RaSTA Protocol Reference Stack

User Manual

Project	:	RaSTA Protocol Reference Stack	Customer	:	SBB
Author	:	Roland Schenk	Version	:	1
Document-Id	:	SBB-RaSTA-084	Status	:	Release
Date	:	20.12.2022	Filename	:	sbb-rasta-084-usermanual.docx

Version history

Version	Date	Author	Changes/Comments
1	20.12.2022	R. Schenk	First release

Contents

List of Figures

0.1 Referenced documents

0.1.1 Compliance document

[1] Elektrische Bahn-Signalanlagen - Teil 200: *Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159* (VDE0831-159); DIN; Juni 2015

[2] Electric signalling systems for railways - Part 200: *Safe transmission protocol according to DIN EN 50159* (VDE 0831-159); English translation of DIN VDE V 0831-200 (VDE V 0831-200):2015-06; DIN; June 2015

[3] *EN 50128:2011 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems*; CENELEC; June 2011

[4] *EN 50129:2018 Railway applications - Communication, signalling and processing systems - Safety-related electronic systems for signalling;* CENELEC; November 2018

[5] *EN 50159:2010 Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems;* CENELEC; September 2010

[6] *EULYNX Baseline Set 4 Release 1 Cover document*; EULYNX; 20220519; Version 1A

0.1.2 Guidance documents

-

0.1.3 Internal documents

All internal documents are listed in the *Document Plan* [7] with their valid version and author.

[7] *Document Plan*, CSA Engineering AG; SBB-RaSTA-008

[8] *Glossary*; CSA Engineering AG; SBB-RaSTA-009

[9] *Software Requirements Specification*; CSA Engineering AG; SBB-RaSTA-012

[10] *Software Architecture and Design Specification*; CSA Engineering AG; SBB-RaSTA-018

[11] *Software Interface Specification*; CSA Engineering AG; SBB-RaSTA-020

[12] *Overall Software Test Case Specification*; SBB-RaSTA-057

0.2 Abbreviation and Terms

Common terms and abbreviations are defined in the overall project glossary *Glossary* [8].

0.3 Scope

This document contains the user manual for the RaSTA Protocol Reference Stack developed in the course of the project called RaSTA Protocol Reference Stack.

The name of the SW under Consideration (SWuC) is RaSTA Protocol Reference Stack SW library.

0.4 Target audience

Target system developers, target system integrators as well as the final user (operator / maintainer) of the SWuC are the primary audience of this paper. The secondary audience are all persons interested in the aspects related to the SWuC developed according to SIL4 requirements.

0.5 Conventions

There are no special conventions.

1 Introduction

This document is the user manual for the RaSTA Protocol Reference Stack SW library.

The RaSTA Protocol Reference Stack SW library has the purpose of realizing a safe and reliable communication protocol for railway signalling in accordance with the generic RaSTA protocol described in *Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159* [1]. The English terms are taken from the English translation of this pre-standard *Safe transmission protocol according to DIN EN 50159* [2].

The RaSTA Protocol Reference Stack SW library is designed to run in the *EULYNX Baseline Set 4 Release 1* [6] environment on an object controller with a partitioned single thread operating system.

1.1 RaSTA overview

1.1.1 RaSTA network

Figure 1 shows an example of a RaSTA network, which consists of three RaSTA instances (A, B, C). The instance A has two RaSTA connections, one to instance B (connection A-B) and one to instance C (connection A-C). To provide this two RaSTA connections, RaSTA instance A has also two redundancy channels. The redundancy channel A1, which uses the redundant transport channels 1 and 2, is used for the connection A-B. And the redundancy channel A2, which uses the redundant transport channels 3 and 4, is used for the connection A-C.

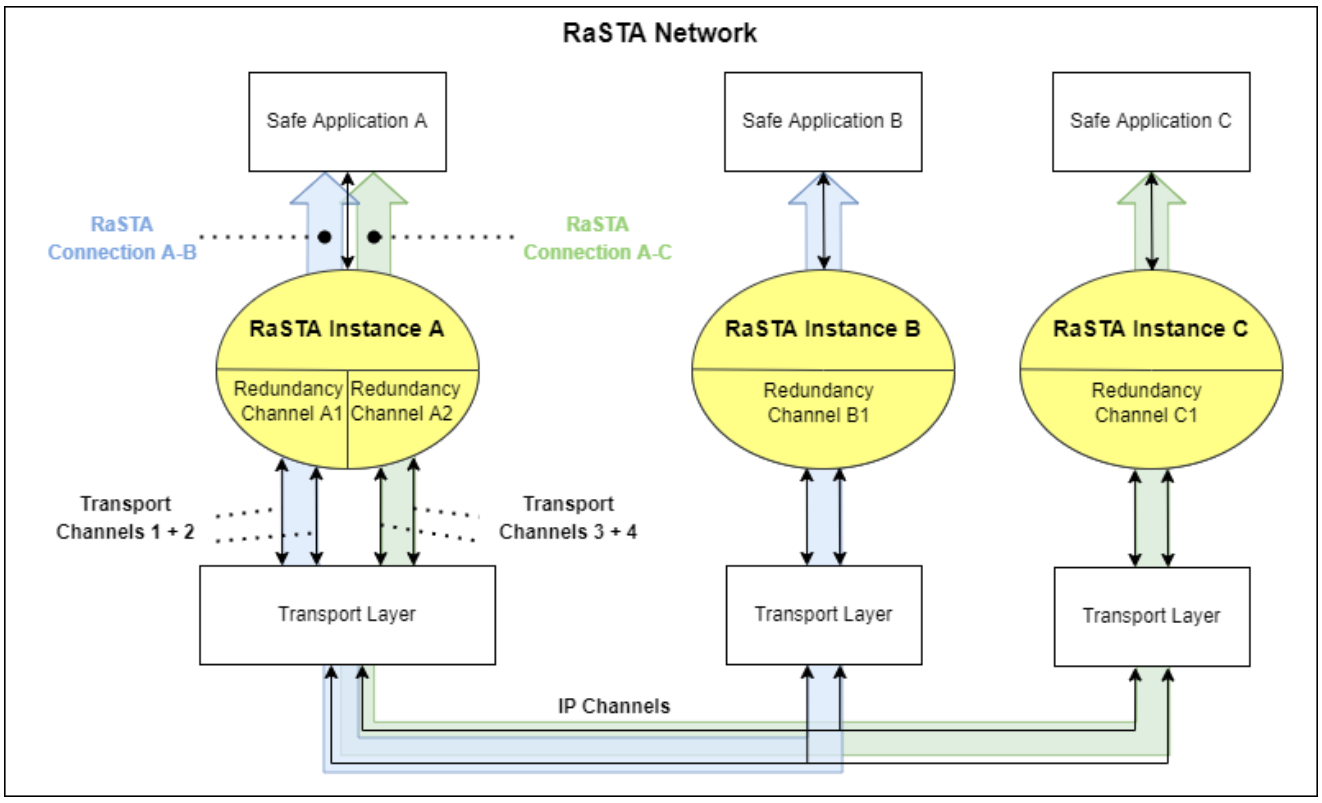

Figure 1: RaSTA Network/Connection/Instance Overview

All RaSTA instances in the same RaSTA network use the same network ID and the same type and initial value of the safety code.

