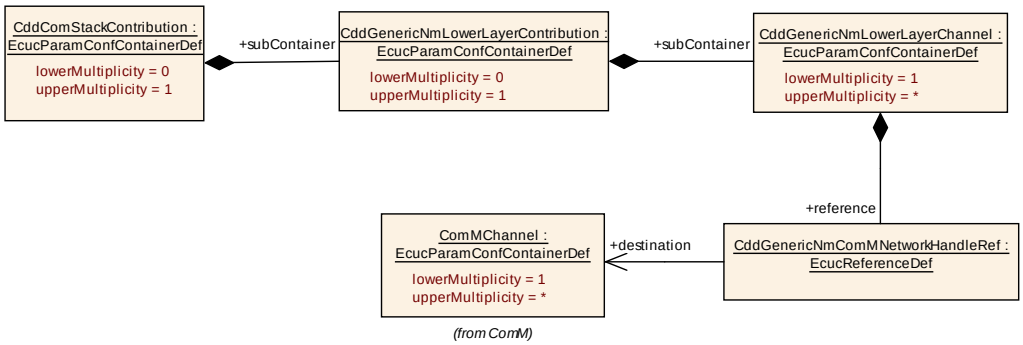

Figure 4.17: GenericNm lower layer contribution

CddGenericNmLowerLayerContribution

SWS Item	[EcuC033_Conf]	
Container Name	CddGenericNmLowerLayerContribution	
Description	Parameters that are necessary for the configuration of a Complex Device 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	[EcuC034_Conf]		
Container Name	CddGenericNmLowerLayerChannel		
Description	NM Channel specific configuration parameters.		
Configuration Parameters			

Name	CddGenericNmComMNetworkHandleRef [EcuC035_Conf]		
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

4.6 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.

To avoid this situation, in both cases (manually or automatically calculated) the same rules shall be applicable that resolve float values into tick units.

These rules shall be used

- manually by the user prior to the call of the code generator (if ticks are specified and expected)
- automatically (implicitly) by the code generators (if floats are specified and expected)

[ecuc_sws_7000] Calculation formula for min values:
IF (((Required Min Time) MOD (Main Function Period Time)) != 0)
THEN Number of Ticks = INT (Required Min Time / Main Function Period Time) +1
ELSE Number of Ticks = INT (Required Min Time / Main Function Period Time)

[ecuc_sws_7001] Calculation formula for max or other values:
Number of Ticks = INT (Required Time / Main Function Period Time)

[ecuc_sws_7002] Restrictions in case of generator usage:

- An error shall be generated if the calculated number of ticks is less than 1 (except for min values) since they are anyway above its limit.
- A warning shall be generated if the calculated number of ticks is not dividable without rest.

Remark: Due to non specified implementation constraints its up to the vendor to define additional checks if necessary.

Example 1 (Main Period = 1.5 ms = 0.0015 sec):

Parameter	Val [ms]	Val [sec]	Calc Val [ticks]	Calc Val [ms]	Message
Avg Time	10	0.01	6.667	9	warning
Min Time	6	0.006	4.0	6	none

Example 2 (Main Period = 10 ms = 0.01 sec):

Parameter	Val [ms]	Val [sec]	Calc Val [ticks]	Calc Val [ms]	Message
Max Time	8	0.008	0.8	0	error
Min Time	5	0.005	0.5	1	warning

4.7 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.

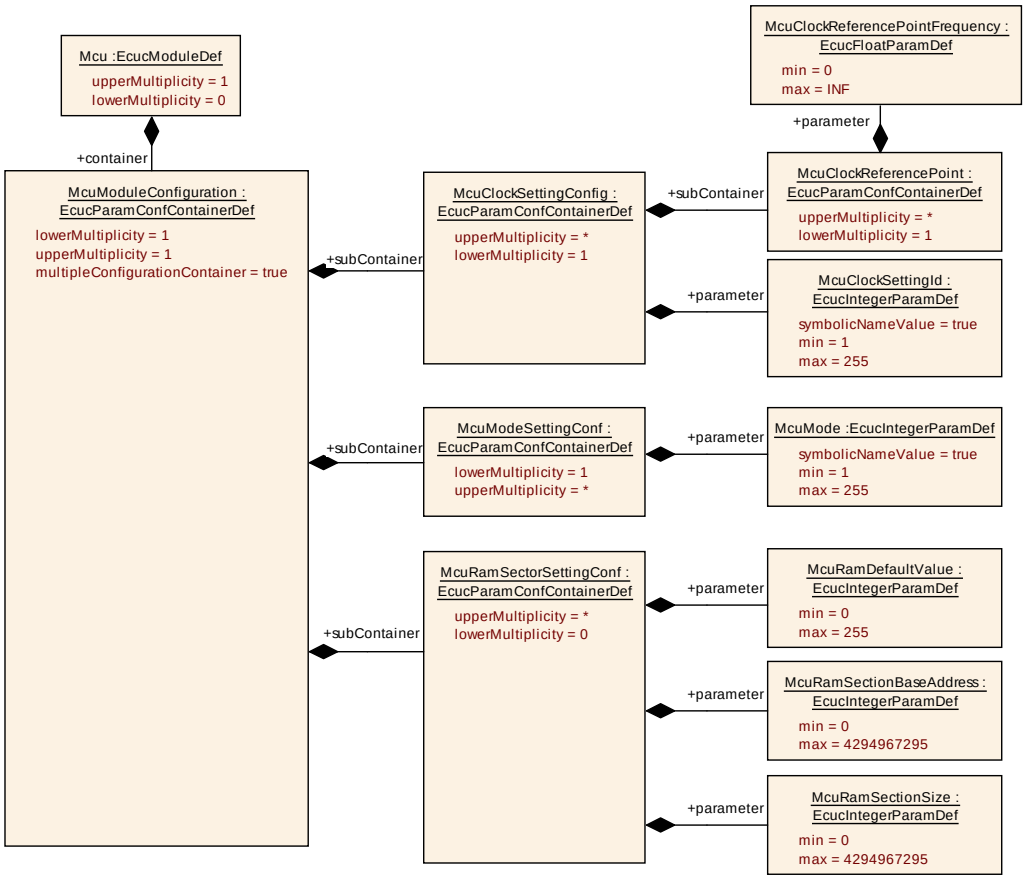

Figure 4.18: MCU Setting

McuClockReferencePoint

SWS Item	[MCU174_Conf]
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 multiplictiy is 1, as even in the simpliest case (only one frequency is used), there is one frequency to be defined.
Configuration Parameters	

Name	McuClockReferencePointFrequency [MCU175_Conf]		
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: Module		
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 4.19). Further adjustments on the clock frequency are under the responsibility of the specific user peripheral's driver.

