



## Revised Requirements of the 4SECURail Case Study

### WP2 Task 2.3 Internal Report

|                                |                                      |
|--------------------------------|--------------------------------------|
| <b>Project acronym:</b>        | 4SECURail                            |
| <b>Starting date:</b>          | 01/12/2019                           |
| <b>Duration (in months):</b>   | 24                                   |
| <b>Call (part) identifier:</b> | H2020-S2R-OC-IP2-2019-01             |
| <b>Grant agreement no:</b>     | 881775                               |
| <b>Due date:</b>               |                                      |
| <b>Actual submission date:</b> |                                      |
| <b>Responsible/Author:</b>     | Franco Mazzanti, Dimitri Belli / CNR |
| <b>Dissemination level:</b>    | PU                                   |
| <b>Status:</b>                 | Draft                                |

#### **Disclaimer**

*The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author’s view – the Joint Undertaking is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability. The content of this deliverable does not reflect the official opinion of the Shift2Rail Joint Undertaking (S2R JU). Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).*



## Table of Contents

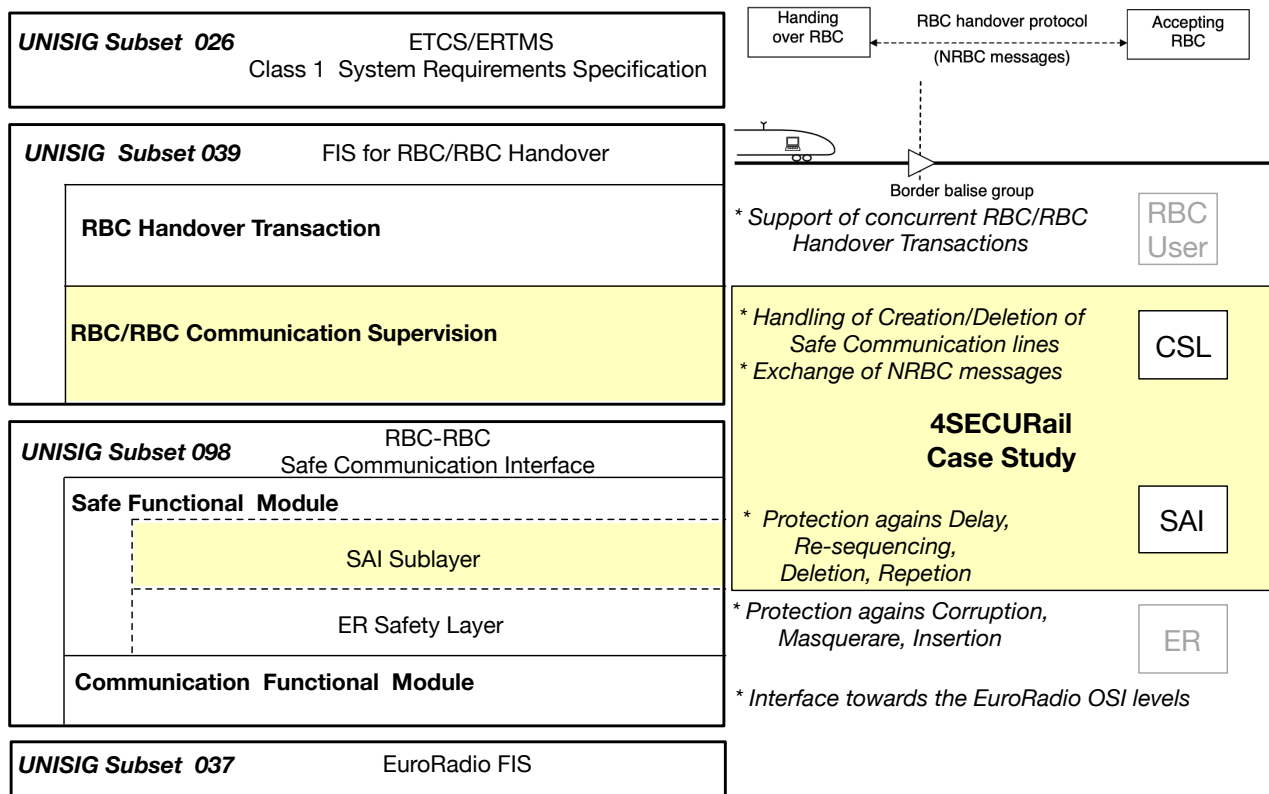
|   |  |    |
|---|--|----|
| 1 | Introduction.....                                    | 3  |
| 2 | Natural Language specifications.....                 | 6  |
| 3 | Graphical semiformal UML state machine designs ..... | 23 |
| 4 | Executable UML Models.....                           | 26 |

# 1 Introduction

The transit of a train from an area supervised by a Radio Block Centre (RBC) to an adjacent area supervised by another RBC occurs during the so-called RBC-RBC handover phase and requires the exchange of information between RBCs according to a specific protocol.

This exchange of information is supported by the communication layer specified within the UNISIG SUBSET-039 and SUBSET-098. Figure 1 hints the overall structure of the UNISIG standards supporting the handover of a train.

The 4SECURail case study focusses on two sub-components of the communication layer underlying the RBC-RBC handover communications. The considered components are the RBC/RBC Communication Supervision Layer (CSL) of the SUBSET-039, and the Safe Application Intermediate SubLayer (SAI) component of the Safe Functional Module (SFM) of the SUBSET-098. These two components are those that support the creation and deletion of safe communication lines (over an unreliable Open System Interconnection Transport Layer), as well as the protected transmission of messages.



In particular, the CSL is responsible for requesting the activation (and in case of failure the reestablishment) of the communication line, and for forwarding the RBC handover transaction messages on the active line.

The SAI is responsible for ensuring the absence of excessive delays, message repetitions, or message reordering during the transmission. This is achieved by adding sequence numbers and time related information to the RBC messages.

The two sides of the communication line are configured as "initiator" side and "called" side.

The 4SECURail case study neither include the rigorous specification and formal modelling of the EuroRadio Safety Layer (ER-SL), which is responsible to prevent corruption, masquerade and insertion issues during the communications, nor the lower ER-SL layers and the Communication Functional Module interface (with which ER-SL interfaces with the environment). For all this parts we abstracted an ER-SL environmental component that behaves according to the requested assumptions and allows the interaction of the two SAIs to each other.

