Document Title	Specification of PDU Router
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	35

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Added multicast (1:n) support from a transport protocol module to local upper layer modules • Added fan-in (n:1) support for multiple communication interface modules to a local upper layer module • Cleaned up chapter 7 and clarified buffering concept • Same `PduRRoutingPath` may be assigned to multiple `PduRRoutingPathGroups` • Inter-Partition Gateway Routing Relations are described in more detail • Clarification and clean up of Multicast TP Tx PDU Forwarding • Editorial changes

2020-11-30	R20-11	AUTOSAR Release Management	• Description of and requirements related to `PduRRoutingPathGroup` has been updated • `CancelTransmit` for gateways has been clarified • Error classification has been harmonized • Structure of “Error Section” has been improved • “Draft” tags have been removed from Multicore Distribution spec. items
2019-11-28	R19-11	AUTOSAR Release Management	• Add Multicore Distribution • Change [SWS_PduR_00783] to process overlength PDUs • Add additional parameters in the `PduRBswModules` container • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Relase Management	• Removal of obsolete elements • Remove dummy implementations for `CancelTransmit` APIs • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation

2017-12-08	4.3.1	AUTOSAR Relase Management	• API parameter `RetryInfoType* retry` has become pointer to `const` in `PduR_<User:LoTp>CopyTxData` • The `ChangeParameter` API has been rendered obsolete • DET Runtime Errors `PDUR_E_TP_TX_REQ_REJECTED` and `PDUR_E_PDU_INSTANCES_LOST` introduced • `Det_ReportRuntimeError` has become a Mandatory Interface and inclusion of DET is not optional anymore • Clarification of the disabled APIs and their behavior if `PduR_DisableRouting` called • Corrections in description of `PduRDestTxBufferRef` and `PduRTxBuffer` • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Reliable `TxConfirmation` • Addressing in Upper Layers using MetaData • Clarification on unknown message length handling for the TP gateway • Added support for n:1 routing • Added support for FIFO for TP messages • Removed module specific dependencies when calling DET

2015-07-31	4.2.2	AUTOSAR Release Management	• Added support of `TriggerTransmit` for dynamic length PDUs • Clarification on output parameter '`availableDataPtr`' of `PduR_<User:LoTp>CopyTxData` • Clarification for releasing of buffer on return of `E_NOT_OK from <DstLoTp_Transmit>` API • Clarified behavior for disabled `TxPduId` of upper layer • Clarified Routing PDUs between local modules • Cleanup of references to former SoAd API • DET Renaming and Extension Incorporation • LdCom as upper module • Clarification for releasing of buffer on return of `E_NOT_OK` from `<DstLoTp_Transmit>` API
2014-10-31	4.2.1	AUTOSAR Release Management	• Support multi-frame TP fanout • CAN-FD and SecOC Concept incorporation • Improved Cancel Transmission handling in case of gatewaying • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Clarified handling of routing `on-the-fly` for unreached TP threshold • Clarify behaviour for `TriggerTransmit` data provision depending on used buffering strategy • Introduced DET when `<DstLo>_Transmit` fails • Harmonize descriptions of identical API functions

2013-10-31	4.1.2	AUTOSAR Release Management	• Revised list of optional interfaces • Deleted handling of misconfigured PDUs during run-time. • Deleted `NotifyResultType` • Added error handling after destination abort in case of gatewaying. • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Added support for extended PDUs as part of the heavy duty vehicle support • Removed minimum routing feature • Improved multicast behavior • Post-build loadable support
2011-12-22	4.0.3	AUTOSAR Administration	• Clarifications regarding non-TP PDU routing • New feature: non-TP PDU routing idependent of the Pdu lengh • FIFO handling for non-TP PDU routing clarified / improved • Service ID's for generic services introduced • Clarification regarding multicast routing of TP-PDU's • DEM error reporting removed
2010-09-30	3.1.5	AUTOSAR Administration	• Introduced new version check • Added `Std_ReturnType` to `PduR_<Lo>TriggerTransmit` • Added functionality of `PduR_<LoTp>CopyTxData` when `TsSduLength` is zero

2010-02-02	3.1.4	AUTOSAR Administration	• The PDU Router module is made generic to allow any combination of busses, TPs and upper modules. The upper and lower modules are modeled generic and handled by the configuration. • The Transport Protocol API has been redesgined. Compatibility between TP in AR3.x and AR4.0 is described. • Cancel transmission of communication interface I-PDUs has been added. • Cancel reception of Transport Protocol I-PDUs has been added. • Change parameter of Transport Protocol parameters has been extended. • Legal disclaimer revised
2009-07-24	3.1.3	AUTOSAR Administration	• Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Correction figure 3
2007-08-10	2.1.17	AUTOSAR Technical Office	• Tables generated from UML-models, UML-diagrams linked to UML-model, general improvements of requirements in preparation of CT-development. No changes in the technical contents of the specification.

2007-07-24	2.1.16	AUTOSAR Administration	• Variants have been renamed. • New Callbacks `PduR_LinTpChangeParameterConfirmation`, `PduR_FrTpChangeParameterConfirmation` • `PduR_CanTpChangeParameterConfirmation` has been added. • New API's `PduR_ChangeParameterRequest`, `PduR_CancelTransmitRequest` has been added • New type defines `PduR_ParameterValueType`, `PduR_CancelReasonType` has been added • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Clarifications added (`FIFO`, `TxConf`, `...`) • Unnecessary development errors removed • `SchM_PduR.h` and `MemMap.h` added • Corrections of configuration parameters • More details in Chapter 13 • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Document structure adapted to common Release 2.0 SWS Template. • Major changes in chapter 10 • Structure of document changed partly • Other changes see chapter
2005-05-31	1.0	AUTOSAR Administration	• Initial Release

Table of Contents

1 Introduction and functional overview

This specification describes the functionality and APIs of the AUTOSAR PDU Router (PduR) module.

The PDU Router module provides services for routing I-PDUs (Interaction Layer Protocol Data Units) using the following module types:

- Communication interface modules, that use the `<Provider:Up>` or `<Provider:Lo>` APIs, e.g. Com, IPduM, LinIf, CanIf, CanNm, FrIf and FrNm;
- Transport Protocol modules, that use the `<Provider:UpTp>` or `<Provider:LoTp>` APIs, e.g. J1939Tp, LinTp (part of LinIf), CanTp, FrTp, Com, Dcm.

The routing of I-PDUs is performed based on statically defined I-PDU identifiers. No I-PDU is routed dynamically during run-time, e.g. dependent on its payload.

The location of related modules can be "upper" (e.g. Dlt, Dcm, Com, IpduM) and/or "lower" (CanIf, FrIf, LinTp, IpduM, CanNm, FrNm). Note that the IpduM is listed as an upper and a lower module because it has two different roles (lower: communication between Com module and IpduM module, upper: communication between IpduM module and lower layer communication interface module).

The PDU Router module is based on a generic approach of interfaced modules. The module that is interfaced is configured in the PDU Router module configuration. The listed modules in parenthesis in the previous paragraph are just examples, and not an exhaustive list. The PDU Router can be easily configured to support other upper and lower layer modules. This approach also allows to integrate Complex Device Drivers (CDDs) as upper or lower layer modules of the PDU Router.

The list of users of the PDU Router module is not fixed. The most common combination of upper and lower layer pairs is listed below:

- Diagnostic Communication Manager (Dcm) and Transport Protocol modules,
- Com and Communication Interface modules, Transport Protocol modules or I-PDU Multiplexer,
- I-PDU Multiplexer and Communication Interface modules.

1.1 AUTOSAR architecture

The PDU Router module is a central module in the AUTOSAR communication structure [1]. Figure 1.1 gives an overview of the AUTOSAR communication structure.

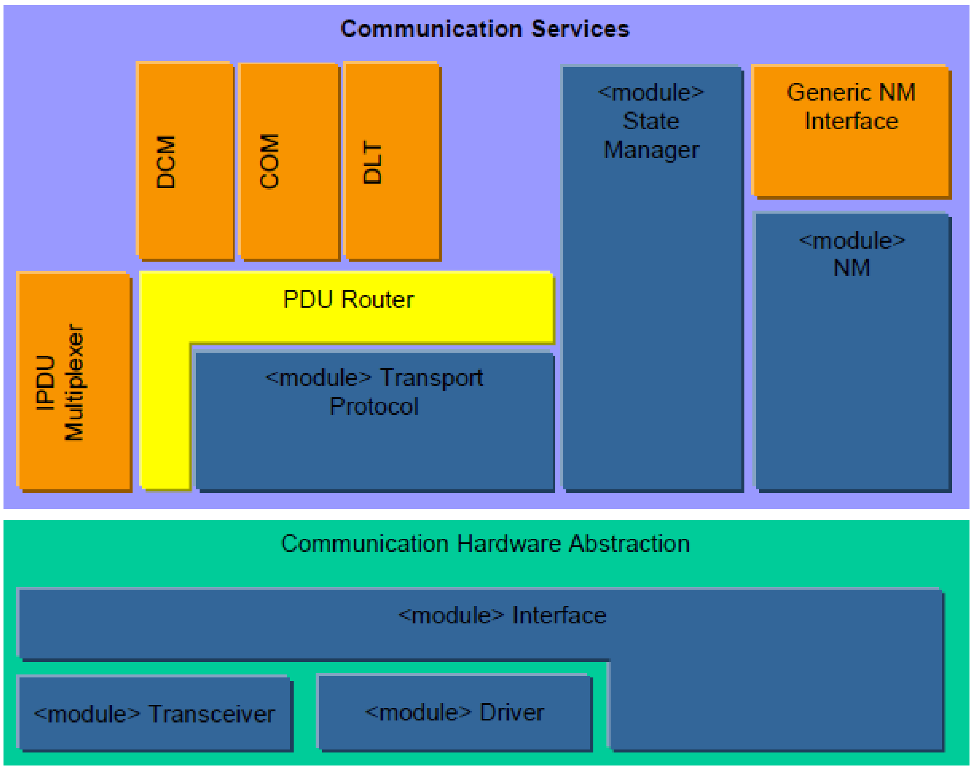

Figure 1.1: CommunicationStructure

1.2 PDU Router module function overview

The PDU Router module is part of the AUTOSAR Basic SW, and is mandatory instantiated in every AUTOSAR ECU.

The detailed PDU Router module structure is shown in Figure 1.2.

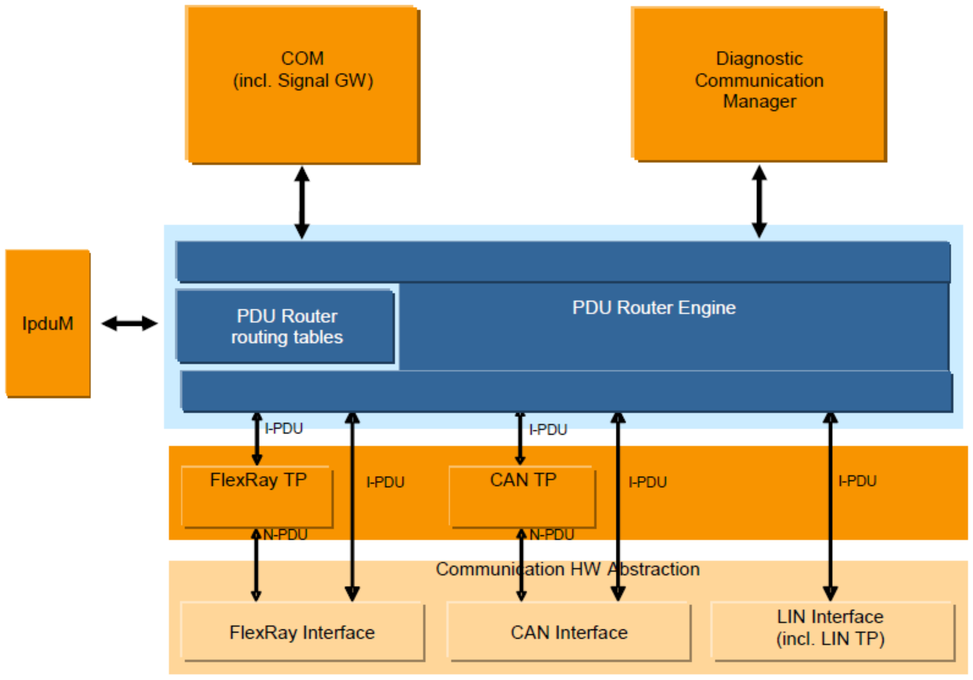

Figure 1.2: Detailed PDU Router Structure showing FlexRAy, CAN and LIN

The PDU Router module mainly consists of two parts:

- The **PDU Router routing paths**: static routing paths describing the routing attributes for each I-PDU to be routed. The routing paths can be (if supported) updated post-build loadable in the programming state of the ECU or selected when initializing the PDU Router by post-build selectable (see section 10.1.1).
- The **PDU Router Engine**: the actual code performing routing actions according to the PDU Router routing paths. The PDU Router Engine has to deal with:
  - Routing the I-PDU from source(s) to destination(s),
  - Translating the source I-PDU ID to the destination I-PDU ID (e.g. `PduR_ComTransmit` to `CanIf_Transmit`, `PduR_CanIfTxConfirmation` to `Com_TxConfirmation`).

1.3 I-PDU handling

I-PDUs are identified by static I-PDU IDs. The PDU Router module determines the destination of an I-PDU by using the I-PDU ID in a static configuration table. I-PDUs are used for the data exchange of the modules directly above the PDU Router module, e.g. the Com module and Dcm module. The routing operation of the PDU Router module does not modify the I-PDU, it simply forwards the I-PDU to the destination module. In case of TP gatewaying, forwarding the I-PDU to the destinaiton(s) may start before the full I-PDU is received ("`on-the-fly gatewaying`").

The I-PDU ID is set in the configuration that also implements the API. This will allow an efficient implementation of look-up tables in each module receiving an I-PDU ID (e.g. the PDU Router module's configuration contains the I-PDU ID for the `PduR_CanIfTxConfirmation`, while CanIf module's configuration contains the I-PDU ID for the `CanIf_Transmit`).

The following list summarizes the routing capabilities of PduR:

1. I-PDU Forwarding
    - Transmission from upper layer
        - Communication Interface
            - Singlecast (1:1) an I-PDU from a local module to a communication interface module.
            - Multicast (1:n) an I-PDU from a local module to communication interface modules.
        - Transport Protocol
            - Singlecast (1:1) an I-PDU (both Single Frame and Multi Frame) from a local module to a transport protocol module.
            - Multicast (1:n) an I-PDU (both Single Frame and Multi Frame) from a local module to transport protocol modules.
    - Reception to upper layer
        - Communication Interface
            - Singlecast (1:1) an I-PDU from a communication interface module to a local module.
            - Multicast (1:n) an I-PDU from a communication interface module to local modules.
            - Fan-in (n:1) an I-PDU from communication interface modules to a local module.
        - Transport Protocol
            - Singlecast (1:1) an I-PDU (both Single Frame and Multi Frame) from a transport protocol module to a local module.
            - Multicast (1:n) an I-PDU (both Single Frame and Multi Frame) from a transport protocol module to local modules.
2. I-PDU Gatewaying
    - Communication Interface

- Gateway (1:1) an I-PDU from a communication interface module to a communication interface module using last-is-best buffer/ FIFO buffer/ no buffer.
- Gateway (1:n) an I-PDU from a communication interface module to multiple communication interface modules using last-is-best buffer/ FIFO buffer/ no buffer.
- Gateway (n:1) an I-PDU from multiple communication interface modules to a communication interface module using last-is-best buffer/ FIFO buffer/ no buffer, only one source shall be enabled at once.

- Transport Protocol
  - Gateway (1:1) an I-PDU from a transport protocol module to a transport protocol module using buffer.
  - Gateway (1:n) an I-PDU from a transport protocol module to multiple transport protocol modules using buffer.
  - Gateway (n:1) an I-PDU from multiple transport protocol modules to a transport protocol module using buffer, only one source shall be enabled at once.

3. Combined I-PDU gatewaying and forwarding
   - Communication Interface
     - An I-PDU may be received by one or more upper modules in the same time as gatewayed to one or more communication interfaces using last-is-best/FIFO/ no buffer.
   - Transport Protocol
     - An I-PDU (only Single Frame) may be received by one or more upper modules in the same time as gatewayed to one or more lower layer transport protocol modules using buffer.

2 Acronyms and abbreviations

The following acronyms and abbreviations have a local scope and are therefore not contained in the AUTOSAR glossary.

Acronym:	Description:
Upper Layer Modules (Up)	Modules above the PDU Router. This layer usually includes Com and Diagnostic Communication Manager (Dcm).
Lower Layer Modules (Lo)	Modules below the PDU Router. This layer may include CAN, LIN, FlexRay, Ethernet Communication Interface modules and the respective TP modules.

▽

△

Acronym:	Description:
PDU Router	Module that transfers I-PDUs from one module to another module. The PDU Router module can be utilized for gateway operations and for internal routing purposes.
`on-the-fly gatewaying`	Gateway capability; routing between two TP modules where forwarding of data is started (when a specified threshold is reached) before all data have been received. If larger amount of data is transported between two interfaces it is desirable to be able to start the transmission on the destination network before receiving all data from the source network. This saves memory and time.
multicast operation	Simultaneous transmission of PDUs to a group of receivers, i.e. 1:n routing.
`data provision`	Provision of data to interface modules. (a) `direct data provision`: data to be transmitted are provided directly at the transmit request. The destination Communication Interface may behave in two ways, either copy the data directly or defer the copy to a trigger transmit. (b) `trigger transmit data provision`: data to be transmitted are not provided at the transmit request, but will be retrieved by the Communication Interface module via a callback function.
`last-is-best buffering`	Buffering strategy where the latest value overwrites the last value.
FIFO buffering	Buffer concept, which uses first in first out strategy.

Abbreviation:	Description:
I-PDU ID	I-PDU Identifier.
I-PDU	Interaction Layer PDU. An I-PDU consists of data (buffer), length and I-PDU ID. The PDU Router will mainly route I-PDUs (exception is `on-the-fly gatewaying`).
N-PDU	Network Layer PDU. Used by Transport Protocol modules to fragment an I-PDU.
L-PDU	Data Link Layer PDU. One or more I-PDUs are packed into one L-PDU. The L-PDU is bus specific, e.g. CAN frame.
SF	Single Frame, Transport Protocol term.
FF	First Frame, Transport Protocol term.
CF	Consecutive Frame, Transport Protocol term.
PDU	Protocol Data Unit.
BSW	Basic Software.
<SrcLo>	Lower layer Communication Interface module acting as a source of the I-PDU. The SrcLo is always one.
<DstLo>	Lower layer Communication Interface module acting as a destination of the I-PDU. The DstLo may by one to many.
<SrcLoTp>	Lower layer Transport Protocol module acting as a source of the I-PDU. The SrcLoTp is always one.
<DstLoTp>	Lower layer Transport Protocol module acting as a destination of the I-PDU. The DstLoTp may by one to many.
<Lo>	Lower layer communication interface module.
<Up>	Upper layer Communication Interface and/or Transport Protocol module.
<LoTp>	Lower layer Transport Protocol module.
<module>	Any type of module <...>.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[3] Requirements on Gateway
AUTOSAR_SRS_Gateway

[4] Specification of I-PDU Multiplexer
AUTOSAR_SWS_IPDUMultiplexer

[5] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[6] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

[7] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes

[8] Guide to BSW Distribution
AUTOSAR_EXP_BSWDistributionGuide

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2], which is also valid for PDU Router.

Thus, the specification SWS BSW General shall be considered as additional and required specification for PDU Router.

4 Constraints and assumptions

4.1 Limitations

The PDU Router module does not:

- have mechanisms for signal extraction or conversion,
- have mechanisms for data integrity checking (like checksums),
- change or modify the I-PDU,

- make any PDU payload dependent routing decisions,
- support routing between TP modules and Communication Interface modules or vice versa,
- support routing of I-PDUs between Communication Interface modules with rate conversion. (This functionality will be supported in cooperation with an upper layer module, e.g. the Com module).

4.1.1 Limitations on supported functionality

The PDU Router module supports fan-out of I-PDUs transmitted from a local module (e.g. Com module) to more than one destinations. There are some limitations if the I-PDU shall be transmitted to more than one destination (fan-out 1:n; n>1), because the upper layer module is not aware how many destinations there are:

- The PDU Router reports `E_OK` for a `Transmit` request from an upper layer if at least one destination lower layer reports `E_OK`.
- The PDU Router gives a `TxConfirmation` to the upper layer when it receives the last `TxConfirmation` from destination lower layer.
- The PDU Router returns `E_OK` for a `CancelTransmit` requested from the upper layer only if all destination lower layers return `E_OK`.

If the I-PDU fan-out is performed by the PDU Router, this has further consequences for the Com as upper layer module:

- Update bits will not work.
- The `TxConfirmation` of the Communication Interface API will be handled in the way that the local module (e.g. Com module) will be informed when the last destination has confirmed the transmission. This means that deadline monitoring is made with respect to the last `TxConfirmation` (i.e. there is no difference if all the I-PDUs were transmitted successfully or not).
- Starting and stopping of I-PDU groups affects all destinations.

Note that above limitations are not set as requirements since they do not concern functionality provided by the PduR module. But implication of the use of the PduR module will affect these functionalities.

If the I-PDU fan-in is performed by the PDU Router, update bits and I-PDU sequence counter will not work with Com module.

4.2 Applicability to car domains

The PDU Router is used in all ECUs where communication is necessary.

The PDU Router module has not been specified to work with MOST communication network. Thus the applicability to multimedia and telematic car domains may be limited.

5 Dependencies to other modules

The PDU Router module depends on the APIs and capabilities of the used communication hardware abstraction layer modules and the used communication service layer modules. Basically the API functions required by the PDU Router module are:

Communication Interface modules:

- `<Lo>_Transmit` (e.g. `CanIf_Transmit`, `FrIf_Transmit`, `LinIf_Transmit`)
- `<Lo>_CancelTransmit` (e.g. `FrIf_CancelTransmit`)

Transport Protocol Modules:

- `<LoTp>_Transmit` (e.g. `CanTp_Transmit`, `FrTp_Transmit`, `LinTp_Transmit`)
- `<LoTp>_CancelTransmit` (e.g. `CanTp_CancelTransmit`, `FrTp_CancelTransmit`)
- `<LoTp>_CancelReceive` (e.g. `CanTp_CancelReceive`, `FrTp_CancelReceive`)

Upper layer modules which use Transport Protocol modules:

- `<Up>_StartOfReception` (e.g. `Dcm_StartOfReception`)
- `<Up>_CopyRxData` (e.g. `Dcm_CopyRxData`)
- `<Up>_CopyTxData` (e.g. `Dcm_CopyTxData`)
- `<Up>_TpRxIndication` (e.g. `Dcm_TpRxIndication`)
- `<Up>_TpTxConfirmation` (e.g. `Dcm_TpTxConfirmation`)

Upper layer modules which process I-PDUs originating from Communication Interface modules:

- `<Up>_RxIndication` (e.g. `Com_RxIndication`),
- `<Up>_TxConfirmation` (e.g. `Com_TxConfirmation`),
- `<Up>_TriggerTransmit` (e.g. `Com_TriggerTransmit`)

5.1 File structure

5.1.1 Code file structure

For details refer to the Chapter 5.1.6 "Code file structure" in [2, SWS_BSWGeneral].

The code file structure is not defined within this specification completely. However to allow integration to other modules the following structure is needed.

5.1.2 Header file structure

[SWS_PduR_00216] ⌈The PDU Router module shall provide the functions used by the different modules in separate header files.⌋*(SRS_BSW_00415)*

Example: If CanIf, CanTp and FrIf are used then the PDU Router module shall provide `PduR_CanIf.h`, `PduR_CanTp.h` and `PduR_FrIf.h`.

[SWS_PduR_00802] ⌈The PduR implementation shall include `Det.h`.⌋*(SRS_BSW_-00350)*

[SWS_PduR_00762] ⌈All PDU Router header files shall contain a software and specification version number.⌋*(SRS_BSW_00003)*

This structure allows the separation between platform, compiler and implementation specific definitions and declarations from general definitions as well as the separation of source code and configuration.

5.2 Version check

For details refer to the chapter 5.1.8 "Version Check" in [2, SWS_BSWGeneral].

6 Requirements Tracing

The following table reference the requirements specified in [2] and [3] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_PduR_00762]
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_PduR_00777]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_PduR_00334]

Requirement	Description	Satisfied by
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_PduR_00777]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_PduR_00777]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_PduR_00777]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_PduR_00777]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_PduR_00777]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_PduR_00777]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_PduR_00777]
[SRS_BSW_00301]	All AUTOSAR Basic Software Modules shall only import the necessary information	[SWS_PduR_00333]
[SRS_BSW_00305]	Data types naming convention	[SWS_PduR_00654] [SWS_PduR_00742] [SWS_PduR_00743] [SWS_PduR_00771]
[SRS_BSW_00310]	API naming convention	[SWS_PduR_00334] [SWS_PduR_00338] [SWS_PduR_00341] [SWS_PduR_00362] [SWS_PduR_00365] [SWS_PduR_00369] [SWS_PduR_00375] [SWS_PduR_00381] [SWS_PduR_00406] [SWS_PduR_00504] [SWS_PduR_00507] [SWS_PduR_00512] [SWS_PduR_00518] [SWS_PduR_00615] [SWS_PduR_00617] [SWS_PduR_00767] [SWS_PduR_00769] [SWS_PduR_00800]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_PduR_00100] [SWS_PduR_00221] [SWS_PduR_00647] [SWS_PduR_00648] [SWS_PduR_00649] [SWS_PduR_00716]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_PduR_00777]
[SRS_BSW_00335]	Status values naming convention	[SWS_PduR_00742] [SWS_PduR_00777]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_PduR_00777]

Requirement	Description	Satisfied by
[SRS_BSW_00337]	Classification of development errors	[SWS_PduR_00100]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_PduR_00777]
[SRS_BSW_00343]	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	[SWS_PduR_00777]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/disabling of detection and reporting of development errors.	[SWS_PduR_00802]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_PduR_00777]
[SRS_BSW_00357]	For success/failure of an API call a standard return type shall be defined	[SWS_PduR_00777]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_PduR_00334]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_PduR_00777]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_PduR_00777]
[SRS_BSW_00384]	The Basic Software Module specifications shall specify at least in the description which other modules they require	[SWS_PduR_00424] [SWS_PduR_91001]
[SRS_BSW_00386]	The BSW shall specify the configuration for detecting an error	[SWS_PduR_00777]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_PduR_00743]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_PduR_00241] [SWS_PduR_00281] [SWS_PduR_00295] [SWS_PduR_00296] [SWS_PduR_00743]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_PduR_00771]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_PduR_00119] [SWS_PduR_00308] [SWS_PduR_00324] [SWS_PduR_00325] [SWS_PduR_00326] [SWS_PduR_00328] [SWS_PduR_00330] [SWS_PduR_00644] [SWS_PduR_00645] [SWS_PduR_00742]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_PduR_00338]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_PduR_00338]

Requirement	Description	Satisfied by
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_PduR_00777]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_PduR_00334]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_PduR_00216]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_PduR_00777]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_PduR_00777]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_PduR_00777]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_PduR_00777]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_PduR_00777]
[SRS_BSW_00438]	Configuration data shall be defined in a structure	[SWS_PduR_00241] [SWS_PduR_00743]
[SRS_BSW_00439]	Enable BSW modules to handle interrupts	[SWS_PduR_00777]
[SRS_BSW_00447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	[SWS_PduR_00777]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ ReturnType	[SWS_PduR_00777]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_PduR_00100]
[SRS_BSW_00459]	It shall be possible to concurrently execute a service offered by a BSW module in different partitions	[SWS_PduR_00841]
[SRS_BSW_00460]	Reentrancy Levels	[SWS_PduR_00841]
[SRS_GTW_06001]	Gateway shall be only be reconfigured while the configuration table to be reconfigured is not in use	[SWS_PduR_00296] [SWS_PduR_00308] [SWS_PduR_00328] [SWS_PduR_00330]
[SRS_GTW_06002]	The Routing Configuration shall be updateable at post-build time	[SWS_PduR_00295]
[SRS_GTW_06003]	Static Routing Configuration shall be defined for gateways	[SWS_PduR_00161] [SWS_PduR_00162] [SWS_PduR_00766]

Requirement	Description	Satisfied by
[SRS_GTW_06012]	PDU router shall route non-TP PDUs with transparency between layers	[SWS_PduR_00164] [SWS_PduR_00255] [SWS_PduR_00256] [SWS_PduR_00307] [SWS_PduR_00362] [SWS_PduR_00365] [SWS_PduR_00369] [SWS_PduR_00406] [SWS_PduR_00430] [SWS_PduR_00436] [SWS_PduR_00437] [SWS_PduR_00621] [SWS_PduR_00626] [SWS_PduR_00627] [SWS_PduR_00629] [SWS_PduR_00638] [SWS_PduR_00640] [SWS_PduR_00665] [SWS_PduR_00666] [SWS_PduR_00667] [SWS_PduR_00669] [SWS_PduR_00670] [SWS_PduR_00744] [SWS_PduR_00745] [SWS_PduR_00746] [SWS_PduR_00783] [SWS_PduR_00784] [SWS_PduR_00785] [SWS_PduR_00786] [SWS_PduR_00787] [SWS_PduR_00788] [SWS_PduR_00793] [SWS_PduR_00807] [SWS_PduR_00808]
[SRS_GTW_06020]	The PDU Router resource usage shall be scalable to zero in case no PDU gateway	[SWS_PduR_00287] [SWS_PduR_00619] [SWS_PduR_00764]

Requirement	Description	Satisfied by
[SRS_GTW_06026]	Data buffers for TP shall be provided on request	[SWS_PduR_00299] [SWS_PduR_00301] [SWS_PduR_00317] [SWS_PduR_00375] [SWS_PduR_00381] [SWS_PduR_00406] [SWS_PduR_00428] [SWS_PduR_00429] [SWS_PduR_00507] [SWS_PduR_00512] [SWS_PduR_00518] [SWS_PduR_00549] [SWS_PduR_00551] [SWS_PduR_00625] [SWS_PduR_00629] [SWS_PduR_00634] [SWS_PduR_00637] [SWS_PduR_00638] [SWS_PduR_00673] [SWS_PduR_00687] [SWS_PduR_00689] [SWS_PduR_00696] [SWS_PduR_00697] [SWS_PduR_00705] [SWS_PduR_00707] [SWS_PduR_00708] [SWS_PduR_00727] [SWS_PduR_00740] [SWS_PduR_00767] [SWS_PduR_00789] [SWS_PduR_00790] [SWS_PduR_00791] [SWS_PduR_00792] [SWS_PduR_00794] [SWS_PduR_00797] [SWS_PduR_00798] [SWS_PduR_00799] [SWS_PduR_00808] [SWS_PduR_00813] [SWS_PduR_00814] [SWS_PduR_00815] [SWS_PduR_00911] [SWS_PduR_00912] [SWS_PduR_00913] [SWS_PduR_00914] [SWS_PduR_00915] [SWS_PduR_00916] [SWS_PduR_00917]

Requirement	Description	Satisfied by
[SRS_GTW_06029]	The PDU Router shall be able to support routing of TP PDUs independent from the source to more than one destinations	[SWS_PduR_00551] [SWS_PduR_00631] [SWS_PduR_00632] [SWS_PduR_00633] [SWS_PduR_00701] [SWS_PduR_00717] [SWS_PduR_00724] [SWS_PduR_00765] [SWS_PduR_00789] [SWS_PduR_00790] [SWS_PduR_00791] [SWS_PduR_00792] [SWS_PduR_00812] [SWS_PduR_00911] [SWS_PduR_00912] [SWS_PduR_00913] [SWS_PduR_00914] [SWS_PduR_00915]
[SRS_GTW_06030]	Routing of non-TP PDUs to more than one destination independent from the source shall be supported by the PDU Router	[SWS_PduR_00164] [SWS_PduR_00436] [SWS_PduR_00633] [SWS_PduR_00701] [SWS_PduR_00717] [SWS_PduR_00723]
[SRS_GTW_06032]	The non-TP transmit buffering strategy shall be configured for each PDU to be routed by the PDU Router	[SWS_PduR_00255] [SWS_PduR_00303] [SWS_PduR_00306] [SWS_PduR_00307] [SWS_PduR_00369] [SWS_PduR_00430] [SWS_PduR_00640] [SWS_PduR_00662] [SWS_PduR_00663] [SWS_PduR_00665] [SWS_PduR_00666] [SWS_PduR_00667] [SWS_PduR_00669] [SWS_PduR_00670] [SWS_PduR_00746] [SWS_PduR_00783] [SWS_PduR_00784] [SWS_PduR_00785] [SWS_PduR_00786] [SWS_PduR_00787] [SWS_PduR_00793] [SWS_PduR_00809] [SWS_PduR_00810]
[SRS_GTW_06049]	PDU buffer content shall be consistent during the time needed to read this data	[SWS_PduR_00160]
[SRS_GTW_06055]	The signal gateway shall provide a mechanism to route individual signals between I-PDUs in a 1:n fashion	[SWS_PduR_00777]
[SRS_GTW_06056]	Signal Groups shall be routed	[SWS_PduR_00777]
[SRS_GTW_06061]	Routers shall map only signals	[SWS_PduR_00777]