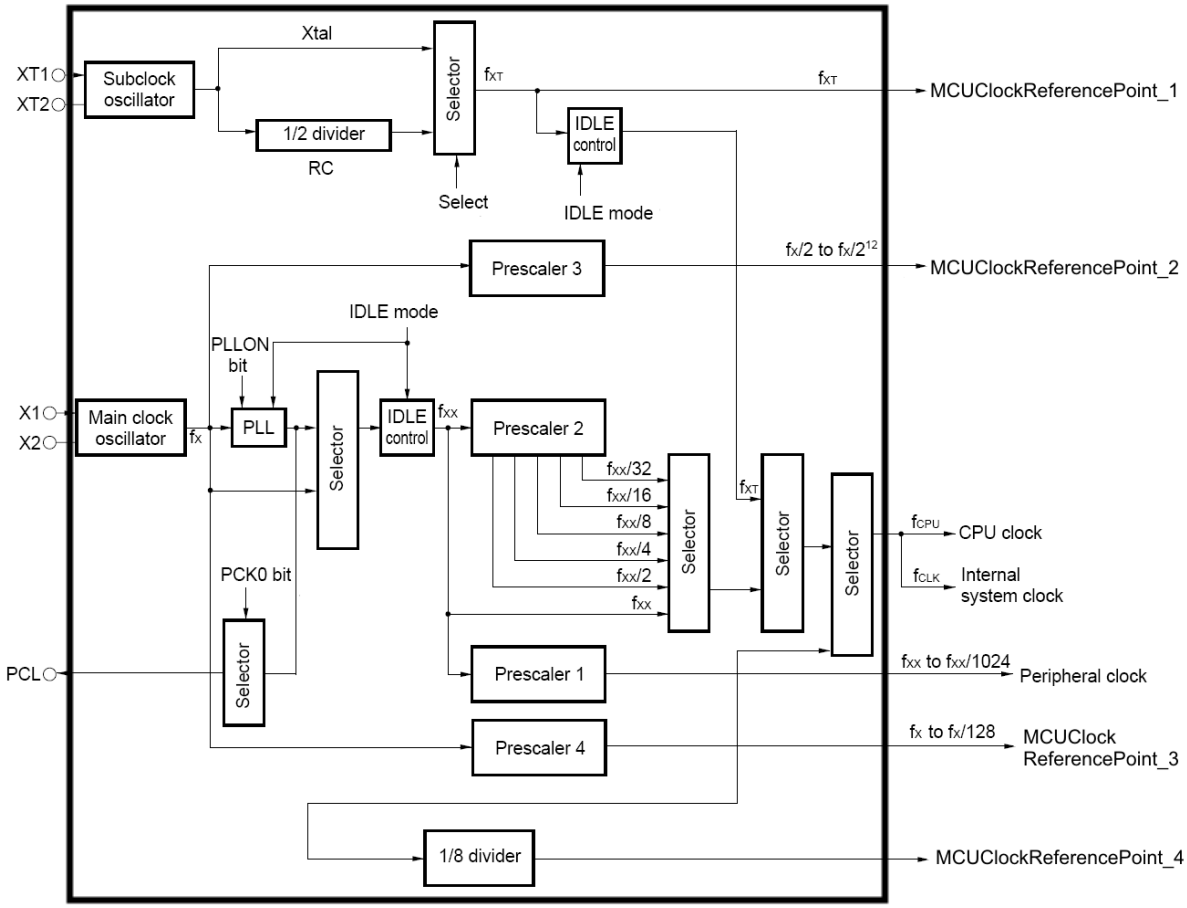

Figure 4.19: Clocktree example

5 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.

5.1 Deriving vendor specific module definitions from standardized module definitions

The following rules must be followed when generating the `Vendor Specific Module Definition` (abbreviated with VSMD in this chapter) from the `Standardized Module Definition` (abbreviated StMD in this chapter). The basic relationship between these two kinds of parameter definitions are depicted in figure 5.1.

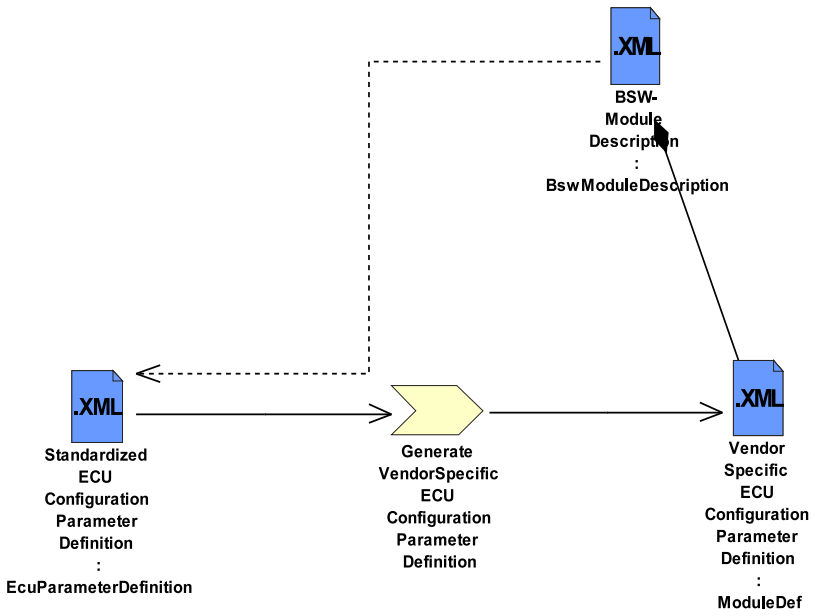

Figure 5.1: Generating Vendor Specific Module Definitions (per module)

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 5.2 shows an example with pure vendor specific containers and references (marked with red color).