The upper level the 4SECURail case study neither includes the rigorous specification and formal modelling of the interactions that the trains perform with the RBC controlling the zone which they are moving in, nor considers the activation of multiple, and possible concurrent, RBC-RBC handover sessions when trains move from a zone supervised by an RBC to another one. From the CSL point of view, only RBC messages are forwarded to/from the upper level, together with the information on the current status of the communication line. Moreover, these upper parts are abstracted as RBC environmental component which behaves according to the requested assumptions.

The rest of the document is structured as follows. Section 2 presents, in natural language, the requirements for the CSL and the SAI components, distinguishing between initiator (ICSL, ISAI) and called (CCSL, CSAI) sides. These requirements are the rewriting of the original standard definition to make the specification more self-contained and clearer. While performing the formal modelling and analysis, these requirements have incrementally been generated and kept aligned with the development of formal and semi-formal models when missing parts or inconsistencies are detected. Section 3 presents a graphical (abstract, semi-formal) representation of the UML state machines describing the expected behavior of the SAI and CSL components. Section 4 presents one complete executable and verifiable version of these components, specified in the syntax accepted by UMC for the specification of UML state machines.

Before moving forward, it is worth pointing out some details.

The UMC notation is used as the basis for the mechanical translation of the UMC formal models

into other notations, like CADP/LNT and ProB, allowing the use of different formal frameworks for the analysis of the system properties. In spite of the differences in the underlying theories and syntax, all these three different formal models can be proved to be strongly equivalent.

To perform the analysis of the specified components, it is necessary to instantiate them with the required parameters and to build closed scenarios in which the designed CSL and SAI components are complemented with a stimulating environment. Different kind of scenarios can be imagined depending on the kind of system properties we are interested to analyse.

Since the modelled system is heavily time dependent, but our formalisms are not natively support time, our formal models include an additional Timer component that allows the various system components to synchronously proceed in parallel, but relatively at the same speed.

Also these environment components (i.e., the Timer, the ER-SL, and the two RBC sides) can be designed as UML state machines and encoded in UMC syntax so that the full resulting system can be mechanically translated in the various formalisms and analysed.

Ultimately, we stress that the overall goal of the 4SECURail project is NOT the validation/verification of some parts of standard documentations, but the observation of the impact (in terms of qualitative/quantitative and cost/benefit analysis) of the use of formal methods within the requirements definition phase. The specific case study focusses on the observation of the improvements that can be achieved in the natural language specification of the interested components while passing from their initial free-style version (subjected to ambiguities, incompleteness, and inconsistencies), to a more rigorous style backed by underlying formal models.

## 2 Natural Language specifications

### Requirements Specification for the *Initiator CSL* Component

#### Configuration Parameters

System parameters,

- *max connection delay*;
- *max delay between send operations*;
- *max delay between receive operations*.

#### External Interactions

The *Initiator CSL* can receive from the *Initiator RBC* component the following message:

- *RBC\_User\_Data.request(RBC\_data\_value)*;

and can send to the *RBC* component the following messages:

- *RBC\_User\_Connect.indication*;
- *RBC\_User\_Disconnect.indication*;
- *RBC\_User\_Data.indication(RBC\_data\_value)*.

The *CSL* can receive from the *Initiator SAI* component the following messages:

- *SAI\_CONNECT.confirm*;
- *SAI\_DISCONNECT.indication*;
- *SAI\_DATA.indication(message\_type<sup>1</sup>, SAI\_data\_value)*;
- *SAI\_ERROR.report*;

and can send to the *SAI* component the following messages:

- *SAI\_CONNECT.request*;
- *SAI\_DISCONNECT.request*;
- *SAI\_DATA.request(message\_type, SAI\_data\_value)*.

#### States

The *CSL* can be in the following four main states:

- *Disconnected*, when the communication is unactive;
- *Connecting*, when the communication is in the establishment phase;
- *Connected*, when the communication is active;
- *Waiting*, when the communication is between the *Connected* and *Disconnected* states.

#### External Guarantees

- *CSL* sends *RBC\_User\_Data.indication* messages only after an *RBC\_User\_Connect.request* not followed by *RBC\_User\_Disconnect.indication*;
- *CSL* sends to the *RBC* component an *RBC\_User\_Disconnect.indication* message only after an *RBC\_User\_Connect.request* message not already followed by *RBC\_User\_Disconnect.indication*;
- *CSL* sends to the *RBC* component an *RBC\_User\_Connect.indication* message only as first message or after an *RBC\_User\_Disconnect.indication* not already followed by *RBC\_User\_Connect.indication*;

---

<sup>1</sup> *message\_type* may refer to either *life\_sign* or *RBC\_data*.

- the first message (possibly) sent to the *RBC* component is an *RBC\_User\_Connect.indication* message;
- the initiator *CSL* periodically sends to the *SAI* component either *SAI\_CONNECT.request* or *SAI\_DATA.request* messages;
- if the initiator *CSL*, while in *Connected (COMMS)* state, does not receive any *SAI\_DATA.indication* message from the *SAI* for a certain specified amount of time, a *SAI\_DISCONNECT.request* message is sent to the *SAI*;
- the initiator *CSL* may send a *SAI\_DISCONNECT.request* message only when in *Connected (COMMS)* state;
- incoming messages are buffered and served with *FIFO* policy.

### External Assumptions

- The *SAI* always replies with a *SAI\_DISCONNECT.indication* message to *SAI\_DISCONNECT.request* messages issued by the *CSL*.

### Behavioral Requirements

**R1:** At startup, the *CSL* is in *Disconnected* state.

#### When in Disconnected State

**R2:** When in *Disconnected* state, the *CSL* immediately sends a *SAI\_CONNECT.request* to the *SAI* component, starts a *connTimer*, and moves to the *Connecting* state.

#### When in Connecting State

**R3:** When in *Connecting* state the *connTimer* expires, the *CSL* moves to *Disconnected* state.

**R4:** When in *Connecting* state is received a *SAI\_CONNECT.confirm* from the *SAI* component, the *CSL* sends an *RBC\_User\_Connect.indication* to the *RBC* component, starts both the *sendTimer* and the *recTimer*, and moves to *Connected* state. It is allowed to set the *sendTimer* so that an initial lifesign is sent without delay.