Requirement	Description	Satisfied by
[SRS_GTW_06064]	The signal gateway shall be scalable to zero size and zero resource usage when signal routing is not required	[SWS_PduR_00777]
[SRS_GTW_06077]	Multiple signals of the same PDU shall be routed	[SWS_PduR_00777]
[SRS_GTW_06089]	The timeout of a deadline monitored signal shall be ignored by the Signal Gateway	[SWS_PduR_00777]
[SRS_GTW_06097]	A Routing Configuration shall be identified by an unique ID number	[SWS_PduR_00280] [SWS_PduR_00281] [SWS_PduR_00341] [SWS_PduR_00771]
[SRS_GTW_06098]	Signal Gateway Error shall be handled with signal routing	[SWS_PduR_00777]
[SRS_GTW_06099]	Signal Gateway Error shall be handled with signal group routing	[SWS_PduR_00777]
[SRS_GTW_06103]	PDU Router error shall be provided for unknown PDU-ID	[SWS_PduR_00221]
[SRS_GTW_06104]	PDU Router error shall be provided for local reception or transmission	[SWS_PduR_00207] [SWS_PduR_00432] [SWS_PduR_00623] [SWS_PduR_00626] [SWS_PduR_00661] [SWS_PduR_00676] [SWS_PduR_00700] [SWS_PduR_00701]
[SRS_GTW_06105]	PDU Router error shall be provided in gateway case	[SWS_PduR_00256] [SWS_PduR_00640] [SWS_PduR_00662] [SWS_PduR_00687] [SWS_PduR_00689] [SWS_PduR_00705] [SWS_PduR_00732] [SWS_PduR_00788] [SWS_PduR_00790] [SWS_PduR_00791] [SWS_PduR_00792] [SWS_PduR_00799] [SWS_PduR_00807] [SWS_PduR_00815] [SWS_PduR_00912] [SWS_PduR_00913] [SWS_PduR_00914] [SWS_PduR_00915]
[SRS_GTW_06106]	PDU Router error shall be provided for FIFO handling	[SWS_PduR_00670]
[SRS_GTW_06114]	The PDU Router provides an interface (API) for usage by COM, to use the PDU Router functionality	[SWS_PduR_00406] [SWS_PduR_00767] [SWS_PduR_00769]
[SRS_GTW_06115]	The PDU Router provides an interface (API) for usage by DCM, to use the PDU Router functionality	[SWS_PduR_00406] [SWS_PduR_00767] [SWS_PduR_00769]

Requirement	Description	Satisfied by
[SRS_GTW_06116]	The PDU Router provides an interface (API) for usage by IpduM, to use the PDU Router functionality	[SWS_PduR_00362] [SWS_PduR_00365] [SWS_PduR_00369] [SWS_PduR_00406] [SWS_PduR_00767] [SWS_PduR_00769]
[SRS_GTW_06117]	The PDU Router provides an interface (API) for usage by bus interfaces, to use the PDU Router functionality	[SWS_PduR_00362] [SWS_PduR_00365] [SWS_PduR_00369] [SWS_PduR_00800]
[SRS_GTW_06119]	Confirmation in case of multicast	[SWS_PduR_00589]
[SRS_GTW_06120]	A predefined set of PDUs shall be enabled and disabled if required	[SWS_PduR_00615] [SWS_PduR_00617] [SWS_PduR_00647] [SWS_PduR_00648] [SWS_PduR_00649] [SWS_PduR_00654] [SWS_PduR_00663] [SWS_PduR_00709] [SWS_PduR_00710] [SWS_PduR_00715] [SWS_PduR_00716] [SWS_PduR_00717] [SWS_PduR_00726] [SWS_PduR_00810] [SWS_PduR_00891] [SWS_PduR_00892] [SWS_PduR_00894] [SWS_PduR_00895] [SWS_PduR_00896] [SWS_PduR_00897] [SWS_PduR_00898] [SWS_PduR_00899]
[SRS_GTW_06121]	J1939 TP as an alternative to CAN TP (ISO 15765-2) shall be supported	[SWS_PduR_00375] [SWS_PduR_00381] [SWS_PduR_00507] [SWS_PduR_00512] [SWS_PduR_00518] [SWS_PduR_00800]
[SRS_GTW_06122]	The PDU Router shall provide a method that enables COM layer to request cancellation of I-PDU transmission	[SWS_PduR_00700] [SWS_PduR_00701] [SWS_PduR_00710] [SWS_PduR_00721] [SWS_PduR_00722] [SWS_PduR_00723] [SWS_PduR_00724] [SWS_PduR_00726] [SWS_PduR_00727] [SWS_PduR_00732] [SWS_PduR_00736] [SWS_PduR_00769]

Requirement	Description	Satisfied by
[SRS_GTW_06123]	The PDU Router shall provide an interface (API) for usage by bus network management, to use the PDU Router functionality for partial networking	[SWS_PduR_00362] [SWS_PduR_00365] [SWS_PduR_00369]
[SRS_GTW_06124]	The TP transmit buffering strategy shall be configured for each PDU to be routed by the PDU Router	[SWS_PduR_00317] [SWS_PduR_00551] [SWS_PduR_00637] [SWS_PduR_00663] [SWS_PduR_00687] [SWS_PduR_00689] [SWS_PduR_00696] [SWS_PduR_00697] [SWS_PduR_00705] [SWS_PduR_00707] [SWS_PduR_00708] [SWS_PduR_00740] [SWS_PduR_00794] [SWS_PduR_00797] [SWS_PduR_00798] [SWS_PduR_00799] [SWS_PduR_00808] [SWS_PduR_00810] [SWS_PduR_00811] [SWS_PduR_00813] [SWS_PduR_00814] [SWS_PduR_00815]
[SRS_GTW_06125]	Multicast implementation in PduR shall behave such that the source module does not need to know that there is more than one destination module configured	[SWS_PduR_00218] [SWS_PduR_00589] [SWS_PduR_00701] [SWS_PduR_00765] [SWS_PduR_00912] [SWS_PduR_00913] [SWS_PduR_00914] [SWS_PduR_00915]
[SRS_GTW_06126]	Routing of non-TP PDUs from more than one source to one destination using a FIFO shall be supported by the PDU Router	[SWS_PduR_00901] [SWS_PduR_00902]

7 Functional Specification

The PDU Router module is an I-PDU transfer unit placed above Communication Interface and Transport Protocol modules (lower layer modules) and below Com and Dcm (upper layer modules), see Figure 1.1.

Beside the PDU Router module there is the I-PDU Multiplexer (IpduM) module [4] that provides support for multiplexed I-PDUs. The IpduM has to be considered as an upper layer module when it calls the PDU Router module to `Transmit` multiplexed I-PDUs or when it is called by the PDU Router module for the `RxIndication` or `TxConfirmation` of multiplexed I-PDUs or to provide data via `TriggerTransmit`. In case the IpduM calls the PDU Router module to forward a `TxConfirmation` or an `RxIndi-`

`cation` to an upper layer (e.g. Com) or when it is called by the PDU Router module to update an I-PDU belonging to a multiplexed I-PDU it has to be considered as lower layer module.

From the ECU point of view, the PDU Router module can perform three different classes of operations:

- **PDU Reception to local module(s)**: receive I-PDUs from lower layer modules and forward them to one or more upper layer modules,
- **PDU Transmission from local module(s)**: transmit I-PDUs to one or more lower layer modules on request of upper layer module,
- **PDU Gateway**:
    1. receive I-PDUs from a Communication Interface module and transmit the I-PDUs immediately via the same or other Communication Interface module(s)
    2. receive I-PDUs from a Transport Protocol module and transmit the I-PDUs via the same or other Transport Protocol module(s).

The combination of PDU Reception and PDU Gateway is allowed. Example: The Com module is receiving an I-PDU in the same time that it is gatewayed to another lower layer module.

[SWS_PduR_00824] ⌈When the PduR detects a development, runtime, or transient error, it shall use the `moduleId` of the calling module as `instanceId` when calling the Default Error Tracer module.⌋*()*

Note: The standardized module ID is found in the List of Basic Software Modules document [5]. The parameter `PduRBswModuleRef` identifies the module used. With this information the `moduleId` can be retrieved in the `BswModuleDescription.moduleId`.

7.1 I-PDU handling

[SWS_PduR_00160] ⌈The PDU Router module shall transfer an I-PDU without modification in a consistent manner from the source module to the destination module(s).⌋ *(SRS_GTW_06049)*

An I-PDU is identified by the I-PDU ID and/or the symbolic name (i.e. the `SymbolicNameValue` of the container of the I-PDU [6, Specification of ECU Configuration]). For post-build the I-PDU ID is required because the I-PDU must be identified after the PDU Router module is compiled. If the PDU Router module is pre-compile (i.e. in source code) the symbolic names may be used, see [6, Specification of ECU Configuration].

Each BSW module that handles I-PDUs and provides an API for I-PDUs must contain a list of I-PDU IDs [6]. This means that each called module will have a look-up table identifying the I-PDU.

Example: The Com module calls `PduR_ComTransmit` (here the PDU Router module configuration contains the I-PDU ID), the PDU Router module will call `CanIf_Transmit` (here the CanIf module configuration contains the I-PDU ID), the CanIf will call `PduR_CanIfTxConfirmation` (here the PDU Router module configuration contains the I-PDU ID), and PDU Router module will call `Com_TxConfirmation` (here the Com module configuration contains the I-PDU ID). The example is illustrated in the following Figure 7.1 (only I-PDU ID is shown as parameter):

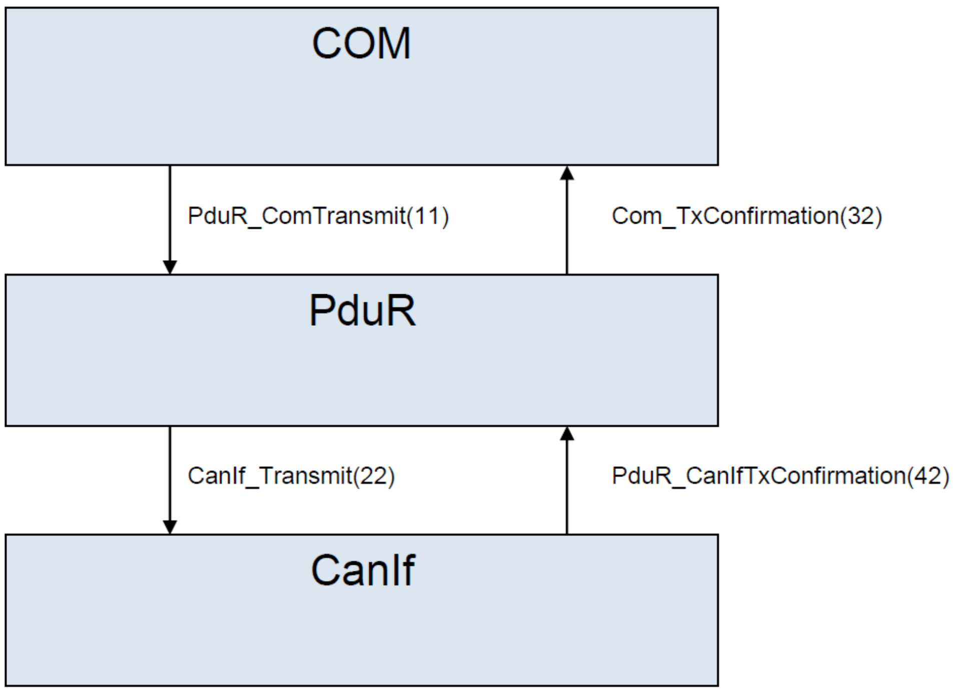

Figure 7.1: I-PDU ID Example

[SWS_PduR_00161] ⌈The PDU Router module shall identify a routing path uniquely by the combination of source module I-PDU ID (located in the PDU Router configuration) and destination I-PDU IDs (located in the called destination module configurations).⌋ *(SRS_GTW_06003)*

[SWS_PduR_00766] ⌈The PDU Router module shall convert the I-PDU ID to the destination module(s) for both `Transmit` path and `TxConfirmation/RxIndication` path.⌋ *(SRS_GTW_06003)*

Example: The Com module transmits an I-PDU to CanIf and LinIf. The `PduR_ComTransmit` is called. The PDU Router module will convert the source I-PDU ID (PDU Router module configuration) to one I-PDU ID for LinIf (LinIf module configuration) and one I-PDU ID for CanIf (CanIf module configuration). The `PduInfoType` value received from the Com module is copied to the CanIf and LinIf modules without change.

Example: The LinIf will call PduR_LinIfTxConfirmation with an I-PDU ID and, dependent on the success of the transmission, with a result E_OK (successful transmission) or E_NOT_OK (not successful transmission). Then the PDU Router module will convert this I-PDU ID and forward the call to Com using Com_TxConfirmation with the converted I-PDU ID and the received result.

[SWS_PduR_00162] ⌈The PDU Router module shall only route I-PDUs according to the routing paths given in the configuration.⌋*(SRS_GTW_06003)*

[SWS_PduR_00828] ⌈PduR generator (validation) shall deny configurations where I-PDUs with different MetaDataTypes are connected by a routing path.⌋*()*

7.1.1 Buffering concept

PduR shall be able to buffer I-PDUs. A routing path is expected to buffer I-PDUs when its related PduRQueueDepth is set. As of today, I-PDU buffering is applicable only for gatewayed I-PDUs and for fan-in reception from multiple comunication interface modules to a local module. Buffering is mandatory in the following cases:

- IF gateway destinations having trigger transmit data provision,
- TP gateways.

In the following chapter the term "FIFO" or "FIFO queue" is used as a synomym for the I-PDU buffer of the PduR. The following subsections explains the type of buffers, their configuration possibilities and relation to routhing paths.

7.1.1.1 Type of buffers

Buffers can be defined by PduRBuffer container. There are two types of buffers from routhing paths' assingment point of view. PduRBuffer which are not referred by any PduRRoutingPath are called global buffers, while the ones which are referred by at least one PduRRoutingPath are the dedicated buffers. Global buffers can be occupied by any PduRRoutingPaths, while dedicated buffers can be occupied only by the PduRRoutingPaths by which they are referred to via PduRDestBufferRef.

The main reason for having dedicated buffers is that functional diagnostic requests and especially OBD request have a very high priority and must not be delayed by buffer allocation strategies that occur because of the lack of memory.

[SWS_PduR_00797] ⌈When an I-PDU needs to get buffered, and the required size is not larger than the configured PduRPduMaxLength of at least one of the free dedicated buffers (PduRBuffer referenced by PduRDestBufferRef), the PduR shall use that dedicated buffer with respect to PduRQueueDepth.⌋*(SRS_GTW_06026, SRS_-GTW_06124)*

[SWS_PduR_00798] ⌈When an I-PDU needs to get buffered, and the required size is larger than the configured PduRPduMaxLength of all free dedicated buffers (PduR-Buffer referenced by PduRDestBufferRef), the PduR shall dynamically allocate a suitably sized global buffer (PduRBuffer not referenced by any PduRDestBuffer-Refs) with respect of PduRQueueDepth.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

7.1.1.2 Buffering strategies

The type of buffering strategy is deteremined by the value of PduRQueueDepth configuration parameter. This parameter specifies the maximum number of PduRBuffers a routing path can occupy simoultanously: PduRBuffers can be taken from the dedicated buffers and form the global buffers (for buffer types see 7.1.1.1 Section). Since a PduRBuffer can hold zero or one I-PDU, PduRQueueDepth implicitly specifies the number of I-PDUs a routing path can buffer.

If the value of PduRQueueDepth is greater than 1, FIFO queue buffering is available. The FIFO has states, and these states may change when various PduR APIs are being called from different contexes. E.g., a PduR_<SrcLo>RxIndication call could be interrupted by a PduR_<DstLo>TxConfirmation call. Thus, there is a need to protect those concurrent calls.

[SWS_PduR_00785] ⌈If PduRQueueDepth is configured to a value greater than 1, the I-PDU buffer shall have a first in - first out (FIFO) behavior.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00787] ⌈In case of FIFO buffering, when a new I-PDU needs to get buffered, and the FIFO queue is not empty then the new I-PDU shall be copied as latest entry.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00307] ⌈In case of multicast routing the PduRQueueDepth and the PduRDestBufferRef can be individually configured. This means buffers of multicast destinations are independent from each other.⌋*(SRS_GTW_06012, SRS_GTW_-06032)*

7.1.1.3 Buffer sharing

On configuration level, it is possible for a PduRBuffer to be referred by multiple PduRDestBufferRefs. Such PduRBuffer can be occupied by any of the referrer PduRRoutingPaths as dedicated buffer, but during runtime, at a specific moment, it can be occupied by only one PduRRoutingPath.

7.1.2 I-PDU Reception to upper layer module(s)

The receive operation of the PDU Router module is always finalized by an `RxIndication` (`PduR_<User:Lo>RxIndication` or `PduR_<User:LoTp>RxIndication`) from a lower layer module (Communication Interface or Transport Protocol module). The `RxIndication` function is originated from the lower layer either in the context of a cyclic function after polling a communication driver or in the context of an interrupt.

[SWS_PduR_00827] ⌈It shall be possible to forward I-PDUs in a n:1 fashion. Only one source shall be enabled at one time, using the `PduR_EnableRouting` and `PduR_DisableRouting` respectively. `PduRIsEnabledAtInit` shall be used to ensure this condition at startup. This does not apply when forwarding from communication interface modules to an upper layer module.⌋*()*

Note: Combined forwarding and gatewaying in n:1 fashion is not supported.

7.1.2.1 Communication Interface

The source Communication Interface module indicates a received I-PDU by calling `PduR_<User:Lo>RxIndication`. The I-PDU may have multiple local destination modules configured by the routing path.

[SWS_PduR_00164] ⌈The PDU Router module shall provide 1:n routing for an I-PDU received from a Communication Interface module and routed to one or more upper layer module(s).⌋*(SRS_GTW_06012, SRS_GTW_06030)*

Example: An I-PDU is received on CanIf and forwarded to Com.

Note: An I-PDU may be received by one or more upper layer modules in the same time as gatewayed to one or more Communication Interface destinations, see 7.1.4.

[SWS_PduR_00621] ⌈When the `PduR_<User:Lo>RxIndication` is called the PDU Router module shall call `<Up>_RxIndication` for each destination upper layer module.⌋*(SRS_GTW_06012)*

[SWS_PduR_00744] ⌈If an I-PDU received by a local module is directly forwarded, the PDU Router shall not check the length of the I-PDU.⌋*(SRS_GTW_06012)*

Since the PDU Router module will not buffer this I-PDU it does not have to reject I-PDU that are longer/shorter than configured.

[SWS_PduR_00901] ⌈The PDU Router module shall provide n:1 routing for I-PDUs received from multiple communication interface modules to one upper layer module.⌋*(SRS_GTW_06126)*

[SWS_PduR_00902] ⌈The PDU Router module shall optionally support a reception FIFO in case of n:1 routing from multiple communication interface modules to one upper layer module.⌋*(SRS_GTW_06126)*

Note: If the user cannot rule out the possibility of concurrent requests for an n:1 routing point, a FIFO is required to serialize the concurrent reception requests.

[SWS_PduR_00903] ⌈If `PduR_<User:Lo>RxIndication` is called for a n:1 routing point without a FIFO configured and the `<Up>_RxIndication` call of the most recent request has not returned, the PDU Router shall return immediately without calling `<Up>_RxIndication` and report `PduR.PDUR_E_PDU_INSTANCES_LOST` to the DET module.⌋*()*

[SWS_PduR_00904] ⌈If `PduR_<User:Lo>RxIndication` is called for a n:1 routing point with a FIFO configured and the `<Up>_RxIndication` call of the most recent request has not returned, the PDU Router shall buffer the I-PDU.⌋*()*

[SWS_PduR_00905] ⌈After `<DstUp>_RxIndication`, called with an I-PDU from the FIFO buffer returns, the I-PDU shall be removed from the FIFO and the next FIFO entry shall be provided to the upper layer, if available.⌋*()*

7.1.2.2 Transport Protocol

The standard use case for reception of I-PDU through Transport Protocol is only one upper layer module configured per routing path.

In case of multiple Software Clusters, this I-PDU may also be received by several upper layer modules.

Example: A functional addressed request is received from the CanTp module (residing in the Host Software Cluster) and routed to two Dcm module instances (residingin different Applicative Software Clusters on the same partition as the Host Software Cluster).

In case source and destination reside on different partitions the inter-partition routing principles described in section 7.10 need to be taken in account.

7.1.2.2.1 1:1 routing for an I-PDU received from a source transport protocol module

In case of a Transport Protocol module the PDU Router module is first notified with a start of reception notification when receiving a first frame (FF) or Single Frame (SF). This call is be forwarded to the related upper layer module by calling `<Up>_StartOfReception`. The payload of each segment (N-PDU) is to be copied in the upper layer destination module within the subsequent `<Up>_CopyRxData` calls. After reception of the last N-PDU the Transport Protocol module will indicate the PDU Router module that the complete I-PDU has been received and the PDU Router module will forward this indication to the related upper layer module by calling `<Up>_TpRxIndication`.

[SWS_PduR_00673] ⌈The PDU Router module shall provide 1:1 routing for an I-PDU received from a source Transport Protocol module and routed to one upper layer destination module.⌋*(SRS_GTW_06026)*

Example: A functional addressed request (in a SF) is received from the CanTp module and routed to the Dcm module.

[SWS_PduR_00549] ⌈When a source Transport Protocol module indicates the start of a reception of a PDU that has only upper layer destination using `PduR_<User:LoTp>StartOfReception`, the PDU Router module shall forward the request to the upper layer destination module by calling `<Up>_StartOfReception`.⌋*(SRS_GTW_06026)*

[SWS_PduR_00623] ⌈The PDU Router shall forward the return value of the `<Up>_StartOfReception` to the source Transport Protocol module.⌋*(SRS_GTW_06104)*

[SWS_PduR_00428] ⌈When a source Transport Protocol module requests the PDU Router module to copy the received data using `PduR_<User:LoTp>CopyRxData`, the PDU Router module shall forward the request to the upper layer destination module by calling `<Up>_CopyRxData`.⌋*(SRS_GTW_06026)*

[SWS_PduR_00429] ⌈When a source Transport Protocol module calls `PduR_<User:LoTp>RxIndication` indicating reception of the complete I-PDU, the PDU Router module shall forward the indication to the upper layer destination module by calling `<Up>_TpRxIndication`.⌋*(SRS_GTW_06026)*

[SWS_PduR_00207] ⌈If the source Transport Protocol module reports an error using `PduR_<User:LoTp>RxIndication`, the PDU Router module shall not perform any error handling other than forwarding the `RxIndication` to the upper layer module.⌋*(SRS_GTW_06104)*

7.1.2.2.2 1:n forwarding for an I-PDU received from a source transport protocol module

[SWS_PduR_00916]{DRAFT} ⌈The PDU Router module shall provide 1:n forwarding for an IPDU received from a source Transport Protocol module and routed to several upper layer destination modules.⌋*(SRS_GTW_06026)*

If the I-PDU is received by more than one local upper layer modules, the forwarding to these upper layers is handled similar to direct Transport Protocol gatewaying. The PduR buffers 1:n forwarding PDUs in dedicated buffers (`PduRBuffer`s) configured via `PduRDestBufferRef` and informs the upper layers in the context of `RxIndication`.

[SWS_PduR_00917]{DRAFT} ⌈When a successful `RxIndication` is received by PduR from the lower layer, the module shall initiate a reception session for each configured upper layer destination: `<UpTp>_StartOfReception`, `<UpTp>_CopyRxData`, and `<UpTp>_RxIndication` will be called in this order.⌋*(SRS_GTW_06026)*

[SWS_PduR_00912]{DRAFT} ⌈In case of several local destinations, the PduR shall perform the error handling individually for each destination.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105, SRS_GTW_06125)*

The other destinations should not be affected by the error of one destination upper layer module.

[SWS_PduR_00913]{DRAFT} ⌈When an error is returned by `PduR_<User:UpTp>StartOfReception` for a multicast with several local destinations, the PduR shall stop the respective upper layer reception without further interaction with the upper layer.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105, SRS_GTW_06125)*

[SWS_PduR_00914]{DRAFT} ⌈When `PduR_<User:UpTp>StartOfReception` returns `BUFREQ_OK`, but the available buffer is too small to receive the whole message, the PduR shall call `PduR_<User:UpTp>RxIndication` with `result = E_NOT_OK` for the respective upper layer module.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105, SRS_GTW_06125)*

[SWS_PduR_00915]{DRAFT} ⌈When `PduR_<User:UpTp>CopyRxData` returns an error, the PduR shall call `PduR_<User:UpTp>RxIndication` with `result = E_NOT_OK` for the respective upper layer module.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105, SRS_GTW_06125)*

7.1.2.2.3 Handling I-PDUs with unknown length

The PduR is able to handle unknown length I-PDUs (i.e. streaming type of data) using the TP API. The definition of unknown length is indicated by `TpSduLength`=0.

[SWS_PduR_00821] ⌈In a local receive situation, when `PduR_<User:LoTp>StartOfReception` is called with `TpSduLength`=0, PduR shall call `<Up>_StartOfReception` with `TpSduLength`=0.⌋*()*

7.1.3 I-PDU Transmission from upper layer module(s)

The transmit operations of the lower layer destination modules are always asynchronous. This means that a transmission service request returns immediately after the I-PDU has been passed by the PDU Router module to the lower layer destination(s). If the PDU Router module is notified by lower layer destination modules via `PduR_<User:Lo>TxConfirmation` (Communication Interface) or `PduR_<User:LoTp>TxConfirmation` (Transport Protocol) after successful or failed transmission of the I-PDU, the PDU Router module will forward this confirmation to the upper layer module via `<Up>_TxConfirmation` (Communication Interface) or `<Up>_TpTxConfirmation` (Transport Protocol).

The transmit operation of the PDU Router module is triggered by a PDU `Transmit` request from an upper layer source module and the PDU Router forwards the request to lower layer destination(s).

[SWS_PduR_00629] ⌈The I-PDU shall not be buffered in the PDU Router module in case of PDU transmission from an upper layer source module.⌋*(SRS_GTW_06012, SRS_GTW_06026)*

7.1.3.1 Multicast

The multicast feature is separated to an own section since there are issues using this feature as described in Section 4.1.1.

Further requirements that are directly handled by the PDU Router module:

[SWS_PduR_00218] ⌈If the provided I-PDU ID represents a group of PDUs (multicast transmit request) and at least one of the forwarded transmit requests returns successfully, the function `PduR_<Up>Transmit` shall return `E_OK`.⌋*(SRS_GTW_06125)*

Note that Communication Interfaces returning with `E_OK` will transmit their data either directly or via trigger transmit.

The other Transport Protocol modules may return `E_NOT_OK`, and therefore these modules will not call the `PduR_<User:LoTp>CopyTxData`. Since the upper layer source module has been informed that at least one transmission was successful, at least one Transport Protocol module will call `PduR_<User:LoTp>CopyTxData`.

[SWS_PduR_00633] ⌈If there are more than one lower layer destination modules in a transmission request (1:n, n>1), all of these modules must either be Communication Interface modules or Transport Protocol modules. Not a mix of them.⌋*(SRS_GTW_06029, SRS_GTW_06030)*

Example: Above requirement means basically that the Com module cannot request a transmission to CanTp and CanIf modules at the same time via `PduR_ComTransmit`.

[SWS_PduR_00589] ⌈In case of a multicast (1:n, n>1) Communication Interface transmission, the PDU Router shall call the transmit confirmation API of the upper layer module when the last transmit confirmation from a Communication Interface module which supports transmit confirmation has been received.⌋*(SRS_GTW_06125, SRS_GTW_06119)*

Note: The above requirement even works if not all destinations provide `TxConfirmations`.

Implementation note: When the source module requests a transmission and the PduR will make a multicast (1:n, n>1), all the I-PDUs in the request and the multicast will have different I-PDU IDs. Therefore the PduR must remember the I-PDU ID from the transmission request so the transmission can be confirmed correctly.

7.1.3.2 Communication Interface

There are three ways that I-PDUs can be transmitted on Communication Interface:

1. `Direct data provision` - where the upper layer module is calling the `PduR_<User:Up>Transmit` function, the PDU Router module forwards the call to `<Lo>_Transmit` and the data is copied by the lower Communication Interface module in the call.
2. `Trigger transmit provision` - where the lower Communication Interface module requests transmission of an I-PDU by using the `PduR_<User:Lo>TriggerTransmit`, and PDU Router module forwards the call to `<Up>_TriggerTransmit` and the data is copied to the destination's buffer by the upper layer module.
3. Where the upper layer module calls the `PduR_<User:Up>Transmit` function, the PDU Router module forwards the call to `<Lo>_Transmit` and the data is not copied by the lower module (Communication Interface module). The data will later be requested by the lower layer using `PduR_<User:Lo>TriggerTransmit`.

The confirmation of the transmission of the I-PDU is the same for the `direct` and `trigger transmit data provision`:

[SWS_PduR_00627] ⌈When the Communication Interface module calls `PduR_<User:Lo>TxConfirmation` the PDU Router shall call `<Up>_TxConfirmation` in the upper layer module and forward the transmission `result` from the lower to the upper layer module.⌋*(SRS_GTW_06012)*

[SWS_PduR_00745] ⌈If the I-PDU is transmitted by an upper layer module the PDU Router module shall not check the length of the I-PDU.⌋*(SRS_GTW_06012)*

[SWS_PduR_00625] ⌈When upper layer source module calls `PduR_<User:Up>Transmit` the PDU Router shall call `<Lo>_Transmit` for each Communication Interface destination module.⌋*(SRS_GTW_06026)*

[SWS_PduR_00626] ⌈If singlecast (1:1) the return value of the `<Lo>_Transmit` call shall be forwarded to the upper layer source module.⌋*(SRS_GTW_06012, SRS_GTW_06104)*

7.1.3.2.1 Trigger transmit data provision

The upper layer module must be informed whether it has to reset the update-bits.

[SWS_PduR_00430] ⌈The PDU Router module shall forward a `PduR_<User:Lo>TriggerTransmit` request by the Communication Interface lower layer module to the upper layer module by calling `<Up>_TriggerTransmit`.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00661] ⌈The PDU Router module shall copy the return value from the `<Up>_TriggerTransmit` to the lower layer module.⌋*(SRS_GTW_06104)*

7.1.3.2.2 Error handling

For errors occurred using singlecast or multicast over Communication Interface modules, no specific error handling is done. Errors in return values are forwarded to the upper layer source module.

7.1.3.3 Transport Protocol

Transmitting I-PDU using Transport Protocol has two flavors, singlecast and multicast. A singlecast (1:1) transmission consists of one upper layer source module and one lower layer Transport Protocol destination module. A multicast (1:n, n>1) transmission consists of more than one lower layer Transport Protocol destination module. The PDU Router module will not check if the transmission request contains a single N-PDU (SF) or multiple N-PDU (FF, CF, . . .).

Initiation of transmission of I-PDU is made by a `PduR_<User:Up>Transmit` request by an upper layer source module. The PduR will forward the request to one or more lower layer Transport Protocol destination modules using `<Lo>_Transmit` according to the routing paths. Note that the `<Lo>_Transmit` may or may not contain data.

The destination module(s) will request data by calling the `PduR_<User:LoTp>CopyTxData`. Retransmission (if supported by the Transport Protocol) of data is made by the `RetryInfoType` parameter. Finalizing the transmission the destination module(s) calls the `PduR_<User:LoTp>TxConfirmation`, which is forwarded to the upper layer source module.

The multicast TP transmission is described in chapter 7.1.3.3.1.