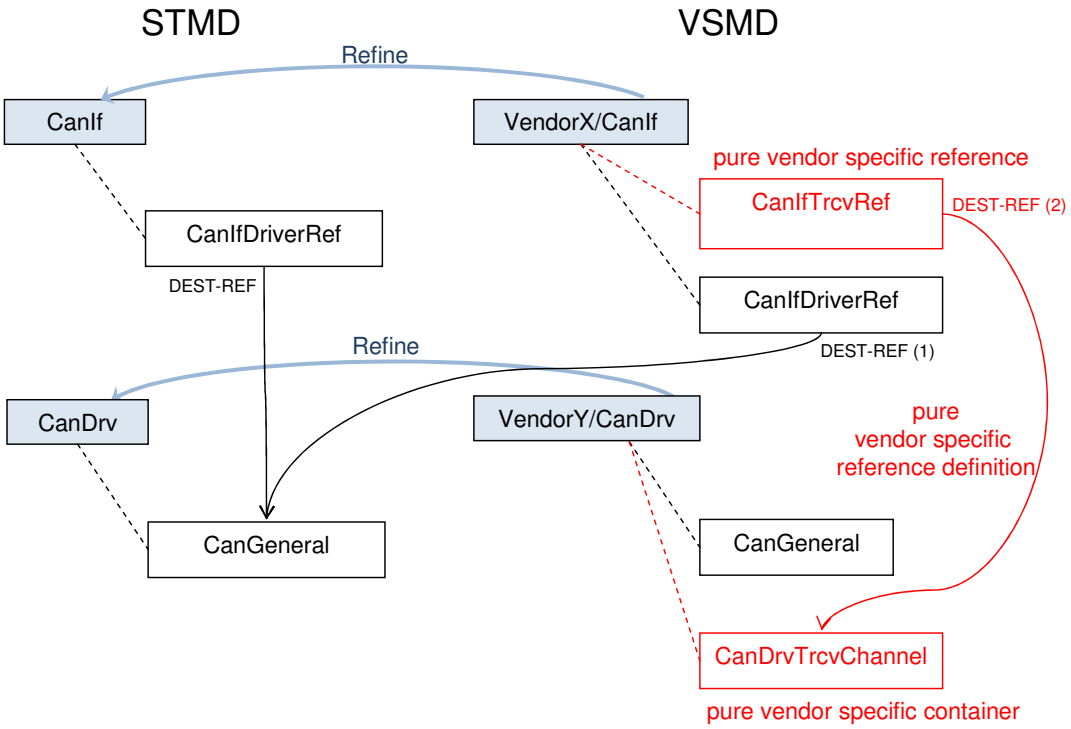

Figure 5.2: Relation between STMD and VSMD

In example 5.1 the StMD of the two modules of figure 5.2 is defined.

Example 5.1

```
<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 5.2 the VSMD of a CanDrv implementation is shown. Here a vendor specific container CanDrvTrcvContainer has been introduced.

Example 5.2

```
<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 5.3 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 5.3

```
<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>
```

[ecuc_sws_6038] The following rules shall be checked by tools that validate whether a SW module implementation conforms to its AUTOSAR specification.

- **[ecuc_sws_1001]** 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.
- **[ecuc_sws_6001]** The `shortName` of a VSMD module shall be the same as the `shortName` of the StMD.
- **[ecuc_sws_6049]** The supported ModuleDef.supportedConfigVariant shall be restricted in the VSMD to the actually supported configuration variants of this implementation. This can be a subset of the ModuleDef.supportedConfigVariant in the StMD.
- **[ecuc_sws_6050]** If the supported ModuleDef.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. ModuleDef.supportedConfigVariant of a VSMD should not contain VariantPostBuild. For compatibility reasons, if it contains VariantPostBuild, it is considered as VariantPostBuildLoadable.
- **[ecuc_sws_6051]** The implementationConfigClass in VSMD must be the same as in StMD with respect to the selected subset defined by the actually implemented ModuleDef.supportedConfigVariant if the scope of the ConfigParameter or ConfigReference in StMD is ECU.

  AUTOSAR ECU Processor Tools which processes ECU Configuration Values shall report an error if the `EcucConfigurationVariantEnum` in the VSMD

of an processed ECU Configuration Values requires a later `EcucConfigurationVariant` than it (the particular procesor) is able to implement.

- **[ecuc_sws_6003]** 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 5.4 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".
- **[ecuc_sws_6015]** 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 5.4 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.
- **[ecuc_sws_6046]** 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 5.4

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 `ContainerDefs` and `ParameterTypes` and `ConfigReferences` defined within the `EcucModuleDef` in the StMD, it holds:

- **[ecuc_sws_6007]** Elements defined in the StMD must be present in the VSMD and must not be omitted, even if the `upperMultiplicity` of an element in the VSMD is set to 0.
- **[ecuc_sws_6008]** The `lowerMultiplicity` of an element in the VSMD must be bigger or equal and the `upperMultiplicity` must be equal or less than in the StMD:
  StMD lowerMult $\leq$ VSMD lowerMult $\leq$ VSMD upperMult $\leq$ StMD upperMult.
- **[ecuc_sws_1034]** Elements taken over from the StMD to the VSMD must use exactly the same `shortName`, since the short name identifies the element. This holds for container definitions and individual parameters.
- **[ecuc_sws_1035]** 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.
- **[ecuc_sws_1005]** 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.
- **[ecuc_sws_1006]** The `defaultValue` attributed may be changed (or added, if missing).
- **[ecuc_sws_1007]** 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.
- **[ecuc_sws_6045]** 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.
- **[ecuc_sws_1009]** For derived parameters defined in the StMD, the values of the `calculationFormula` and `calculationLanguage` may change in the VSMD.
- **[ecuc_sws_1011]** Additional vendor specific `choices` (i.e. aggregated `ParamConfContainerDefs`) may be added for `EcucChoiceReferenceDef` in the VSMD.
- **[ecuc_sws_1013]** Additional vendor specific references may be added for `ChoiceReferenceParamDef` in the VSMD.

- **[ecuc_sws_1014]** Additional vendor specific parameter definitions (using `ParameterTypes`), container definitions and references shall be added to the VSMD according to the alphabetical order.
- **[ecuc_sws_1015]** The `origin` attribute of vendor specific additional elements shall contain the name of the vendor that defines the element.
- **[ecuc_sws_2084]** For an `EcucEnumerationParamDef` from the StMD there can be additional `EcucEnumerationLiteralDef` added in the VSMD if the scope of the `EcucEnumerationParamDef` is local.
- **[ecuc_sws_6002]** An `EcucEnumerationLiteralDef` from an `EcucEnumerationParamDef` can be removed in the VSMD if the `scope` of the `EcucEnumerationParamDef` is defined as `local`[1].
- **[ecuc_sws_5002]** Induce VSMD in to the StMD in a simplified manner, so that the configuration can be carried out without any disarray.
- **[ecuc_sws_5003]** The `desc` in VSMD can be used to specify detailed information about the respective parameter.