1.1.1.1 RaSTA instance

A RaSTA instance is one unit of the SWuC, consisting of a safety and retransmission layer and a redundancy layer (ex. RaSTA Instance A).

1.1.1.2 RaSTA connection

A RaSTA connection is a point-to-point data transfer connection with an unambiguous pair of sender ID and receiver ID with a defined connection ID.

A RaSTA connection provides the following features:

- Timeliness of message transmission
- Connection monitoring using heartbeats
- Retransmission of missing messages
- Data integrity of received messages using an MD4 safety code

1.1.1.3 Redundancy channel

Redundancy channels are used to improve the availability of the RaSTA network by using redundant transport channels. A redundancy channel consists of own or two transport channels. With one transport channel the data transmission is possible, but there is no redundant transmission.

The redundancy channels are identified by a redundancy channel ID. Each RaSTA connection uses a dedicated redundancy channel with the redundancy channel ID equal to the ID of the associated RaSTA connection.

1.1.1.4 Transport channel

The transport channels are used by the redundancy channels for a redundant transport of messages from one RaSTA instance to another. In *EULYNX Baseline Set 4 Release 1* [6] the use of TCP/IP and TLS is possible. In this case a transport channel can be understood as an IP socket pair, which transfers data from a defined source address to a defined destination address and vice versa.

1.1.2 RaSTA stack overview

Figure 2 shows an example of a RaSTA instance on a EULYNX object controller HW. The following SW elements are identified:

- The EULYNX OC application which includes the EULYNX application configuration
- The safety and retransmission layer which includes safety and retransmission layer configuration
- The safety and retransmission layer adapter, which passes data and events from the SIL4 part to the SIL0 part of the system and vice versa, is divided into two independent sections. One section is developed according to SIL4 requirements and runs in the SIL4 part. The other section is developed according to SIL0 requirements and runs in the SIL0 part. These two sections communicate via operating system specific ports. Therefore, the two sections of the safety and retransmission layer adapter must be implemented by the integrator for the respective operating system. The requirements for the safety and retransmission layer adapter are defined in the SRAC.
- The redundancy layer which includes the redundancy layer configuration
- The transport layer which includes the transport layer configuration
- The system adapter which provides access to operating system functions (ex. system time counter, fatal error handler, etc.)

The yellow marked areas are part of the SWuC.