[SWS_PduR_00634] ⌈When an upper layer module calls the `PduR_<User:Up>Transmit` the PDU Router module shall call `<LoTp>_Transmit` for each Transport Protocol destination module.⌋*(SRS_GTW_06026)*

[SWS_PduR_00299] ⌈When a Transport Protocol destination module calls `PduR_<User:LoTp>CopyTxData` the PDU Router module shall call `<Up>_CopyTxData` in the upper layer source module.⌋*(SRS_GTW_06026)*

[SWS_PduR_00676] ⌈The return value from the `<Up>_CopyTxData` shall be forwarded to the calling lower layer Transport Protocol destination module.⌋*(SRS_GTW_06104)*

[SWS_PduR_00301] ⌈In case of singlecast the PDU Router module shall forward the confirmation `PduR_<User:LoTp>TxConfirmation` from the lower layer Transport Protocol destination module to upper layer source module using `<Up>_TpTxConfirmation`.⌋*(SRS_GTW_06026)*

[SWS_PduR_00432] ⌈In case of singlecast and after calling `<Lo>_Transmit` then the PDU Router module shall return with the same return value to the calling `PduR_<User:Up>Transmit` from upper layer source module.⌋*(SRS_GTW_06104)*

7.1.3.3.1 Multicast transmission

This subsection contains specific requirements for the multicast transmission of I-PDU using Transport Protocol modules.

Since the 1:n, n>1 routing is made in the PDU Router module the PDU Router module must request the same data several times from the upper layer source module. Also the confirmation of the multicast must be handled specifically.

As the upper layer shall copy the same data several times, the PDU Router will use the `RetryInfoPtr` [7] in order to query the same data several times. The `RetryInfo Ptr` contains a state type called `TpDataState` .

Therefore the transport protocol destinations do not set the `TpDataState` to `TP_DATARETRY`.

[SWS_PduR_00871] ⌈When `PduR_<User:LoTp>CopyTxData` is called with a `Tp DataState` set to `TP_DATARETRY` for a multicast TP transmission type routing path, PduR shall return `E_NOT_OK`.⌋*()*

The multicast transmission of N-PDUs is performed in a lock-step mode, where the slowest destination dictates the rate of transmission. The use-case behind the multicast multiframe transmission is the broadcast of J1939Tp BAM messages (messages like DM1 or CommandedAddress have more than 8 Bytes and need to be broadcast to several destinations).

[SWS_PduR_00872] ⌈When `PduR_<User:LoTp>CopyTxData`is called for a multicast TP transmission session for a destination which previously had already fetched more data than the upper layer buffer's read index currently points to, PduR shall return `BUFREQ_E_BUSY`.⌋*()*

Implementation hint: In this case for each destination PduR must remember how many bytes have been transmitted so far and also shall keep tracking.

[SWS_PduR_00631] ⌈For a new multicast TP transmission session, the request of `PduR_<User:LoTp>CopyTxData` of the first destination shall be forwarded with `Tp DataState` set to `TP_CONFPENDING`.⌋*(SRS_GTW_06029)*

[SWS_PduR_00632] ⌈For all subsequential calls of `PduR_<User:LoTp>CopyTxData` for the same multicast TP transmission session, PduR shall overwrite `TpData State` to `TP_DATARETRY` and adjust `Tx TpData Cnt` to request data to be transmitted from the upper layer source buffer from the previous transmission point related to that destination.⌋*(SRS_GTW_06029)*

Note: `Tx TpData Cnt` is an "Offset from the current position." see Specification of Communication Stack Types [7].

[SWS_PduR_00812] ⌈After all Transport Protocols have received their data the PDU Router module may confirm the data to the upper layer module.⌋*(SRS_GTW_-06029)*

[SWS_PduR_00765] ⌈In case of multicast transmission, the PDU Router module shall call the upper layer module using `<Up>_TpTxConfirmation` after receiving the last `PduR_<User:LoTp>TxConfirmation` from the lower layer Transport Protocol modules. The `result` parameter shall be `E_OK` if at least one `PduR_<User:LoTp>Tx-Confirmation` reported `E_OK`.⌋*(SRS_GTW_06029, SRS_GTW_06125)*

7.1.3.3.2 Error handling

The PDU Router module will not take specific actions on errors occurred, the errors will be forwarded to the upper layer source module. Appropriate error handling is in the responsibility of the upper layer module.

7.1.3.3.3 Handling I-PDUs with unknown length

The PduR is able to handlle unknown length I-PDUs (i.e. streaming type of data) using the TP API. The definition of unknown length is indicated by `TpSduLength`=0.

[SWS_PduR_00822] ⌈In a local transmit situation, when `PduR_<User:Up>Trans-mit` is called with `PduInfoType.SduLength=0` and I-PDU is routed to a TP-module, the PduR shall call `<LoTp>_Transmit` with `PduInfoType.SduLength=0` to all destination TP modules.⌋*()*

7.1.4 I-PDU Gatewaying

The PDU Router module supports gatewaying of I-PDUs from one source bus to one or more destination buses. In addition, it is possible to forward gatewayed I-PDUs to upper layer modules. Moreover, multiple sources can be configured and enabled at the same time with one destination set, to realize a fan-in gateway. The difference from a transmission and reception from/to a local module is that the PDU Router module must be a receiver and transmitter at the same time, and in some cases also provides buffering for the I-PDUs.

The gateway requirements are deliberately separated to allow an efficient implementation of the PDU Router module in case gatewaying is not needed. In case the PDU Router module allows gatewaying of I-PDUs, these requirements are seen as additional and not replacing previous requirements.

Following list gives an overview of the features of the I-PDU gateway:

- I-PDUs may be gatewayed from a source of a Communication Interface module to one (1:1) or more destinations of Communication Interface module(s) (1:n I-PDU gateway)
  - PDU Router module may set the type of buffering for each destination independently (i.e., FIFO if more than one I-PDU).
  - An I-PDU may be received by destinations of upper layer module(s) at the same time as gatewayed to n destinations of Communication Interface.
- I-PDUs transported using TP may be gatewayed to one or more destinations of TP module(s), with the following scope:
  - Both Single Frames and Multi Frames can be gatewayed to more than one destination of TP module(s) and in the meantime forwarded to one or more destinations of local module(s) (e.g., Dcm).
  - Multi Frames may be gatewayed "`on-the-fly gatewaying`" to one destination, meaning that complete I-PDU does not need to be received before starting transmission on the destination TP module
  - TP gateway I-PDUs must be buffered, and the buffer depth must be fonfigurable. Last is best behaviour of Single Frames are not appicable.

This means the PDU Router module shall forward an I-PDU received from one lower layer module (source network) to lower layer modules (destination networks) identified by the provided I-PDU ID.

Note that in this section "`Src`" and "`Dst`" are used for the configurable APIs. This is just to be clear which call belongs to the source module and destination module.

[SWS_PduR_00638] ⌈An I-PDU may only be gatewayed either between Communication Interface modules or between Transport Protocol modules, not a mix of them.⌋ *(SRS_GTW_06012, SRS_GTW_06026)*

Example: An I-PDU received from FrIf may not be gatewayed to CanTp.

[SWS_PduR_00825] ⌈It shall be possible in a gatewaying situation, to gateway I-PDUs in a n:1 fashion.⌋*()*

[SWS_PduR_00826] ⌈If using n:1 gatewaying the PduR shall ensure that the sequence of incoming I-PDUs is preserved on the destination.⌋*()*

Note: Combined forwarding and gatewaying in n:1 fashion is not supported.

[SWS_PduR_00829] ⌈In a gateway situation, Meta Data is contained and buffering is needed, the PduR shall in addition to the I-PDU also buffer the Meta Data.⌋*()*

7.1.4.1 Communication interface

An I-PDU can be configured to be received on one Communication Interface module and gatewayed to n destinations of Communication Interface modules including local module(s), i.e. 1:n gatewaying. For gatewaying it is also possible to configure a buffer for each lower layer Communication Interface destination of module (not local module however).

General requirements applicable for Communication Interface type gateways are listed below:

[SWS_PduR_00436] ⌈The PDU Router module shall support routing of I-PDUs between a source Communication Interface module and one or more destinations of Communication Interface destination module(s) (1:n gatewaying).⌋*(SRS_GTW_06012, SRS_GTW_06030)*

[SWS_PduR_00437] ⌈The PDU Router module shall support routing of I-PDUs between Communication Interface modules with immediate transmission (without rate generation by PDU Router).⌋*(SRS_GTW_06012)*

Routing of I-PDUs between Communication Interface modules with different period or rate (rate conversion) is not supported, this can be done via the Com module using signal gateway. In this case the I-PDU has to be routed to the Com module.

There are two flavors of gatewaying an I-PDU depending on the the Communication Interface destination module. The used flavor is controlled by the configuration:

- **[SWS_PduR_00303]** ⌈`Direct data provision`: The `PduRDestPduDataProvision` of the destination I-PDU is configured to `PDUR_DIRECT`. When `<DstLo>_Transmit` is called the `<DstLo> module` copies the data and the PDU Router does not buffer the transmitted I-PDU any longer.⌋*(SRS_GTW_06032)*
- **[SWS_PduR_00306]** ⌈`Trigger transmit data provision`: The `PduRDestPduDataProvision` of the destination I-PDU is configured to `PDUR_TRIGGERTRANSMIT`. When `<DstLo>_Transmit` is called the <DstLo> module does not copy the data and the PDU Router module shall buffer the I-PDU and wait for the `PduR_<DstLo>TriggerTransmit` call from the `<DstLo>` module.⌋*(SRS_GTW_06032)*

7.1.4.1.1 Buffered gatewaying

Please note that, for Communication Interface gateway destinations having `PDUR_DIRECT` type `PduRDestPduDataProvision`, buffering is not mandatory. See 7.1.1.2 Section.

It is possible that an I-PDU that will be gatewayed will have different lengths. Therefore:

[SWS_PduR_00746] ⌈In case the I-PDU is buffered in the PDU Router module: The PDU Router module shall copy the data of the I-PDU up to smallest of the following values:

- the received data length (PduLength of received I-PDU)
- the configured length of the destination I-PDU. In this case the rest of the received I-PDU shall be dropped.

⌋*(SRS_GTW_06012, SRS_GTW_06032)*

Note: [SWS_PduR_00746] gives the possibility to avoid buffer over run when PduR_<DstLo>TriggerTransmit is called. The dedicated buffer length can not be greater than the length of the destination I-PDU.

[SWS_PduR_00784] ⌈If direct data provision is used with a FIFO: when the I-PDU is transmitted from the PduR buffer to the destination module, the PduR shall pass the number of bytes which was copied to the buffer as SduLength.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00793] ⌈If direct data provision is used with a FIFO: the PduR shall enqueue new data in the FIFO when PduR_<SrcLo>RxIndication is called and the last transmission of the same PDU has not yet been confirmed via PduR_<DstLo>TxConfirmation.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00665] ⌈If direct data provision is used with a FIFO: when PduR_<SrcLo>RxIndication is called and the FIFO queue is empty and no confirmation is outstanding for the same PDU, <DstLo>_Transmit shall be called directly. The FIFO stays empty.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00667] ⌈If direct data provision is used with a FIFO: when PduR_<DstLo>TxConfirmation is called and the FIFO queue is not empty <DstLo>_Transmit shall be called with the oldest I-PDU of the FIFO. The transmitted I-PDU shall be removed afterwards.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00786] ⌈If trigger transmit data provision is used with a FIFO: when PduR_<SrcLo>RxIndication is called and the FIFO queue is empty the received I-PDU shall be enqueued into the FIFO and <DstLo>_Transmit shall be called.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

[SWS_PduR_00662] ⌈If trigger transmit data provision is used with a FIFO: when the destination comunication interface module is requesting the I-PDU buffer using PduR_<DstLo>TriggerTransmit and the FIFO is empty the return value E_NOT_OK shall be used.⌋*(SRS_GTW_06032, SRS_GTW_06105)*

Note that for a gateway of an I-PDU the PduR_<DstLo>TxConfirmation is not interesting (except for FIFO of a direct data provision I-PDU).

[SWS_PduR_00640] ⌈If the Communication Interface destination module confirms the transmission of the I-PDU (successful or failed) using PduR_<DstLo>TxConfirmation and destination is not a direct data provision PDU with FIFO buffer, the

PDU Router module shall not do anything.⌋*(SRS_GTW_06012, SRS_GTW_06032, SRS_GTW_06105)*

[SWS_PduR_00819] ⌈If `trigger transmit data provision` is used with a FIFO: when `PduR_<DstLo>TriggerTransmit` is called to copy an I-PDU from the PduR buffer to the destination module, the PduR shall check the lower layer's buffer size provided as `SduLength`. In case the buffer is too small for the stored PDU data, the PduR shall return `E_NOT_OK` and not process the `TriggerTransmit` call any further.⌋*()*

Note: Not processing the `TriggerTransmit` call as defined in [SWS_PduR_00819] does mean that the PDU shall not be removed from the PduR buffer.

[SWS_PduR_00666] ⌈If `trigger transmit data provision` is used with a FIFO: when `PduR_<DstLo>TriggerTransmit` is called and will return `E_OK` according to [SWS_PduR_00819], the oldest FIFO entry shall be copied and then removed. If afterwards the FIFO queue is not empty `<DstLo>_Transmit` shall be called with the oldest I-PDU of the FIFO.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

Note: In case of the destination module is FrIf the FrIfCounterLimit of the PDU needs to be configured > 1 because the new transmit will be called before the counter is decremented. For LinIf there is no such a constraint, however FIFO queue routing to sporadic frames is not supported.

[SWS_PduR_00809] ⌈If `trigger transmit data provision` is used with `last-is-best buffering` (`PduRQueueDepth` set to 1): the PDU Router shall buffer the latest I-PDU.⌋*(SRS_GTW_06032)*

The reason why it must be stored for `trigger transmit data provision` is that the Communication Interface destination may transmit the I-PDU according to a schedule. Then the Communication Interface will call the `PduR_<DstLo>TriggerTransmit` without a preceding `<DstLo>_Transmit` call.

7.1.4.1.2 Immediate gatewaying

Immediate gatewaying means that an I-PDU must be gatewayed without any buffering mechanism of PduR. This can be realized only by Communication Interface gateways having destination direct data provision type.

It is possible that an I-PDU that will be gatewayed will have different lengths. Therefore:

[SWS_PduR_00783] ⌈In case the I-PDU is gatewayed without buffering in the PDU Router, the PDU Router shall forward the length of the I-PDU up to the smallest of the following values:

- either the received data length `(PduLength` of received I-PDU)
- or the configured data length of the destination I-PDU (`PduLength` of `PduRDestPdu`

⌋*(SRS_GTW_06012, SRS_GTW_06032)*

7.1.4.2 Transport Protocol

Gatewaying I-PDU from a source Transport Protocol to one or more destinations of transport protocol destination module(s) may either be gatewayed direct as a complete I-PDU (complete set of N-PDUs building up the I-PDU is received before transmitted) or as fragmented I-PDU (`on-the-fly gatewaying`) where a configured number of bytes `PduRTpThreshold` are received before transmission.

In general the PDU Router will gateway the payload only, and will not be aware of Transport Protocol details such as SF, FF, CF, PCI etc. But the PduR shall also support gatewaying of I-PDUs with MetaData, configured using the `MetaDataType` of the global PDU. This type of I-PDUs requires no special treatment during interface routing or forwarding, but for TP routing, the additional information has to be forwarded separately. The following requirement is relevant both for direct gatewaying and `on-the-fly gatewaying`:

[SWS_PduR_00794] ⌈The MetaData of I-PDUs provided by `PduR_<SrcLoTp>StartOfReception` shall be stored and provided with the I-PDU to `<DstLoTp>_Transmit`.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

On a Transport Protocol module an I-PDU can be transported in multiple N-PDUs (FF and CFs) or in a single N-PDU (SF). One use-case is that an I-PDU transported in multiple N-PDUs is not multicast (i.e., physical addressing) and in single N-PDU may be multicast (i.e. functional addressing). Another use-case is multicast of a Multi Frame message to a local receiver and to multiple gateway destinations.

For example: A SF received on CAN and shall be transmitted on two LIN busses. The received SF can carry up to data 6 bytes but a SF on LIN only up to data 5 bytes. Therefore the SF on CAN is limited to data 5 bytes if gatewayed to the two LIN busses.

Note that an I-PDU transported over Transport Protocol modules may also be gatewayed frame by frame directly through the Communication Interfaces (i.e. by gatewaying the N-PDUs directly). This requires no special treatment here of the PDU Router module and can be handled by gatewaying through Communication Interface modules, see Section 7.1.4.1. However, this requires that the source and destination busses have exactly same packing of N-PDUs (e.g. from CAN to CAN).

[SWS_PduR_00830] ⌈When `PduR_<SrcLoTp>StartOfReception` the PduR shall allocate enough buffers ([SWS_PduR_00797], [SWS_PduR_00798]) from `PduRBuffer` (for each destination).⌋*()*

[SWS_PduR_00831] ⌈The PduR shall start transmission on destination Transport Protocol when either `PduRTpThreshold` or complete (`PduR_<DstLoTp>RxIndication` is called) I-PDU is received.⌋*()*

[SWS_PduR_00832] ⌈If another PduR_<SrcLoTp>StartOfReception for same routing path(s) is called, then PduR shall enqueue the I-PDU instance into the FIFO.⌋ *()*

[SWS_PduR_00833] ⌈When PduR_<DstLoTp>TxConfirmation is received from Transport Protocol destination module, the PduR shall start transmission of next I-PDU if FIFO is not empty.⌋*()*

[SWS_PduR_00835] ⌈If FIFO queue is already containing at least one entry the received message shall be stored in the FIFO, and <DstLoTp>_Transmit shall be called as soon as this FIFO queue entry is due for transmission (i.e. when this message is first on the FIFO).⌋*()*

Note: The effect of on-the-fly gatewaying using FIFO is that it will be a faster way to gateway the TP messages. Obviously if the FIFO is not empty then the message must be stored and not to be forwarded to the desintation TP.

[SWS_PduR_00696] ⌈If PduR_<DstLoTp>CopyTxData is called and state is TP_DATACONF then the PDU Router may free the already copied data.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00637] ⌈When the PDU Router module receives the PduR_<DstLoTp>TxConfirmation, the PDU Router shall free the I-PDU buffer for this destination.⌋ *(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00740] ⌈If the Transport Protocol module calls PduR_<DstLoTp>CopyTxData or PduR_<SrcLoTp>CopyRxData with zero length (PduInfoType.SduLength = 0) the PDU Router module shall return the size of the currently available buffer or the currently available data respectively.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00818] ⌈For TP gateway scenario, the availableDataPtr of the PduR_<DstLoTp>CopyTxData shall indicate the remaining number of bytes that are available in the PduR's (gateway) TP buffer.⌋*()*

7.1.4.2.1 Direct gatewaying

[SWS_PduR_00551] ⌈The <DstLoTp>_Transmit shall be called on each destination of transport protocol module(s) within the PduR_<SrcLoTp>RxIndication, if result is E_OK.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06124)*

[SWS_PduR_00697] ⌈For Single Frame transmission, if PduR_<DstLoTp>CopyTxData is called with TpDataState TP_CONFPENDING or TP_DATACONF or when the RetryInfoType pointer is NULL, the PDU Router shall copy SduLength bytes of data. If not enough data is avilalble, the PDU Router shall return BUFREQ_E_BUSY without copying any data.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00705] ⌈For Single Frame transmission, if PduR_<DstLoTp>CopyTxData is called with TpDataState TP_DATARETRY, the PDU Router shall set back the current position by Tx TpData Cnt bytes and copy SduLength bytes of data. If the PDU Router cannot set back the position as requested, it shall return BUFREQ_E_NOT_OK without changing the current position or copying any data. If, after resetting the current position, not enough data is available for copying, the PDU Router shall return BUFREQ_E_BUSY without copying any data.⌋*(SRS_GTW_06026, SRS_GTW_06124, SRS_GTW_06105)*

[SWS_PduR_00813] ⌈For Multi Frame transmission, if PduR_<DstLoTp>CopyTxData is called with TpDataState TP_CONFPENDING or TP_DATACONF or when the RetryInfoType pointer is NULL, the PDU Router shall copy SduLength bytes of data. Only the call from the last destination module shall increase the buffer position.⌋ *(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00814] ⌈If not enough data is availalble or not all other destination Transport Protocoll modules have called PduR_<DstLoTp>CopyTxData for the previous frame, the PDU Router shall return BUFREQ_E_BUSY without copying any data.⌋ *(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00815] ⌈For Multi Frame transmission, if PduR_<DstLoTp>CopyTxData is called with TpDataState TP_DATARETRY, the PDU Router shall return BUFREQ_E_NOT_OK without copying any data.⌋*(SRS_GTW_06026, SRS_GTW_06124, SRS_GTW_06105)*

7.1.4.2.2 On-the-fly gatewaying

In on-the-fly gatewaying the PDU Router module will start transmission to the Transport Protocol destination module when a specific threshold (configured by PduRTpThreshold) is reached.

[SWS_PduR_00708] ⌈Using on-the-fly gatewaying only one destination of transport protocol module is allowed.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00317] ⌈The PDU Router module shall start the TP transmission on the destination by calling <DstLoTp>_Transmit as soon as the PduRTpThreshold has been reached for the specific destination.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00811] ⌈If a TP transmission is started via PduR_<SrcLoTp>StartOfReception, the PDU Router module shall directly call <DstLoTp>_Transmit if PduRTpThreshold = 0.⌋*(SRS_GTW_06124)*

[SWS_PduR_00808] ⌈The PDU Router module shall start the TP transmission on the destination by calling <DstLoTp>_Transmit if result value is E_OK in the PduR_<SrcLoTp>RxIndication even if the PduRTpThreshold was not reached.⌋*(SRS_GTW_06012, SRS_GTW_06026, SRS_GTW_06124)*

If on-the-fly gatewaying is used the PDU Router shall not support retransmission of already transmitted data:

[SWS_PduR_00707] ⌈If `PduR_<DstLoTp>CopyTxData` is called with `TpDataState TP_DATACONF` or if the `RetryInfoType` pointer is `NULL`, the PDU Router shall copy `SduLength` bytes of data.⌋*(SRS_GTW_06026, SRS_GTW_06124)*

[SWS_PduR_00823] ⌈In a gateway situation and `PduR_<SrcLo>StartOfReception` is called with `TpSduLength`=0 only `on-the-fly gatewaying` is supported.⌋*()*

7.1.4.3 Forwarding to upper layers

If the I-PDU is gatewayed (direct Transport Protocol gateway) to one or more destinations of transport protocol module(s), this I-PDU may be also received by a local upper layer module.

In case of multiple Software Clusters, this I-PDU may also be received by several upper layer modules. As the relevance of such scenario is limited to functional addressing of multiple Dcm instances in a clusters software architecture, the constraint to support it for direct gatewaying only, does not pose a problem because only I-PDUs of limited size can be expected.

Implementations may choose to report (via DET) whenever the upper layer reception was not successful. The gatewaying to lower layers should not be aborted in this case.

The reception to the upper layers of direct gatewaying is specified below:

[SWS_PduR_00789] ⌈In case of gatewaying, when a successful `RxIndication` is received by PduR from the lower layer, the module shall initiate a reception session for a configured upper layer destination: `<UpTp>_StartOfReception`, `<UpTp>_CopyRxData`, and `<UpTp>_RxIndication` will be called in this order.⌋*(SRS_GTW_06026, SRS_GTW_06029)*

[SWS_PduR_00911]{DRAFT} ⌈In case of forwarding to multiple upper layers, the module shall initiate a reception session for each configured upper layer destination.⌋*(SRS_GTW_06026, SRS_GTW_06029)*

[SWS_PduR_00790] ⌈When an error is returned by `<UpTp>_StartOfReception` for a multicast TP gatewaying with configured local destination, the PduR shall stop the upper layer reception without further interaction with the upper layer.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105)*

[SWS_PduR_00791] ⌈When `<UpTp>_StartOfReception` returns `BUFREQ_OK`, but the available buffer is too small to receive the whole message, the PduR shall call `<UpTp>_RxIndication` with `result = E_NOT_OK`.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105)*

[SWS_PduR_00792] ⌈When `<UpTp>_CopyRxData` returns an error, the PduR shall call `<UpTp>_RxIndication` with `result = E_NOT_OK`.⌋*(SRS_GTW_06026, SRS_GTW_06029, SRS_GTW_06105)*

7.1.4.4 Error handling

[SWS_PduR_00788] ⌈If `<DstLo/DstLoTp>_Transmit()`, called with an I-PDU from the FIFO buffer, returns `E_NOT_OK` the I-PDU shall be removed from the FIFO and the next FIFO entry shall be transmitted, if available.⌋*(SRS_GTW_06012, SRS_GTW_06105)*

[SWS_PduR_00807] ⌈When `<DstLo/DstLoTp>_Transmit()` returns `E_NOT_OK` for a routing path using a FIFO, the PDU Router shall report `PduR.PDUR_E_PDU_INSTANCES_LOST` to the DET module.⌋*(SRS_GTW_06012, SRS_GTW_06105)*

7.1.4.4.1 Communication interface

The PDU Router module shall not perform any error handling for an I-PDU instance if an interface module rejects a transmit request which belongs to a gateway operation.

[SWS_PduR_00256] ⌈The PDU Router module shall not retry transmission if the Communication Interface destination module returns `E_NOT_OK` after calling `<DstLo>_Transmit`.⌋*(SRS_GTW_06012, SRS_GTW_06105)*

Here the destination returned `E_NOT_OK` for some reason, will also report this error. The PDU Router module cannot do anything else than discarding the I-PDU.

[SWS_PduR_00255] ⌈If the FIFO is full and a new `PduR_<SrcLo>RxIndication` is called, the FIFO shall be flushed⌋*(SRS_GTW_06012, SRS_GTW_06032)*

Note: That means in case of `PduRQueueDepth` is configured to 1 and the `PduRDestPduDataProvision` is configured to `PDUR_TRIGGERTRANSMIT` the new I-Pdu will be always copied within the next `PduR_<SrcLo>TriggerTransmit` call. That is a "`last-is-best buffering`" behaviour.

[SWS_PduR_00669] ⌈If the FIFO is flushed the new I-PDU delivered by the `PduR_<SrcLo>RxIndication` shall be handled as if the FIFO is empty.⌋*(SRS_GTW_06012, SRS_GTW_06032)*

The new I-PDU that causes the FIFO flush will be served and not discarded.

[SWS_PduR_00670] ⌈If the FIFO is flushed the PDU Router shall report `PduR.PDUR_E_PDU_INSTANCES_LOST` to the DET module.⌋*(SRS_GTW_06012, SRS_GTW_06032, SRS_GTW_06106)*

[SWS_PduR_00806] ⌈In case of gatewaying between IFs, when one destination fails (Transmit returns `E_NOT_OK`), the other destinations shall continue.⌋*()*

7.1.4.4.2 Transport protocol

Error handling for I-PDUs gatewayed using Transport Protocol modules is simple: the PDU Router module will not do anything and rely on that the Transport Protocol modules handles the communication errors properly.

[SWS_PduR_00799] ⌈If no buffer could be allocated during the call of `PduR_<SrcLoTp>StartOfReception` for the reception of a gatewayed TP PDU, the PduR shall immediately stop further processing of this I-PDU and return `BUFREQ_E_OVFL`.⌋ *(SRS_GTW_06026, SRS_GTW_06124, SRS_GTW_06105)*

[SWS_PduR_00687] ⌈If `PduR_<SrcLoTp>CopyRxData` is called and the provided data cannot be stored in the buffer, then `BUFREQ_E_NOT_OK` shall be returned and the execution of the I-PDU gateway shall be stopped.⌋*(SRS_GTW_06026, SRS_GTW_06124, SRS_GTW_06105)*

[SWS_PduR_00689] ⌈If the `result` value is not `E_OK` in the `PduR_<SrcLoTp>RxIndication`, the PDU Router shall immediately stop further processing of the I-PDU.⌋ *(SRS_GTW_06026, SRS_GTW_06124, SRS_GTW_06105)*

Note: The PDU Router shall not expect a `PduR_<SrcLoTp>RxIndication` after `PduR_<SrcLo>StartOfReception` failed. The PDU Router shall not expect a `PduR_<DstLoTp>TxConfirmation` after `<LoTp>_Transmit` failed.

[SWS_PduR_00803] ⌈In case of gatewaying between TPs, when one destination fails (Transmit returns `E_NOT_OK` or `TpTxConfirmation` is called with an error), the other destinations shall continue.⌋*()*

[SWS_PduR_00804] ⌈In case of gatewaying between TPs, when all destinations fail, the reception side shall be stopped by returning `BUFREQ_E_NOT_OK` for the current call of `CopyRxData` or `StartOfReception`.⌋*()*

7.2 Cancel transmission

An upper layer module may request cancellation of an I-PDU (transported by Communication Interface module or Transport Protocol module). The PDU Router module will forward the request to either one destination module (singlecast) or multiple destination modules (multicast).

The `PduR_<Up>CancelTransmit` is used to cancel Communication Interface I-PDU and to cancel Transport Protocol I-PDUs in the case of forwarding.

The cancel transmission is optional and enabled in the configuration per module, see `PduRCancelTransmit` configuration parameter.

In case of forwarding, an upper layer module requests cancellation of an I-PDU, and the PDU Router will forward the request to one or more destination modules according to the routing path.

In case of gatewaying, PDU Router can optionally cancel the transmission when a connection is stopped based on e.g. after a negative return value or a negative `PduR_<SrcLoTp>RxIndication` from the source side of the routing path.

Note: This may happen e.g. during TP `on-the-fly gatewaying` or disabling a routing path group.

[SWS_PduR_00710] ⌈If the routing path for the requested I-PDU is disabled, then `PduR_<Up>CancelTransmit` shall return `E_NOT_OK` directly without any further action.⌋*(SRS_GTW_06122, SRS_GTW_06120)*

Following requirements describes the behavior in the PDU Router module when the `PduR_<Up>CancelTransmit` is called:

[SWS_PduR_00721] ⌈Communication Interface module

`PduR_<Up>CancelTransmit` and single destination: The PDU Router module shall call the `<Lo>_CancelTransmit` for the destination module of the I-PDU.⌋*(SRS_GTW_06122)*

[SWS_PduR_00723] ⌈Communication Interface module

`PduR_<Up>CancelTransmit` and multiple destinations: The PDU Router module shall call the `<Lo>_CancelTransmit` for each destination module of the I-PDU.⌋*(SRS_GTW_06122, SRS_GTW_06030)*

[SWS_PduR_00722] ⌈Transport Protocol module

`PduR_<Up>CancelTransmit` and single destination: The PDU Router module shall call the `<LoTp>_CancelTransmit` for the destination module of the I-PDU.⌋*(SRS_GTW_06122)*

[SWS_PduR_00724] ⌈Transport Protocol module

`PduR_<Up>CancelTransmit` and multiple destinations: The PDU Router module shall call the `<LoTp>_CancelTransmit` for each destination module of the I-PDU.⌋*(SRS_GTW_06122, SRS_GTW_06029)*

Following requirements describes the behavior in the PDU Router module when the return value of `<Lo>_CancelTransmit` or `<LoTp>_CancelTransmit` is received:

[SWS_PduR_00700] ⌈For a single destination, the PDU Router module shall return the same return value to the calling upper layer module.⌋*(SRS_GTW_06122, SRS_GTW_06104)*

[SWS_PduR_00701] ⌈For multiple destinations, `E_OK` shall be returned to the calling upper layer if all destination modules return `E_OK`. Otherwise, `E_NOT_OK` shall be returned.⌋*(SRS_GTW_06029, SRS_GTW_06030, SRS_GTW_06125, SRS_GTW_06122, SRS_GTW_06104)*

7.3 Cancel reception

An upper layer module may request cancellation of an I-PDU transported on Transport Protocol module(s). The PDU Router module will get a request through the `PduR_<Up>CancelReceive`. The confirmation of the cancellation request is made through the return value of `<LoTp>_CancelReceive` that is forwarded to the upper layer module as return value of `PduR_<Up>CancelReceive`.

[SWS_PduR_00726] ⌈If the routing path for the requested I-PDUs is disabled, then `PduR_<Up>CancelReceive` shall return `E_NOT_OK` directly without any further action.⌋*(SRS_GTW_06122, SRS_GTW_06120)*

The flow of the I-PDU id on the receiving side is from lower to upper layer modules. Here the flow is from upper to lower modules, since the id belongs to an already (partially) received I-PDU for which the reception shall be canceled.