#### When in Waiting State

**R5:** When in *Waiting* state is received a *SAI\_DISCONNECT.indication* from the *SAI* component, the *CSL* moves to *Disconnected* state.

#### When in Connected State

**R6:** When in *Connected* state the *recTimer* expires, the *CSL* sends a *SAI\_DISCONNECT.request* to the *SAI* component, an *RBC\_User\_Disconnect.indication* to the *RBC* and moves to *Waiting* state.

**R7:** Each time that in *Connected* state the *sendTimer* expires, the *CSL* sends a *SAI\_DATA.request* with a *life\_sign* to the *SAI* component.

**R8:** When in *Connected* state is received an *RBC\_User\_Data.request* with *RBC\_data* from the *RBC* component, the *CSL* sends a *SAI\_DATA.request* with such *RBC\_data* to the *SAI* component.

**R9:** When in *Connected* state is received a *SAI\_DATA.indication* with *SAI\_data* from the *SAI* component, the *CSL* sends an *RBC\_User\_Data.indication* with such *SAI\_data* to the *RBC* component and restarts the *recTimer*.

**R10:** When in *Connected* state is received a *SAI\_DATA.indication* with a *life\_sign* from the *SAI* component, the *CSL* restarts the *recTimer*.

**R11:** When in *Connected* state is received a *SAI\_DISCONNECT.indication* from the *SAI* component, the *CSL* sends an *RBC\_User\_Disconnect.indication* to the *RBC* component and moves to *Disconnected* state.

#### Discarding of Messages

**RD1:** When in *Connecting* state, the *CSL* discards any message except for *SAI\_CONNECT.confirm* from the *SAI* component.

**RD2:** When in *Waiting* state, the *CSL* discards any message except for *SAI\_DISCONNECT.indication* from the *SAI* component.

**RD3:** When in *Connected* state, the *CSL* component discards only *SAI\_CONNECT.confirm* and *SAI\_ERROR.report* messages from the *SAI* component.



## Requirements Specification for the *Initiator SAI* Component

### Configuration Parameters

Initialization kind: *Execution Cycle* option.

System parameters,

- for *Execution Cycle* procedure:
  - *maximum initialization delay*
  - *Mec* (limit of the execution cycle counters);
  - *K* (max acceptable transmission delay for a message);
- for *ACK* procedure:
  - *ack\_request\_period*;
  - *ack\_response\_timeout*;
- for *sequence number*:
  - *N* (limit of acceptable, consecutive message losses,  $N = 1$  means no losses);
  - *M* (limit of the sequence number values, which have range  $0..M-1$ ).

### External Interactions

The *Initiator SAI* can receive from the *Initiator CSL* component the following messages:

- *SAI\_CONNECT.request*;
- *SAI\_DISCONNECT.request*;
- *SAI\_DATA.request* (*message\_type*<sup>2</sup>, *RBC\_data\_value*);

and can send to the *CSL* component the following messages:

- *SAI\_CONNECT.confirm*;
- *SAI\_DISCONNECT.indication*;
- *SAI\_DATA.indication*(*message\_type*, *RBC\_data\_value*);
- *SAI\_ERROR.report*.

The *SAI* can receive from the *EuroRadio Safety Layer* (henceforth *ER-SL*) the following messages:

- *Sa\_CONNECT.confirm*;
- *Sa\_DISCONNECT.indication*;
- *Sa\_DATA.indication*(*message\_type*, *data\_value*, *ack\_request*, *ack\_response*, *sequence\_number*, *execution\_cycle\_number*);
- *Sa\_ExecutionCycleStart*(*sequence\_number*, *execution\_cycle\_counter*);

and can send to the *ER-SL* the following messages:

- *Sa\_CONNECT.request*;
- *Sa\_DISCONNECT.request*;
- *Sa\_DATA.request*(*message\_type*, *data\_value*, *ack\_request*, *ack\_response*, *sequence\_number*, *execution\_cycle\_number*);
- *Sa\_ExecutionCycle*(*sequence\_number*, *execution\_cycle\_counter*).

### Internal Variables

- *sequence\_number*;
- *execution\_cycle\_counter*;

---

<sup>2</sup> *message\_type* may refer to either *life\_sign* or *RBC\_data*.

- *last\_received\_sequence\_number*;
- *last\_received\_execution\_cycle\_counter*;
- *execution\_cycle\_OFFSET*.

### States

The *SAI* can be in the following four main states:

- *Connected*, when the communication is active;
- *Connecting*, when the communication is in the establishment phase;
- *Initializing*, while performing the execution cycle start procedure;
- *Disconnected*, when the communication is unactive.

### External Guarantees

- The *SAI* always replies with a *SAI\_DISCONNECT.indication* message to *SAI\_DISCONNECT.request* messages issued by the *CSL*;
- the data messages delivered to the *CSL* are valid (i.e., arrived with a limited delay), not duplicated, not reordered messages;
- no more than one data message per execution cycle is sent to the *ER-SL*;
- incoming messages are buffered and served with *FIFO* policy.

### External Assumptions

- The *ER-SL* always eventually replies either with a *Sa\_DISCONNECT.indication* or with a *SAI\_CONNECT.confirm* to *Sa\_CONNECT.request* messages issued by the *SAI*;
- the initiator *CSL*, after having sent a *SAI\_CONNECT.request* message to the *SAI*, does not send a *SAI\_DISCONNECT.request* message until *SAI\_CONNECT.indication* messages is received.

### Behavioral Requirements

**R1:** At startup, the *SAI* is in *Disconnected* state.

#### When in Disconnected State

**R2:** When in *Disconnected* state is received a *SAI\_CONNECT.request* from the *CSL* component, the *SAI* sends a *Sa\_CONNECT.request* to the *ER-SL* and moves to *Connecting* state.

**R3:** When in *Disconnected* state is received a *SAI\_DISCONNECT.request* from the *CSL* component, the *SAI* replies with a *SAI\_DISCONNECT.indication* to the *CSL* component.

#### When in Connecting State

**R4:** When in *Connecting* state is received a *Sa\_DISCONNECT.indication* from the *ER-SL*, the *SAI* moves to *Disconnected* state.