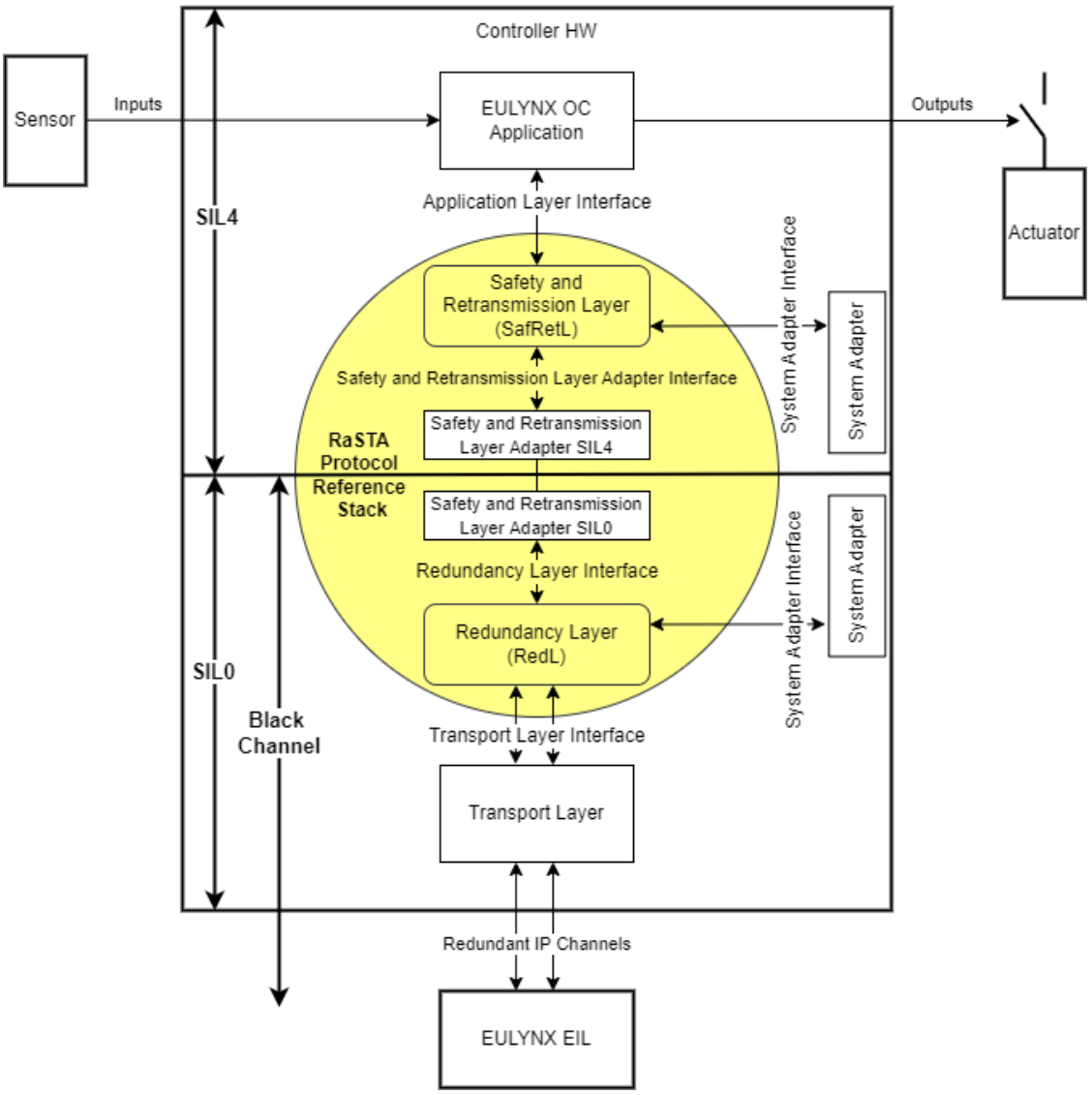

Figure 2: RaSTA Instance Overview

1.1.3 RaSTA stack layers

The SWuC contains the following layers the Safety and retransmission layer (SafRetL) and Redundancy layer (RedL).

1.1.3.1 Safety and retransmission layer

The purpose of this communication layer is to provide a safe communication mechanism in accordance with *EN 50159* [5] for networks of Category 1 and Category 2. In addition to that, it offers the retransmission of lost or distorted data within a defined period of time [2].

Therefore the SafRetL is developed as SIL4 SW according to *EN 50128* [3].

1.1.3.2 Redundancy layer

The redundancy layer provides a highly available communications service in the form of redundancy channels. A redundancy channel is implemented by using one or more transport channels. The transport channels are used to transmit information via a transport service. When using several transport channels, information which is lost or distorted on only one of the transport channels has no effect on the communication [2].

The RedL only improves the availability of the connection and is not safety relevant. Therefore it is developed as SIL0 SW according to *EN 50128* [3].

1.1.4 RaSTA messages

Figure 1 shows the different messages used in the different layers of the SWuC.

If a message is sent, the application message is passed form the EULYNX OC application to the SafRetL.

The SafRetL takes the application message as payload data and extends it with the SafRetL PDU message header and the safety code to a SafRetL PDU message. This message is passed to the RedL. The SafRetL PDU message header contains the message size, type, sender and receiver ID, sequence numbers and timestamps. The MD4 safety code is used to ensure the whole message data integrity.

The RedL takes the SafRetL PDU message as payload data and extends it with the RedL PDU message header and the optional check code to the RedL PDU message. This message is passed to the transport layer. The RedL PDU message header contains the message size and the RedL sequence number. The optional CRC check code can be used in non-safe networks as minimal data integrity check.

If a message is received, it passes the SWuC layers bottom-up.

The protocol data of the RaSTA messages are always transmitted in the little-endian format. Exempt from this rule is the data type “byte array” e.g., application message.

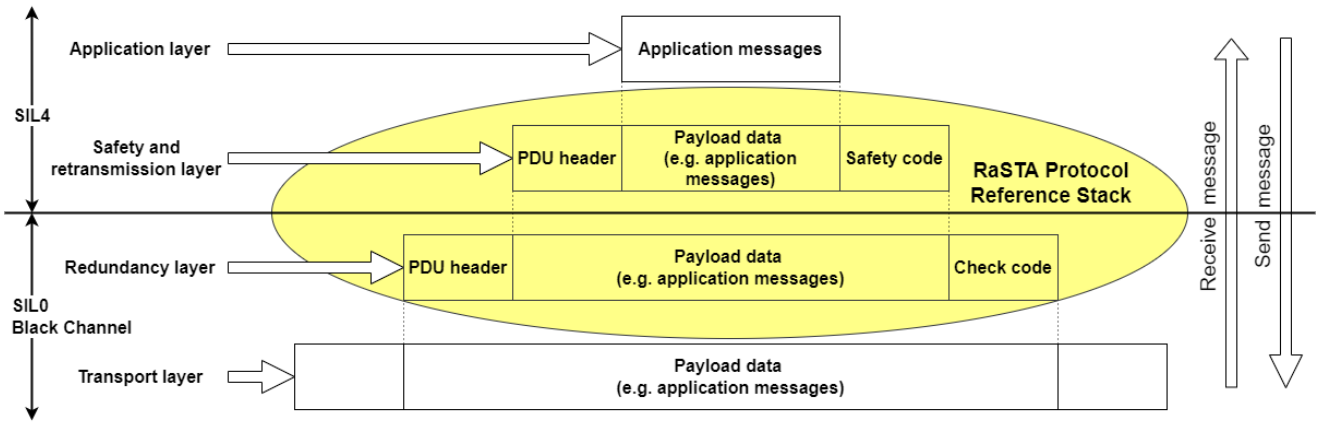

Figure 3: Overview of message frames for the different layers

2 RaSTA Protocol Reference Stack SW architecture

In Figure 4 the package diagram and the dependencies between the packages are shown. The packages of the SWuC are highlighted in yellow. The SWuC is split into packages for the two layers and a common package. The common package contains functionalities and definitions that are used by both layers. Packages of the SWuC:

- rasta_safety_retransmission
- rasta_redundancy
- rasta_common

The other packages are not part of the SWuC and must be implemented by the target system integrator.