[SWS_PduR_00736] ⌈The I-PDU id provided in the call is Rx I-PDU ID and therefore the PDU Router module shall be able to identify this I-PDU correctly.⌋*(SRS_GTW_06122)*

[SWS_PduR_00727] ⌈When the `PduR_<Up>CancelReceive` is called the PDU Router module shall call the `<LoTp>_CancelReceive` for the Transport Protocol destination module of the I-PDU.⌋*(SRS_GTW_06026, SRS_GTW_06122)*

[SWS_PduR_00732] ⌈The return value of the `<LoTp>_CancelReceive` shall be forwarded to the upper layer module.⌋*(SRS_GTW_06122, SRS_GTW_06105)*

In case of gatewaying, PDU Router can optionally cancel the reception when a connection is stopped, e.g. after a negative return value or a negative `PduR_<DstLoTp>TxConfirmation` from the destination of the routing path.

Note: This may happen e.g. during TP `on-the-fly gatewaying` or disabling a routing path group.

7.4 Zero Cost Operation

Zero cost operation is an optimization that may be done where source and destination modules are single and in source code (one of the modules must be in source code otherwise the PDU Router must create glue-code for the function call). For example an ECU with a Com module and a single CAN bus, the `PduR_ComTransmit` may directly call the `CanIf_Transmit` without any logic inside the PduR Module. The PDU Router becomes a macro layer.

This optimization is only possible where routing paths are of configuration class `Pre-Compile`.

[SWS_PduR_00287] ⌈If `PduRZeroCostOperation` is set to true and all routing paths are of configuration class `Pre-Compile`; modules directly above or below the

PDU Router may directly call each other without using PduR module functions.⌋*(SRS_-GTW_06020)*

[SWS_PduR_00619] ⌈If `PduRZeroCostOperation` is set to true and at least one routing path is not of configuration class `Pre-Compile`; the PDU Router module configuration generator shall report an error.⌋*(SRS_GTW_06020)*

7.5 State Management

The state machine of the PDU Router module is depicted in Figure 7.2.

[SWS_PduR_00644] ⌈Only one instance of the state machine shall exist in the PDU Router module.⌋*(SRS_BSW_00406)*

[SWS_PduR_00324] ⌈The PDU Router module shall consist of two states, `PDUR_-UNINIT` and `PDUR_ONLINE` as defined in `PduR_StateType`⌋*(SRS_BSW_00406)*

[SWS_PduR_00325] ⌈The PDU Router module shall be in the state `PDUR_UNINIT` after power up the PDU Router module (i.e. before calling the `PduR_Init` function).⌋*(SRS_BSW_00406)*

[SWS_PduR_00326] ⌈The PDU Router module shall change to the state `PDUR_ON-LINE` when the PDU Router has successfully been initialized via the function `PduR_-Init`⌋*(SRS_BSW_00406)*

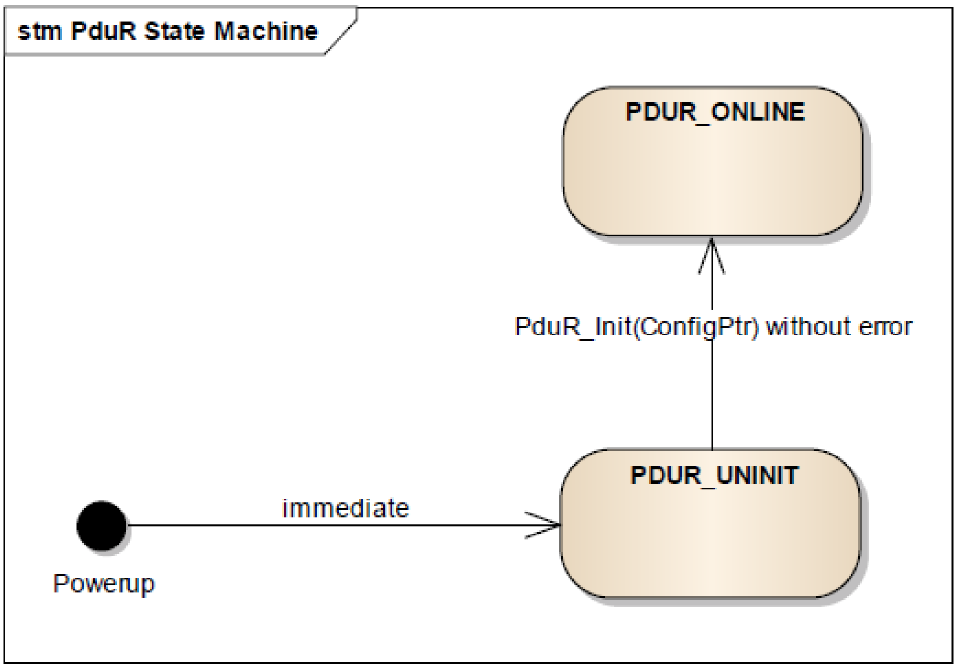

Figure 7.2: PDU Router states

[SWS_PduR_00328] ⌈The PDU Router module shall perform routing of PDUs according to the PDU Router routing tables only when it is in the online state PDUR_ONLINE⌋ *(SRS_BSW_00406, SRS_GTW_06001)*

[SWS_PduR_00330] ⌈The PDU Router module shall perform no routing when it is in the uninitialized state PDUR_UNINIT⌋*(SRS_BSW_00406, SRS_GTW_06001)*

[SWS_PduR_00645] ⌈The PDU Router module shall release all buffers in the PduR_Init function.⌋*(SRS_BSW_00406)*

[SWS_PduR_00308] ⌈The function PduR_Init shall initialize all configured default value to the PDU transmit buffers.⌋*(SRS_BSW_00406, SRS_GTW_06001)*

[SWS_PduR_00119] ⌈If the PDU Router module has not been initialized (state PDUR_UNINIT all functions except PduR_Init and PduR_GetVersionInfo shall report the error PduR.PDUR_E_UNINIT via the DET when called, when PduRDevErrorDetect is enabled.⌋*(SRS_BSW_00406)*

7.6 Routing path groups

A routing path group is a group of routing paths that can be disabled and enabled during runtime. This allows for enabling/disabling a set of routing paths related to a certain network or a certain kind of PDUs.

Enabling and disabling of routing path groups is typically used by the BswM module.

7.6.1 PduRRoutingPathGroup definitions

For a PduRRoutingPathGroup the following rules apply:

1. A PduRRoutingPath can belong to any PduRRoutingPathGroup.
2. A PduRRoutingPath shall be enabled, if it belong to a enabled PduRRoutingPathGroup (see [SWS_PduR_00891]). If a PduRRoutingPath, does not belong to any PduRRoutingPath, the PduRRoutingPath is always enabled.

[SWS_PduR_00891] ⌈A PduRRoutingPath shall be enabled under either of the following conditions:

- if at least one PduRRoutingPathGroup is enabled it refers to
- if the PduRRoutingPath does not reference any PduRRoutingPathGroup at all

⌋*(SRS_GTW_06120)*

7.6.2 Initialization of PduRRoutingPathGroups

[SWS_PduR_00892] ⌈A PduRRoutingPath shall be disabled, if all PduRRoutingPathGroups it references are disabled.⌋*(SRS_GTW_06120)*

[SWS_PduR_00894] ⌈All PduRRoutingPathGroups where PduRIsEnabledAtInit is set to TRUE shall be enabled after initialization of the PDU Router module.⌋*(SRS_GTW_06120)*

[SWS_PduR_00895] ⌈All PduRRoutingPathGroups where PduRIsEnabledAtInit is set to FALSE shall be disabled.⌋*(SRS_GTW_06120)*

7.6.3 Switching of PduRRoutingPathGroups

[SWS_PduR_00897] ⌈If a PduRRoutingPathGroup is disabled by PduR_DisableRouting, the PDU Router module shall re-evaluate all PduRRoutingPaths which are referencing the affected PduRRoutingPathGroup according to [SWS_PduR_00892].⌋*(SRS_GTW_06120)*

[SWS_PduR_00898] ⌈If a PduRRoutingPathGroup is enabled by PduR_EnableRouting, the PDU Router module shall re-evaluate all PduRRoutingPaths which are referencing the affected PduRRoutingPathGroup according to [SWS_PduR_00891].⌋*(SRS_GTW_06120)*

[SWS_PduR_00717] ⌈If the I-PDU ID associated with a disabled routing path is used in any API, then E_NOT_OK (if possible) shall be returned with no further action.⌋*(SRS_GTW_06120, SRS_GTW_06029, SRS_GTW_06030)*

[SWS_PduR_00715] ⌈Enabling of I-PDU routing path groups shall be immediate⌋*(SRS_GTW_06120)*

Example: A subsequent call to PduR_<Up>Transmit shall serve this I-PDU directly.

[SWS_PduR_00805] ⌈If a routing path group is disabled (by the call PduR_DisableRouting) the PduR shall directly return for following functions for this routing path group:

- PduR_<User:Up>Transmit,
- PduR_<User:Lo>RxIndication,
- PduR_<User:Lo>TriggerTransmit,
- PduR_<User:LoTp>StartOfReception,
- PduR_<User:LoTp>CopyRxData,
- PduR_<User:LoTp>CopyTxData.

If the function has a Std_ReturnType, it shall return E_NOT_OK. If function has a BufReq_ReturnType, it shall return BUFREQ_E_NOT_OK.⌋*()*

Note: This does not affect PduR_<User:LoTp>RxIndication

[SWS_PduR_00810] ⌈When a routing path associated with a single buffer PduRQueueDepth == 1) is stopped, the according buffer shall be set to the default value if PduR_DisableRouting is called with initialize set to true, otherwise the current value shall be retained.⌋*(SRS_GTW_06120, SRS_GTW_06032, SRS_GTW_-06124)*

[SWS_PduR_00663] ⌈When a routing path associated with a FIFO PduRQueueDepth > 1) is stopped, the according FIFO shall be flushed, and the PduR shall report PduR.-PDUR_E_PDU_INSTANCES_LOST to the DET if DET reporting is enabled.⌋*(SRS_-GTW_06120, SRS_GTW_06032, SRS_GTW_06124)*

Example: If a gateway operation is made and the PDU Router module has buffered an I-PDU and is waiting for the destination communication module to call trigger transmit, the buffer is cleared and the buffer not available is returned to the destination Communication Interface.

7.7 Complex Driver Interaction

Besides the AUTOSAR Com and Dcm modules, Complex Drivers (CDD) are also possible as upper or lower layer modules for the PduR.

Whether a CDD is an upper layer or a lower layer module for the PduR is configurable via the PduRUpperModule or PduRLowerModule configuration parameters of the PduRBswModules configuration.

A CDD can require both a Communication Interface API or a Transport Protocol API, depending on the configuration parameters PduRCommunicationInterface and PduRTransportProtocol of the PduRBswModules configuration. The API functions provided by the PduR for the CDD interaction contain the CDD's service prefix as specified by the apiServicePrefix configuration parameter, see [SWS_PduR_00504].

The PduR provides the unique transmit function PduR_<Cdd>Transmit for each upper layer CDD. When a callout function of the PduR is invoked from a lower layer module for a PDU that is transmitted or received by an upper layer CDD, the PduR invokes the corresponding target function of the CDD.

For a lower layer CDD that requires a Communication Interface API, the PduR provides a unique set of Communication Interface API functions PduR_<Cdd>RxIndication and - if configured - PduR_<Cdd>TxConfirmation and PduR_<Cdd>TriggerTransmit, see Section 8.3.3.

For a lower layer CDD that requires a Transport Protocol API, the PduR provides a unique set of Transport Protocol API functions PduR_<Cdd>CopyRxData, PduR_<Cdd>CopyTxData, PduR_<Cdd>RxIndication, PduR_<Cdd>StartOfReception and PduR_<Cdd>TxConfirmation, see Section 8.3.4.

When an API function of the PduR is invoked from an upper layer module for a PDU that is transmitted or received by a lower layer CDD, the PduR invokes the corresponding target function of the CDD.

To determine if a PDU is transmitted or received by a CDD, the PduR has to examine the origin of the references to the PDU list in the EcuC module:

- If the source PDU of a routing path references a PDU in the PDU list that is also referenced by an upper layer CDD, the PDU is transmitted by the CDD.
- If the destination PDU of a routing path references a PDU in the PDU list that is also referenced by an upper layer CDD, the PDU is received by the CDD.
- If the source PDU of a routing path references a PDU in the PDU list that is also referenced by a lower layer CDD, the PDU is received from the CDD.
- If the destination PDU of a routing path references a PDU in the PDU list that is also referenced by a lower layer CDD, the PDU is transmitted via the CDD.

[SWS_PduR_00504] ⌈The PduR shall use the `apiServicePrefix` attribute of the CDD's vendor specific module definition (`EcucModuleDef` element) to replace the `<Lo>`, `<Up>`, and `<LoTp>` tags of the `GenericComServices APIs`. The CDD's vendor specific module definition can be indirectly accessed via the configuration parameter `PduRBswModuleRef` which references the top-level element of the concrete configuration of the CDD (i.e., `EcucModuleConfigurationValues` element) which references the CDD's vendor specific module definition (`EcucModuleDef element`).⌋ *(SRS_BSW_00310)*

7.8 Error classification

For details refer to the Chapter 7.2 "Error classification" in [2, SWS BSW General].

Note that the PduR does not report production errors.

7.8.1 Development Errors

[SWS_PduR_00100] ⌈

Type of error	*Related error code*	*Error value*
Invalid configuration pointer	PDUR_E_INIT_FAILED	0x00
API service (except PduR_GetVersionInfo) used without module initialization or PduR_Init called in any state other than PDUR_UNINIT	PDUR_E_UNINIT	0x01
Invalid PDU identifier	PDUR_E_PDU_ID_INVALID	0x02
If the routing table is invalid that is given to the PduR_EnableRouting or PduR_DisableRouting functions	PDUR_E_ROUTING_PATH_GROUP_ID_INVALID	0x08

▽

△

Type of error	*Related error code*	*Error value*
Null pointer has been passed as an argument	PDUR_E_PARAM_POINTER	0x09

⌋*(SRS_BSW_00337, SRS_BSW_00323, SRS_BSW_00452)*

7.8.2 Runtime Errors

[SWS_PDUR_00816] ⌈

Type of error	*Related error code*	*Error value*
TP module rejects a transmit request for a valid PDU identifier	PDUR_E_TP_TX_REQ_REJECTED	0x03
Loss of a PDU instance (buffer overrun in gateway operation)	PDUR_E_PDU_INSTANCES_LOST	0x0a

⌋*()*

7.8.3 Transient Faults

There are no transient faults.

7.8.4 Production Errors

There are no production errors.

7.8.5 Extended Production Errors

There are no extended production errors.

7.9 API parameter checking

[SWS_PduR_00221] ⌈If development error detection is enabled, a PDU identifier is not within the specified range, and the PDU identifier is configured to be used by the PDU Router module, the PDU Router module shall report the error `PduR.PDUR_E_PDU_ID_INVALID` to the DET module, when `PduRDevErrorDetect` is enabled.⌋ *(SRS_GTW_06103, SRS_BSW_00323)*

7.10 Multicore Distribution

The PduR module is distributed over all partitions and acts as a central inter-partition dispatcher for any inter-partition (inter-core) routing path(s), mainly for gateway and multicast use cases.

For further explanations about the distribution principles, please see chapter "Com-Stack Distribution" within the [8, Guide to BSW Distribution].

[SWS_PduR_00836] ⌈The PDU Router module configuration generator shall examine the partition assignment of each source/destination I-PDU of all routing path instances.⌋*()*

[SWS_PduR_00837] ⌈The PDU Router module configuration generator shall take over the dedicated partition reference if an `EcucPduDedicatedPartition` container is available for the respective module, referred by `EcucPduDedicatedPartitionBswModuleRef`.⌋*()*

[SWS_PduR_00838] ⌈In case no module individual dedicated partition reference (`EcucPduDedicatedPartition`) is available for the respective module, the PDU Router module configuration generator shall take over the default partition reference of the I-PDU.⌋*()*

[SWS_PduR_00839] ⌈The PDU Router module configuration generator shall process a routing path as intra-partition communication, if both connected I-PDUs (source and target) are assigned to the same partition.⌋*()*

[SWS_PduR_00840] ⌈The PDU Router module configuration generator shall process a routing path as inter-partition communication, if the connected source and target I-PDU are assigned to different partitions.⌋*()*

[SWS_PduR_00841] ⌈In configurations, in which upper and/or lower layers reside in different partitions, PduR may provide one init function per ECUC partition.⌋*(SRS_BSW_00459, SRS_BSW_00460)*

[SWS_PduR_CONSTR_00861] ⌈The PDU Router shall only accept PduRRoutingPath Groups, which contain `PduRRoutingPaths` having all destination I-PDUs assigned to the same partition.⌋*()*

7.10.1 Intra-partition Routing Path

The intra-partition communication behavior should not be altered, even though there are upper and/or lower layers, which reside in different partitions.

[SWS_PduR_CONSTR_00842] ⌈For routing paths with TriggerTransmit Transmission from an upper layer module the PDU Router shall accept intra-partition routings only.⌋
()

7.10.2 Inter-partition Routing Path

This chapter describes, how the PduR shall handle inter-partition routing paths. This means a destination PDU is mapped to a different ECUC partition than the corresponding source Pdu.

PduR shall perform the actual cross-partition communication with respect to data and call context.

Note: The [8, Guide to BSW Distribution] describes the ways how to solve a context switch in a multicore environment within its chapter "BSW Distribution in Multi-Core Systems".

7.10.2.1 Upper layer module interaction

[SWS_PduR_00843] ⌈For inter-partition routing paths, the return value of the function `PduR_<User:Up>Transmit` shall reflect if the PduR itself has accepted the transmit request or not.⌋*()*

[SWS_PduR_00844] ⌈For inter-partition routing paths, the call to the function `PduR_<User:Up>Transmit` shall only execute code that is assigned to the same ECUC partition. PduR shall not call the `<Provider:Lo>_Transmit` in the call context of `PduR_<User:Up>Transmit`. PduR shall call `<Provider:Lo>_Transmit` in the ECUC partition context of the `<Provider:Lo>` module (in fact of the ECUC partition defined via the `EcucPduDefaultPartitionRef` or the `EcucPduDedicatedPartitionRef` of the `PduRDestPduRef`).⌋*()*

[SWS_PduR_00845] ⌈For inter-partition routing paths, the return value of the function `PduR_<User:Up>CancelTransmit` shall reflect if the PduR itself has accepted the transmit cancellation or not.⌋*()*

[SWS_PduR_00846] ⌈For inter-partition routing paths, the call to the function `PduR_<User:Up/UpTp>CancelTransmit` shall only execute code that is assigned to the same ECUC partition. PduR shall not call the `<Provider:Lo/LoTp>_Cancel Transmit` in the call context of `PduR_<User:Up/UpTp>CancelTransmit`. PduR shall call `<Provider:Lo/LoTp>_CancelTransmit` in the ECUC partition context of the `<Provider:Lo/LoTp>` module (in fact of the ECUC partition context defined via the `EcucPduDefaultPartitionRef` or the `EcucPduDedicatedPartitionRef` of the `PduRDestPduRef`).⌋*()*

[SWS_PduR_00847] ⌈For inter-partition routing paths, the return value of the function `PduR_<User:Up>CancelReceive` shall reflect if the PduR itself has accepted the reception cancellation or not.⌋*()*

[SWS_PduR_00848] ⌈For inter-partition routing paths, the call to the function `PduR_<User:UpTp>CancelReceive` shall only execute code that is assigned to the same ECUC partition. PduR shall not call the `<Provider:LoTp>_CancelReceive` in the call context of `PduR_<User:UpTp>CancelReceive`. PduR shall call`<Provider:Lo`

Tp>_CancelReceive in the ECUC partition context of the <Provider:LoTp> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRSrcPduRef).⌋*()*

7.10.2.2 Lower layer Communication Interface module interaction

[SWS_PduR_00849] ⌈For inter-partition routing paths, the call to the function PduR_<User:Lo>RxIndication shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:Up>_RxIndication in the call context of PduR_<User:Lo>RxIndication. PduR shall call <Provider:Up>_RxIndication in the ECUC partition context of the <Provider:Up> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRSrcPduRef).⌋*()*

[SWS_PduR_00850] ⌈For inter-partition routing paths, the call to the function PduR_<User:Lo>TxConfirmation shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:Up>_TxConfirmation in the call context of PduR<User:Lo>TxConfirmation. PduR shall call <Provider:Up>_TxConfirmation in the ECUC partition context of the <Provider:Up> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRSrcPduRef).⌋*()*

7.10.2.3 Lower layer Transport Protocol module interaction

[SWS_PduR_00851] ⌈For inter-partition routing paths, the call to the function PduR_<User:LoTp>CopyRxData shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:UpTp>_CopyRxData in the call context of PduR<User:LoTp>CopyRxData. PduR shall call <Provider:UpTp>_CopyRxData in the ECUC partition context of the <Provider:UpTp> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRDestPduRef).⌋*()*

[SWS_PduR_00852] ⌈For inter-partition routing paths, the return value of the function PduR_<User:LoTp>CopyRxData shall reflect the status of the PduR but not of Provider:UpTp.⌋*()*

[SWS_PduR_00853] ⌈For inter-partition routing paths, the call to the function PduR_<User:LoTp>RxIndication shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:UpTp>_RxIndication in the call context of PduR<User:LoTp>RxIndication. PduR shall call <Provider:UpTp>_RxIndication in the ECUC partition context of the <Provider:UpTp> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRDestPduRef).⌋*()*

[SWS_PduR_00854] ⌈For inter-partition routing paths, the call to the function PduR_-<User:LoTp>StartOfReception shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:UpTp>_StartOfReception in the call context of PduR_<User:LoTp>StartOfReception. PduR shall call <Provider:UpTp>_StartOfReception in the ECUC partition context of the <Provider:UpTp> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRDestPduRef).⌋*()*

[SWS_PduR_00855] ⌈For inter-partition routing paths, the return value of the function PduR_<User:LoTp>StartOfReception shall reflect the status of the PduR but not of Provider:UpTp.⌋*()*

[SWS_PduR_00856] ⌈For inter-partition routing paths, the call to the function PduR_-<User:LoTp>CopyTxData shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:UpTp>_CopyTxData in the call context of PduR_<User:LoTp>CopyTxData. PduR shall call <Provider:UpTp>_CopyTxData in the ECUC partition context of the <Provider:UpTp> module (in fact of the ECUC partition context defined via the EcucPduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRSrcPduRef).⌋*()*

[SWS_PduR_00857] ⌈For inter-partition routing paths, the return value of the function PduR_<User:LoTp>CopyTxData shall reflect the status of the PduR but not of Provider:UpTp.⌋*()*

[SWS_PduR_00858] ⌈For inter-partition routing paths, the call to the function PduR_-<User:LoTp>TxConfirmation shall only execute code that is assigned to the same ECUC partition. PduR shall not call the <Provider:UpTp>_TxConfirmation in the call context of PduR_<User:LoTp>TxConfirmation. PduR shall call <Provider:UpTp>_TxConfirmation in the ECUC partition context of the <Provider:UpTp> module (in fact of the ECUC partition context defined via the Ecuc PduDefaultPartitionRef or the EcucPduDedicatedPartitionRef of the PduRSrcPduRef).⌋*()*

[SWS_PduR_00859] ⌈For inter-partition Transport Protocol module interactions, PduR shall be the relay between <User:Lo/UpTp> and <Provider:Lo/UpTp> routing paths and decouple the call context by intermediate buffering.⌋*()*

Note: The reception behavior on the source bus depends on the PduR's configuration (PduRTpThreshold/PduRSourcePduBlockSize).

[SWS_PduR_00860] ⌈PduR shall treat inter-partition Transport Protocol module interactions as gateway routings.⌋*()*

Note: Configurations, in which upper and/or lower Transport Protocol layers reside in different partitions will have an effect on the timing (latency) and potentially the behavior on the bus (late abort of Rx transmission in case the <User:Lo/UpTp> was not ready to receive). If this cannot be tolerated, all participating modules have to be mapped to the same partition.

7.10.2.4 Communication Interface Gatewaying

[SWS_PduR_00881] ⌈For inter-partition routing paths, the call to the following functions shall only execute code that is assigned to the same ECUC partition:

- `PduR_<SrcLo>RxIndication`,
- `PduR_<DstLo>TriggerTransmit`,
- `PduR_<DstLo>TxConfirmation`.

PduR shall not call the `<DstLo>_Transmit` in the call context of `PduR_<SrcLo>RxIndication`. PduR shall call `<DstLo>_Transmit` in the ECUC partition context of the `<DstLo>` module (in fact of the ECUC partition context defined via the `EcucPduDefaultPartitionRef` or the `EcucPduDedicatedPartitionRef` of the `PduR-DestPduRef`).⌋*()*

7.10.2.5 Transport Protocol Gatewaying

[SWS_PduR_00882] ⌈For inter-partition routing paths, the call to the following functions shall only execute code that is assigned to the same ECUC partition:

- `PduR_<SrcLoTp>StartOfReception`,
- `PduR_<SrcLoTp>CopyRxData`,
- `PduR_<DstLoTp>CopyTxData`,
- `PduR_<SrcLoTp>RxIndication`,
- `PduR_<DstLoTp>TxConfirmation`.

PduR shall not call the `<DstLoTp>_Transmit` in the call context of `PduR_<SrcLoTp>CopyRxData`, `PduR_<DstLoTp>TxConfirmation`. PduR shall call `<DstLoTp>_Transmit` in the ECUC partition context of the `<DstLoTp>` module (in fact of the ECUC partition context defined via the `EcucPduDefaultPartitionRef` or the `EcucPduDedicatedPartitionRef` of the `PduRDestPduRef`).⌋*()*

[SWS_PduR_00883] ⌈For inter-partition routing paths, PduR shall allow concurrent invocations of the following API-pairs from different ECUC partitions

- `PduR_<SrcLoTp>CopyRxData` and `PduR_<DstLoTp>CopyTxData`,
- `PduR_<SrcLoTp>RxIndication` and `PduR_<DstLoTp>CopyTxData`,
- `PduR_<SrcLoTp>CopyRxData` and `PduR_<DstLoTp>TxConfirmation`,
- `PduR_<SrcLoTp>RxIndication` and `PduR_<DstLoTp>TxConfirmation`.

⌋*()*

8 API specification

The following paragraphs specify the API of the PDU Router module.

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_PduR_00333] ⌈

Module	***Header File***	***Imported Type***
ComStack_Types	ComStack_Types.h	BufReq_ReturnType
	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
	ComStack_Types.h	RetryInfoType
	ComStack_Types.h	TpDataStateType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋*(SRS_BSW_00301)*

8.2 Type definitions

8.2.1 PduR_PBConfigType

The post-build-time configuration fulfills two functionalities:

- Post-build selectable, where more than one configuration is located in the ECU, and one is selected at init of the PDU Router module
- Post-build loadable, where one configuration is located in the ECU. This configuration may be reprogrammed after compile-time

Basically there is no restriction to mix both selectable and loadable. Typically the post-build loadable is located in its own flash sector where it can be reprogrammed without affecting other modules/applications.

[SWS_PduR_00743] ⌈

Name	PduR_PBConfigType
Kind	Structure
Elements	–

▽

△

	Type	–
	Comment	implementation specific
Description	Data structure containing post-build-time configuration data of the PDU Router.	
Available via	PduR.h	

⌋*(SRS_BSW_00400, SRS_BSW_00438, SRS_BSW_00404, SRS_BSW_00305)*

[SWS_PduR_00241] ⌈The type `PduR_PBConfigType` is an external data structure containing post-build-time configuration data of the PDU Router module which shall be implemented in PduR_PBcfg.c.⌋*(SRS_BSW_00438, SRS_BSW_00404)* (see Chapter subsection 5.1.1 and section 10.2).

8.2.2 PduR_PBConfigIdType

This type is returned by the `PduR_GetConfigurationId` API.

[SWS_PduR_00771] ⌈

Name	PduR_PBConfigIdType
Kind	Type
Derived from	uint16
Description	Identification of the post-build configuration currently used for routing I-PDUs. An ECU may contain several configurations (post-build selectable), each have unique Id.
Available via	PduR.h

⌋*(SRS_BSW_00405, SRS_BSW_00305, SRS_GTW_06097)*

8.2.3 PduR_RoutingPathGroupIdType

The routing path group ID is used for identifying a specific group of routing paths. Since a `PduRRoutingPath` links one source I-PDU and one destination I-PDU, it is possible to enable/disable routing per network or per PDU kind.

[SWS_PduR_00654] ⌈

Name	PduR_RoutingPathGroupIdType
Kind	Type
Derived from	uint16
Description	Identification of a Routing Table
Available via	PduR.h

⌋*(SRS_BSW_00305, SRS_GTW_06120)*

8.2.4 PduR_StateType

[SWS_PduR_00742] ⌈

Name	PduR_StateType		
Kind	Enumeration		
Range	PDUR_UNINIT	–	PDU Router not initialised
	PDUR_ONLINE	–	PDU Router initialized successfully
Description	States of the PDU Router		
Available via	PduR.h		

⌋*(SRS_BSW_00305, SRS_BSW_00335, SRS_BSW_00406)*

8.3 Function definitions

8.3.1 General functions provided by the PDU Router

8.3.1.1 PduR_Init

[SWS_PduR_00334] ⌈

Service Name	PduR_Init	

Syntax	`void PduR_Init (`
	`  const PduR_PBConfigType* ConfigPtr`
	`)`	

Service ID [hex]	0xf0	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to post build configuration
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Initializes the PDU Router	
Available via	PduR.h	

⌋*(SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00310)*

Integration note: To avoid problems calling the PDU Router module uninitialized it is important that the PDU Router module is initialized before interfaced modules.

[SWS_PduR_00709] ⌈After initialization all I-PDU routing groups shall be enabled according enable at start configuration parameter.⌋*(SRS_GTW_06120)*

Note: NULL pointer checking is specified within document [2, SWS BSW General].

8.3.1.2 PduR_GetVersionInfo

[SWS_PduR_00338] ⌈

Service Name	PduR_GetVersionInfo	
Syntax	`void PduR_GetVersionInfo ( Std_VersionInfoType* versionInfo )`	
Service ID [hex]	0xf1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versionInfo	Pointer to where to store the version information of this module.
Return value	None	
Description	Returns the version information of this module.	
Available via	PduR.h	

⌋*(SRS_BSW_00407, SRS_BSW_00411, SRS_BSW_00310)*

8.3.1.3 PduR_GetConfigurationId

[SWS_PduR_00341] ⌈

Service Name	PduR_GetConfigurationId	
Syntax	`PduR_PBConfigIdType PduR_GetConfigurationId ( void )`	
Service ID [hex]	0xf2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	None	
Return value	PduR_PBConfigIdType	Identifier of the post-build time configuration
Description	Returns the unique identifier of the post-build time configuration of the PDU Router	
Available via	PduR.h	

⌋*(SRS_GTW_06097, SRS_BSW_00310)*

[SWS_PduR_00280] ⌈The function `PduR_GetConfigurationId` shall return the unique identifier of the post-build time configuration of the PDU Router module.⌋*(SRS_-GTW_06097)*

8.3.1.4 PduR_EnableRouting

[SWS_PduR_00615] ⌈

Service Name	PduR_EnableRouting	
Syntax	`void PduR_EnableRouting ( PduR_RoutingPathGroupIdType id )`	
Service ID [hex]	0xf3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the routing path group. Routing path groups are defined in the PDU router configuration.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Enables a routing path group.	
Available via	PduR.h	

⌋*(SRS_GTW_06120, SRS_BSW_00310)*

[SWS_PduR_00647] ⌈If the routing path group id does not exist, then the PDU Router module shall return with no action.⌋*(SRS_GTW_06120, SRS_BSW_00323)*

[SWS_PduR_00648] ⌈If the routing path group id does not exist and the `PduRDevErrorDetect` is enabled, the PDU Router module shall report `PduR.PDUR_E_ROUTING_PATH_GROUP_ID_INVALID`.⌋*(SRS_GTW_06120, SRS_BSW_00323)*

[SWS_PduR_00899] ⌈If the routing path group exists and the routing path group is already enabled, then the PDU Router module shall return with no action.⌋*(SRS_GTW_06120)*

8.3.1.5 PduR_DisableRouting

[SWS_PduR_00617] ⌈

Service Name	PduR_DisableRouting	
Syntax	`void PduR_DisableRouting ( PduR_RoutingPathGroupIdType id, boolean initialize )`	
Service ID [hex]	0xf4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the routing path group. Routing path groups are defined in the PDU router configuration.