**R6:** When in *Connecting* state is received a *Sa\_CONNECT.confirm* from the *ER-SL*, the *SAI* replies with a *Sa\_ExecutionCycle(seqnum, ecnum)* to the *ER-SL* and moves to the *Initializing* state, waiting for a *Sa\_ExecutionCycleStart* message from the *ER-SL* within a *maximum initialization delay*. The management of the *Sa\_ExecutionCycleStart* parameters is done according to the requirements in the following *Sequence Numbers Management* and *Execution Cycle Counters Management* sections.

### When in Initializing State

**R7:** When in *Initializing* state the *maximum initialization delay* expires, the SAI sends an *SAI\_ERROR.report* to the CSL component, a *Sa\_DISCONNECT.request* to the ER-SL and moves to *Disconnected* state.

**R9:** When in *Initializing* state is received a *Sa\_DISCONNECT.indication* from the ER-SL, the SAI moves to *Disconnected* state.

**R11:** When in *Initializing* state is received a *Sa\_ExecutionCycleStart(seqnum, ecnum)* from the ER-SL, the SAI sends a *SAI\_CONNECT.confirm* to the CSL component and moves to *Connected* state. The received *seqnum* is accepted as initial remote sequence number and the *ecnum* is accepted as initial value of the remote execution cycle counter. The *execution\_cycle\_OFFSET* variable is set as the difference between the current execution cycle counter and the received execution cycle counter. While the *last\_received\_sequence\_number* variable is set to the received sequence number.

### When in Connected State

**R12:** When in *Connected* state is received a *SAI\_DISCONNECT.request* from the CSL component, the SAI replies with a *SAI\_DISCONNECT.indication* to the CSL component, sends a *Sa\_DISCONNECT.request* to the ER-SL, and moves to *Disconnected* state.

**R13a:** When in *Connected* state is received a *SAI\_DATA.request(msgtype, data)* from the CSL component, and yet no other data message has been sent in this cycle, the SAI sends a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* to the ER-SL.

The *ackreq* and *ackresp* parameters are set according to REQ\_ACKs.

The *seqnum* parameter is set according to SEQ\_NUMs and the *ecnum* parameter is set according to REQ\_ECNUMs.

**R13b:** When in *Connected* state is received a *SAI\_DATA.request(msgtype, data)* from the CSL component, but another data message has already been sent in this cycle, the *SAI\_DATA.request* is saved in a FIFO dataout buffer (see also REQ\_OUTDATABUFF).

**R14:** Each time that in *Connected* state the *set\_ack\_response* expires, the SAI sends a *SAI\_ERROR.report* to the CSL component.

**R15:** When in *Connected* state is received a *Sa\_DISCONNECT.indication* from the ER-SL, the SAI sends a *SAI\_DISCONNECT.indication* to the CSL component and moves to *Disconnected* state.

**R16:** When in *Connected* state is received a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the ER-SL we can have four cases, depending on the received *seqnum* and *ecnum* values (see SEQ\_NUMs and REC\_ECNUMs Management).

\* The *seqnum* is the one EXPECTED and *ecnum* is VALID: In this case the SAI sends a *SAI\_DATA.indication(msgtype, data)* to the CSL component.

\* The *seqnum* is ACCEPTABLE and the *ecnum* is VALID: in this case the SAI sends a *SAI\_DATA.indication(msgtype, data)* and a *SAI\_ERROR.report* to the CSL component.

\* The *seqnum* is OLD or (the *seqnum* is ACCEPTABLE and the *ecnum* is VALID): In this case the SAI sends a *SAI\_ERROR.report* to the CSL component and discards the *Sa\_DATA.indication* message.

\* The *seqnum* is NOT\_ACCEPTABLE: In this case the SAI component sends a *Sa\_DISCONNECT.request* to ER-SL and a *SAI\_DISCONNECT.indication* to the CSL component, and then moves to *Disconnected* state.

### OUTDATA Buffer Management

**REQ\_OUTDATABUFF1:** At the beginning of each cycle, if the *dataout buffer* is not empty, the first *SAI\_DATA.request(msgtype, data)* in the queue is removed and its data are used to send a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* to the ER-SL.

The *ackreq, ackresp, seqnum, ecnum* parameters are set according to *REQ\_ECNUM, REQ\_ACK, and REQ SEQNUM* requirements.

**REQ\_OUTDATABUFF2:** When the *SAI* moves from the *Connected* state to the *Disconnected* state, the *dataout buffer* is emptied and the possibly waiting messages are discarded.

### Execution Cycle Counters Management

**REQ\_ECNUM1:** When entering in the *Initializing* state, the initial value of the *execution cycle counter* is set to 0.

**REQ\_ECNUM2:** While in the *Initializing* or *Connected* state, the *execution cycle counter* is incremented modulo *Mec* at every cycle.

**REQ\_ECNUM3:** When sending a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message or a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* the value of the *ecnum* parameter is set to the current value of the *execution cycle counter*.

**REQ\_ECNUM4:** When receiving a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message from the ER-SL, the value of the *ecnum* parameter is used to compute the *EC\_OFFSET* as difference between the current value of the *execution cycle counter* and the received *seqnum* value.

**REQ\_ECNUM5:** When receiving a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message from ER-SL, the message is considered VALID if the message delay is less than *K*, where the message delay is computed as follows<sup>3</sup>:

```
message_delay = (execution_cycle_counter - EC_OFFSET) mod Mec4 - ecnum;
if message_delay < -Mec/2 then
    message_delay := message_delay + Mec;
elseif message_delay > Mec/2 then
    message_delay := message_delay - Mec;
end if
```

### Sequence Numbers Management

**SEQ\_NUM1:** When entering in the state *Connected*, the *sequence\_number* is set to 0.

**SEQ\_NUM2:** When in *Connecting* state a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message is sent to the ER-SL, the *seqnum* parameter is set to the current value of *sequence\_number*.

<sup>3</sup> This is a simplification from what required by UNISIG-098 as we assume that the EC period is 1 cycle for both SAI sides.

<sup>4</sup> Also when applied to negative numbers,  $(N \bmod M)$  is assumed to be equal to  $((N+M) \bmod M)$ .