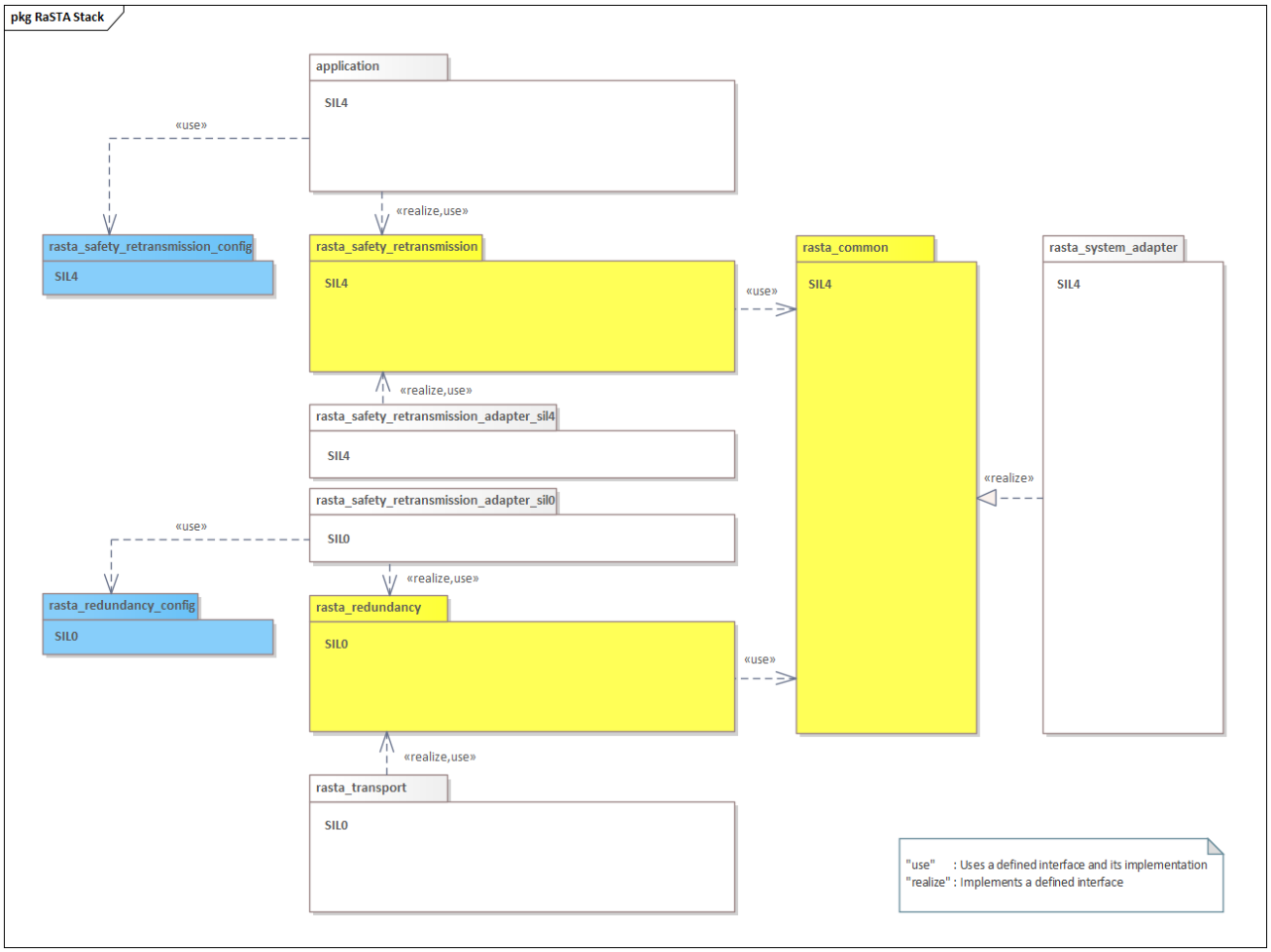

Figure 4: SWuC package diagram

For details about SW architecture and SW design see *Software Architecture and Design Specification* [10]. For details about SW interfaces see *Software Interface Specification* [11].

2.1 RaSTA common package

The RaSTA common package contains definitions and functions, which are used in both, the SafRetL and the RedL. Therefore it is developed as SIL4 SW according to *EN 50128* [3].

The RaSTA common package contains the following public interfaces:

- raas_rasta_assert.h: Assert functions
- radef_rasta_definitions.h: Global definitions
- rahlp_rasta_helper.h: Helper functions

- ralog_rasta_logger.h: Debug logger to stdout which is activated in the debug build only
- rasys_rasta_system_adapter.h: System adapter interface which has to be implemented by the target system designer.

The source code and the software component design specification of the RaSTA common package is located in the RaSTA_Component_SIL4_Source_Code_Vx.y.z.zip.

2.2 RaSTA safety and retransmission package

The SafRetL package contains the implementation of the SafRetL and is developed as SIL4 SW according to *EN 50128* [3].

The SafRetL package contains the following public interfaces:

- sradin_sr_adapter_interface.h: SafRetL adapter interface which has to be implemented by the target system designer.
- sradno_sr_adapter_notifications.h: SafRetL adapter notifications to the SafRetL
- srapi_sr_api.h: SafRetL interface to the application
- sraty_sr_api_types.h: SafRetL application interface data type definitions
- srcty_sr_config_types.h: SafRetL configuration data type definitions
- srnot_sr_notifications.h: SafRetL notifications interface to the application which has to be implemented by the target system designer.

The source code and the software component design specification of the SafRetL package is located in the RaSTA_Component_SIL4_Source_Code_Vx.y.z.zip.

2.3 RaSTA redundancy layer package

The RedL package contains the implementation of the RedL and is developed as SIL0 SW according to *EN 50128* [3].

The RedL package contains the following public interfaces:

- redtri_transport_interface.h: Transport layer interface which has to be implemented by the target system designer.
- redtrn_transport_notifications.h: Transport layer notifications to the RedL
- redint_red_interface.h: RedL interface to the SafRetL adapter
- sraty_sr_api_types.h: RedL interface data type definitions
- redcty_red_config_types.h: RedL configuration data type definitions
- rednot_red_notifications.h: RedL notifications interface to the SafRetL adapter which has to be implemented by the target system designer.

The source code and the software component design specification of the RedL package is located in the RaSTA_Component_Basic_Integrity_Source_Code_Vx.y.z.zip.

3 Target system requirements

3.1 Target system hardware requirements

The target HW has to provide a dual electronic structure based on composite fail-safety with fail-safe comparison according to *EN 50129* [4] to run the SIL4 parts of the SW.