Figure 5.3 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.

[ecuc_sws_6009] 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).

[ecuc_sws_6010] If the `lowerMultiplicity` of a parameter definition is bigger than 0, the parameter must exist in the parameter Value description (6).

[ecuc_sws_6011] Missing parameters shall be detected by tools (8).

[ecuc_sws_6012] Parameters without parameter definitions shall be ignored by tools (9).

[1] For the definition of the `scope` please refer to the 'General Requirements on Basic Software Modules' [4] [BSW00394]

[ecuc_sws_6013] The number of parameters in the ECUC Value description shall not exceed the `upperMultiplicity` of the parameter definition in the VSMD (7).

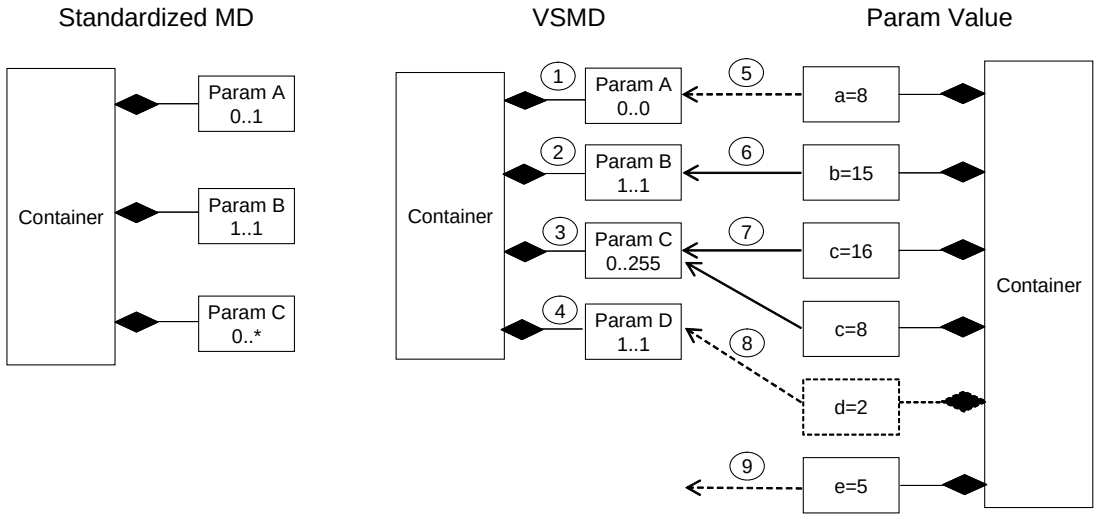

Figure 5.3: Relation between standardized module definition and vendor specific module definition

Example 5.5 depicts the usage of VSMD in case of parameter definition.

Example 5.5

```
<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 5.6 depicts the usage of VSMD in case of parameter description.

Example 5.6