**SEQ\_NUM3:** When in the *Initializing* or *Connected* state a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message is sent to the *ER-SL*, the *seqnum* parameter is set to the current value of *sequence\_number*, and the *sequence\_number* is incremented by  $1 \bmod M$ .

**SEQ\_NUM4:** When in *Initializing* state is received a *Sa\_ExecutionCycleStart(seqnum, ecnum)* message from *ER-SL*, the value of the *seqnum* parameter is saved as *last\_received\_sequence\_number*.

**SEQ\_NUM5:** When in the *Initializing* or *Connected* state is received a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *ER-SL*, the distance of the current message from the last received one is computed as follows:

```
distance := seq_num - last_received_sequence_number;  
if (distance < -M/2) then {distance := distance + M};  
else if (distance > M/2) then {distance := distance - M};
```

If the distance value is equal to 1, the *seqnum* is considered EXPECTED.

If the distance value is lower than 1, the *seqnum* is considered OLD.

If the distance value is greater than 1 and less or equals to *N*, the *seqnum* is considered ACCEPTABLE.

If the distance value is greater than *N*, the *seqnum* is considered NOT\_ACCEPTABLE.

### ACK Management

**REQ\_ACK1:** When in *Connected* state, the *SAI* periodically (with a configurable *ack\_request\_period*) sets an *ackreq* flag to the first *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message to be forwarded to the *ER-SL* and starts an *ack\_response\_timer* with a *max\_response\_delay* limit.

The *ackreq* flag is not set and the timer is not started if the *SAI* is still waiting for the response to a previous ack request.

**REQ\_ACK2:** When the *ack\_response\_timeout* expires, if a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message with an *ackresp* parameter set has not yet been received from the *ER-SL*, the *SAI* sends a *SAI\_ERROR.report* to the *CSL* component and restarts the *ack request timer*.

**REQ\_ACK3:** While in *Connected* or *Initializing* state, when it is received a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *CSL* component, the *SAI* sets the *ackresp* parameter in next *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message to be sent to the *ER-SL*.

### Discarding of Messages

**RD1:** When in *Disconnected* state the *SAI* discards any message except for,

- *SAI\_CONNECT.request* and *SAI\_DISCONNECT.request* from the *CSL* component.

**RD2:** When in *Connecting* state, the *SAI* discards any message except for,

- *Sa\_DISCONNECT.indication*, and *Sa\_CONNECT.confirm* from the *ER-SL*;

**RD3:** When in *Initializing* state, the *SAI* discards any message except for,

- *Sa\_DISCONNECT.indication* and *Sa\_ExecutionCycleStart* from the *ER-SL*;

**RD4:** When in *Connected* state, the *SAI* discards any message except for,



- *Sa\_DISCONNECT.indication* and *Sa\_DATA.indication* from the *ER-SL*;
- *SAI\_DISCONNECT.request*, *SAI\_DATA.request* from the *CSL* component.

## Requirements Specification for the *Called CSL* Component

### Configuration Parameters

System parameters,

- *max delay between send operations*;
- *max delay between receive operations*.

### External Interactions

The *Called CSL* can receive from the *Called RBC* component the following message:

- *RBC\_User\_Data.request(RBC\_data\_value)*;

and can send to the *RBC* component the following messages:

- *RBC\_User\_Connect.indication*;
- *RBC\_User\_Disconnect.indication*;
- *RBC\_User\_Data.indication(RBC\_data\_value)*.

The *CSL* can receive from the *Called SAI* component the following messages:

- *SAI\_CONNECT.indication*;
- *SAI\_DISCONNECT.indication*;
- *SAI\_DATA.indication(message\_type<sup>5</sup>, SAI\_data\_value)*;
- *SAI\_ERROR.report*;

and can send to the *SAI* component the following messages:

- *SAI\_CONNECT.request*;
- *SAI\_DISCONNECT.request*;
- *SAI\_DATA.request(message\_type, SAI\_data\_value)*.

### States

The *CSL* can be in the following two states:

- *Disconnected (NOCOMMS)*, when the communication is unactive;
- *Connected (COMMS)*, when the communication is active.

### External Guarantees

- The frequency of messages being sent by *CSL* to *RBC* is limited by an upper bound;
- the frequency of messages being sent by *CSL* to *SAI* is limited by an upper bound;
- *CSL* sends *RBC\_User\_Data.indication* messages only after an *RBC\_User\_Connect.request* not followed by *RBC\_User\_Disconnect.indication*;
- *CSL* sends to the *RBC* component an *RBC\_User\_Disconnect.indication* message only after an *RBC\_User\_Connect.indication* message not already followed by *RBC\_User\_Disconnect.indication*;
- the first message (possibly) sent to the *RBC* component is an *RBC\_User\_Connect.indication* message;
- *CSL* sends to the *RBC* component an *RBC\_User\_Connect.indication* message only as first message or after an *RBC\_User\_Disconnect.indication* not already followed by *RBC\_User\_Connect.indication*;
- the called *CSL*, while in the *Connected (COMMS)* state periodically sends to the *SAI* component *SAI\_DATA.request* messages;

---

<sup>5</sup> *message\_type* may refer to either *life\_sign* or *RBC\_data*.



- if the called CSL, while in the *Connected (COMMS)* state, does not receive any *SAI\_DATA.indication* message from the SAI for a certain specified amount of time, a *SAI\_DISCONNECT.request* message is sent to the SAI;
- incoming messages are buffered and served with *FIFO* policy.

## Behavioral Requirements

**R1:** At startup, the *CSL* is in *Disconnected* state.

### When in Disconnected State

**R2:** When in *Disconnected* state is received a *SAI\_CONNECT.indication* from the *SAI* component, the *CSL* sends an *RBC\_User\_Connect\_indication* to the *RBC* component, starts both the *sendTimer* and the *recTimer*, and moves to *Connected* state. It is allowed to set the *sendTimer* so that an initial lifesign is sent without delay.

### When in Connected State

**R4:** When in *Connected* state is received an *RBC\_User\_Data.request(userdata)* from the *RBC* component, the *CSL* sends a *SAI\_DATA.request(RBC\_data,userdata)* to the *SAI* component.