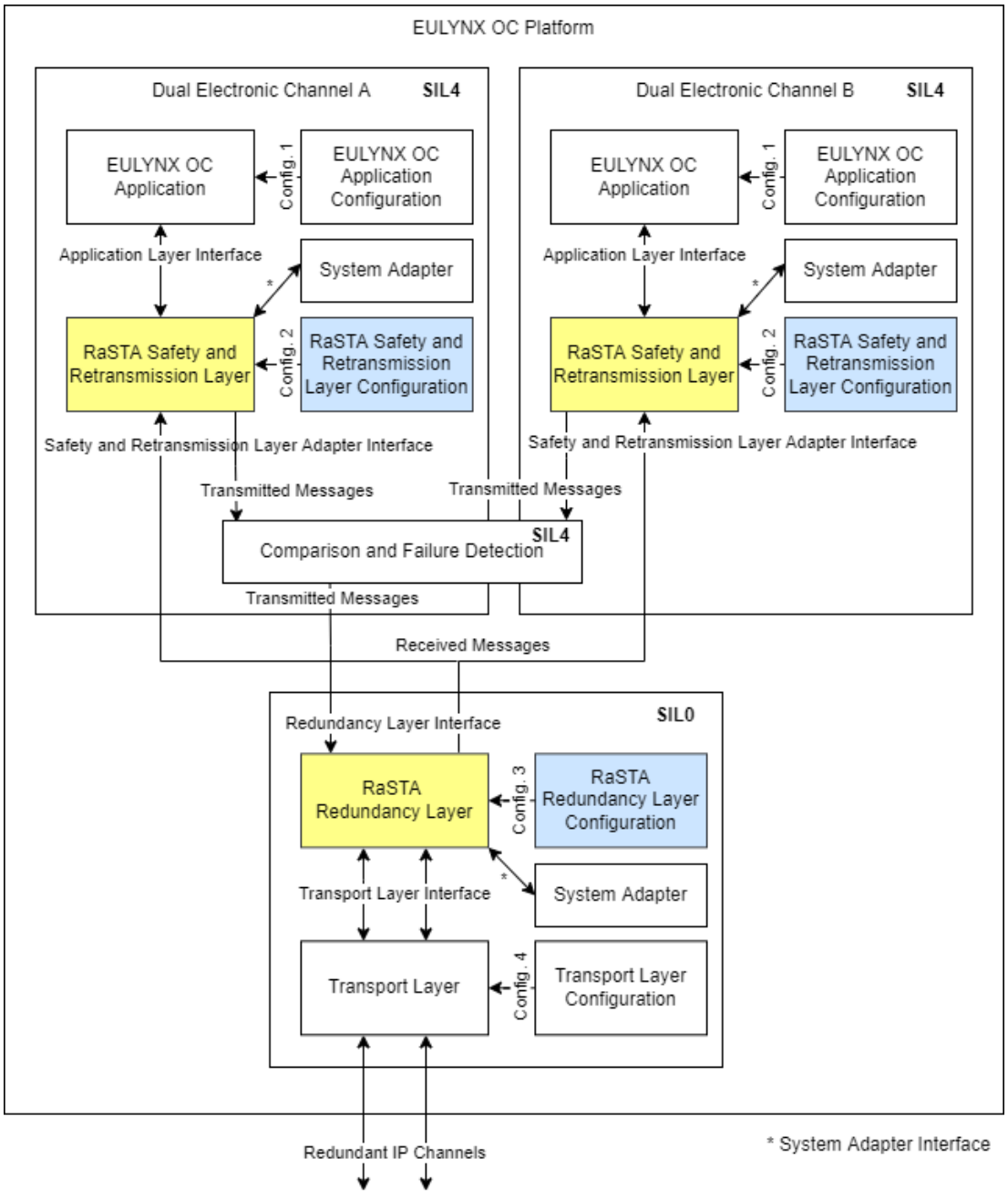

Figure 5: System architecture overview

3.1.1 Memory requirements

The memory requirements of the SWuC are dependent on the target system, the target system compiler and the target system compiler options.

A rough of estimation of the memory usage of the SWuC is calculated from a build with the GCC compiler without optimization on a i368 32bit platform:

- Common package
  - Section .text: 1’252 byte
  - Section .data: 0 bytes
  - Section .bss: 224 bytes
  - Section .rdata: 68 bytes

- Safety and retransmission package
  - Section .text: 39’040 bytes
  - Section .data: 0 bytes
  - Section .bss: 121’520 bytes
  - Section .rdata: 3984 bytes

- Redundancy package
  - Section .text: 18’656 bytes
  - Section .data: 4 bytes
  - Section .bss: 96’520 bytes
  - Section .rdata: 2’596 bytes

Take in mind that the common package is used in the SafRetL in the SIL4 partition and in the RedL in the SIL0 partition.

For the stack usage of the SafRetL see chapter 4.2.2.1 and for the RedL see chapter 4.4.2.1.

The SWuC does not use any heap memory.

3.1.2 Performance requirements

The performance requirements are dependent on the OC application requirements. The SWuC also has some timing requirements which must be feasible with the target system performance:

- SafRetL:
  - Tmax: Maximum accepted age of a message: Default value 1000ms
  - Th: Heartbeat interval: Default value 300ms
- RedL:
  - Tseq: Defer time: Default value 50ms

3.1.3 Endianness

The SWuC is designed to run on little- and big-endian systems.

3.2 Target operating system requirements

The SWuC is designed to run on a target operating system which provides the following features:

- Suitable to run on a dual electronic structure based on composite fail-safety with fail-safe comparison according to *EN 50129* [4]
- SIL4 approbation according to *EN 50128* [3]
- Partitioned single thread operating system without interrupts

The target operating system has to provide the following functions to the SWuC:

- Fatal Error Function
- Get Random Number Function
- Get Timer Granularity Function
- Get Timer Value Function

The requirements and interfaces of these functions are defined in the system adapter interface. For details see *Software Interface Specification* [11].

3.3 Target system compiler

The SWuC is implemented in C99 programming language without the use of any compiler specific extensions. The target system designer has to choose a target system compiler for C99 which is validated as T3 tool according to *EN 50128* [3].

4 Integration

The target system designer and the target system integrator have to do the integration of SWuC on the target system. One part is the integration of the SafRetL and the RedL provided by the SWuC and the other part is the development of the surrounding SW parts.

The SWuC provides the following SW parts to integrate:

- SafRetL (SIL4)
- RedL (SIL0)

The target system designer has to develop the following SW parts:

- EULYNX OC application (SIL4)
- SafRetL Adapter SIL4 part (SIL4)
- SafRetL Adapter SIL0 part (SIL0)
- Transport layer
- System adapter (SIL4)

4.1 OC application

The EULYNX OC application has to provide the main functionality of the OC. It must be implemented as SIL4 SW according to *EN 50128* [3] and it must be placed in a SIL4 partition of the target system.

4.1.1 Init

At the start-up of the OC application the srapi_Init() function has to be called with a pointer to the SafRetL configuration, to make the SafRetL ready for operation.

4.1.2 Open connection

To open a RaSTA connection the OC application has to call the srapi_OpenConnection() function.

If the OC application acts as server in a RaSTA connection, it has to open the connection immediately after the start-up of the target system or for reconnection.