```
<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>
```

5.2 Rules for building the Base ECU configuration

Chapter 2.3.3 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:

- **[ecuc_sws_1016]** For mandatory containers, parameters and references (i.e. with `lowerMultiplicity` > 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.

- **[ecuc_sws_1017]** 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.

- **[ecuc_sws_1018]** For containers with variable multiplicity (i.e. `lowerMultiplicity` $<$ `upperMultiplicity`), 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.

- **[ecuc_sws_1019]** For the setting of the initial values for configuration parameters, the following sources shall be used (in decreasing priority)

  - **[ecuc_sws_1020]** 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 must be included in the base ECU configuration Value description and shall not be changed in later editing.

  - **[ecuc_sws_1021]** Values derived from the ECU extract of the system configuration. 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.

- **[ecuc_sws_1022]** 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.
- **[ecuc_sws_1023]** 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.

[ecuc_sws_1024] If no initial value can be derived from any of these sources for a parameter, the parameter will be generated without an initial value.

[ecuc_sws_4004] 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`.

5.3 Rules for Configuration Editors

Chapter 2.3.4 describes the methodology for editing configuration parameters. The following rules apply for a configuration editor supporting the methodology:

- **[ecuc_sws_4001]** The ECU Configuration Editor shall be able to generate the files containing parameters affecting other parameters. That is the xml-files of type `Module Configurations` described in chapter 2.3.4.
- **[ecuc_sws_4002]** The ECU Configuration Editor shall be able to perform a simple merge of `ECU Configuration Value descriptions` as described in chapter 2.3.4.
- **[ecuc_sws_4003]** 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.
- **[ecuc_sws_4005]** The ECU Configuration Editor shall be able to generate and import files describing a specific aspect of the configuration of a module. The files that shall be generated and imported are `ModuleConfigurations`. The rationale for this is to support post-build time loadable configuration from a Configuration Management perspective. See chapter 2.3.2.3.1.

- **[ecuc_sws_6071]** The ECU Configuration Editor shall be able to read parameter values in any ordering according to the input.
- **[ecuc_sws_6073]** 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:

[ecuc_sws_2088] When a `longName` (`LONG-NAME` in XML) is provided for a configuration element the Configuration Editor shall display the content of the `longName` this to it's users.

For referencing the following requirements apply:

[ecuc_sws_6047] 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 5.4 defines a reference inside the CanDrv module. Thus the values can only refer to container instances within the respective CanDrv configuration instance.

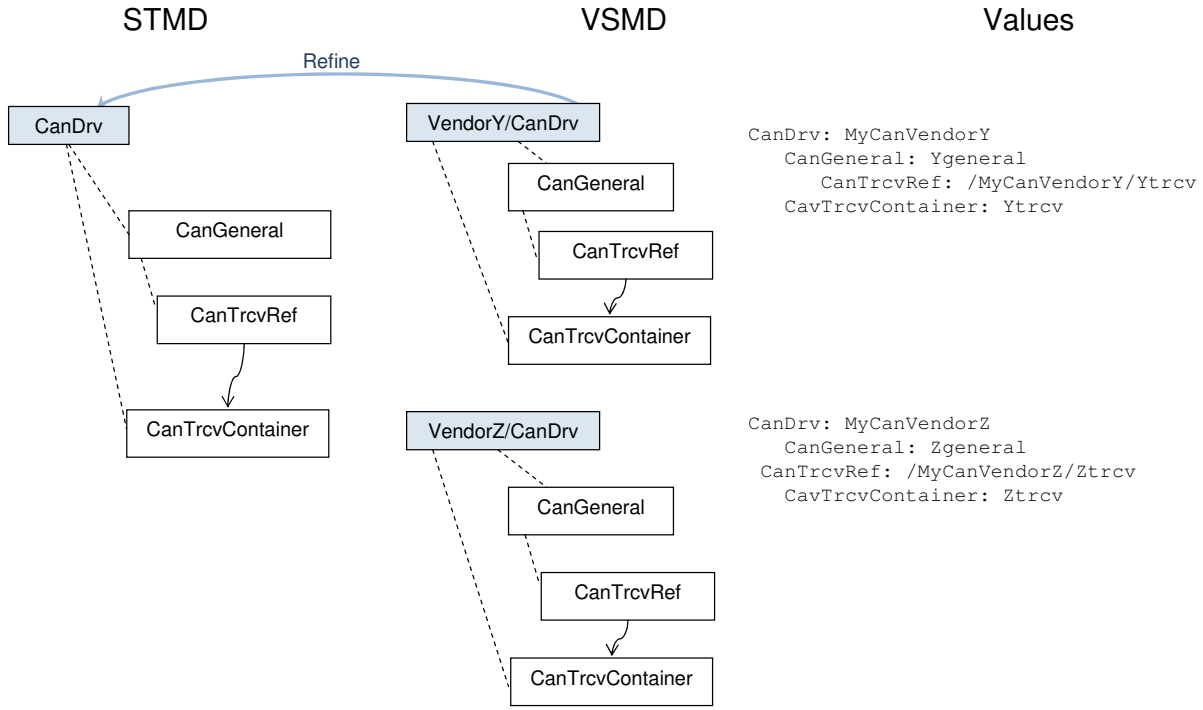

Figure 5.4: Reference inside a module

[ecuc_sws_6048] 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 5.5 defines a reference between the CanIf and the CanDrv module. Thus the values can refer to container instances of different CanDrv configuration instances.

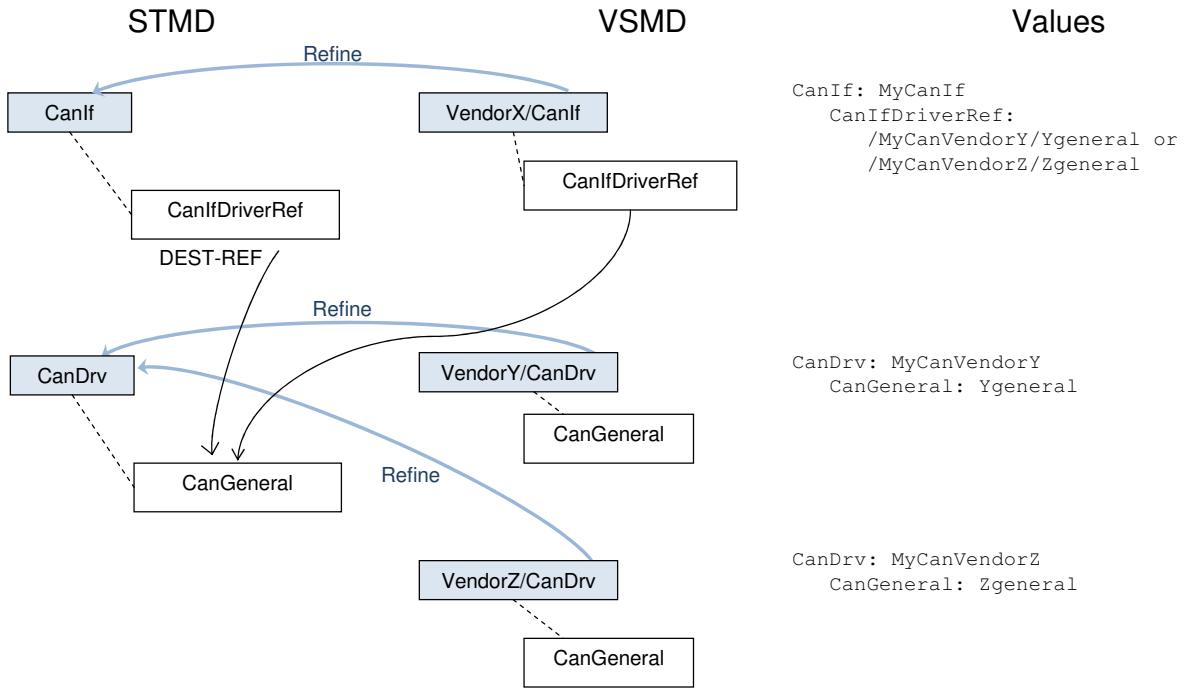

Figure 5.5: Reference between modules