▽

△

	initialize	true: initialize single buffers to the default value false: retain current value of single buffers
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Disables a routing path group.	
Available via	PduR.h	

⌋*(SRS_GTW_06120, SRS_BSW_00310)*

[SWS_PduR_00716] ⌈If the routing path group id does not exist, then the PDU Router module shall return with no action.⌋*(SRS_GTW_06120, SRS_BSW_00323)*

[SWS_PduR_00649] ⌈If the routing path table id does not exist and the `PduRDevErrorDetect` is enabled, the PDU Router module shall report `PduR.PDUR_E_ROUTING_PATH_GROUP_ID_INVALID`.⌋*(SRS_GTW_06120, SRS_BSW_00323)*

[SWS_PduR_00896] ⌈If the routing path group exists and the routing path group is already disabled, then the PDU Router module shall return with no action.⌋*(SRS_GTW_06120)*

8.3.2 Configurable interfaces definitions for interaction with upper layer module

Since the API description now has a generic approach, the `serviceId`s of the upper layer API functions are generic as well. To differ between several upper layers, the PduR uses the `moduleId`s of the upper layer modules as the `instanceId` argument in the Det call.

8.3.2.1 PduR_<User:Up>Transmit

[SWS_PduR_00406] ⌈

Service Name	PduR_<User:Up>Transmit	
Syntax	`Std_ReturnType PduR_<User:Up>Transmit ( PduIdType TxPduId, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x49	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	Identifier of the PDU to be transmitted
	PduInfoPtr	Length of and pointer to the PDU data and pointer to MetaData.

▽

△

Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Transmit request has been accepted. E_NOT_OK: Transmit request has not been accepted.
Description	Requests transmission of a PDU.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06012, SRS_GTW_06026, SRS_GTW_06114, SRS_GTW_06115, SRS_GTW_06116, SRS_BSW_00310)*

8.3.2.2 PduR_<User:Up>CancelTransmit

[SWS_PduR_00769] ⌈

Service Name	PduR_<User:Up>CancelTransmit	

Syntax	`Std_ReturnType PduR_<User:Up>CancelTransmit (`
	`  PduIdType TxPduId`
	`)`	

Service ID [hex]	0x4a	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	Identification of the PDU to be cancelled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06122, SRS_GTW_06114, SRS_GTW_06115, SRS_GTW_06116, SRS_BSW_00310)*

8.3.2.3 PduR_<User:Up>CancelReceive

[SWS_PduR_00767] ⌈

Service Name	PduR_<User:Up>CancelReceive

Syntax	`Std_ReturnType PduR_<User:Up>CancelReceive (`
	`  PduIdType RxPduId`
	`)`

▽

△

Service ID [hex]	0x4c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	RxPduId	Identification of the PDU to be cancelled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description	Requests cancellation of an ongoing reception of a PDU in a lower layer transport protocol module.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06026, SRS_GTW_06114, SRS_GTW_06115, SRS_GTW_06116, SRS_BSW_00310)*

8.3.3 Configurable interfaces definitions for lower layer communication interface module interaction

Since the API description now has a generic approach, the `serviceId`s of the lower layer API functions are generic as well. To differ between several lower layers, the PduR uses the `moduleId`s of the lower layer modules as the `instanceId` argument in the Det call.

8.3.3.1 PduR_<User:Lo>RxIndication

[SWS_PduR_00362] ⌈

Service Name	PduR_<User:Lo>RxIndication	

Syntax	`void PduR_<User:Lo>RxIndication (`
	`  PduIdType RxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID [hex]	0x42	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	RxPduId	ID of the received PDU.
	PduInfoPtr	Contains the length (SduLength) of the received PDU, a pointer to a buffer (SduDataPtr) containing the PDU, and the MetaData related to this PDU.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	Indication of a received PDU from a lower layer communication interface module.
Available via	PduR_<module>.h

⌋(*SRS_GTW_06012*, *SRS_GTW_06116*, *SRS_GTW_06117*, *SRS_GTW_06123*, *SRS_BSW_00310*)

8.3.3.2 PduR_<User:Lo>TxConfirmation

[SWS_PduR_00365] ⌈

Service Name	PduR_<User:Lo>TxConfirmation	
Syntax	`void PduR_<User:Lo>TxConfirmation ( PduIdType TxPduId, Std_ReturnType result )`	
Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	ID of the PDU that has been transmitted.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.	
Available via	PduR_<module>.h	

⌋(*SRS_GTW_06012*, *SRS_GTW_06116*, *SRS_GTW_06117*, *SRS_GTW_06123*, *SRS_BSW_00310*)

8.3.3.3 PduR_<User:Lo>TriggerTransmit

[SWS_PduR_00369] ⌈

Service Name	PduR_<User:Lo>TriggerTransmit
Syntax	`Std_ReturnType PduR_<User:Lo>TriggerTransmit ( PduIdType TxPduId, PduInfoType* PduInfoPtr )`
Service ID [hex]	0x41
Sync/Async	Synchronous
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.

▽

△

Parameters (in)	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout)	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: SDU has been copied and SduLength indicates the number of copied bytes. E_NOT_OK: No SDU data has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06012, SRS_GTW_06032, SRS_GTW_06116, SRS_GTW_06117, SRS_GTW_06123, SRS_BSW_00310)*

8.3.4 Configurable interfaces definitions for lower layer transport protocol module interaction

Since the API description now has a generic approach, the `serviceId`s of the lower layer transport protocol API functions are generic as well. To differ between several lower layers, the PduR uses the `moduleId`s of the lower layer modules as the `instanceId` argument in the Det call.

8.3.4.1 PduR_<User:LoTp>CopyRxData

[SWS_PduR_00512] ⌈

Service Name	PduR_<User:LoTp>CopyRxData	

Syntax	`BufReq_ReturnType PduR_<User:LoTp>CopyRxData (`
	`  PduIdType id,`
	`  const PduInfoType* info,`
	`  PduLengthType* bufferSizePtr`
	`)`	

Service ID [hex]	0x44	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the received I-PDU.
	info	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available buffer in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.

▽

△

Parameters (inout)	None	
Parameters (out)	bufferSizePtr	Available receive buffer after data has been copied.
Return value	BufReq_ReturnType	BUFREQ_OK: Data copied successfully BUFREQ_E_NOT_OK: Data was not copied because an error occurred.
Description	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining buffer is written to the position indicated by bufferSizePtr.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06026, SRS_GTW_06121, SRS_BSW_00310)*

8.3.4.2 PduR_<User:LoTp>RxIndication

[SWS_PduR_00375] ⌈

Service Name	PduR_<User:LoTp>RxIndication	

Syntax	`void PduR_<User:LoTp>RxIndication (`
	`  PduIdType id,`
	`  Std_ReturnType result`
	`)`	

Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the received I-PDU.
	result	E_OK: The PDU was received. E_NOT_OK: Reception of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06026, SRS_GTW_06121, SRS_BSW_00310)*

8.3.4.3 PduR_<User:LoTp>StartOfReception

[SWS_PduR_00507] ⌈

Service Name	PduR_<User:LoTp>StartOfReception

▽

△

Syntax	```BufReq_ReturnType PduR_<User:LoTp>StartOfReception (PduIdType id, const PduInfoType* info, PduLengthType TpSduLength, PduLengthType* bufferSizePtr)```	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the I-PDU.
	info	Pointer to a PduInfoType structure containing the payload data (without protocol information) and payload length of the first frame or single frame of a transport protocol I-PDU reception, and the MetaData related to this PDU. If neither first/single frame data nor MetaData are available, this parameter is set to NULL_PTR.
	TpSduLength	Total length of the N-SDU to be received.
Parameters (inout)	None	
Parameters (out)	bufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute the Block Size (BS) in the transport protocol module.
Return value	BufReq_ReturnType	BUFREQ_OK: Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr. BUFREQ_E_NOT_OK: Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged. BUFREQ_E_OVFL: No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Description	This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSdu Length equal to 0.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06026, SRS_GTW_06121, SRS_BSW_00310)*

8.3.4.4 PduR_<User:LoTp>CopyTxData

[SWS_PduR_00518] ⌈

Service Name	PduR_<User:LoTp>CopyTxData
Syntax	```BufReq_ReturnType PduR_<User:LoTp>CopyTxData (PduIdType id, const PduInfoType* info, const RetryInfoType* retry, PduLengthType* availableDataPtr)```
Service ID [hex]	0x43
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	id	Identification of the transmitted I-PDU.
	info	Provides the destination buffer (SduDataPtr) and the number of bytes to be copied (SduLength). If not enough transmit data is available, no data is copied by the upper layer module and BUFREQ_E_BUSY is returned. The lower layer module may retry the call. An SduLength of 0 can be used to indicate state changes in the retry parameter or to query the current amount of available data in the upper layer module. In this case, the Sdu DataPtr may be a NULL_PTR.
	retry	This parameter is used to acknowledge transmitted data or to retransmit data after transmission problems. If the retry parameter is a NULL_PTR, it indicates that the transmit data can be removed from the buffer immediately after it has been copied. Otherwise, the retry parameter must point to a valid RetryInfoType element. If TpDataState indicates TP_CONFPENDING, the previously copied data must remain in the TP buffer to be available for error recovery. TP_DATACONF indicates that all data that has been copied before this call is confirmed and can be removed from the TP buffer. Data copied by this API call is excluded and will be confirmed later. TP_DATARETRY indicates that this API call shall copy previously copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset in bytes from the current data copy position.
Parameters (inout)	None	
Parameters (out)	availableDataPtr	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer. availableDataPtr can be used by TP modules that support dynamic payload lengths (e.g. FrIsoTp) to determine the size of the following CFs.
Return value	BufReq_ReturnType	BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_BUSY: Request could not be fulfilled, because the required amount of Tx data is not available. The lower layer module may retry this call later on. No data has been copied. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Description	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_ DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.	
Available via	PduR_<module>.h	

⌋*(SRS_GTW_06026, SRS_GTW_06121, SRS_BSW_00310)*

8.3.4.5 PduR_<User:LoTp>TxConfirmation

[SWS_PduR_00381] ⌈

Service Name	PduR_<User:LoTp>TxConfirmation

▽

△

Syntax	`void PduR_<User:LoTp>TxConfirmation ( PduIdType id, Std_ReturnType result )`	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	id	Identification of the transmitted I-PDU.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.	
Available via	PduR_<module>.h	

⌋(*SRS_GTW_06026*, *SRS_GTW_06121*, *SRS_BSW_00310*)

8.4 Scheduled functions

As any PDU Router operation is triggered by an adjacent communication module the PDU Router does not require scheduled functions.

8.5 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

The PDU router module is modeled as a generic module that can interface to different upper and lower modules. The approach taken to model this generic approach is to have a virtual module called `GenericComServices`. This virtual module contains a set of APIs that the PDU router will call in upper layer or lower layer modules. These APIs are generic in the way that they contain a tag `<Lo>`, `<Up>` and `<LoTp>` that is replaced with the interfaced module. The tag is set by the configuration in the `PduRBswModules` container using the `PduRBswModuleRef` reference parameter.

8.5.1 Mandatory Interfaces

The PDU Router does not require mandatory interfaces. The required API functions depend on the configuration.

[SWS_PduR_91001] ⌈

API Function	***Header File***	***Description***
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋*(SRS_BSW_00384)*

8.5.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

[SWS_PduR_00424] ⌈

API Function	***Header File***	***Description***
<Provider:Lo>_CancelTransmit	–	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.
<Provider:Lo>_Transmit	–	Requests transmission of a PDU.
<Provider:LoTp>_CancelReceive	–	Requests cancellation of an ongoing reception of a PDU in a lower layer transport protocol module.
<Provider:LoTp>_CancelTransmit	–	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.
<Provider:LoTp>_Transmit	–	Requests transmission of a PDU.
<Provider:Up>_RxIndication	–	Indication of a received PDU from a lower layer communication interface module.
<Provider:Up>_TriggerTransmit	–	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->Sdu Length. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->Sdu Length. If not, it returns E_NOT_OK without changing PduInfoPtr.
<Provider:Up>_TxConfirmation	–	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.
<Provider:UpTp>_CopyRxData	–	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining buffer is written to the position indicated by bufferSizePtr.
<Provider:UpTp>_CopyTxData	–	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.

▽

△

API Function	***Header File***	***Description***
<Provider:UpTp>_StartOfReception	–	This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSdu Length equal to 0.
<Provider:UpTp>_TpRxIndication	–	Called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.
<Provider:UpTp>_TpTxConfirmation	–	This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.
Det_ReportError	Det.h	Service to report development errors.

⌋*(SRS_BSW_00384)*

9 Sequence diagrams

The goal of this chapter is to make the understanding of the PDU Router easier. For this purpose sequence diagrams which show different communication scenarios are used. Please consider that the sequence diagrams are not exhaustive and are only used to support the functional specification (Chapter 7) and API specification (Chapter 8).

Focus of the sequence diagrams is the PDU Router and therefore interactions between other modules (e.g. between an interface and its driver) are not shown.

Note: The sequence diagrams of the I-PDU Multiplexer are shown in [4]. Depending on the interaction scenario the IpduM has to be considered as an upper layer or a lower layer module of the PDU Router.

Note: The diagrams in this chapter are to show specific use-cases. They are not requirements for an implementation of the PDU Router module.

9.1 I-PDU Reception

The reception of an I-PDU received from a Communication Interface module or from Transport Protocol module and forwarded to the COM module.

Note that the PDU Router is not the only customer for the Communication Interface modules and I-PDUs. Other modules such as NM and TP modules receive PDUs directly from the Communication Interface modules.

9.1.1 CanIf module I-PDU reception

Following Figure 9.1 shows reception of I-PDU from the CanIf module to the COM module.

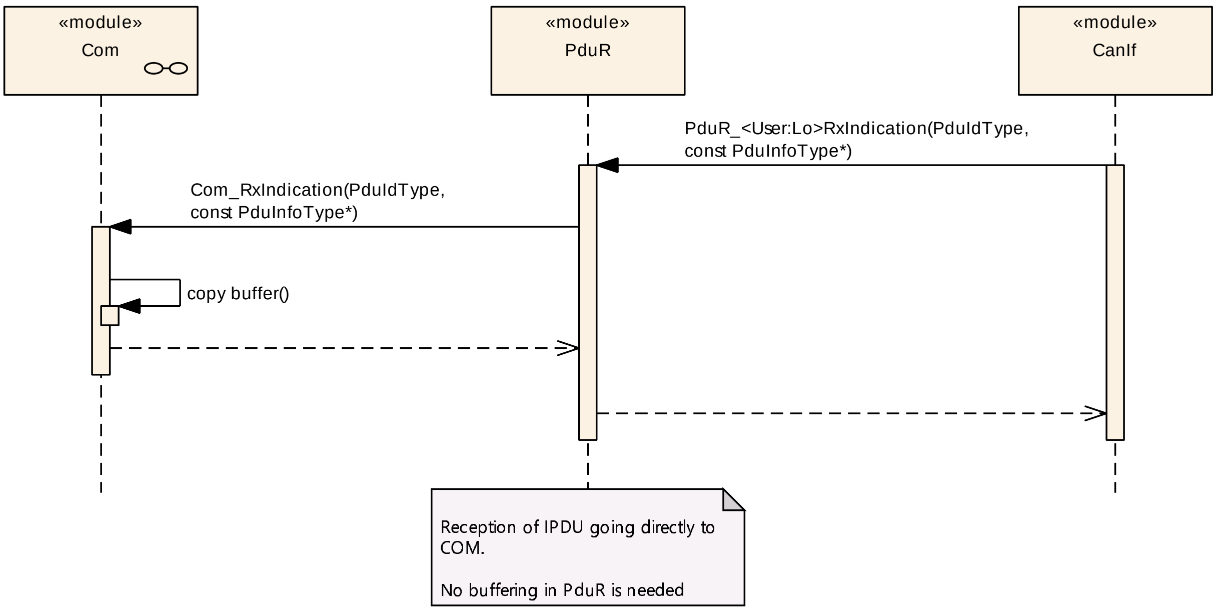
«module»
Com
«module»
PduR
«module»
CanIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
Com_RxIndication(PduIdType, const PduInfoType*)
copy buffer()
Reception of IPDU going directly to COM.
No buffering in PduR is needed

Figure 9.1: CanIf to COM I-PDU reception

9.1.2 FrIf module I-PDU reception

Following Figure 9.2 shows reception of I-PDU from the FrIf module to the COM module.

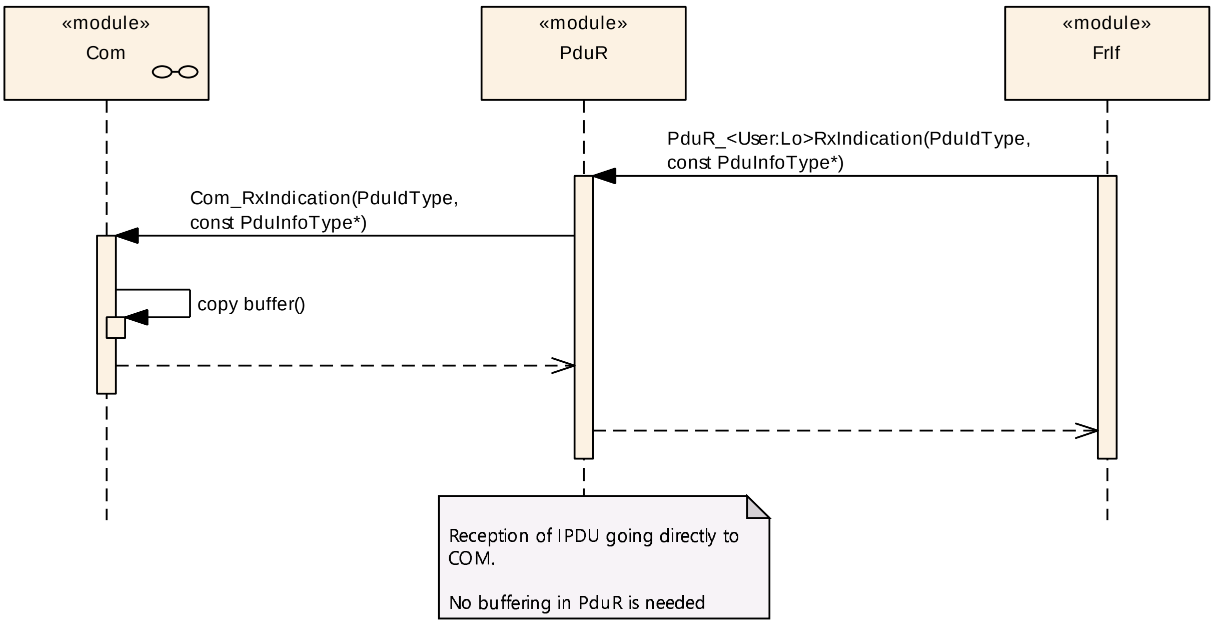
«module»
Com
«module»
PduR
«module»
FrIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
Com_RxIndication(PduIdType, const PduInfoType*)
copy buffer()
Reception of IPDU going directly to COM.
No buffering in PduR is needed

Figure 9.2: FrIf to COM I-PDU reception

9.1.3 LinIf module I-PDU reception

Following Figure 9.3 shows reception of I-PDU from the LinIf module to the COM module.

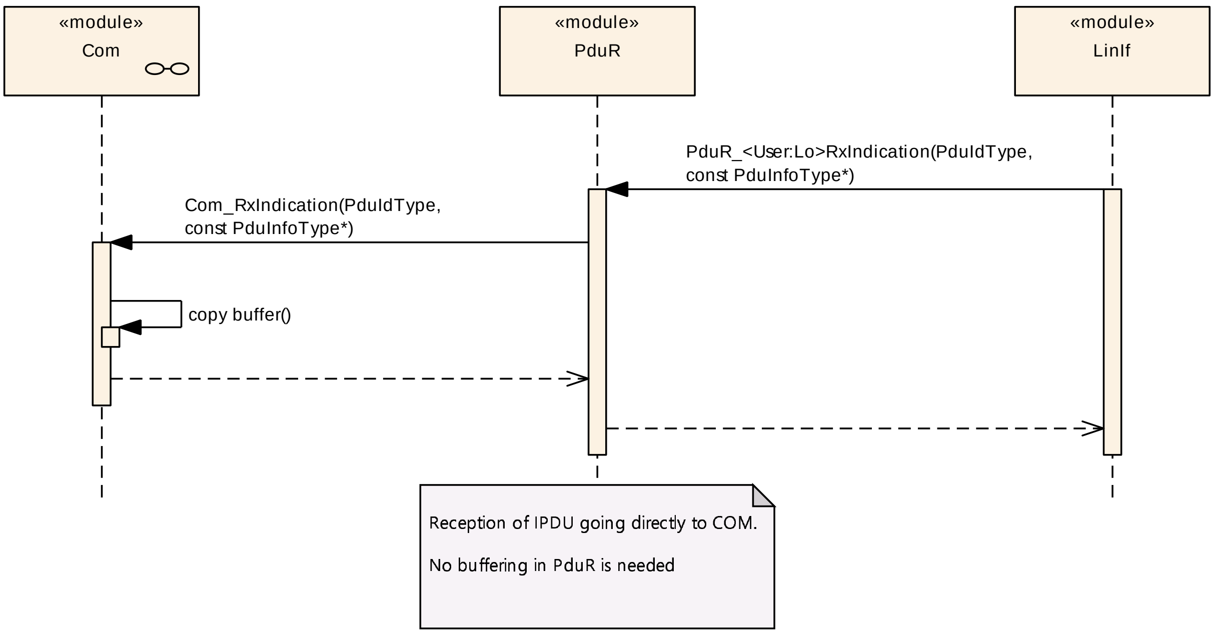
«module»
Com
«module»
PduR
«module»
LinIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
Com_RxIndication(PduIdType, const PduInfoType*)
copy buffer()
Reception of IPDU going directly to COM.
No buffering in PduR is needed

Figure 9.3: LinIf to COM I-PDU reception

9.1.4 CanTp module I-PDU reception

Following Figure 9.4 shows reception of I-PDU from the CanTp module to the DCM module. The reception is made using the Transport Protocol APIs.

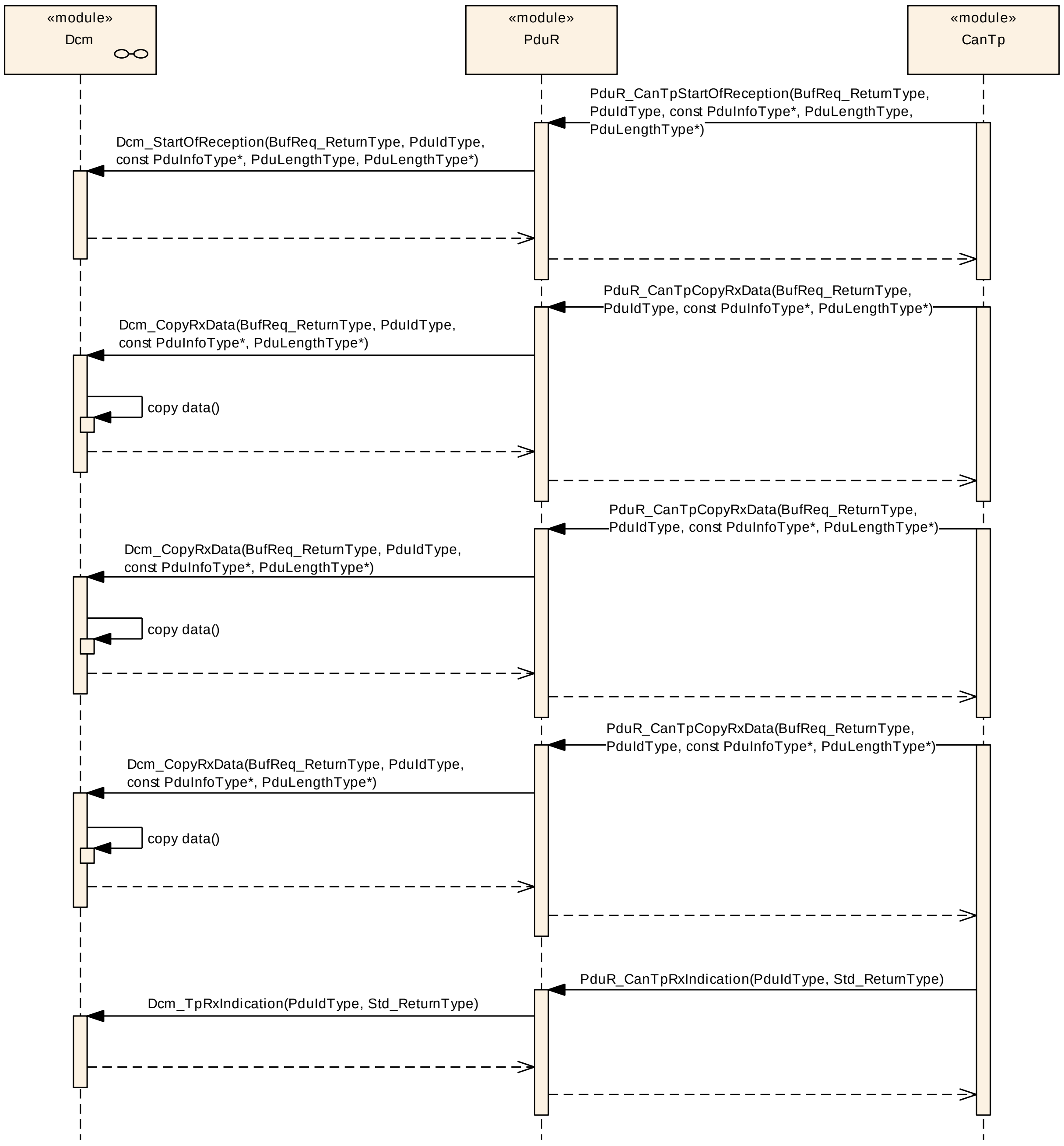
«module» Dcm
«module» PduR
«module» CanTp
PduR_CanTpStartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
Dcm_StartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
PduR_CanTpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
PduR_CanTpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
PduR_CanTpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
PduR_CanTpRxIndication(PduIdType, Std_ReturnType)
Dcm_TpRxIndication(PduIdType, Std_ReturnType)

Figure 9.4: CanTp to Dcm TP PDU reception

9.2 I-PDU transmission

The transmission of an I-PDU transmitted from the COM module to a Communication Interface module or a Transport Protocol module.

9.2.1 CanIf module I-PDU transmission

Following Figure 9.5 shows transmission of I-PDU from the COM module to the CanIf module.

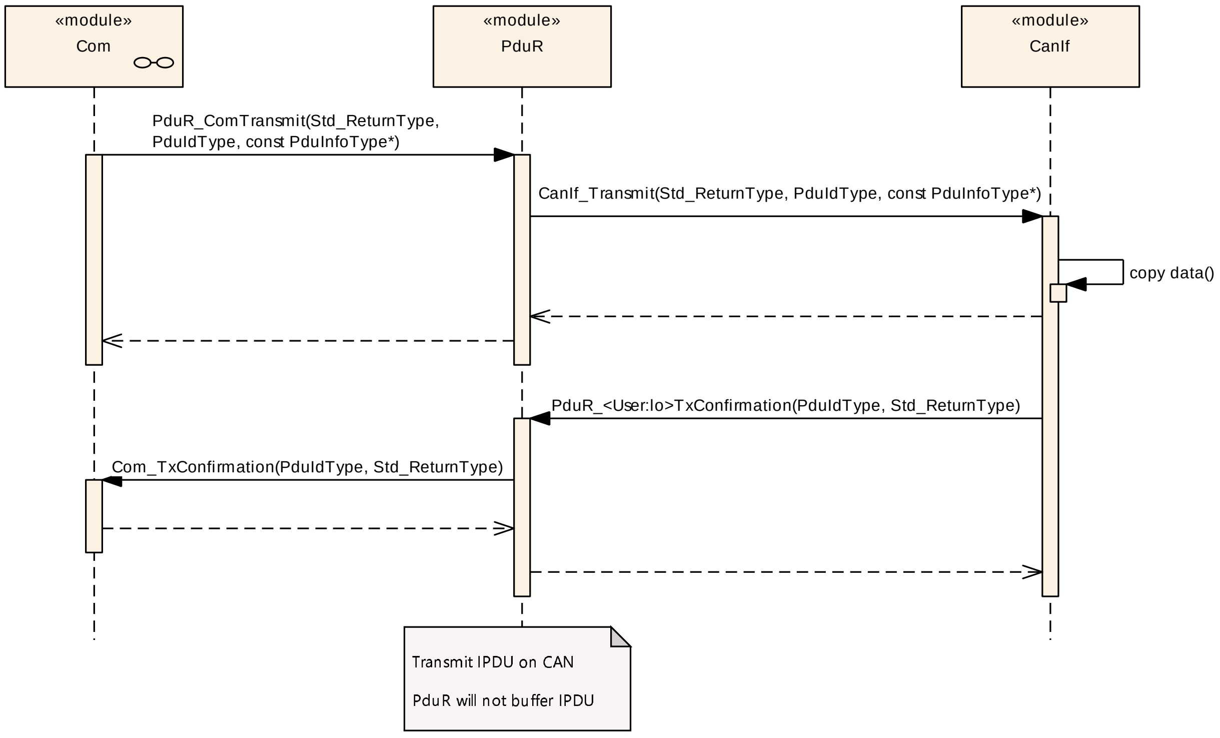
«module» Com
«module» PduR
«module» CanIf
PduR_ComTransmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
copy data()
PduR_<User:lo>TxConfirmation(PduIdType, Std_ReturnType)
Com_TxConfirmation(PduIdType, Std_ReturnType)
Transmit IPDU on CAN
PduR will not buffer IPDU

Figure 9.5: Com to CanIf I-PDU transmission

9.2.2 FrIf module I-PDU transmission

Following Figure 9.6 shows transmission of I-PDU from the COM module to the FrIf module using trigger transmit.

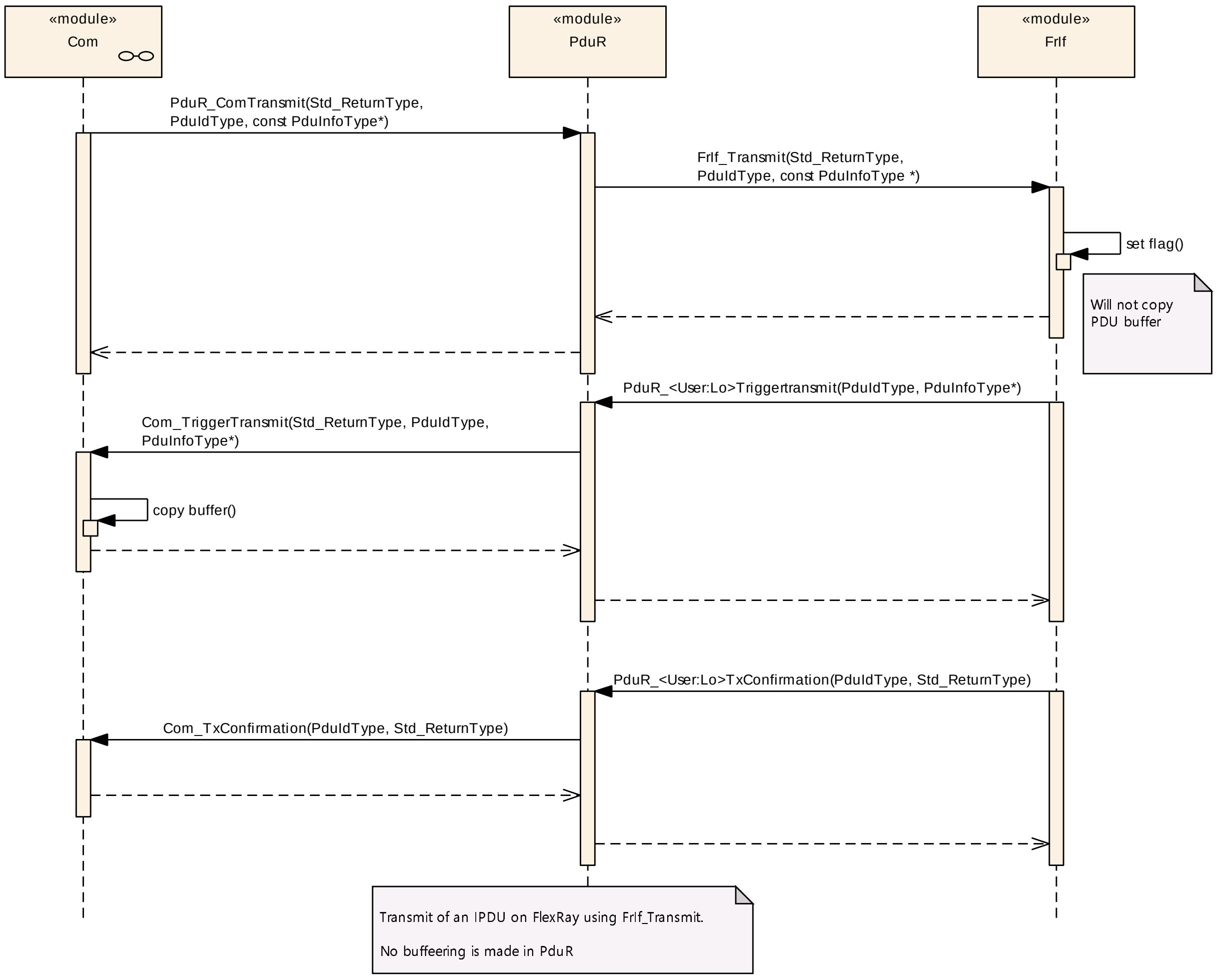
«module» Com
«module» PduR
«module» FrIf
PduR_ComTransmit(Std_ReturnType, PduIdType, const PduInfoType*)
FrIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType *)
set flag()
Will not copy PDU buffer
PduR_<User:Lo>Triggertransmit(PduIdType, PduInfoType*)
Com_TriggerTransmit(Std_ReturnType, PduIdType, PduInfoType*)
copy buffer()
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Com_TxConfirmation(PduIdType, Std_ReturnType)
Transmit of an IPDU on FlexRay using FrIf_Transmit.
No buffeering is made in PduR

Figure 9.6: Com to FrIf I-PDU transmission

9.2.3 LinIf module I-PDU transmission

Following Figure 9.7 shows transmission of I-PDU from the COM module to the LinIf module using transmit and later trigger transmit functions. In this case the I-PDU is a LIN sporadic frame.