If the OC application acts as client in a RaSTA connection, it has to try to open the connection periodically until it is established.

The roles of the client and server in a particular connection are defined by the value of the receiver and sender identifications: The communication party with the higher value is the server, the one with the lower value the client of a connection [2].

4.1.3 Connection state notification

A changed connection state is notified from the SafRetL to the OC application by calling the srnot_ConnectionStateNotification() function, which must be implemented in the OC application.

4.1.4 Send / receive messages

The OC application has to define the payload data of sent application messages and has to interpret the payload data of received messages.

To send messages with the srapi_SendData() function, the RaSTA connection must be in the state “up”. The current connection state can be read with the srapi_GetConnectionState() function.

A received message is notified from the SafRetL by calling the srnot_MessageReceivedNotification() function which must be implemented in the OC application.

The received message payload data can be read with the srapi_ReadData() function.

4.1.5 Check timings

The OC application must call the srapi_CheckTimings() function periodically in an appropriate interval related to the configured connection timings. This is mandatory to ensure the correct operation of timely message transmission and connection monitoring.

4.1.6 Diagnostic notifications

New diagnostic data from the SafRetL is notified to the OC application by calling the srnot_SrDiagnosticNotification() function, which must be implemented in the OC application.

New diagnostic data from the RedL is notified to the OC application by calling the srnot_RedDiagnosticNotification() function, which must be implemented in the OC application.

4.1.7 Close connection

The OC application can close an open RaSTA connection by calling the srapi_CloseConnection() function.

4.2 Safety and retransmission layer

The SafRetL provides the main functionality of the SWuC. It is implemented as SIL4 SW according to *EN 50128* [3] and it must be placed in a SIL4 partition of the target system.

4.2.1 Connection ID

The RaSTA connection ID has the purpose to uniquely identify a RaSTA connection on the SafRetL interface.

The srapi_OpenConnection() function returns the connection ID of the desired connection, as it is defined in SafRetL configuration data.

4.2.2 Memory usage

For static memory usage see chapter 3.1.1

4.2.2.1 Stack usage

An appropriate minimum stack size for the SIL4 partition of the target system where the SafRetL is running must be defined.

A rough estimation of the maximum stack usage of the SafRetL is calculated by the pc-lint tool with the assumption that a bare function call puts 8 bytes on the stack and an external called function puts 32 bytes on the stack. This estimation showed a maximum stack usage of 437 bytes for the SafRetL.

The real stack usage is dependent on the target system and the compiler used on the target system. Take also in mind to add the stack usage of other software, like the SafRetL adapter and OC application, running in the same SIL4 partition.

4.2.2.2 Heap usage

The SafRetL does not use any heap memory.

4.3 Safety and retransmission layer adapter

The SafRetL adapter must be implemented by the target system designer. It has the purpose to forward sent messages and actions from the SafRetL in the SIL4 partition to the RedL in the SIL0 partition and vice versa, to forward received messages and notifications from the RedL in the SIL0 partition to the SafRetL in the SIL4 partition.

The implementation of the SafRetL adapter must be divided into a SIL4 part and a SIL0 part. The two parts must communicate with suitable operating system functions (e.g. ports) over the SIL4 - SIL0 partition boundary.

The SIL4 part of the SafRetL adapter must be developed as SIL4 software according to *EN 50128* [3].

Figure 6 shows an example of the SW layering of the integration of the SWuC on a partitioned operating system using ports for the communication between the SIL4 and the SIL0 partitions.

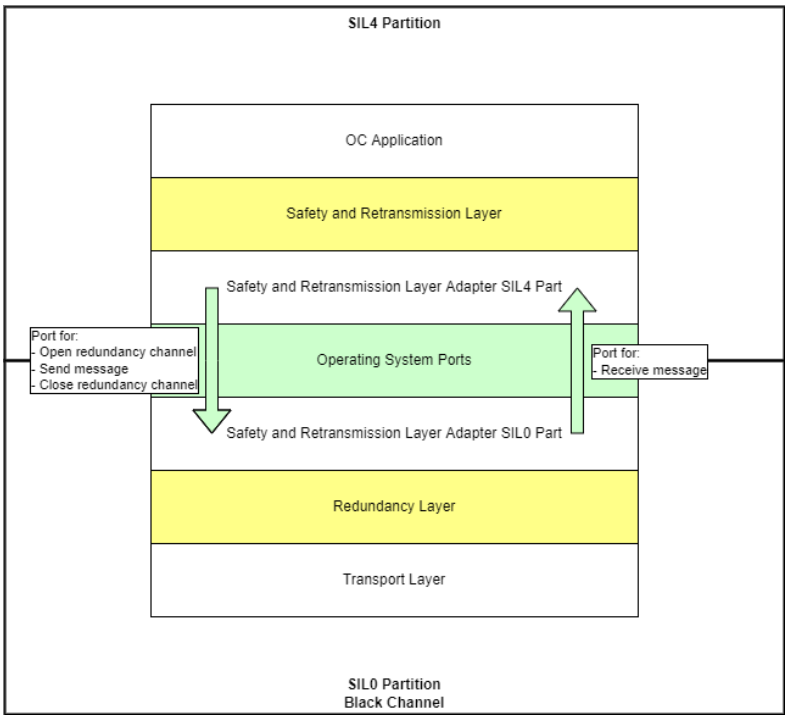

Figure 6: SafRetL operating system ports

4.3.1 Init

The sradin_Init() function has to initialize the SafRetL adapter SIL4 part to make it ready for the use by the RaSTA reference stack. The SafRetL adapter SIL0 part also needs an initialization function, which calls the RedL redint_Init() function with a pointer to the RedL configuration data and makes the SafRetL adapter SIL0 part ready for operation.

These two initialization functions must be called immediately after the start-up of the operating system to make the SafRetL adapter and the RedL ready for operation.

4.3.2 Open redundancy channel

The sradin_OpenRedundancyChannel() function has to forward the open redundancy channel action from the SafRetL in the SIL4 partition to the RedL redint_OpenRedundancyChannel() function in the SIL0 partition.

4.3.3 Close redundancy channel

The sradin_CloseRedundancyChannel() function has to forward the close redundancy channel action from the SafRetL in the SIL4 partition to the RedL redint_CloseRedundancyChannel() function in the SIL0 partition.