5.4 Rules for navigating in Ecu Configuration Artifacts

The following rules apply for tools that are navigating in Ecu Configuration Artifacts:

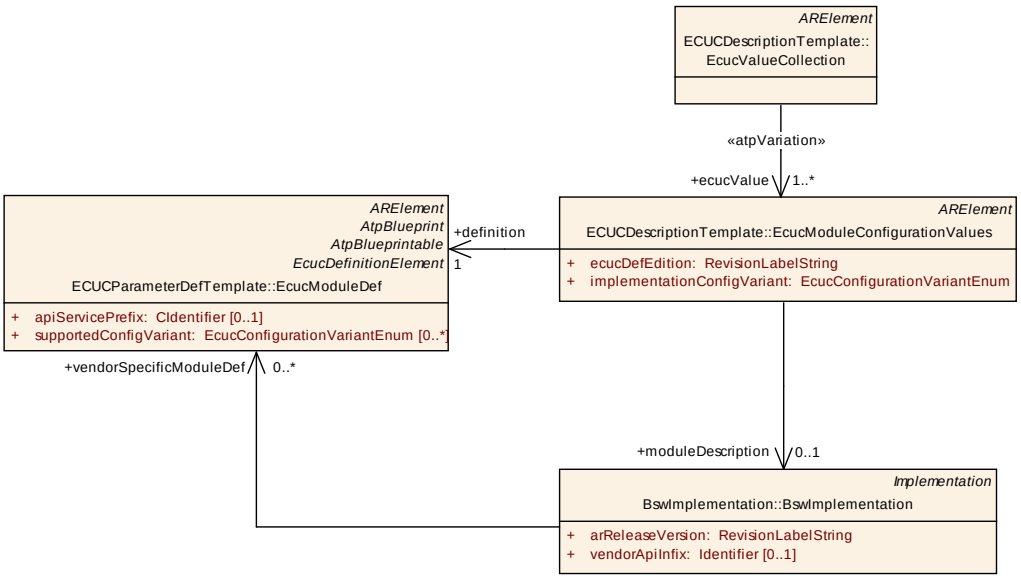

Figure 5.6: Ecu Configuration Artifacts

- **[ecuc_sws_6039]** The tool knows his BSW Implementation element and subsequently the `vendorSpecificModuleDef`.
- **[ecuc_sws_6040]** The tool shall get the `EcucValueCollection` as input information. Please note that according to the IOAT [10] the input can be provided as multiple files and can be structured in an arbitrary package structure.
- **[ecuc_sws_6041]** The tool shall respect only those `EcucModuleConfigurationValues` elements which are referenced by the `EcucValueCollection`.
- **[ecuc_sws_6042]** 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.

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

[ecuc_sws_2124] 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.

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)

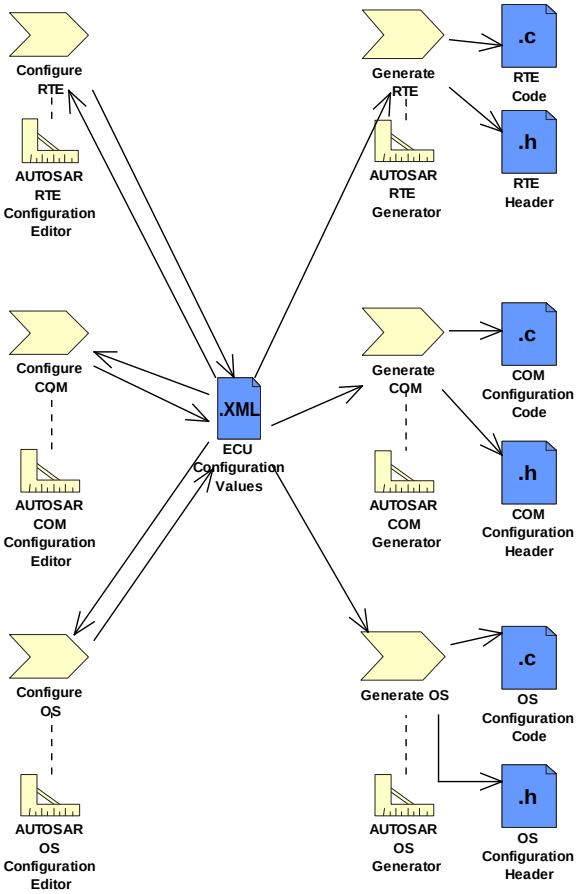

Figure A.1: Custom Editors and Generators

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)

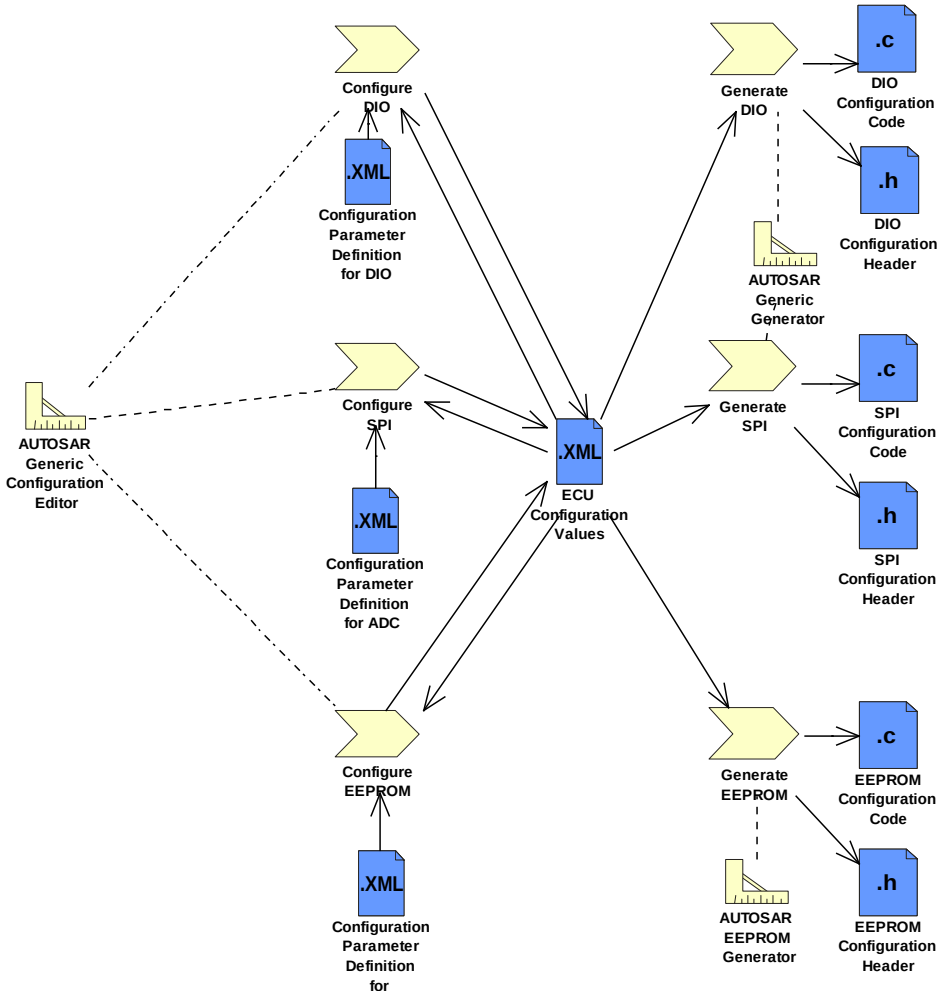

Figure A.2: Generic Configuration Editor

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)

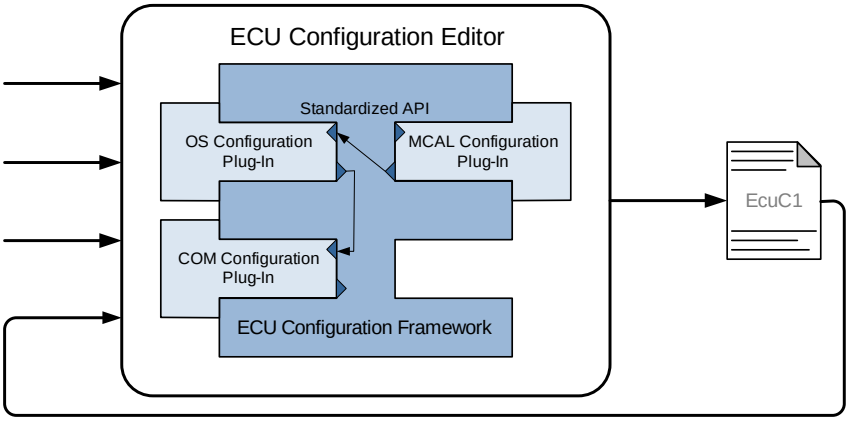

Figure A.3: Framework Tools

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.

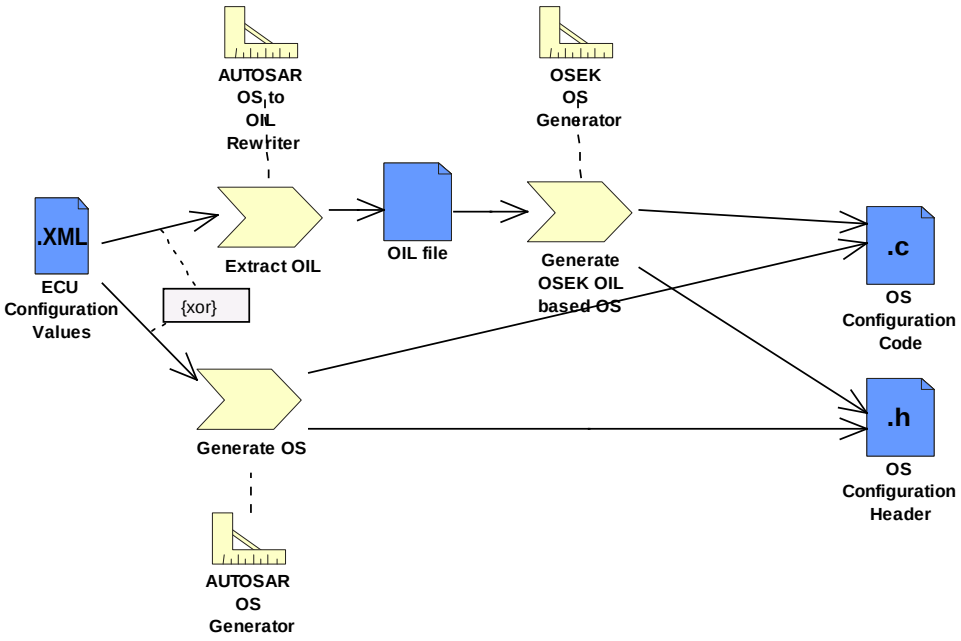

Figure A.4: Module generator implementation alternatives, example for OS

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.

[ecuc_sws_1025] 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 Change History

B.1 Change History between AUTOSAR R4.0.1 against R3.1.5

B.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 B.1: Renamed meta-model elements

B.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 [ecuc_sws_2124] 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 [ecuc_sws_2012].
[ecuc_sws_1000]	Replaced requirement by [ecuc_sws_6038].
[ecuc_sws_1002]	Replaced with [ecuc_sws_6007] and [ecuc_sws_6008] for clarification of derivation rules.
[ecuc_sws_1003]	Replaced with [ecuc_sws_6007] and [ecuc_sws_6008] for clarification of derivation rules.
[ecuc_sws_1010]	Removed for clarification of derivation rules.
[ecuc_sws_1012]	Removed for clarification of derivation rules.

Table B.2: Deleted SWS Items

B.1.3 Changed SWS Items

SWS Item	*Rationale*
[ecuc_sws_2029]	Refined specification of String Parameter definition.
[ecuc_sws_2030]	Refined specification of String Parameter definition.
[ecuc_sws_2031]	Refined specification of String Parameter definition.
[ecuc_sws_2108]	Refined how symbolic names are generated to header files.
[ecuc_sws_5001]	Refined because of unclear requirement.
[ecuc_sws_3021]	Changed due to changes on the derived parameter structure.
[ecuc_sws_2047]	Changed due to changes on the derived parameter structure.
[ecuc_sws_2087]	Changed due to changes in Ecu Extract description.
[ecuc_sws_2000]	Changed sws_2000 due to changes in the generic structure template
[ecuc_sws_2084]	Incompatible inter module queries.
[ecuc_sws_6002]	Incompatible inter module queries.
[ecuc_sws_1032]	Refined because of unclear requirement.
[ecuc_sws_2030]	Refined because of unclear requirement.
[ecuc_sws_2095]	Refined specification.
[ecuc_sws_3007]	Refined specification.
[ecuc_sws_3034]	Refined specification.
[ecuc_sws_3010]	Refined specification.
[ecuc_sws_3040]	Refined specification.
[ecuc_sws_2107]	Refined specification.
[ecuc_sws_2000]	Refined specification.