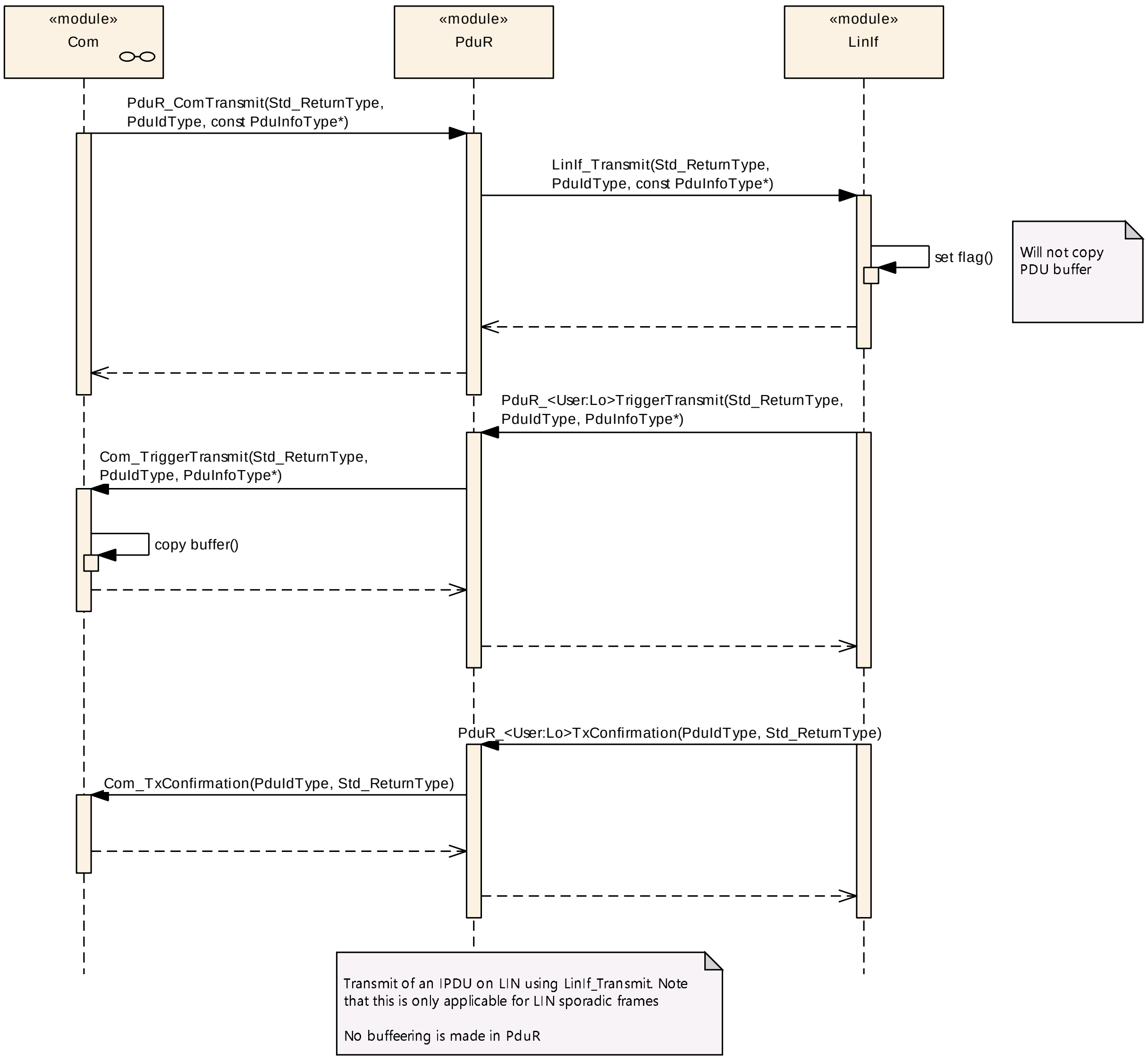
«module» Com
«module» PduR
«module» LinIf
PduR_ComTransmit(Std_ReturnType, PduIdType, const PduInfoType*)
LinIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
set flag()
Will not copy PDU buffer
PduR_<User:Lo>TriggerTransmit(Std_ReturnType, PduIdType, PduInfoType*)
Com_TriggerTransmit(Std_ReturnType, PduIdType, PduInfoType*)
copy buffer()
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Com_TxConfirmation(PduIdType, Std_ReturnType)
Transmit of an IPDU on LIN using LinIf_Transmit. Note that this is only applicable for LIN sporadic frames
No buffeering is made in PduR

Figure 9.7: Com to LinIf I-PDU transmission (LIN sporadic frame)

Following Figure 9.8 shows transmission of I-PDU from the COM module to the Lin If module using trigger transmit. In this case the I-PDU is all other types except LIN sporadic frame.

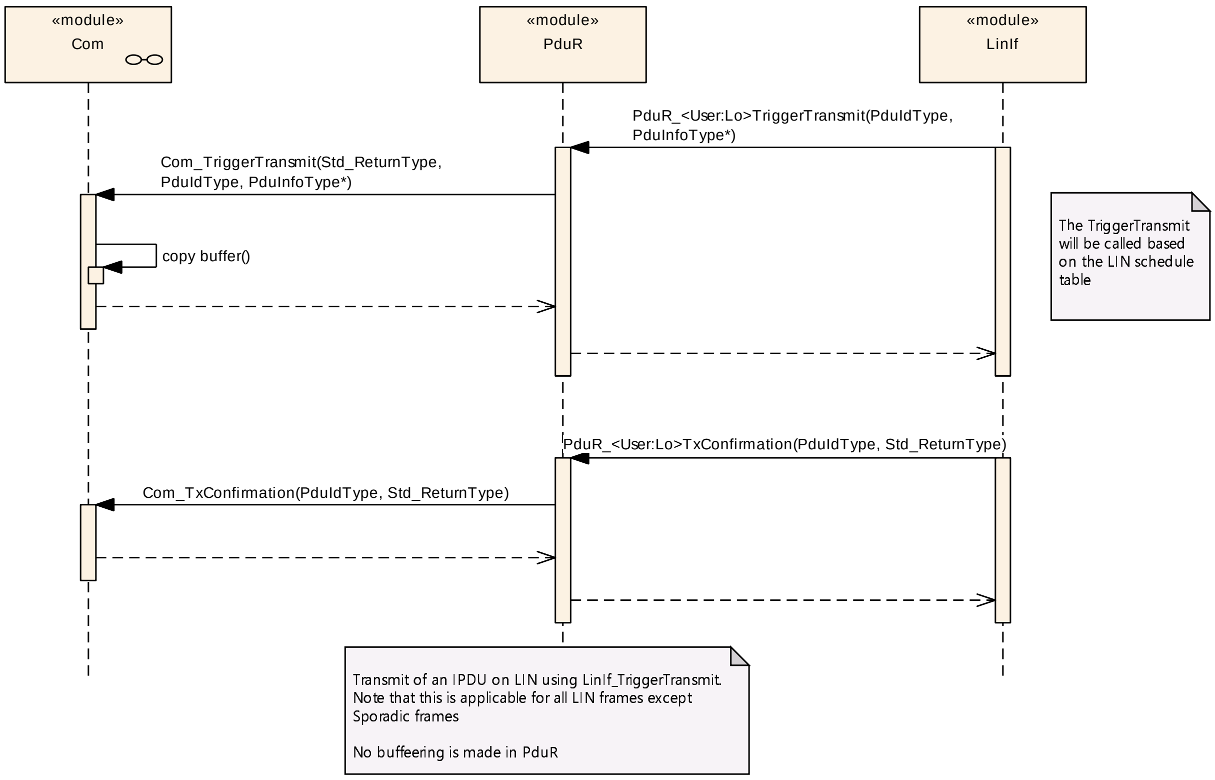
«module» Com
«module» PduR
«module» LinIf
PduR_<User:Lo>TriggerTransmit(PduIdType, PduInfoType*)
Com_TriggerTransmit(Std_ReturnType, PduIdType, PduInfoType*)
copy buffer()
The TriggerTransmit will be called based on the LIN schedule table
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Com_TxConfirmation(PduIdType, Std_ReturnType)
Transmit of an IPDU on LIN using LinIf_TriggerTransmit. Note that this is applicable for all LIN frames except Sporadic frames
No buffeering is made in PduR

Figure 9.8: Com to LinIf I-PDU transmission (LIN sporadic non-frame)

9.2.4 CanTp module I-PDU transmission

Following Figure 9.9 shows transmission of I-PDU from the DCM module to the CanTp module using the Transport Protocol API.

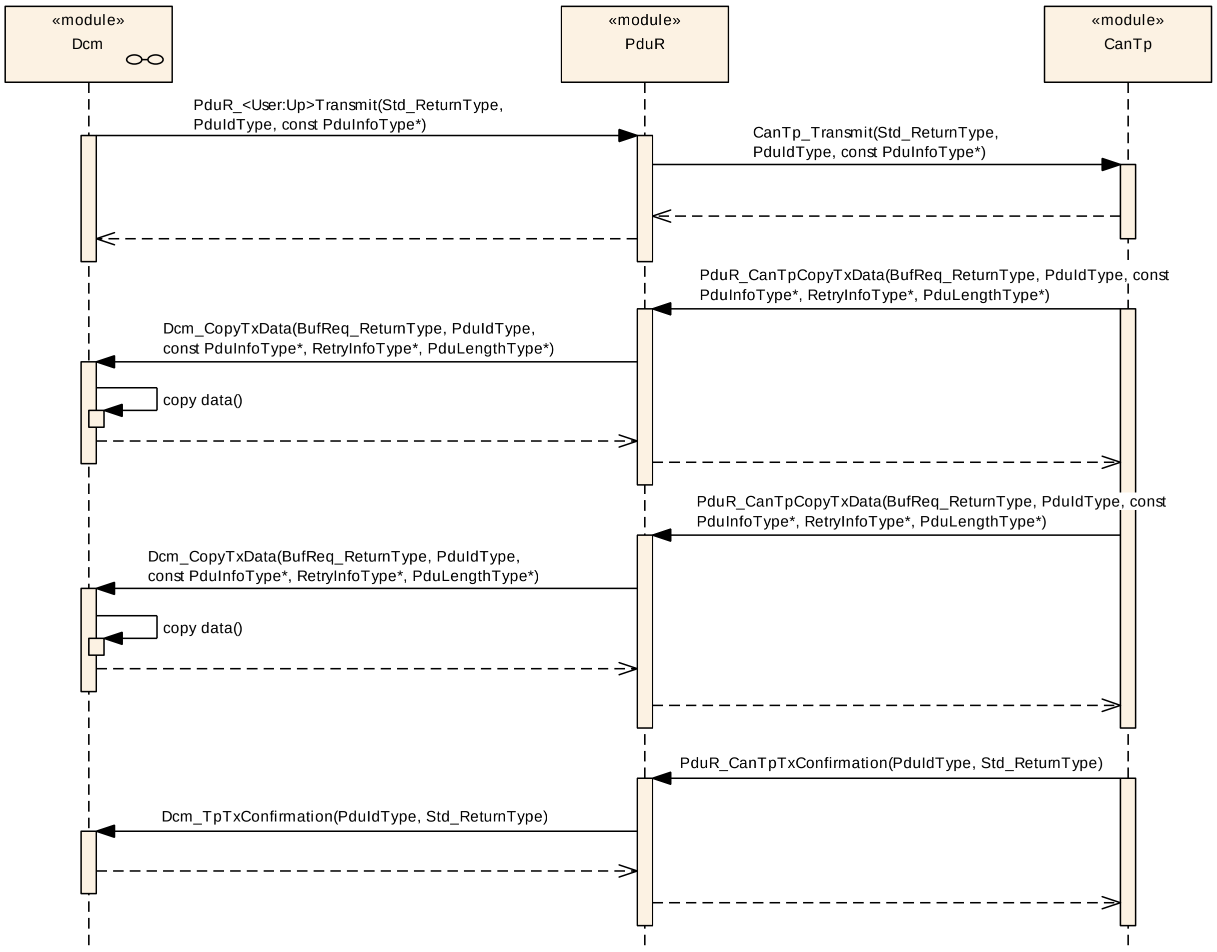
«module»
Dcm
«module»
PduR
«module»
CanTp
PduR_<User:Up>Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
PduR_CanTpTxConfirmation(PduIdType, Std_ReturnType)
Dcm_TpTxConfirmation(PduIdType, Std_ReturnType)

Figure 9.9: Dcm to CanTp PDU transmission

9.2.5 Multicast I-PDU transmission on Transport Protocol modules

Following Figure 9.10 shows transmission of I-PDU from the DCM module to the Can Tp, FrTp and LinTp (LinIf includes the Transport Protocol module) module using the Transport Protocol API.

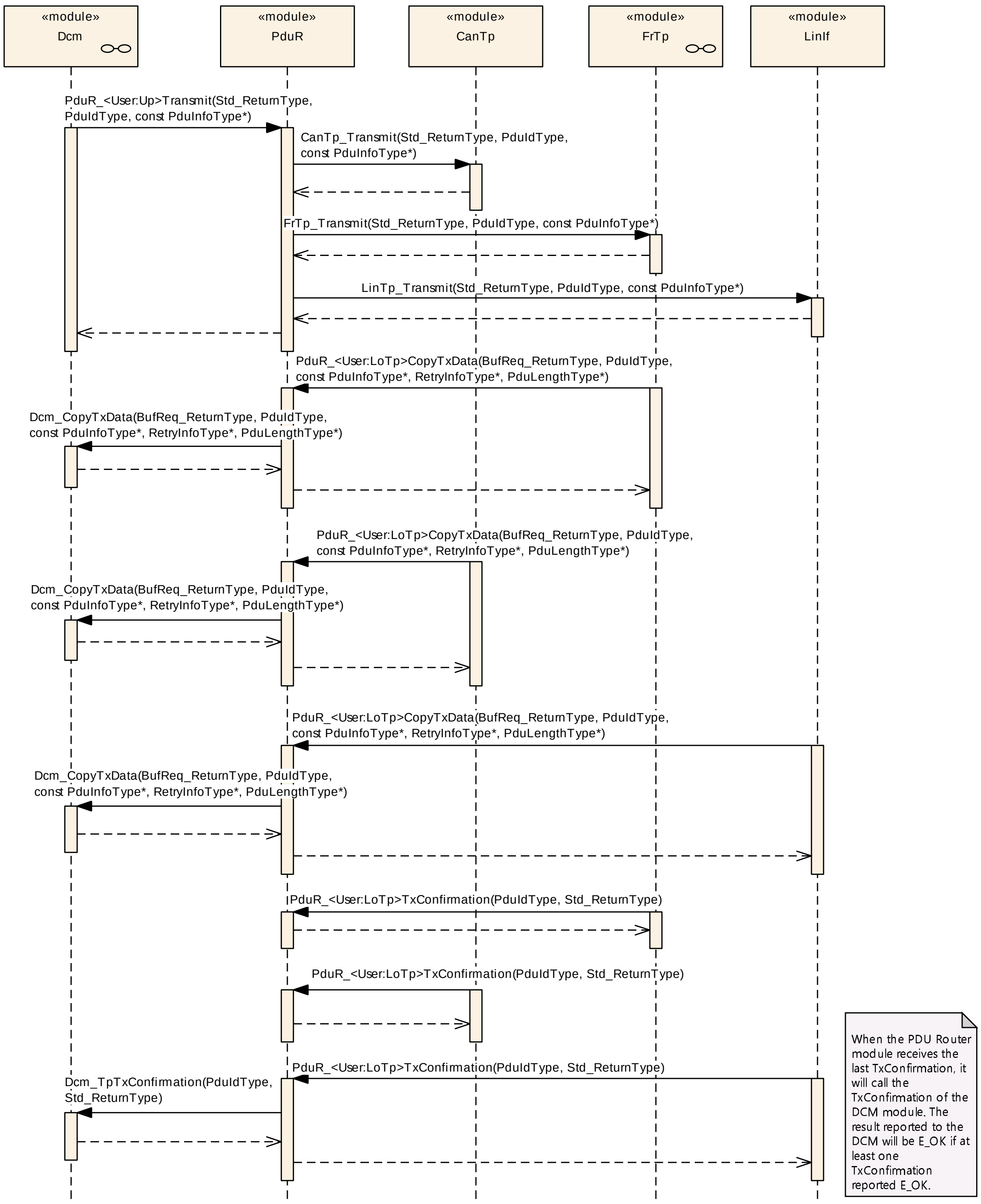
«module» Dcm
«module» PduR
«module» CanTp
«module» FrTp
«module» LinIf
PduR_<User:Up>Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
FrTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
LinTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
PduR_<User:LoTp>CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
PduR_<User:LoTp>CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
PduR_<User:LoTp>CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Dcm_CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
PduR_<User:LoTp>TxConfirmation(PduIdType, Std_ReturnType)
PduR_<User:LoTp>TxConfirmation(PduIdType, Std_ReturnType)
PduR_<User:LoTp>TxConfirmation(PduIdType, Std_ReturnType)
Dcm_TpTxConfirmation(PduIdType, Std_ReturnType)
When the PDU Router module receives the last TxConfirmation, it will call the TxConfirmation of the DCM module. The result reported to the DCM will be E_OK if at least one TxConfirmation reported E_OK.

Figure 9.10: Local PDU transmission on Transport Protocol from Dcm to CAN, FlexRay and LIN

9.3 Gateway of I-PDU

Following use-cases shows how the PDU Router modules will gateway I-PDUs.

9.3.1 Gateway between two CanIfs

Following Figure 9.11 shows how an I-PDU is gatewayed between two CAN networks (CAN1 and CAN2) using CanIf.

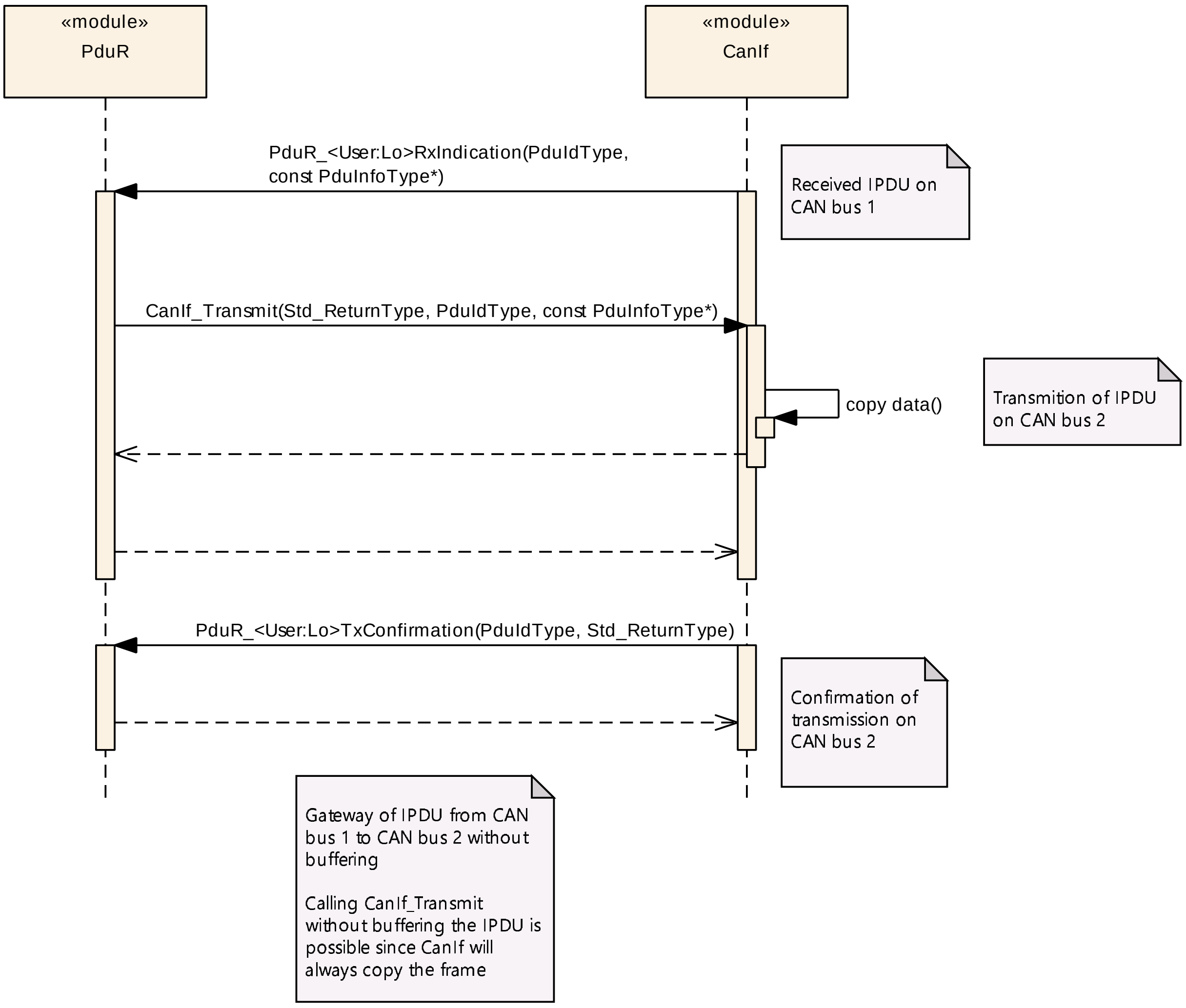
«module»
PduR
«module»
CanIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
Received IPDU on CAN bus 1
CanIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
copy data()
Transmition of IPDU on CAN bus 2
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Confirmation of transmission on CAN bus 2
Gateway of IPDU from CAN bus 1 to CAN bus 2 without buffering
Calling CanIf_Transmit without buffering the IPDU is possible since CanIf will always copy the frame

Figure 9.11: Gateway of I-PDU from CAN1 to CAN2

9.3.2 Gateway from CAN to FlexRay

Following Figure 9.12 shows how an I-PDU is gatewayed between CAN and FlexRay, using trigger transmit (with buffering and without buffering).

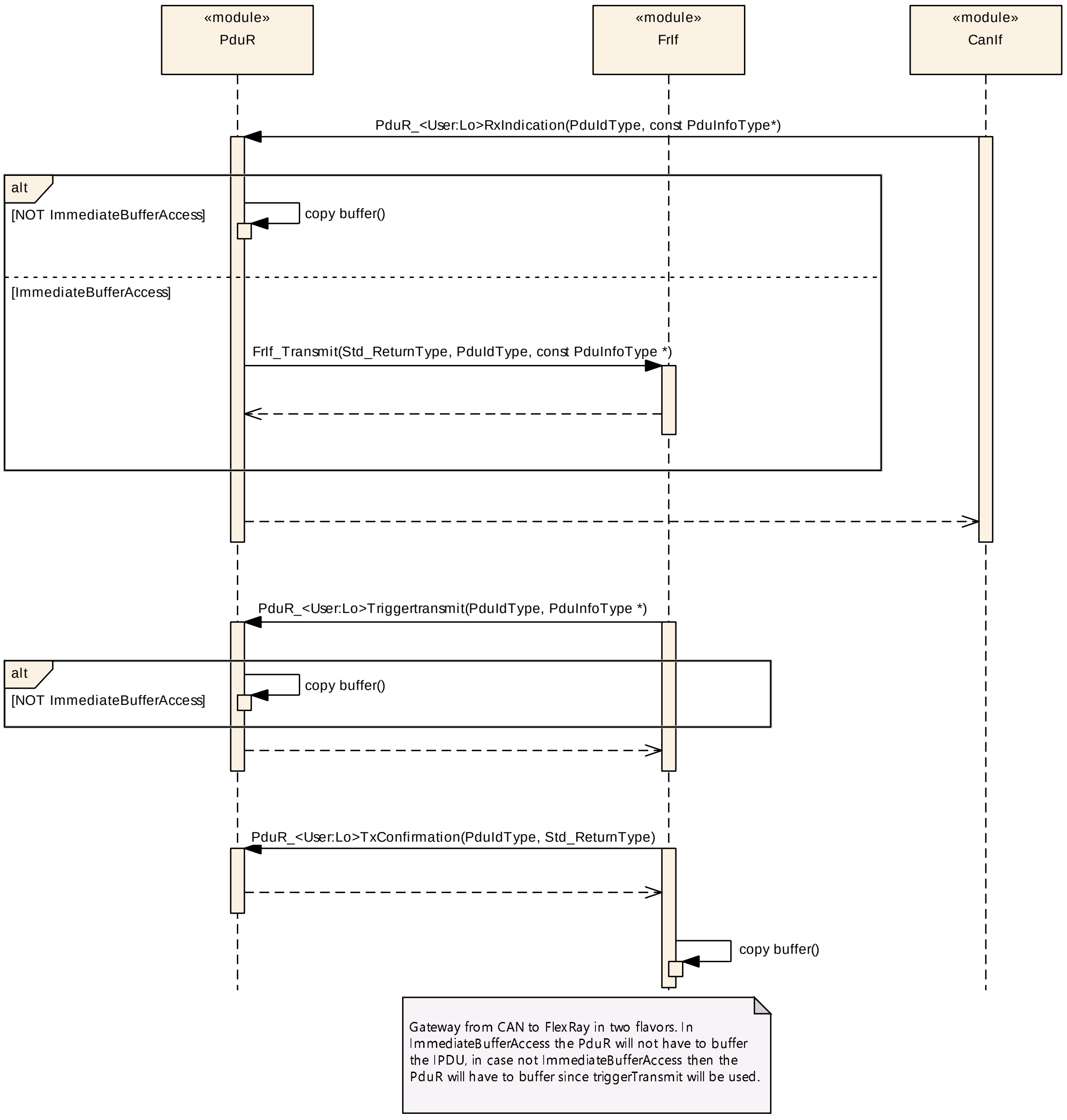
«module» PduR
«module» FrIf
«module» CanIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
alt
[NOT ImmediateBufferAccess]
copy buffer()
[ImmediateBufferAccess]
FrIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType *)
PduR_<User:Lo>Triggertransmit(PduIdType, PduInfoType *)
alt
[NOT ImmediateBufferAccess]
copy buffer()
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
copy buffer()
Gateway from CAN to FlexRay in two flavors. In ImmediateBufferAccess the PduR will not have to buffer the IPDU, in case not ImmediateBufferAccess then the PduR will have to buffer since triggerTransmit will be used.

Figure 9.12: Gateway of I-PDU from CAN to FlexRay

9.3.3 Gateway from CAN to LIN

Following Figure 9.13 shows how an I-PDU is gatewayed from CAN to LIN, using trigger transmit (LIN sporadic frame and all other LIN frame types).

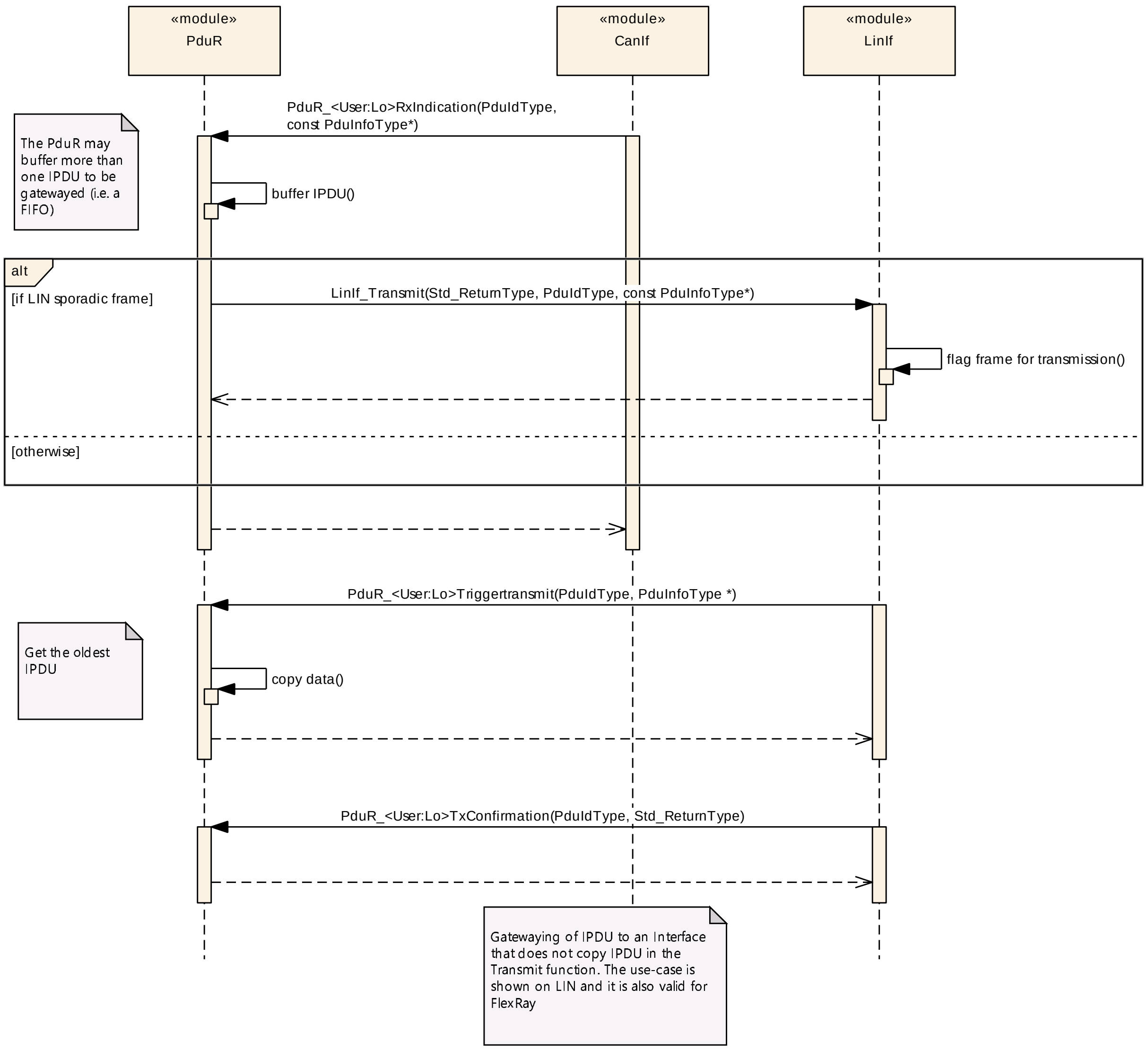
«module»
PduR
«module»
CanIf
«module»
LinIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
The PduR may buffer more than one IPDU to be gatewayed (i.e. a FIFO)
buffer IPDU()
alt
[if LIN sporadic frame]
LinIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
flag frame for transmission()
[otherwise]
PduR_<User:Lo>Triggertransmit(PduIdType, PduInfoType *)
Get the oldest IPDU
copy data()
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Gatewaying of IPDU to an Interface that does not copy IPDU in the Transmit function. The use-case is shown on LIN and it is also valid for FlexRay

Figure 9.13: Gateway of I-PDU from CAN to LIN

9.3.4 Gateway from CAN to CAN and received by the COM module

Following Figure 9.14 shows how an I-PDU is gatewayed from CAN1 to CAN2 and also received locally by the COM module.