**R5:** Each time that in *Connected* state the *sendTimer* expires, the *CSL* sends a *SAI\_DATA.request* with a *life\_sign* to the *SAI* component.

**R6:** When in *Connected* state is received a *SAI\_DATA.indication* with a *life\_sign* from the *SAI* component, the *CSL* restarts the *recTimer*.

**R7:** When in *Connected* state is received a *SAI\_DATA.indication* with *SAI\_data* from the *SAI* component, the *CSL* sends an *RBC\_User\_Data.indication* with such *SAI\_data* to the *RBC* component and restarts the *recTimer*.

**R8:** When in *Connected* state is received a *SAI\_DISCONNECT.indication* from the *SAI* component, the *CSL* sends an *RBC\_User\_Disconnect.indication* to the *RBC* component and moves to *Disconnected* state.

**R9:** When in *Connected* state the *recTimer* expires, the *CSL* sends a *SAI\_DISCONNECT.request* to the *SAI* component, an *RBC\_User\_Disconnect.indication* to the *RBC* component and moves to *Disconnected* state.

### Discarding of Messages

**RD1:** When in *Disconnected* state the *CSL* does not accept any kind of message except for *SAI\_CONNECT.indication* from the *SAI* component.

**RD2:** When in *Connected* state the *CSL* discards *SAI\_CONNECT.indication* and *SAI\_ERROR.report* messages from the *SAI* component.



## Requirements Specification for the *Called SAI* Component

### Configuration Parameters

Initialization kind: *Execution Cycle* option.

System parameters,

- for *Execution Cycle* procedure:
  - *maximum initialization delay*
  - *Mec* (limit of the execution cycle counters);
  - *K* (max acceptable transmission delay for a message);
- for *ACK* procedure:
  - *ack\_request\_period*;
  - *ack\_response\_timeout*;
- for *sequence number*:
  - *N* (limit of acceptable, consecutive message losses,  $N = 1$  means no losses);
  - *M* (limit of the sequence number values, which have range  $0..M-1$ ).

### External Interactions

The *Called SAI* can receive from the *Called CSL* component the following messages:

- *SAI\_DISCONNECT.request*;
- *SAI\_DATA.request(message\_type<sup>6</sup>, RBC\_data\_value)*;

and can send to the *CSL* the following messages:

- *SAI\_CONNECT.indication*;
- *SAI\_DISCONNECT.indication*;
- *SAI\_DATA.indication(message\_type, RBC\_data\_value)*;
- *SAI\_ERROR.report*.

The *SAI* can receive from the *EuroRadio Safety Layer* (henceforth *ER-SL*) the following messages:

- *Sa\_CONNECT.indication*;
- *Sa\_DISCONNECT.indication*;
- *Sa\_DATA.indication(message\_type, SAI\_data\_value, ack\_request, ack\_response, sequence\_number, execution\_cycle\_number)*;
- *Sa\_ExecutionCycleStart(sequence\_number, execution\_cycle\_counter)*;

and can send to the *ER-SL* the following messages:

- *Sa\_CONNECT.response*;
- *Sa\_DISCONNECT.request*;
- *Sa\_DATA.request(message\_type, SAI\_data\_value, ack\_request, ack\_response, sequence\_number, execution\_cycle\_number)*;
- *Sa\_ExecutionCycle(sequence\_number, execution\_cycle\_counter)*.

### Internal Variables

- *sequence\_number*;
- *execution\_cycle\_counter*;
- *last\_received\_sequence\_number*;

---

<sup>6</sup> *message\_type* may refer to either *life\_sign* or *RBC\_data*.

- *last\_received\_execution\_cycle\_counter*;
- *execution\_cycle\_OFFSET*.

### States

The *SAI* can be in the following four main states:

- *Connected*, when the communication is active;
- *Connecting*, when the communication is in the establishment phase;
- *Initializing*, while performing the execution cycle start procedure;
- *Disconnected*, when the communication is unactive.

### External Guarantees

- The data messages delivered to the *CSL* are valid (i.e., arrived with a limited delay), neither duplicated nor reordered;
- no more than one data message per execution cycle is sent to the *ER-SL*;
- incoming messages are buffered and served with *FIFO* policy.

### Behavioral Requirements

**R1:** At startup, the *SAI* is in *Disconnected* state.

#### When in Disconnected State

**R2:** When in *Disconnected* state is received a *Sa\_CONNECT.indication* from the *ER-SL*, the *SAI* replies with a *Sa\_CONNECT.response* to the *ER-SL* and moves to *Connecting* state.

#### When in Connecting State

**R2b:** When in *Connecting* state is received a *Sa\_CONNECT.indication* from the *ER-SL*, the *SAI* replies with a *Sa\_CONNECT.response* to the *ER-SL* and remains in the *Connecting* state.

**R3:** When in *Connecting* state is received a *Sa\_DISCONNECT.indication* from the *ER-SL*, the *SAI* moves to *Disconnected* state.

**R5:** When in *Connecting* state is received a *Sa\_ExecutionCycleStart(seqnum, ecnum)* from the *ER-SL*, the *SAI* replies with a *Sa\_ExecutionCycle(seqnum, ecnum)* to the *ER-SL*, starts an *initTimer* set to the *maximum initialization delay*, and moves to *Initializing* state. The management of the *Sa\_ExecutionCycleStart* parameters are done according to the rules in the *Sequence Numbers Management* and *Execution Cycle Counters Management* sections.

#### When in Initializing State

**R2c:** When in *Initializing* state is received a *Sa\_CONNECT.indication* from the *ER-SL*, the *SAI* replies with a *Sa\_CONNECT.response* to the *ER-SL* and moves to *Connecting* state.

**R6:** When in *Initializing* state the *maximum initialization delay* expires, the *SAI* sends a *SAI\_ERROR.report* to the *CSL* component and moves to *Disconnected* state.

**R8:** When in *Initializing* state is received a *Sa\_DISCONNECT.indication* from the *ER-SL*, the *SAI* moves to *Disconnected* state.

**R9:** When in *Initializing* state is received a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *ER-SL* may have four cases, depending on the received *seqnum* and *ecnum* values (see *REQ\_SEQ\_NUMs* and *REC\_ECNUMs*).