[ecuc_sws_2001]	Refined specification.
[ecuc_sws_2002]	Refined specification.
[ecuc_sws_6004]	Refined specification.
[ecuc_sws_2067]	Refined specification.
[ecuc_sws_2012]	Refined specification.
[ecuc_sws_2009]	Refined specification.
[ecuc_sws_2029]	Refined specification.
[ecuc_sws_2087]	Refined specification.

Table B.3: Changed SWS Items

B.1.4 Added SWS Items

SWS Item	*Rationale*
[ecuc_sws_2110]	Implement Variant Handling Concept.
[ecuc_sws_2111]	Implement Variant Handling Concept.
[ecuc_sws_2112]	Implement Variant Handling Concept.
[ecuc_sws_2113]	Implement Variant Handling Concept.
[ecuc_sws_2114]	Implement Variant Handling Concept.
[ecuc_sws_2115]	Implement Variant Handling Concept.
[ecuc_sws_2116]	Implement Variant Handling Concept.
[ecuc_sws_2117]	Implement Variant Handling Concept.
[ecuc_sws_2119]	Implement Variant Handling Concept.
[ecuc_sws_2120]	Implement Variant Handling Concept.
[ecuc_sws_2121]	Implement Variant Handling Concept.
[ecuc_sws_2122]	Implement Variant Handling Concept.
[ecuc_sws_2118]	Implement Documentation Concept.
[ecuc_sws_2123]	Implement Documentation Concept.

[ecuc_sws_2124]	New requirement id for old [ecuc_sws_1031] due to double usage.
[ecuc_sws_2125]	Added due to changes on the derived parameter structure.
[ecuc_sws_2126]	Added to remove type specific value definitions.
[ecuc_sws_2127]	Implement Variant Handling Concept.
[ecuc_sws_2128]	Added due to changes on the derived parameter structure.
[ecuc_sws_2129]	Added due to changes on the derived parameter structure.
[ecuc_sws_2130]	Extended Package structure for parameter definition.
[ecuc_sws_6014]	Changed due to changes in Ecu Extract description.
[ecuc_sws_6015]	Refined specification
[ecuc_sws_6016]	Implement Variant Handling concept
[ecuc_sws_6017]	Implement Variant Handling Concept
[ecuc_sws_6018]	Implemented calculation formula concept
[ecuc_sws_6019]	Implemented calculation formula concept
[ecuc_sws_6020]	Implemented calculation formula concept
[ecuc_sws_6021]	Implemented calculation formula concept
[ecuc_sws_6022]	Implemented calculation formula concept
[ecuc_sws_6023]	Implemented calculation formula concept
[ecuc_sws_6024]	Implemented calculation formula concept
[ecuc_sws_6025]	Implemented calculation formula concept
[ecuc_sws_6026]	Implemented calculation formula concept
[ecuc_sws_6027]	Implemented calculation formula concept
[ecuc_sws_6028]	Implemented calculation formula concept
[ecuc_sws_6029]	Implemented calculation formula concept
[ecuc_sws_6030]	Refined specification
[ecuc_sws_6031]	Implement CDD Concept
[ecuc_sws_6032]	Refined specification
[ecuc_sws_6033]	Refined specification
[ecuc_sws_6034]	Refined specification
[ecuc_sws_1032]	Refined specification
[ecuc_sws_6036]	Implement CDD Concept
[ecuc_sws_6037]	Implement CDD Concept
[ecuc_sws_6038]	Requirements traceability
[ecuc_sws_6043]	Refined specification
[ecuc_sws_6044]	Refined specification
[ecuc_sws_6035]	Refined regular expression.
[ecuc_sws_6006]	Refined regular expression.
[ecuc_sws_6007]	Added for clarification of derivation rules.
[ecuc_sws_6008]	Added for clarification of derivation rules.
[ecuc_sws_6009]	Added for clarification of derivation rules.
[ecuc_sws_6010]	Added for clarification of derivation rules.
[ecuc_sws_6011]	Added for clarification of derivation rules.
[ecuc_sws_6012]	Added for clarification of derivation rules.
[ecuc_sws_6013]	Added for clarification of derivation rules.

Table B.4: Added SWS Items

B.2 Change History between AUTOSAR R4.0.2 against R4.0.1

B.2.1 Changed SWS Items

SWS Item	*Rationale*