4.3.4 Send message

The sradin_SendMessage() function has to forward a message to send of a RaSTA connection from the SafRetL in the SIL4 partition to the RedL redint_SendMessage() function in the SIL0 partition.

This function gets called up to RADEF_SEND_BUFFER_SIZE times for one RaSTA connection in one program execution cycle. Therefore the SafRetL adapter has to provide a send messages buffer for each redundancy channel with a size of at least RADEF_SEND_BUFFER_SIZE messages.

The constant definition RADEF_SEND_BUFFER_SIZE defines the number of send buffer entries and is set to 20.

4.3.5 Read message

The sradin_ReadMessagefunction() has to read a received message form the RedL redint_ReadMessage() function in the SIL0 partition and has to forward it as output parameter of the SafRetL Adapter sradin_ReadMessagefunction() in the SIL4 partition.

4.3.6 Check timings

The SIL0 part of the SafRetL adapter has to call the redint_CheckTimings() function, to ensure the proper operation of the RedL defer queue, in appropriate interval related to the configured Tseq timing.

4.3.7 Redundancy layer notifications

The RedL notifications must be implemented by the target system designer. They have the purpose to forward notifications from the RedL in the SIL0 partition to the SafRetL in the SIL4 partition.

The implementation of the RedL notifications must be divided into a SIL0 part and a SIL4 part. The two parts must communicate with suitable operating system functions (e.g. ports) over the SIL0 - SIL4 partition boundary.

The SIL4 part of the RedL notifications must be developed as SIL4 software according to *EN 50128* [3].

Figure 7 shows an example of the SW layering of the integration of the SWuC on a partitioned operating system using a port to forward the notifications from the SIL0 partition to the SIL4 partition.

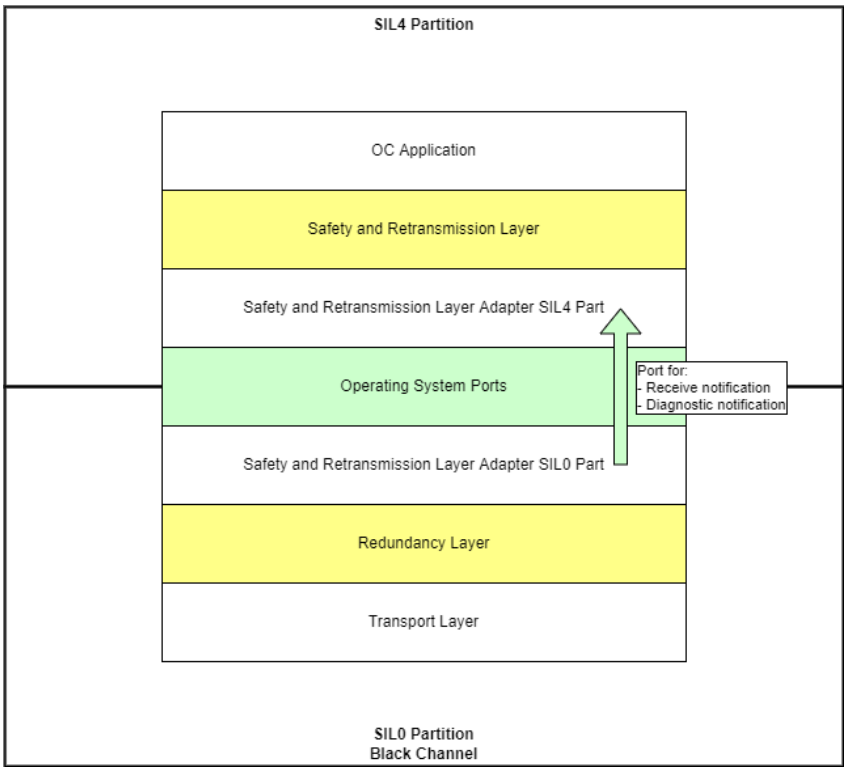

Figure 7: Redundancy notification operating system port

4.3.7.1 Message received notification

The srnot_MessageReceivedNotification() function has to forward the message received notification from the RedL in the SIL0 partition to the message received notification function sradno_MessageReceivedNotification() of SafRetL adapter in the SIL4 partition.

This function gets called up to RADEF_MAX_DEFER_QUEUE_SIZE +1 (=11) times per redundancy channel in one program execution cycle.

The constant definition RADEF_MAX_DEFER_QUEUE_SIZE defines the maximum defer queue size and is set to 10.

4.3.7.2 Diagnostic notification

The rednot_DiagnosticNotification() function has to forward the diagnostic notification from the RedL in the SIL0 partition to the diagnostic notification function sradno_DiagnosticNotification() of SafRetL adapter in the SIL4 partition.

This function is called only at the end of a RedL diagnostic window (every 10 to every 1000 messages).

4.4 Redundancy layer

The RedL provides the highly available communications service functionality of the SWuC and it contains no safety related functionality. Therefore it is implemented as SIL0 SW according to *EN 50128* [3] and it must be placed in a SIL0 partition of the target system.

4.4.1 Redundancy channel ID

The redundancy channel ID has the purpose to uniquely identify a redundancy channel on the RedL interface.

The value of a redundancy channel ID must be equal to the value of the associated RaSTA connection ID. The configuration data of the SafRetL and the RedL have to be aligned accordingly.

4.4.2 Memory usage

For static memory usage see chapter 3.1.1

4.4.2.1 Stack usage

An appropriate minimum stack size for the SIL0 partition of the target system where the RedL is running must be defined.

A rough estimation of the maximum stack usage of the RedL is calculated by the pc-lint tool with the assumption that a bare function call puts 8 bytes on the stack and an external called function puts 32 bytes on the stack. This estimation showed a maximum stack usage of 4679 bytes for the RedL.

The real stack usage is dependent on the target system and the compiler used on the target system. Take also in mind to add the stack usage of other software, like the SafRetL adapter, running in the same SIL0 partition.

4.4.2.2 Heap usage

The RedL does not use any heap memory.

4.5 Transport layer

The transport layer is intended to provide a transparent service, which accepts messages and sends messages to a destination address. A transport channel represents a communication link between two instances of a transport layer. For the purpose of redundant transmission, RaSTA can use several transport channels [2].

4.5.1 Start-up and opening transport channels

At the start-up of the target system the transport layer has to be initialized and the transport channels have to be opened. The transport channels must be open bevor a RaSTA connection can be established. Else the connection request / response is not working. To ensure reconnection, the transport channels must remain open after a RaSTA connection is closed.