\* The *seqnum* is the one *EXPECTED* and *ecnum* is *VALID*: In this case the *SAI* moves to *Connected* state and sends both a *SAI\_CONNECT.indication* and a *SAI\_DATA.indication(msgtype, data)* to the *CSL* component.

\* The *seqnum* is *ACCEPTABLE* and the *ecnum* is *VALID*: in this case the *SAI* moves to *Connected* state and sends a *SAI\_CONNECT.indication*, a *SAI\_DATA.indication(msgtype, data)* and a *SAI\_ERROR.report* to the *CSL* component.

\* The *seqnum* is *NOT\_ACCEPTABLE*: in this case the *SAI* component sends a *Sa\_DISCONNECT.request* to *ER-SL* and moves to *Disconnected* state.

\* The *seqnum* is *OLD* or (the *seqnum* is *ACCEPTABLE* and the *ecnum* is *VALID*): In this case the *SAI* sends a *SAI\_ERROR.report* to the *CSL* component and discards the *Sa\_DATA.indication* message.

#### When in Connected State

**R10:** When in *Connected* state is received a *SAI\_DISCONNECT.request* from the *CSL* component, the *SAI* replies with a *SAI\_DISCONNECT.indication* to the *CSL* component, sends a *Sa\_DISCONNECT.request* to the *ER-SL*, and moves to *Disconnected* state.

**R11:** When in *Connected* state is received a *Sa\_DISCONNECT.indication* from the *ER-SL*, the *SAI* sends a *SAI\_DISCONNECT.indication* to the *CSL* component and moves to *Disconnected* state.

**R12:** When in *Connected* state is received a *Sa\_CONNECT.indication* from the *ER-SL*, the *SAI* replies with a *Sa\_CONNECT.response* to the *ER-SL*, sends a *SAI\_DISCONNECT.indication* to the *CSL* component, and moves to *Connecting* state.

**R13a:** When in *Connected* state is received a *SAI\_DATA.request(msgtype, data)* from the *CSL* component, and yet no other data message has been sent in this cycle, the *SAI* sends a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* to the *ER-SL*.

The *ackreq* and *ackresp* parameters are set according to *REQ\_ACKs*.

The *seqnum* parameter is set according to *SEQ\_NUMs* and the *ecnum* parameter is set according to *REQ\_ECNUMs*.

**R13b:** When in *Connected* state is received a *SAI\_DATA.request(msgtype, data)* from the *CSL* component, but another data message has already been sent in this cycle, the *SAI\_DATA.request* is saved in a *FIFO dataout buffer* (see also *REQ\_OUTDATABUFF*).

**R14:** When in *Connected* state is received a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *ER-SL* we can have four cases, depending on the received *seqnum* and *ecnum* values (see *SEQ\_NUMs* and *REC\_ECNUMs*).

\* The *seqnum* is the one *EXPECTED* and *ecnum* is *VALID*: In this case the *SAI* sends a *SAI\_DATA.indication(msgtype, data)* to the *CSL* component.

Depending on the received values of the *ackreq* and *ackresp* parameters, appropriate actions are performed (see *REQ\_ACKs*).

\* The *seqnum* is *ACCEPTABLE* and the *ecnum* is *VALID*: in this case the *SAI* sends a *SAI\_DATA.indication(msgtype, data)* and a *SAI\_ERROR.report* to the *CSL* component.

Depending on the received values of the *ackreq* and *ackresp* parameters, appropriate actions are performed (see *REQ\_ACKs*).

- \* The *seqnum* is OLD or (the *seqnum* is ACCEPTABLE and the *ecnum* is VALID): In this case the SAI sends a *SAI\_ERROR.report* to the CSL component and discards the *Sa\_DATA.indication* message.
- \* The *seqnum* is NOT\_ACCEPTABLE: In this case the SAI component sends a *Sa\_DISCONNECT.request* to ER-SL, a *SAI\_DISCONNECT.indication* to the CSL component, and then moves to *Disconnected* state.

### OUTDATA Buffer Management

**REQ\_OUTDATABUFF1:** At the beginning of each cycle, if the *dataout buffer* is not empty, the first *SAI\_DATA.request(msgtype, data)* in the queue is removed and its data are used to send a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* to the ER-SL. The *ackreq, ackresp, seqnum, ecnum* parameters are set according to *REQ\_ECNUM, REQ\_ACK, and REQ SEQNUM* requirements.

**REQ\_OUTDATABUFF2:** When the SAI moves from the *Connected* state to the *Disconnected* state, the *dataout buffer* is emptied and the possibly waiting messages are discarded.

### Execution Cycle Counters Management

**REQ\_ECNUM1:** When entering in the *Initializing* state, the initial value of the *execution cycle counter* is set to 0.

**REQ\_ECNUM2:** While in the *Initializing* or *Connected* state, the *execution cycle counter* is incremented modulo *Mec* at every cycle.

**REQ\_ECNUM3:** When sending a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message or a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* the value of the *ecnum* parameter is set to the current value of the *execution cycle counter*.

**REQ\_ECNUM4:** When receiving a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message from the ER-SL, the value of the *ecnum* parameter is used to compute the *EC\_OFFSET* as difference between the current value of the *execution cycle counter* and the received *seqnum* value.

**REQ\_ECNUM5:** When receiving a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message from ER-SL, the message is considered VALID if the message delay is less than *K*, where the message delay is computed as follows<sup>7</sup>:

```

message_delay = (execution_cycle_counter - EC_OFFSET) mod Mec8 - ecnum;
if message_delay < -Mec/2 then
    message_delay := message_delay + Mec;
elseif message_delay > Mec/2 then
    message_delay := message_delay - Mec;
end if

```

### Sequence Numbers Management

**SEQ\_NUM1:** When entering in the state *Connected*, the *sequence\_number* is set to 0.

<sup>7</sup> This is a simplification from what required by UNISIG-098 as we assume that the EC period is 1 cycle for both SAI sides.

<sup>8</sup> Also when applied to negative numbers,  $(N \bmod M)$  is assumed to be equal to  $((N+M) \bmod M)$ .

**SEQ\_NUM2:** When in *Connecting* state a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message is sent to the *ER-SL*, the *seqnum* parameter is set to the current value of *sequence\_number*.