[ecuc_sws_2072]	Remove restriction on hex-representation of integers.
[ecuc_sws_2095]	CATEGORY should not be Identifier.

Table B.5: Changed SWS Items

B.2.2 Added SWS Items

SWS Item	*Rationale*
[ecuc_sws_2131]	Added origin to EcucEnumerationLiteralDef.
[ecuc_sws_6039]	Update how to look up the correct module configuration in an EcuC description.
[ecuc_sws_6040]	Update how to look up the correct module configuration in an EcuC description.
[ecuc_sws_6041]	Update how to look up the correct module configuration in an EcuC description.
[ecuc_sws_6042]	Update how to look up the correct module configuration in an EcuC description.
[ecuc_sws_6043]	CATEGORY should not be Identifier.
[ecuc_sws_6044]	CATEGORY should not be Identifier.
[ecuc_sws_6045]	Allow Infinity to float parameters min max values.

Table B.6: Added SWS Items

B.3 Change History between AUTOSAR R4.0.3 against R4.0.2

B.3.1 Deleted SWS Items

SWS Item	*Rationale*
[ecuc_sws_5001]	Removed because it was unclear.

Table B.7: Deleted SWS Items

B.3.2 Changed SWS Items

SWS Item	*Rationale*
[ecuc_sws_2041]	Clarified modeling of destinationType and destinationContext.
[ecuc_sws_2082]	Clarified modeling of destinationType and destinationContext.
[ecuc_sws_2083]	Clarified modeling of destinationType and destinationContext.
[ecuc_sws_6002]	Clarified scope of parameters.
[ecuc_sws_6015]	Update semantics of definitionRef.
[ecuc_sws_6025]	Properly support String Arguments in FormulaExpression
[ecuc_sws_6037]	Storage of CDD module abbreviation.
[ecuc_sws_2108]	Clarified usage of symbolic name references.

Table B.8: Changed SWS Items

B.3.3 Added SWS Items

SWS Item	*Rationale*
[ecuc_sws_2132]	Clarified postBuildChangeable and multipleConfigurationContainer
[ecuc_sws_2133]	Clarified postBuildChangeable and multipleConfigurationContainer
[ecuc_sws_6046]	Update semantics of definitionRef.
[ecuc_sws_6047]	Update semantics of definitionRef.
[ecuc_sws_6048]	Update semantics of definitionRef.
[ecuc_sws_6049]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6050]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6051]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6052]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6053]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6054]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6055]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6056]	Clarification of PostBuildSelectable, PostBuildLoadable in VSMD
[ecuc_sws_6057]	Properly support String Arguments in FormulaExpression
[ecuc_sws_6058]	Properly support String Arguments in FormulaExpression
[ecuc_sws_6059]	Properly support String Arguments in FormulaExpression
[ecuc_sws_6060]	Sequence of Ecuc parameters
[ecuc_sws_6061]	Sequence of Ecuc parameters
[ecuc_sws_6062]	Sequence of Ecuc parameters
[ecuc_sws_6063]	Sequence of Ecuc parameters
[ecuc_sws_6064]	Sequence of Ecuc parameters
[ecuc_sws_6065]	Sequence of Ecuc parameters
[ecuc_sws_6066]	Sequence of Ecuc parameters
[ecuc_sws_6067]	Sequence of Ecuc parameters
[ecuc_sws_6068]	Sequence of Ecuc parameters
[ecuc_sws_6069]	Sequence of Ecuc parameters
[ecuc_sws_6070]	Sequence of Ecuc parameters
[ecuc_sws_6071]	Sequence of Ecuc parameters
[ecuc_sws_6072]	Sequence of Ecuc parameters
[ecuc_sws_6073]	Package Structure of value descriptions
[ecuc_sws_6074]	Symbolic names in multiple Configuration sets

Table B.9: Added SWS Items

B.3.4 Added Constraints

Number	**Heading**
[constr_3022]	EcucModuleDef category restriction
[constr_3023]	Usage of apiServicePrefix

Table B.10: Added Constraints in R4.0.3