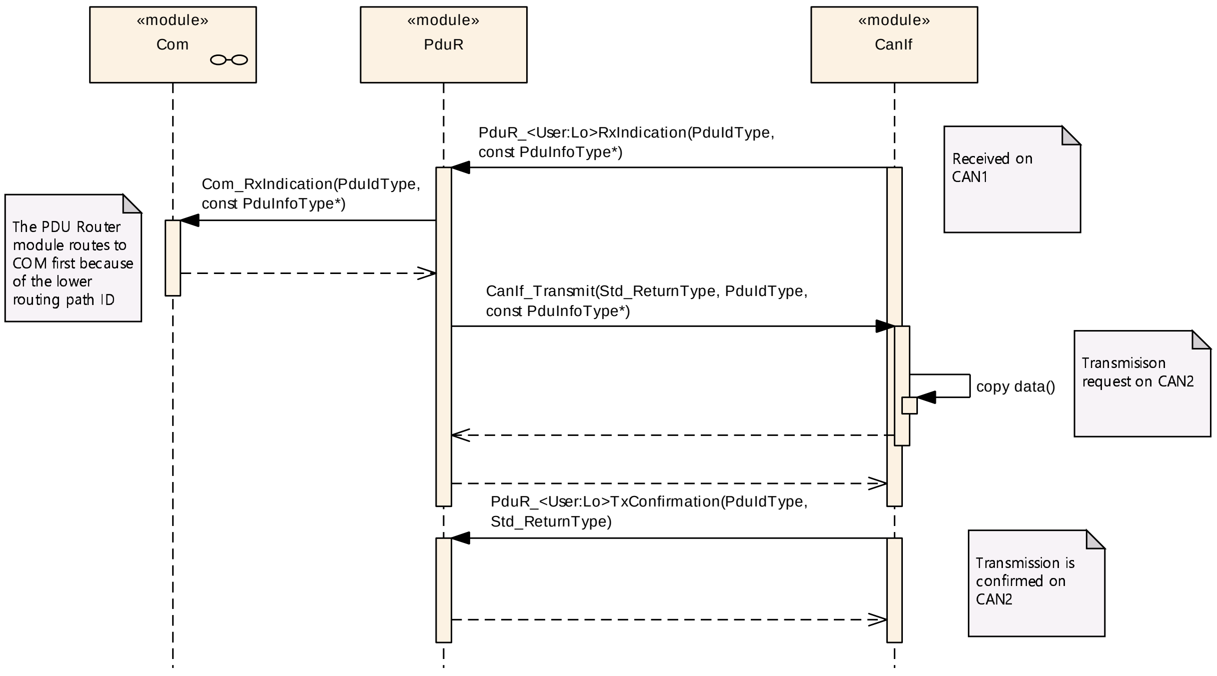
«module» Com
«module» PduR
«module» CanIf
PduR_<User:Lo>RxIndication(PduIdType, const PduInfoType*)
Received on CAN1
Com_RxIndication(PduIdType, const PduInfoType*)
The PDU Router module routes to COM first because of the lower routing path ID
CanIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
copy data()
Transmisison request on CAN2
PduR_<User:Lo>TxConfirmation(PduIdType, Std_ReturnType)
Transmission is confirmed on CAN2

Figure 9.14: Gateway of I-PDU from CAN to CAN and Com

9.3.5 Singlecast-Gateway I-PDU using Transport Protocol modules

Following Figure 9.15 shows how an (multi N-PDU) I-PDU is gatewayed between two CAN networks, using Transport Protocol module.

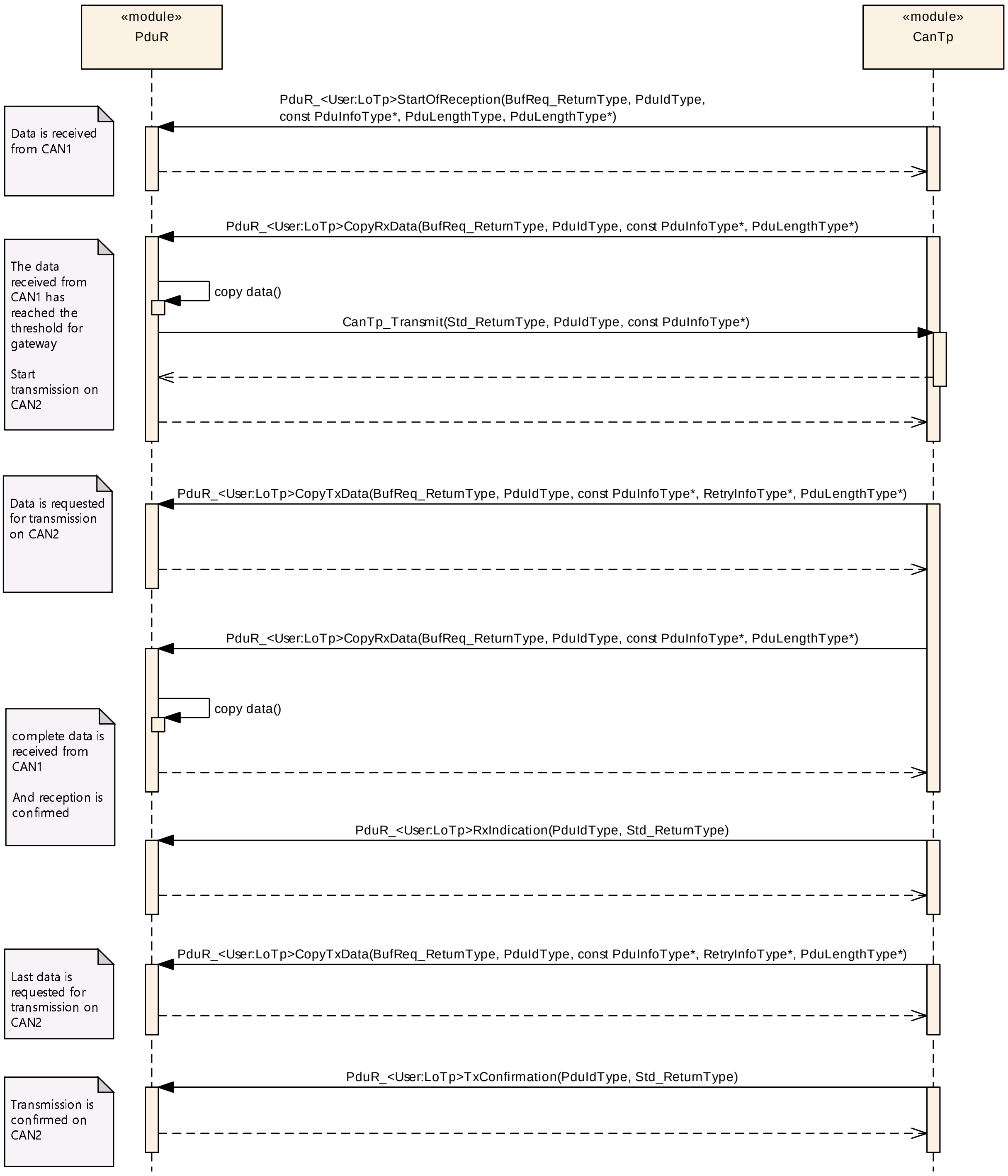
«module» PduR
«module» CanTp
PduR_<User:LoTp>StartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
Data is received from CAN1
PduR_<User:LoTp>CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
The data received from CAN1 has reached the threshold for gateway
Start transmission on CAN2
PduR_<User:LoTp>CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Data is requested for transmission on CAN2
PduR_<User:LoTp>CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
complete data is received from CAN1
And reception is confirmed
PduR_<User:LoTp>RxIndication(PduIdType, Std_ReturnType)
PduR_<User:LoTp>CopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Last data is requested for transmission on CAN2
PduR_<User:LoTp>TxConfirmation(PduIdType, Std_ReturnType)
Transmission is confirmed on CAN2

Figure 9.15: Gateway of I-PDU (multi N-PDU) between two CAN networks

9.3.6 Multicast-Gateway I-PDU using Transport Protocol modules

The following Figure 9.16 shows how an (multi N-PDU) I-PDU is gatewayed from J1939Tp to two CAN-Networks, using Transport Protocol module.

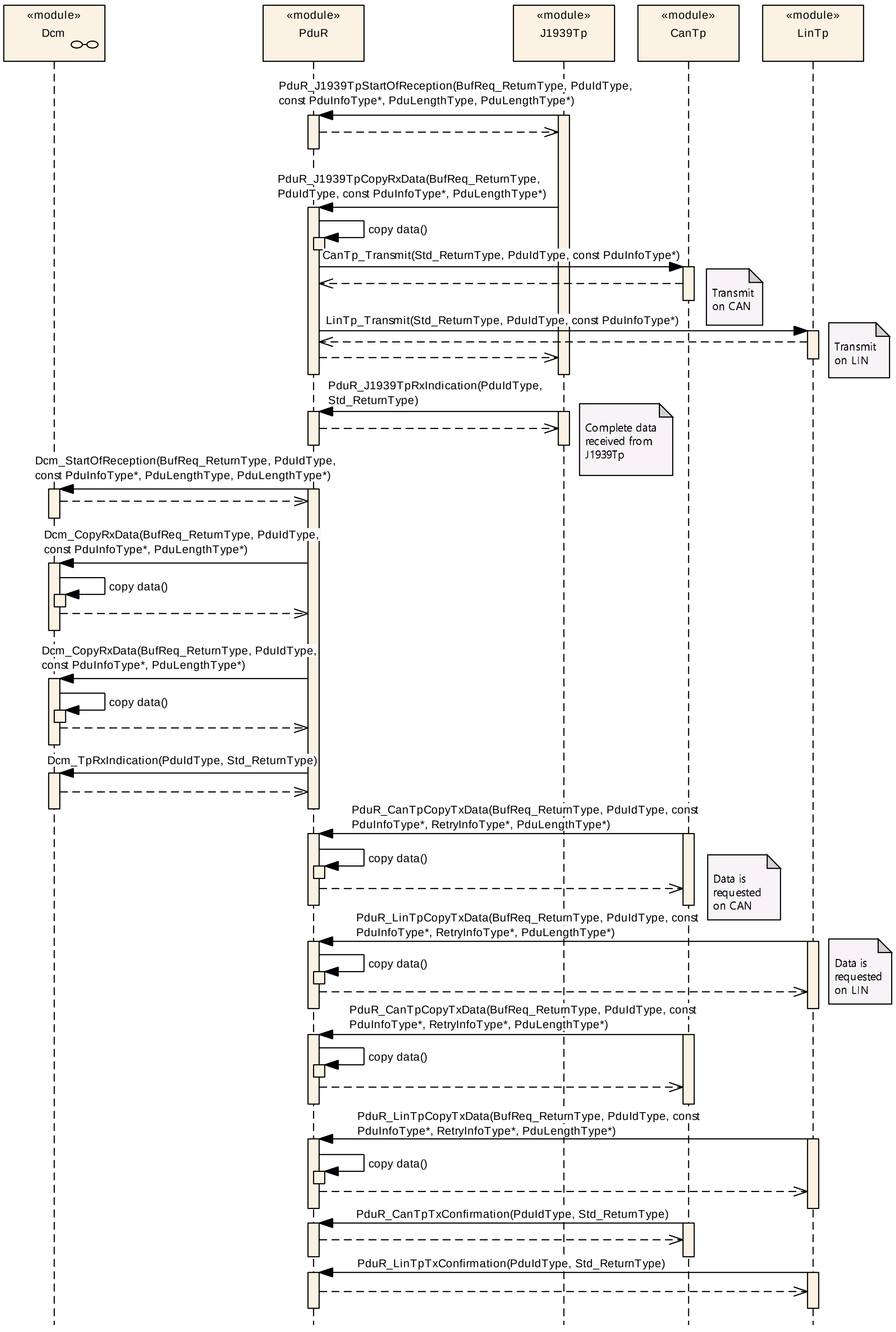
«module» Dcm
«module» PduR
«module» J1939Tp
«module» CanTp
«module» LinTp
PduR_J1939TpStartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
PduR_J1939TpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
Transmit on CAN
LinTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
Transmit on LIN
PduR_J1939TpRxIndication(PduIdType, Std_ReturnType)
Complete data received from J1939Tp
Dcm_StartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
Dcm_TpRxIndication(PduIdType, Std_ReturnType)
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
Data is requested on CAN
PduR_LinTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
Data is requested on LIN
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
PduR_LinTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
PduR_CanTpTxConfirmation(PduIdType, Std_ReturnType)
PduR_LinTpTxConfirmation(PduIdType, Std_ReturnType)

Figure 9.16: Gateway of I-PDU (multi N-PDU) between J1939Tp and two CanTp

9.3.7 Gateway Single Frame I-PDU from CAN1 to DCM and CAN2

The following use-case, Figure 9.17, shows an I-PDU (contained in a SF) received from CAN1 Transport Protocol and gatewayed to DCM (internal) and gatewayed to CAN2 Transport Protocol.

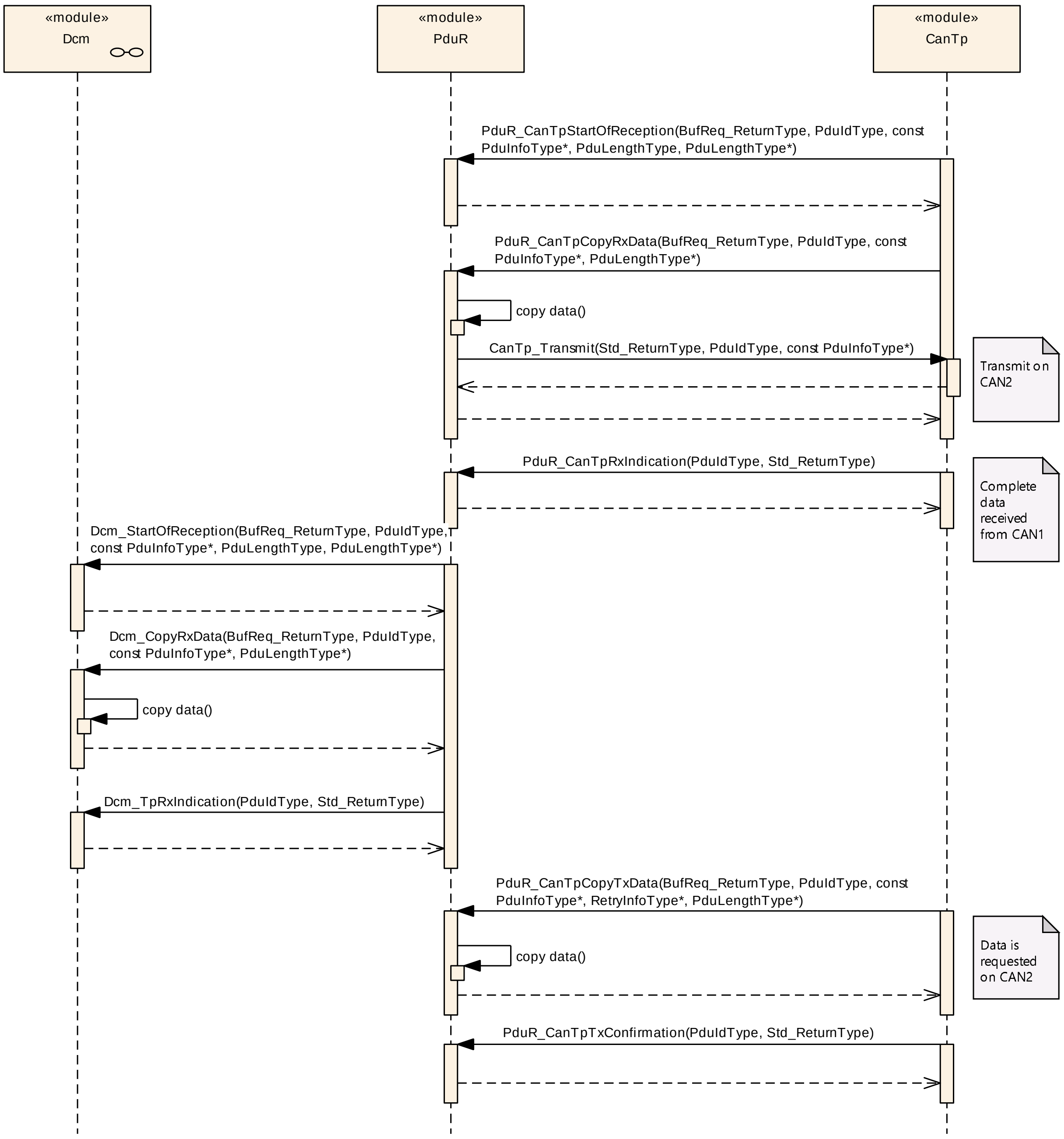
«module» Dcm
«module» PduR
«module» CanTp
PduR_CanTpStartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
PduR_CanTpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
Transmit on CAN2
PduR_CanTpRxIndication(PduIdType, Std_ReturnType)
Complete data received from CAN1
Dcm_StartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
Dcm_CopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
Dcm_TpRxIndication(PduIdType, Std_ReturnType)
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
copy data()
Data is requested on CAN2
PduR_CanTpTxConfirmation(PduIdType, Std_ReturnType)

Figure 9.17: Gateway of I-PDU to DCM and CAN

The I-PDU must be buffered in the PDU Router since the DCM module is not aware of that it will be gatewayed to CAN2. Such gatewaying is controlled by the configuration and cannot be processed by the DCM.

9.3.8 Gateway Multi Frame I-PDU from J1939Tp to DCM, CAN and LIN

The following Figure 9.18 shows how routing of a broadcast TP protocol (e.g. BAM of J1939Tp) is performed.

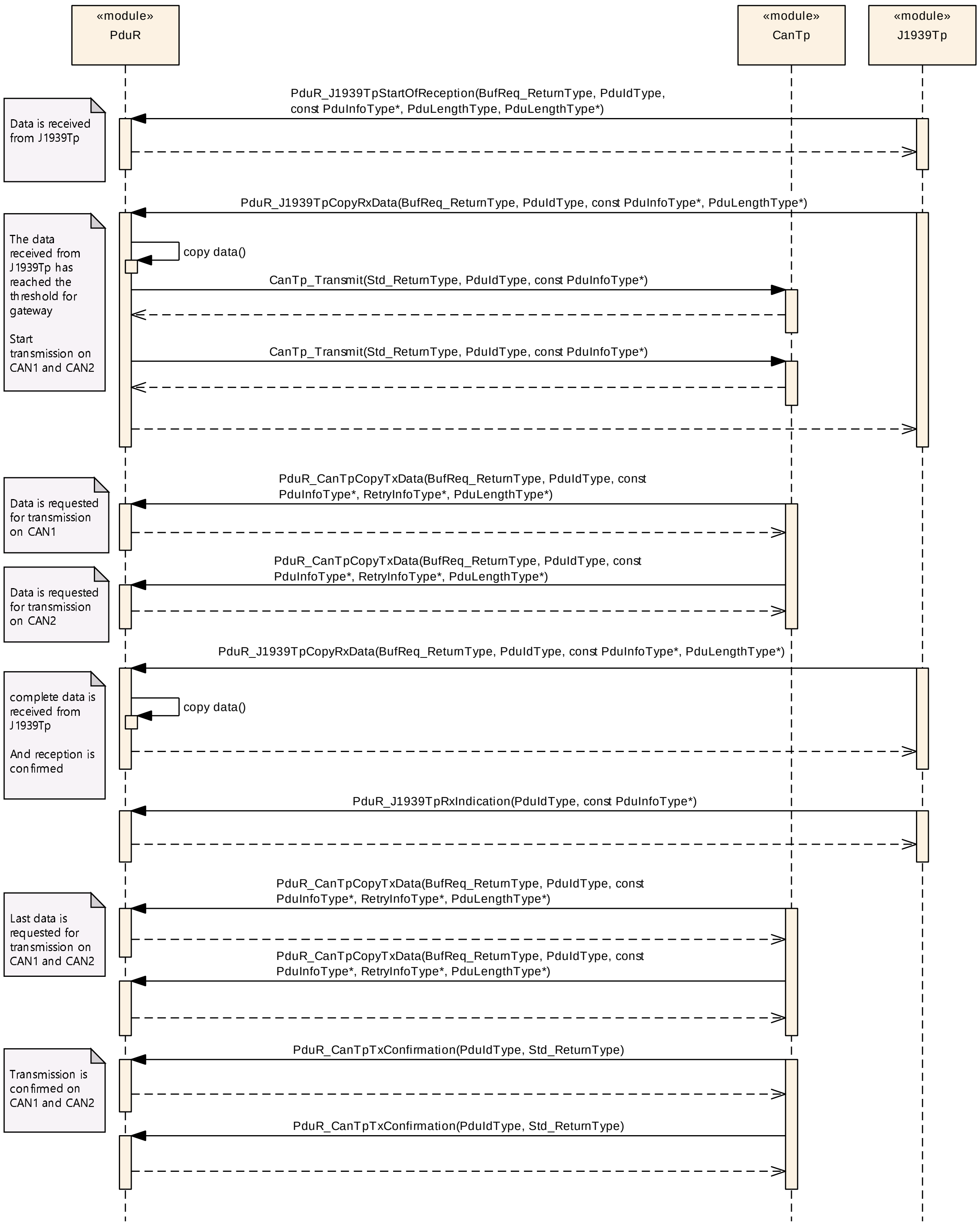
«module» PduR
«module» CanTp
«module» J1939Tp
Data is received from J1939Tp
PduR_J1939TpStartOfReception(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType, PduLengthType*)
The data received from J1939Tp has reached the threshold for gateway
Start transmission on CAN1 and CAN2
PduR_J1939TpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
Data is requested for transmission on CAN1
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Data is requested for transmission on CAN2
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
complete data is received from J1939Tp
And reception is confirmed
PduR_J1939TpCopyRxData(BufReq_ReturnType, PduIdType, const PduInfoType*, PduLengthType*)
copy data()
PduR_J1939TpRxIndication(PduIdType, const PduInfoType*)
Last data is requested for transmission on CAN1 and CAN2
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
PduR_CanTpCopyTxData(BufReq_ReturnType, PduIdType, const PduInfoType*, RetryInfoType*, PduLengthType*)
Transmission is confirmed on CAN1 and CAN2
PduR_CanTpTxConfirmation(PduIdType, Std_ReturnType)
PduR_CanTpTxConfirmation(PduIdType, Std_ReturnType)

Figure 9.18: Routing of Broadcast Tp protocol

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module PDU Router.

Chapter 10.3 specifies published information of the module PDU Router.

10.1 How to read this chapter

For details refer to the chapter 10.1 "Introduction to configuration specification" in [2, SWS_BSWGeneral].

10.1.1 Variants

[SWS_PduR_00295] ⌈The PDU Router module shall support the update of the routing configuration (i.e. the PDU Router routing tables) at post build-time if this variant is supported.⌋*(SRS_GTW_06002, SRS_BSW_00404)*

Support of post-build update of the routing table is not always desired. Therefore post-build update of the routing table is only supported in the variant post-build of the PDU Router module, see further section 10.1.1.

The post-build comes in two flavors: Selectable and Loadable, there is no restriction on using any of them in the PDU Router module or even a combination of them.

[SWS_PduR_00296] ⌈If the variant post-build is supported, the update of the routing tables shall only be possible when the PDU Router module is uninitialized.⌋*(SRS_-GTW_06001, SRS_BSW_00404)*

Remark: The process how the update of the routing tables is performed is not restricted. Most likely a reflashing of the memory segment that holds the table will be done by the bootloader - a separate program which may be loaded after a reboot to update the ECU.

[SWS_PduR_00281] ⌈The post-build time configuration of the PDU Router module shall be identifiable by the unique configuration identifier: `PduRConfigurationId`⌋*(SRS_GTW_06097, SRS_BSW_00404)*

Remark: The unique configuration identifier is not used to select one of multiple post-build configuration sets of the PDU Router module, but for unique identification of the current PDU Router module post-build configuration, e.g. for Diagnostics or for checking at runtime that the post-build configurations of related communication modules match. The configuration identifier can be read via the API `PduR_GetConfigurationId` see section 8.3.1.3.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in chapter 7 and chapter 8. An overview of the top-level PDU Router configuration container PduR is shown in Figure 10.1.

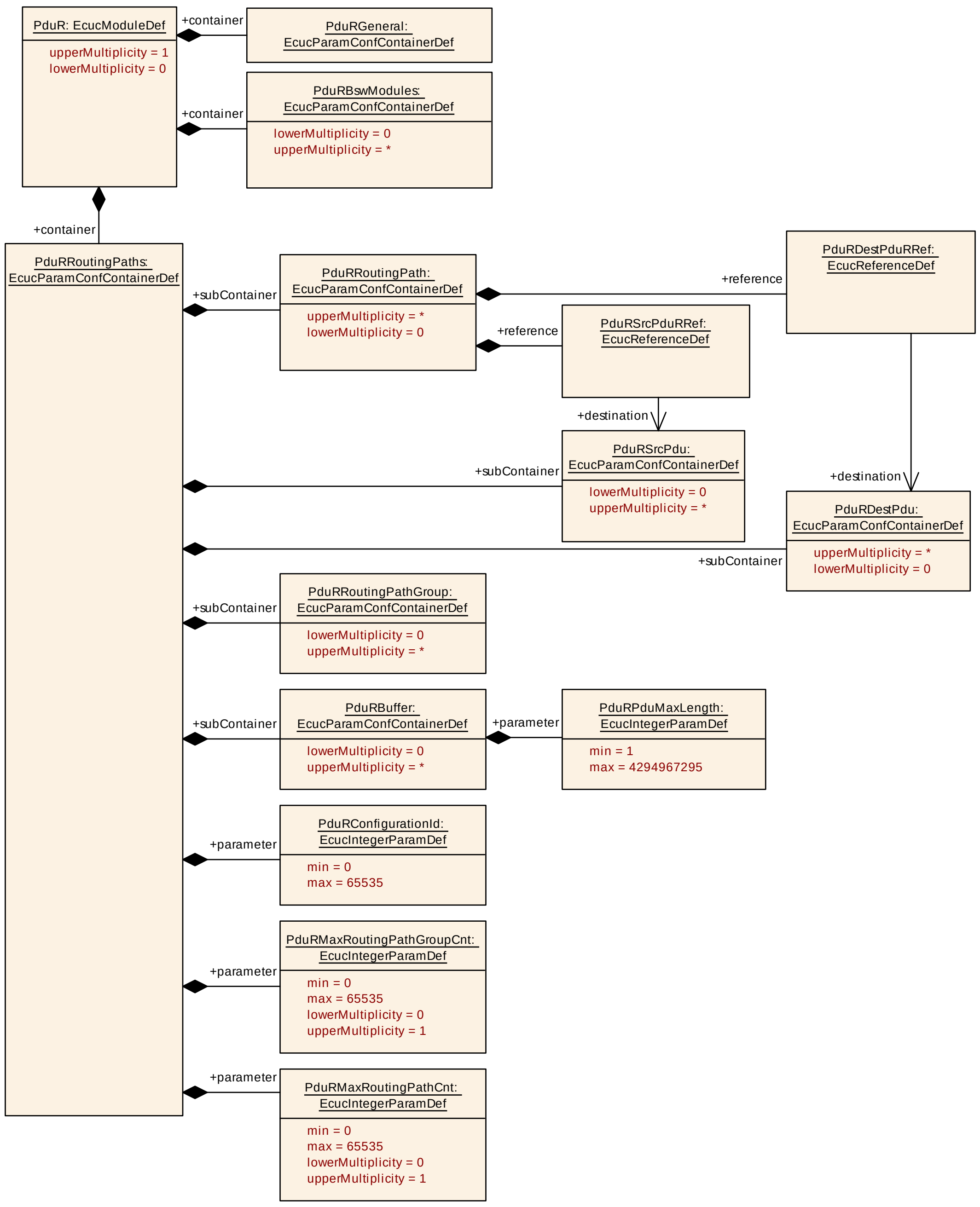

Figure 10.1: PDU Router Configuration Overview - PduR

10.2.1 PduR

Module SWS Item	ECUC_PduR_00293	
Module Name	PduR	
Module Description	Configuration of the PduR (PDU Router) module.	
Post-Build Variant Support	true	
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
PduRBswModules	0..*	Each container describes a specific BSW module (upper/CDD/lower/IpduM) that the PDU Router shall interface to. The reason to have it as own configuration container instead of implication of the routing path is to be able to configure CDDs properly and to force module's to be used in a post-build situation even though no routing is made to/from this module (future configurations may include these modules).
PduRGeneral	1	This container is a subcontainer of PduR and specifies the general configuration parameters of the PDU Router.
PduRRoutingPaths	1	Represents one table of routing paths. This routing table allows multiple configurations that can be used to create several routing tables in the same configuration. This is mainly used for post-build (e.g. post-build selectable) but can be used by pre-compile and link-time for variant handling.