4.5.2 Transport channel IDs

The transport channel ID has the purpose to uniquely identify a transport channel on the transport layer interface.

The transport channel IDs has to be in the range from 0 to the product of max. number of redundancy channels with the max. number of transport channels.

The used transport channel IDs have to correspond with the configuration of transport channel IDs in the redundancy layer configuration.

4.5.3 Receiving messages

Received data must be portioned into complete RedL PDU messages and must be forwarded individually to the RedL. For this purpose the message size in the first two bytes of each RedL PDU message can be used.

For each received message the function redtrn_MessageReceivedNotification() has to be called.

The RedL uses the redtri_ReadMessage() function to read a received message.

4.5.4 Sending messages

The RedL uses the redtri_SendMessage() function to send the messages over the transport channels.

This function gets called up to RADEF_SEND_BUFFER_SIZE times for one RaSTA connection in one program execution cycle. Therefore the SafRetL adapter has to provide a send messages buffer for each redundancy channel with a size of at least RADEF_SEND_BUFFER_SIZE messages.

The constant definition RADEF_SEND_BUFFER_SIZE defines the number of send buffer entries and is set to 20.

4.5.5 Configuration

The target system designer and the target system integrator are responsible for the transport layer configuration. If TCP/IP is used, IP addresses and ports have to be defined and configured for each transport channel.

4.5.6 Security measures

If the SWuC is used in environments requiring Cat. 3 networks according *EN 50159* [5], use security measures like TLS, as proposed in *EULYNX Baseline Set 4 Release 1* [6] or newer.

4.6 System adapter

The purpose of the system adapter is the abstraction of target system specific functions which are used by the SWuC. The target system designer is responsible for the implementation of the system adapter. The system adapter functions are used in the SafRetL and the RedL. Therefore the system adapter must be developed as SIL4 software according to *EN 50128* [3].

The system adapter has to provide following functions as specified in the *Software Architecture and Design Specification* [10] and the *Software Interface Specification* [11]:

- Fatal Error Function
- Get Random Number Function
- Get Timer Granularity Function
- Get Timer Value Function

5 Configuration

The target system integrator has to adjust the configuration parameters of the SWuC according to the requirements of the application and the requirements of the RaSTA network where it gets connected to.

The configuration data of the SafRetL and the RedL has to be stored in a memory area which cannot be changed during the runtime of the SWuC.

5.1 Safety retransmission layer configuration

The SafRetL provides the following configuration parameters:

- RaSTA network ID
- Tmax: Maximum accepted age of a message.
- Th: Time period for sending heartbeats.
- Safety code type (The option “No safety code” is not allowed for safety related applications.)
- Initial safety code value (The standard initial value “0123456789abcdeffedcba9876543210” of the MD4 may be used only if an implementation of the MD4 by non-safe network users can be excluded [2].)
- MWA: Maximum number of received, unconfirmed messages.
- NsendMax: Maximum number of messages which the communication party may send without receiving a confirmation (receive buffer size).
- NmaxPacket: Packetization factor (must always be 1)
- NdiagWindow: SafRetL diagnosis window size
- Number of used RaSTA connections
- Connection configuration: (for each RaSTA connection)
  - Connection ID (starting form 0, equal to the index in the configuration data structure)
  - Sender ID
  - Receiver ID
- Diagnostic timing intervals

For details (unit, resolution, valid range, default value) about this configuration parameters see *Software Requirements Specification* [9] chapter 3.1.1.1 and *Software Interface Specification* [11] chapter 5.

The SafRetL configuration data is safety relevant and must be placed in the SIL4 partition of the target system.

5.2 Redundancy layer configuration

The RedL provides the following configuration parameters:

- Check code type
- Tseq: Time period indicating how long a message is buffered that was received outside the sequence.
- Ndiagnosis: RedL diagnosis window size
- NdeferQueueSize: Maximum number of entries in the waiting queue deferQueue.
- Number of used redundancy channels (must be equal to the number of used RaSTA connections)
- Redundancy channel configuration: (for each redundancy channel)
  - Redundancy channel ID (starting form 0, equal to the index in the configuration data structure)
  - Number of used transport channels
  - Used transport channel IDs

For details (unit, resolution, valid range, default value) about this configuration parameters see *Software Requirements Specification* [9] chapter 3.1.2.1 and *Software Interface Specification* [11] chapter 5.

The RedL configuration data is not safety relevant and must be placed in the SIL0 partition of the target system.

6 Verification kit

This chapter shows an overview of tests which were already done during the SW development process of the SWuC and test which must be planned and executed by the target system integrator to achieve SIL4 according to *EN 50128* [3].

6.1 Component tests

Unit tests of the components of the SWuC were performed using the Google test framework gTest/gMock, the GCC compiler and Gcov on little- and big-endian host systems during the development process of the SWuC.

The unit test specification, source code and test report files for the SafRetL and the RaSTA common package are located in the RaSTA_Component_Test_SIL4_Vx.y.z.zip

The unit test specification, source code and test report files for the RedL package are located in the RaSTA_Component_Test_Basic_Integrity_Vx.y.z.zip

6.2 Integration tests

SW - SW integration tests of the components inside the SafRetL and the components inside the RedL were performed using the Google test framework gTest/gMock, the GCC compiler on little- and big-endian host systems during the development process of the SWuC.

The integration test specification, source code and test report files for the SafRetL package are located in the RaSTA_Integration_Test_SIL4_Vx.y.z.zip

The integration test specification, source code and test report files for the RedL package are located in the RaSTA_ Integration _Test_Basic_Integrity_Vx.y.z.zip

Due to non-existing target system integration, the following integration tests were not performed during the development of the SWuC. Therefore the target system integrator is responsible for the planning and the execution of the following tests:

- SW - HW integration tests
- Integration test of the SIL4 partition (Application, SafRetL, SafRetL Adapter) with the SIL0 partition (RedL, Transport Layer) on the target system.
- Performance tests

6.3 Overall SW tests

During the development process of the SWuC all test cases of type “OST-*” from the *Overall Software Test Case Specification* [12] were already executed.

Due to non-existing target system integration, the test cases of type “IST-*” from the *Overall Software Test Case Specification* [12] were not executed during the development of the SWuC.

The target system integrator has to execute all test cases of type “IST-*” from the *Overall Software Test Case Specification* [12].