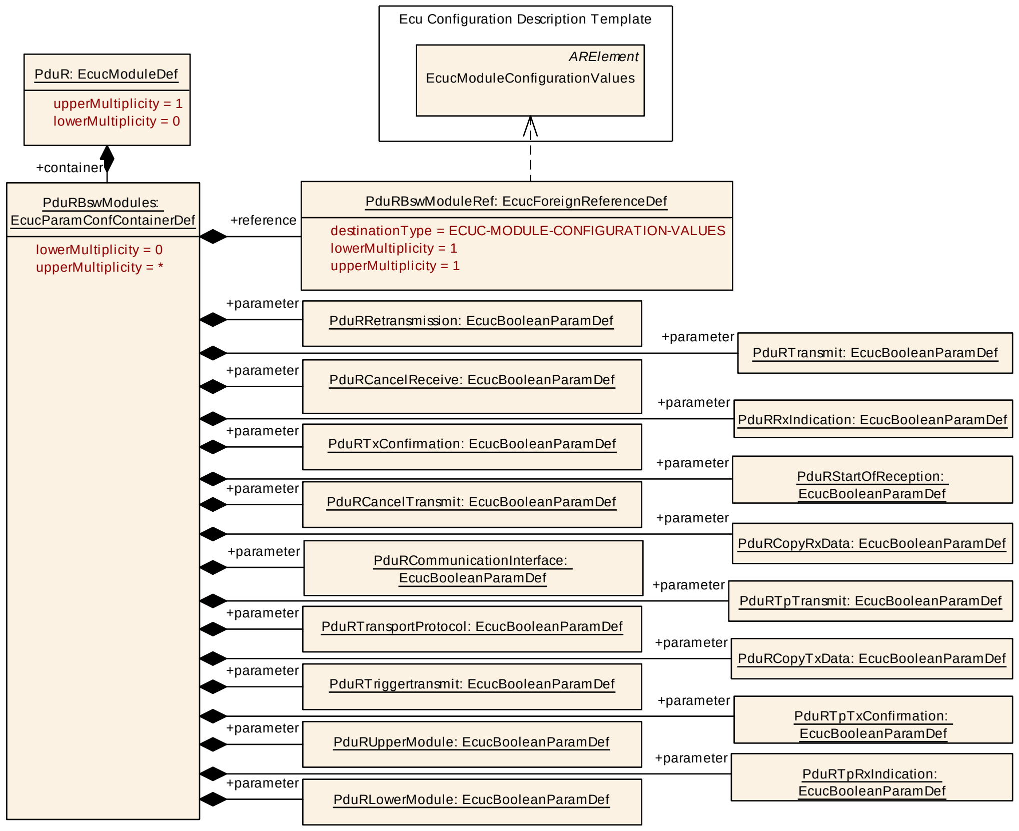
Ecu Configuration Description Template
ARElement
EcucModuleConfigurationValues
PduR: EcucModuleDef
upperMultiplicity = 1
lowerMultiplicity = 0
+container
PduRBswModules: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
PduRBswModuleRef: EcucForeignReferenceDef
destinationType = ECUC-MODULE-CONFIGURATION-VALUES
lowerMultiplicity = 1
upperMultiplicity = 1
+parameter
PduRRetransmission: EcucBooleanParamDef
PduRCancelReceive: EcucBooleanParamDef
PduRTxConfirmation: EcucBooleanParamDef
PduRCancelTransmit: EcucBooleanParamDef
PduRCommunicationInterface: EcucBooleanParamDef
PduRTransportProtocol: EcucBooleanParamDef
PduRTriggertransmit: EcucBooleanParamDef
PduRUpperModule: EcucBooleanParamDef
PduRLowerModule: EcucBooleanParamDef
PduRTransmit: EcucBooleanParamDef
PduRRxIndication: EcucBooleanParamDef
PduRStartOfReception: EcucBooleanParamDef
PduRCopyRxData: EcucBooleanParamDef
PduRTpTransmit: EcucBooleanParamDef
PduRCopyTxData: EcucBooleanParamDef
PduRTpTxConfirmation: EcucBooleanParamDef
PduRTpRxIndication: EcucBooleanParamDef

Figure 10.2: PduRBswModules

10.2.2 PduRBswModules

SWS Item	[ECUC_PduR_00295]		
Container Name	PduRBswModules		
Parent Container	PduR		
Description	Each container describes a specific BSW module (upper/CDD/lower/IpduM) that the PDU Router shall interface to. The reason to have it as own configuration container instead of implication of the routing path is to be able to configure CDDs properly and to force module's to be used in a post-build situation even though no routing is made to/from this module (future configurations may include these modules).		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	PduRCancelReceive [ECUC_PduR_00340]		
Parent Container	PduRBswModules		
Description	Specifies if the Transport protocol module supports the CancelReceive API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRCancelTransmit [ECUC_PduR_00297]		
Parent Container	PduRBswModules		
Description	Specifies if the BSW module supports the CancelTransmit API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRCommunicationInterface [ECUC_PduR_00298]		
Parent Container	PduRBswModules		
Description	Specifies if the BSW module supports the Communication Interface APIs or not. Value true the APIs are supported. A module can have both Communication Interface APIs and Transport Protocol APIs (e.g. the COM module).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRCopyRxData [ECUC_PduR_00360]		
Parent Container	PduRBswModules		
Description	Specifies if the Transport protocol module supports the CopyRxData API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRCopyTxData [ECUC_PduR_00362]		
Parent Container	PduRBswModules		
Description	Specifies if the Transport protocol module supports the CopyTxData API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRLowerModule [ECUC_PduR_00307]		
Parent Container	PduRBswModules		
Description	The PduRLowerModule will decide who will call the APIs and who will implement the APIs. For example, if the CanIf module is referenced then the PDU Router module will implement the PduR_CanIfRxIndication API. And the PDUR module will call the CanIf_Transmit API. Other APIs are of course also covered. An upper module can also be an lower module (e.g. the IpduM module).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Scope / Dependency	scope: local		

Name	PduRRetransmission [ECUC_PduR_00332]		
Parent Container	PduRBswModules		
Description	If set to true this means that the destination transport protocol module will use the retransmission feature. This parameter might be set to false if the retransmission feature is not used, even though the destination transport protocol is supporting it. This parameter is only valid for transport protocol modules and gateway operations. If transmission from a local upper layer module this module will handle the retransmission.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRRxIndication [ECUC_PduR_00358]		
Parent Container	PduRBswModules		
Description	Specifies if BSW module supports the RxIndication API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRStartOfReception [ECUC_PduR_00359]
Parent Container	PduRBswModules
Description	Specifies if the Transport protocol module supports the StartOfReception API or not. Value true the API is supported.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	
Post-Build Variant Value	false

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTpRxIndication [ECUC_PduR_00364]		
Parent Container	PduRBswModules		
Description	Specifies if the Transport protocol module supports the TpRxIndication API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTpTransmit [ECUC_PduR_00361]		
Parent Container	PduRBswModules		
Description	Specifies if BSW module supports the TP Transmit API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTpTxConfirmation [ECUC_PduR_00363]
Parent Container	PduRBswModules
Description	Specifies if the Transport protocol module supports the TpTxConfirmation API or not. Value true the API is supported.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	
Post-Build Variant Value	false

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTransmit [ECUC_PduR_00357]		
Parent Container	PduRBswModules		
Description	Specifies if BSW module supports the (IF) Transmit API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTransportProtocol [ECUC_PduR_00312]		
Parent Container	PduRBswModules		
Description	The PDU Router module shall use the API parameters specified for transport protocol interface.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRTriggertransmit [ECUC_PduR_00313]
Parent Container	PduRBswModules
Description	Specifies if the BSW module supports the TriggerTransmit API or not. Value true means that the BSW module supports the TriggerTransmit interface which a lower layer module can call and also that it can call the TriggerTransmit interface of an upper layer module. Value false means that the BSW module does not support the TriggerTransmit interface which a lower layer module can call and also that it shall not call the TriggerTransmit interface of an upper layer module.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRTxConfirmation [ECUC_PduR_00314]		
Parent Container	PduRBswModules		
Description	Specifies if the BSW module supports the TxConfirmation API or not. Value true the API is supported.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRUpperModule [ECUC_PduR_00338]		
Parent Container	PduRBswModules		
Description	The PduRUpperModule will decide who will call the APIs and who will implement the APIs. For example, if the COM module is referenced then the PDU Router module will implement the PduR_Transmit API. And the PDUR module will call the Com_RxIndication API. Other APIs are of course also covered. An upper module can also be an lower module (e.g. the IpduM module).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRBswModuleRef [ECUC_PduR_00294]		
Parent Container	PduRBswModules		
Description	This is a reference to one BSW module's configuration (i.e. not the ECUC parameter definition template). Example, there could be several configurations of LinIf and this reference selects one of them.		
Multiplicity	1		
Type	Foreign reference to ECUC-MODULE-CONFIGURATION-VALUES		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

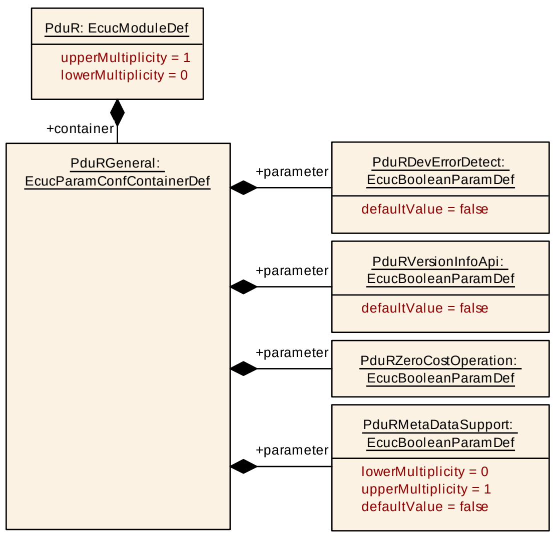

Figure 10.3: PduRGeneral

10.2.3 PduRGeneral

SWS Item	[ECUC_PduR_00305]
Container Name	PduRGeneral
Parent Container	PduR
Description	This container is a subcontainer of PduR and specifies the general configuration parameters of the PDU Router.
Configuration Parameters	

Name	PduRDevErrorDetect [ECUC_PduR_00302]		
Parent Container	PduRGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRMetaDataSupport [ECUC_PduR_00347]		
Parent Container	PduRGeneral		
Description	Enable support for MetaData handling. The MetaData is defined by the referenced MetaDataType of the global PDU definitions. This feature may be used for efficient address based routing and generic CAN-CAN-routing, where the MetaData contains the CAN ID.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRVersionInfoApi [ECUC_PduR_00316]
Parent Container	PduRGeneral
Description	If true the PduR_GetVersionInfo API is available.
Multiplicity	1
Type	EcucBooleanParamDef
Default Value	false
Post-Build Variant Value	false

Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRZeroCostOperation [ECUC_PduR_00317]		
Parent Container	PduRGeneral		
Description	If set the PduR configuration generator will report an error if zero-cost-operation cannot be fulfilled. This parameter shall be seen as an input requirement to the configuration generator.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

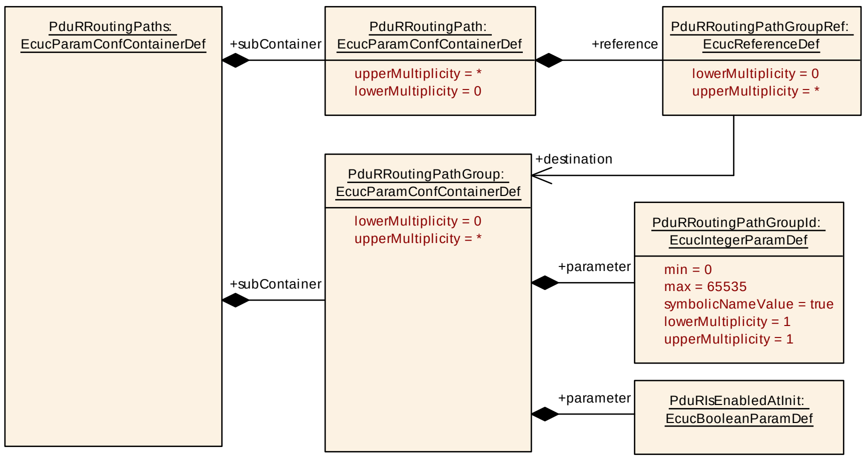

Figure 10.4: PduRRoutingPathGroup

10.2.4 PduRRoutingPathGroup

SWS Item	[ECUC_PduR_00308]
Container Name	PduRRoutingPathGroup
Parent Container	PduRRoutingPaths

Description	This container groups routing paths. By this grouping, it is possible to switch all routings related to one network, or to one kind of PDUs. PduRRoutingPaths link one source with one destination. Enabling and disabling of routing path groups is done using the PduR API.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	PduRIsEnabledAtInit [ECUC_PduR_00329]		
Parent Container	PduRRoutingPathGroup		
Description	If set to true this routing path group will be enabled after initializing the PDU Router module (i.e. enabled in the PduR_Init function).		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRRoutingPathGroupId [ECUC_PduR_00309]		
Parent Container	PduRRoutingPathGroup		
Description	Identification of the routing group. The identification will be used by the disable/enable API in the PDU Router module API.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

No Included Containers

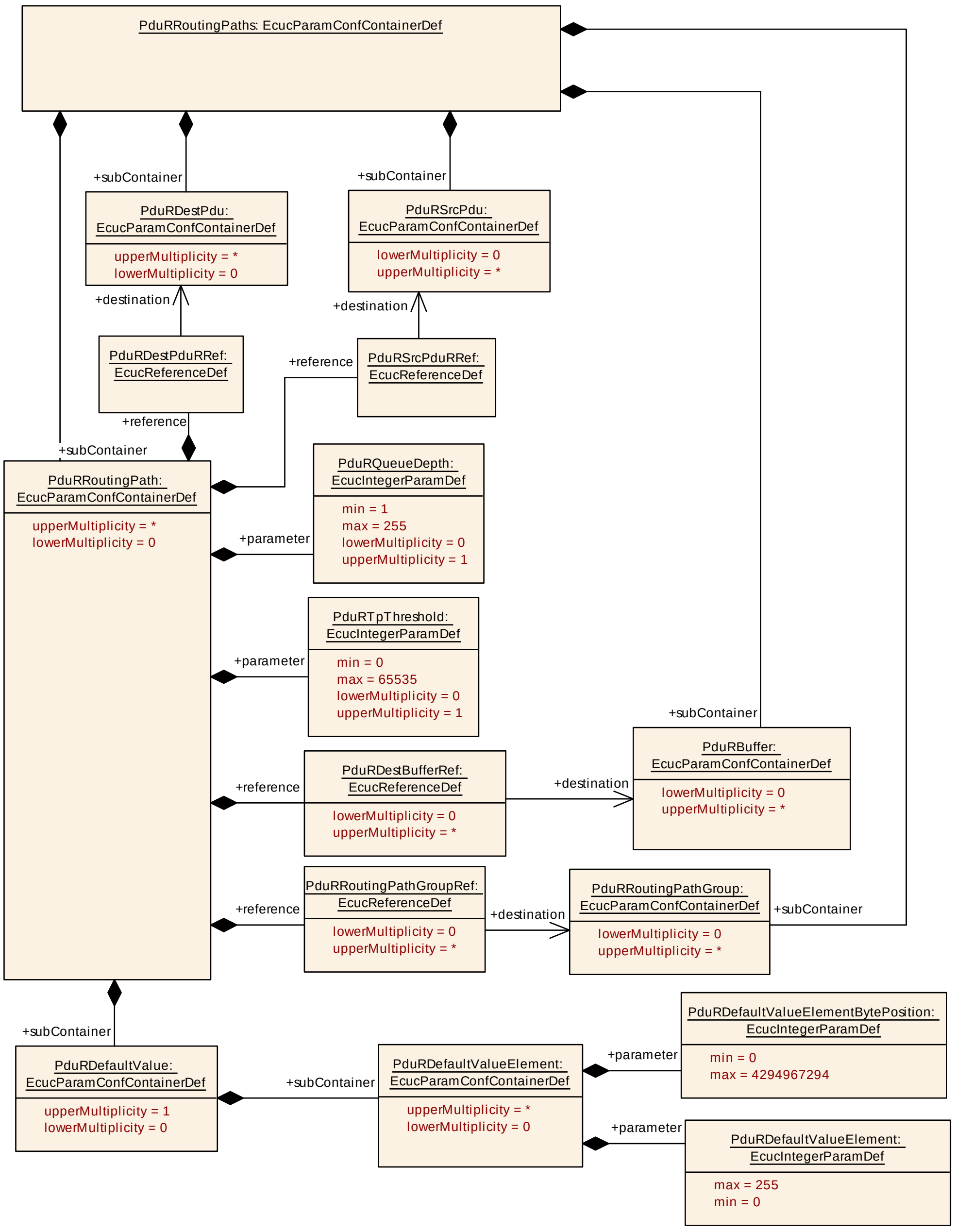

Figure 10.5: PduRRoutingPath

10.2.5 PduRRoutingPaths

SWS Item	[ECUC_PduR_00310]
Container Name	PduRRoutingPaths
Parent Container	PduR

Description	Represents one table of routing paths. This routing table allows multiple configurations that can be used to create several routing tables in the same configuration. This is mainly used for post-build (e.g. post-build selectable) but can be used by pre-compile and link-time for variant handling.
Configuration Parameters	

Name	PduRConfigurationId [ECUC_PduR_00327]		
Parent Container	PduRRoutingPaths		
Description	Identification of the configuration of the PduR configuration. This identification can be read using the PduR API.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRMaxRoutingPathCnt [ECUC_PduR_00350]		
Parent Container	PduRRoutingPaths		
Description	Maximum number of RoutingPaths in all RoutingTables. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRMaxRoutingPathGroupCnt [ECUC_PduR_00348]		
Parent Container	PduRRoutingPaths		
Description	Maximum number of RoutingPathGroups. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
PduRBuffer	0..*	Specifies a buffer used for gatewaying via communication interfaces or transport protocols, transport protocol 1:n receiving, or for fan-in reception routing for communication interface modules.
PduRDestPdu	0..*	This container is a subcontainer of PduRRoutingPath and specifies one destination for the PDU to be routed.
PduRRoutingPath	0..*	This container is a subcontainer of PduRRoutingTable and specifies the routing path of a PDU.
PduRRoutingPathGroup	0..*	This container groups routing paths. By this grouping, it is possible to switch all routings related to one network, or to one kind of PDUs. PduRRoutingPaths link one source with one destination. Enabling and disabling of routing path groups is done using the PduR API.
PduRSrcPdu	0..*	This container is a subcontainer of PduRRoutingPath and specifies the source of the PDU to be routed.

10.2.6 PduRRoutingPath

SWS Item	[ECUC_PduR_00248]
Container Name	PduRRoutingPath
Parent Container	PduRRoutingPaths
Description	This container is a subcontainer of PduRRoutingTable and specifies the routing path of a PDU.
Post-Build Variant Multiplicity	true

Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	PduRQueueDepth [ECUC_PduR_00356]		
Parent Container	PduRRoutingPath		
Description	This parameter defines the queue depth for this routing path.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRTpThreshold [ECUC_PduR_00320]		
Parent Container	PduRRoutingPath		
Description	This parameter is only relevant for TP routings. When configured, it enables on-the-fly routing and defines the number of bytes which must have been received before transmission on the destination bus may start. When omitted, direct TP routing is enforced. The PduRouter shall ensure that a buffer is allocated for this routing path which is at least as large as the threshold.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRDestBufferRef [ECUC_PduR_00304]		
Parent Container	PduRRoutingPath		
Description	Reference to a buffer in the PduR. This buffer is required for communication interface gatewaying, and for transport protocol gatewaying or for fan-in reception routing for communication interface modules.		
Multiplicity	0..*		
Type	Reference to PduRBuffer		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRDestPduRRef [ECUC_PduR_00354]		
Parent Container	PduRRoutingPath		
Description			
Multiplicity	1		
Type	Reference to PduRDestPdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Name	PduRRoutingPathGroupRef [ECUC_PduR_00352]		
Parent Container	PduRRoutingPath		
Description	Reference to routing paths.		
Multiplicity	0..*		
Type	Reference to PduRRoutingPathGroup		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRSrcPduRRef [ECUC_PduR_00353]		
Parent Container	PduRRoutingPath		
Description			
Multiplicity	1		
Type	Reference to PduRSrcPdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
PduRDefaultValue	0..1	Specifies the default value of the I-PDU. Only required for gateway operation and if at least one PDU specified by PduRDestPdu uses TriggerTransmit Data provision. Represented as an array of IntegerParamDef.

10.2.7 PduRDestPdu

SWS Item	[ECUC_PduR_00249]
Container Name	PduRDestPdu
Parent Container	PduRRoutingPaths
Description	This container is a subcontainer of PduRRoutingPath and specifies one destination for the PDU to be routed.
Post-Build Variant Multiplicity	true

Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	PduRDestPduDataProvision [ECUC_PduR_00289]		
Parent Container	PduRDestPdu		
Description	Specifies how data are provided: direct (as part of the Transmit call) or via the TriggerTransmit callback function. Only required for non-TP gatewayed I-PDUs.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	PDUR_DIRECT	The PDU Router module shall call the transmit function in the destination module and not buffer the I-PDU	
	PDUR_TRIGGERTRANSMIT	The PDU Router module shall call the transmit function in the destination module. The destination module will request the I-PDU using the triggerTransmit function. The I-PDU is shall be buffered.	
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: In case of PDUR_TRIGGERTRANSMIT the parameter PduRDestBufferRef is required.		

Name	PduRDestPduHandleId [ECUC_PduR_00322]	
Parent Container	PduRDestPdu	
Description	PDU identifier assigned by PDU Router. Used by communication interface and transport protocol modules for confirmation (PduR_<Lo>TxConfirmation) and for TriggerTransmit (PduR_<Lo>TriggerTransmit).	
Multiplicity	0..1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default Value		
Post-Build Variant Multiplicity	false	

Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRTransmissionConfirmation [ECUC_PduR_00339]		
Parent Container	PduRDestPdu		
Description	This parameter is only for communication interfaces. Transport protocol modules will always call the TxConfirmation function. If set the destination communication interface module will call the TxConfirmation. However the TxConfirmation may be not called due to error. So the PduR shall not block until the TxConfirmation is called. One background for this parameter is for the PduR to know when all modules have confirmed a multicast operation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRDestPduRef [ECUC_PduR_00291]
Parent Container	PduRDestPdu
Description	Destination PDU reference; reference to unique PDU identifier which shall be used by the PDU Router instead of the source PDU ID when calling the related function of the destination module.
Multiplicity	1
Type	Reference to Pdu
Post-Build Variant Value	true

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.8 PduRSrcPdu

SWS Item	[ECUC_PduR_00288]
Container Name	PduRSrcPdu
Parent Container	PduRRoutingPaths
Description	This container is a subcontainer of PduRRoutingPath and specifies the source of the PDU to be routed.
Configuration Parameters	

Name	PduRSourcePduBlockSize [ECUC_PduR_00355]		
Parent Container	PduRSrcPdu		
Description	Minimum amount of buffer space required by receiving transport protocol layer to continue reception.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	–	
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	PduRSourcePduHandleId [ECUC_PduR_00311]	
Parent Container	PduRSrcPdu	
Description	PDU identifier assigned by PDU Router.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 65535	
Default Value		

Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	PduRSrcPduUpTxConf [ECUC_PduR_00351]		
Parent Container	PduRSrcPdu		
Description	When enabled, the TxConfirmation will be forwarded to the upper layer. Prerequisites: Lower layer and upper layer support TxConfirmation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	true		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Name	PduRSrcPduRef [ECUC_PduR_00318]		
Parent Container	PduRSrcPdu		
Description	Source PDU reference; reference to unique PDU identifier which shall be used for the requested PDU Router operation.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.9 PduRDefaultValue

SWS Item	[ECUC_PduR_00299]
Container Name	PduRDefaultValue
Parent Container	PduRRoutingPath
Description	Specifies the default value of the I-PDU. Only required for gateway operation and if at least one PDU specified by PduRDestPdu uses TriggerTransmit Data provision. Represented as an array of IntegerParamDef.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
PduRDefaultValue Element	0..*	Each value element is represented by the element and the position in an array.

10.2.10 PduRDefaultValueElement

SWS Item	[ECUC_PduR_00300]		
Container Name	PduRDefaultValueElement		
Parent Container	PduRDefaultValue		
Description	Each value element is represented by the element and the position in an array.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	PduRDefaultValueElement [ECUC_PduR_00290]		
Parent Container	PduRDefaultValueElement		
Description	The default value consists of a number of elements. Each element is one byte long and the number of elements is specified by SduLength. The position of this parameter in the container is specified by the PduRElementBytePosition parameter.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	PduRDefaultValueElementBytePosition [ECUC_PduR_00292]		
Parent Container	PduRDefaultValueElement		
Description	This parameter specifies the byte position of the element within the default value		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967294		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.11 PduRBuffer

SWS Item	[ECUC_PduR_00336]		
Container Name	PduRBuffer		
Parent Container	PduRRoutingPaths		
Description	Specifies a buffer used for gatewaying via communication interfaces or transport protocols, transport protocol 1:n receiving, or for fan-in reception routing for communication interface modules.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE, VARIANT-POST-BUILD
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	PduRPduMaxLength [ECUC_PduR_00324]		
Parent Container	PduRBuffer		
Description	Length of the PDU buffer in bytes. This parameter limits the size of buffered routed PDUs.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Scope / Dependency	scope: local dependency: When this buffer is used for TP routing path or 1:n reception the PduRPduMaxLength has to be large enough to contain the largest possible single frame of the source network.

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 "Published Information" in [2, SWS_BSWGeneral].

11 PDU Router module design notes

This chapter collects a set of notes that describes features of this document.

11.1 Configuration parameter considerations

The configuration parameters listed in chapter 10 contain restrictions for the parameters themselves. But no restrictions are set that affects more than one parameter. The intention of this section is to list some of these to better understand the configuration parameters and also to allow a simpler configuration generator tool for the PDU Router module.

The buffers needed for gatewaying (communication interface and transport protocol) are specified per destination in the configuration. Since no specific traffic shaping can be specified it is assumed that worst case where all I-PDUs are gatewayed at the same time. It is possible to extend the configuration with parameters that allow more efficient usage of buffers.

11.2 Generic interfaces concept

The provided and used APIs of the PDU Router module are not connected to specific busses. The API names in chapter 8.3.1.3 have a generic part (<Up>, <Lo>, etc) that will be exchanged with the name of the module using or implementing the API.

The way to identify the name is using the reference to an ECUC description, see Figure 10.2. The short-name will be used in the referenced ECUC description.

The `PduRBswModules` container contain parameters that describe the supported functionality (if it is communication interface, transport layer, upper layer, lower layer, etc.) of the BSW module.

[SWS_PduR_00800] ⌈In case the lower layer module supports both TP and IF, the infixes Tp and If shall be added to the function names directly in front of the function, e.g. `<Lo>_[Tp]Transmit`, `PduR_<Lo>[If]TxConfirmation`.⌋*(SRS_GTW_06117, SRS_GTW_06121, SRS_BSW_00310)*

The connection between the generic interface configuration of a BSW module and the I-PDUs are made using the routing paths and the I-PDU configuration in the ECUC module.

11.3 Example structure of Routing tables

This chapter shows example structures of routing tables that contain the properties of each I-PDU. It does not specify the internals of the PDU Router but shall rather serve as example for better understanding of APIs and I-PDUs.

The IpduM is not considered by these examples.

Note: This chapter is by no means the recommended implementation way. The chapter focuses more on understandability than optimizing implementation.

11.3.1 Single and Multicast transmission via communication interface modules

Routing table used by PduR_ComTransmit for I-PDUs transmitted by Com:

PduR_ComTransmit (PduIdType TxPduId,const PduInfoType* `PduInfoPtr`)			
id	TargetFctPtr	TargetPduId	Description
0	CanIf_Tansmit	0	Transmission on CanIf
1	FrIf_Transmit	0	Transmission on FrIf
2	CanIf_Tansmit	1	Transmission on CanIf
3	CanIf_Transmit CanIf_Transmit	0 2	Multicast using CanIf on two CAN busses
4	LinIf_Transmit CanIf_Transmit	2 3	Multicast using CanIf and LinIf. Note that for LinIf this is a sporadic frame (will later be a TriggerTransmit call).

The first three entries represent normal PDU transmit operations from Com via CanIf or FrIf respectively, the remaining two entries are related to multicast I-PDU transmit operations from Com module to two different CAN busses and Com module to LinIf and CanIf. For the latter an internal PDU Router function (MultiIf_Transmit) and an additional routing table is used.

The destination module will confirm the transmission of the I-PDU using the configured I-PDU id, and it might not be the same as in the `<User:Lo>_Transmit` call.

11.3.2 Reception and gatewaying via communication interface modules

Routing table used by `PduR_<User:Lo>RxIndication` for receiving I-PDUs received from the lower layer communication interfaces:

`PduR_<User:Lo>RxIndication` (PduIdType `RxPduId` const PduInfoType* `PduInfoPtr`)			
id	TargetFctPtr1	TargetPduId	Description
0	Com_RxIndication	0	Routed to Com module
1	Com_RxIndication CanIf_Transmit	0 1	Routed to Com and gatewayed to CanIf
2	CanIf_Transmit LIN	1 2	Gatewayed to CanIf and to LinIf. In the LinIf case the LinIf will later call Trigger Transmit. The PDU Router ill not call LinIf_Transmit

11.4 Configuration generator

The PDU Router configuration generator will take the ECU configuration description XML file containing the PDU Router configuration as input. And the generator will produce `.c` and `.h` files containing the configuration.

One aim of the configuration generator is to allow the generator to produce an efficient PDU Router module implementation. Since the PDU Router module is a central module it is important that the final executable including configuration be as efficient as possible:

[SWS_PduR_00764] ⌈The PDU Router module generator shall be able to optimize away features based on if they are used or not. At least following features shall be considered:

- Transport protocol
- Communication interfaces
- Gateway
- FIFO queue handling

⌋*(SRS_GTW_06020)*

One part of the job made by the generator is to lookup all routing paths and produces the correct look-up tables and the correct APIs to be used. Here are some examples how the generator may handle the routing paths.

11.4.1 CanIf and Com routing path example

This is an example that shows how an I-PDU received by the CanIf module and forwarded by the Com module is handled.

In Figure 11.1 the configuration of CanIf, Com and PDU Router is shown. The PDU Router has a routing path with a source I-PDU `PduRSrcPduRef` and destination I-PDU `PduRDestPduRef`. When following the I-PDU `PduRSrcPduRef` it is found that the CanIf `PduIdRef` is pointing at the same I-PDU in the ECUC. The `PduRDestPduRef` is followed and it is found that Com `PduIdRef` is pointing at the same I-PDU in the ECUC.

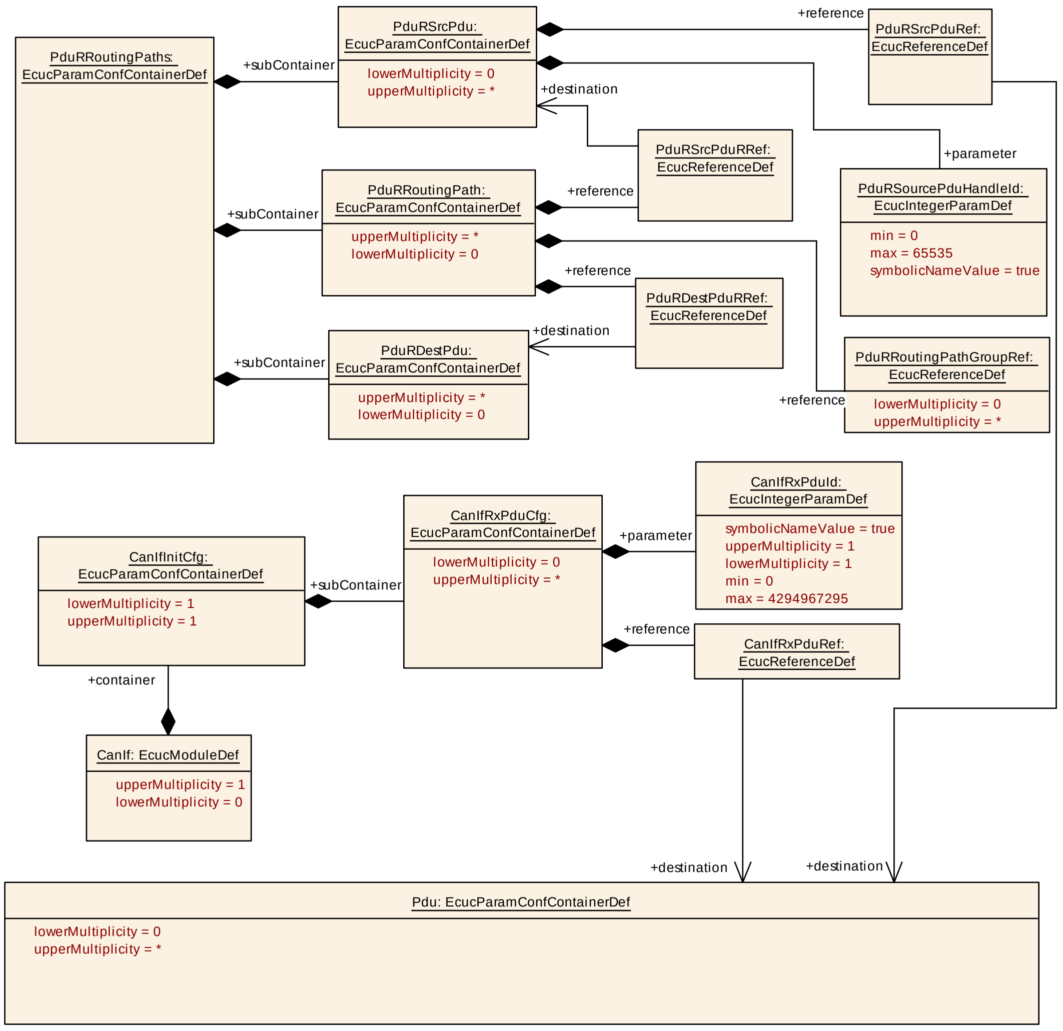

Figure 11.1: PDU Router, CanIf and Com configuration example

The `CanIfCanRxPduId` reveals the I-PDU ID for the source I-PDU and the `ComIPdu HandleId` reveals the I-PDU ID for the destination I-PDU.

The shortname of the CanIf module and the Com module (and that the I-PDU is transported on a communication interface module) will generate the routing table and APIs to be used:

`PduR_<User:Lo>RxIndication` (PduIdType `RxPduId` const PduInfoType* `PduInfoPtr`)			
id	Source	TargetPduId	Destination
12	PduR_CanIfRxIndication	13	Com_RxIndication

`PduR_CanIf.h`

```
void PduR_CanIfRxIndication(PduIdType RxPduId const PduInfo
Type* PduInfoPtr);
```

If `PduRZeroCostOperation` is enabled and the CanIf module only forwards (through PDU Router module) to the Com module, the PduR generator may optimize the generated code (if source code is used):

```
#define PduR_CanIfRxIndication Com_RxIndication
```

11.5 Post-build considerations

This section describes some important behavior when using the post-build variant of the PDU Router. It contains no requirements, just important issues that need to be considered.

NVRAM and RAM memory size can potentially grow if a new post-build configuration is downloaded into the ECU. Estimation at design time must be done to allow such grow so other areas are not overwritten (in case of RAM) or memory borders are not crossed.

It is not possible to configure restrictions/locations/etc of memory in the PduR module configuration since this is implementation specific and relitevly difficult to implement (pre-compile and link-time does not really need this). It is recommended for implementations of PduR module generators to extend the configuration with specific memory constraints if needed.

A Not applicable requirements

[SWS_PduR_00777] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00170, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00437, SRS_BSW_00168, SRS_BSW_00425, SRS_BSW_00432, SRS_BSW_00336, SRS_BSW_00417, SRS_BSW_00386, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00164, SRS_BSW_00343, SRS_BSW_00160, SRS_BSW_00413, SRS_BSW_00335, SRS_BSW_00447, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00439, SRS_BSW_00449, SRS_BSW_00357, SRS_BSW_00172, SRS_BSW_00341, SRS_BSW_00334, SRS_GTW_06055, SRS_GTW_06056, SRS_GTW_06061, SRS_GTW_06098, SRS_GTW_06099, SRS_GTW_06077, SRS_GTW_06064, SRS_GTW_06089)*