**SEQ\_NUM3:** When in the *Initializing* or *Connected* state a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message is sent to the *ER-SL*, the *seqnum* parameter is set to the current value of *sequence\_number*, and the *sequence\_number* is incremented by  $1 \bmod M$ .

**SEQ\_NUM4:** When in *Initializing* state is received a *Sa\_ExecutionCycleStart(seqnum,ecnum)* message from *ER-SL*, the value of the *seqnum* parameter is saved as *last\_received\_sequence\_number*.

**SEQ\_NUM5:** When in the *Initializing* or *Connected* state is received a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *ER-SL*, the distance of the current message from the last received one is computed as follows:

$$\begin{aligned} \text{distance} &:= \text{last\_received\_sequence\_number} - \text{seq\_num}; \\ \text{if } (\text{distance} < -M/2) &\text{ then } \{\text{distance} := \text{distance} + M\}; \\ \text{else if } (\text{distance} > M/2) &\text{ then } \{\text{distance} := \text{distance} - M\}; \end{aligned}$$

If the distance value is equal to 1, the *seqnum* is considered EXPECTED.

If the distance value is lower than 1, the *seqnum* is considered OLD.

If the distance value is greater than 1 and less or equals to *N*, the *seqnum* is considered ACCEPTABLE.

If the distance value is greater than *N*, the *seqnum* is considered NOT\_ACCEPTABLE.

### ACK Management

**REQ\_ACK1:** When in *Connected* state, the *SAI* periodically (with a configurable *ack\_request\_period*) sets an *ackreq* flag to the first *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message to be forwarded to the *ER-SL* and starts an *ack\_response\_timer* with a *max\_response\_delay* limit.

The *ackreq* flag is not set and the timer is not started if the *SAI* is still waiting for the response to a previous ack request.

**REQ\_ACK2:** When the *ack\_response\_timeout* expires, if a *Sa\_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message with an *ackresp* parameter set has not yet been received from the *ER-SL*, the *SAI* sends a *SAI\_ERROR.report* to the *CSL* component and restarts the *ack request timer*.

**REQ\_ACK3:** While in *Connected* or *Initializing* state, when it is received a *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* from the *CSL* component, the *SAI* sets the *ackresp* parameter in next *Sa\_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)* message to be sent to the *ER-SL*.

### Discarding of Messages

**RD1:** When in *Disconnected* state the *SAI* discards any message except for *Sa\_CONNECT.indication* from the *ER-SL*.

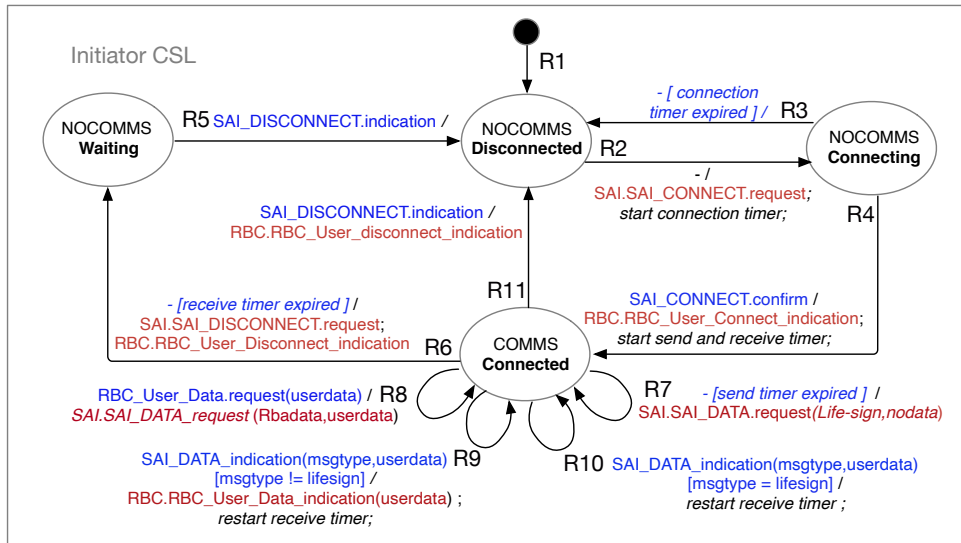
**RD2:** When in *Connecting* state, the *SAI* discards any message except for, *Sa\_DISCONNECT.indication* and *Sa\_ExecutionCycleStart* from the *ER-SL*;

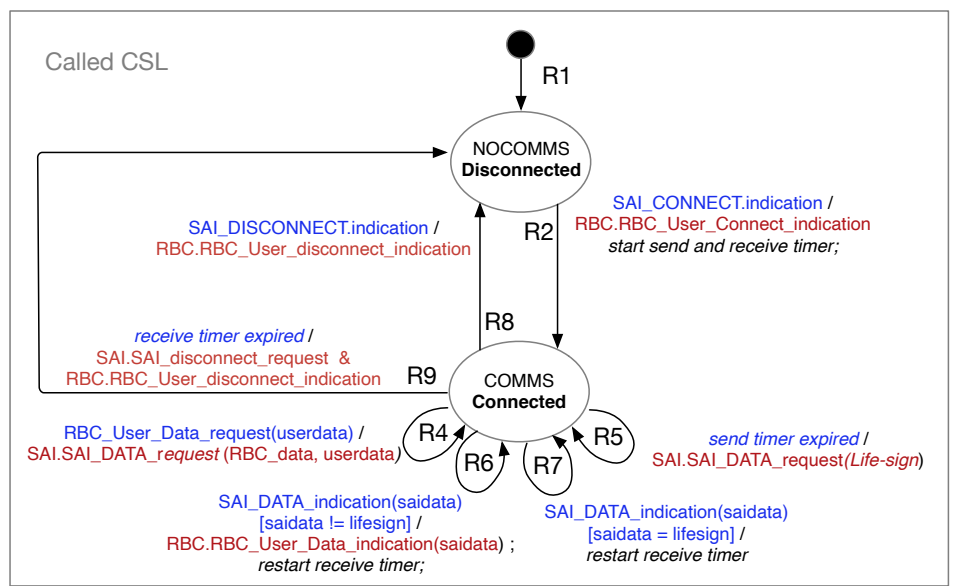
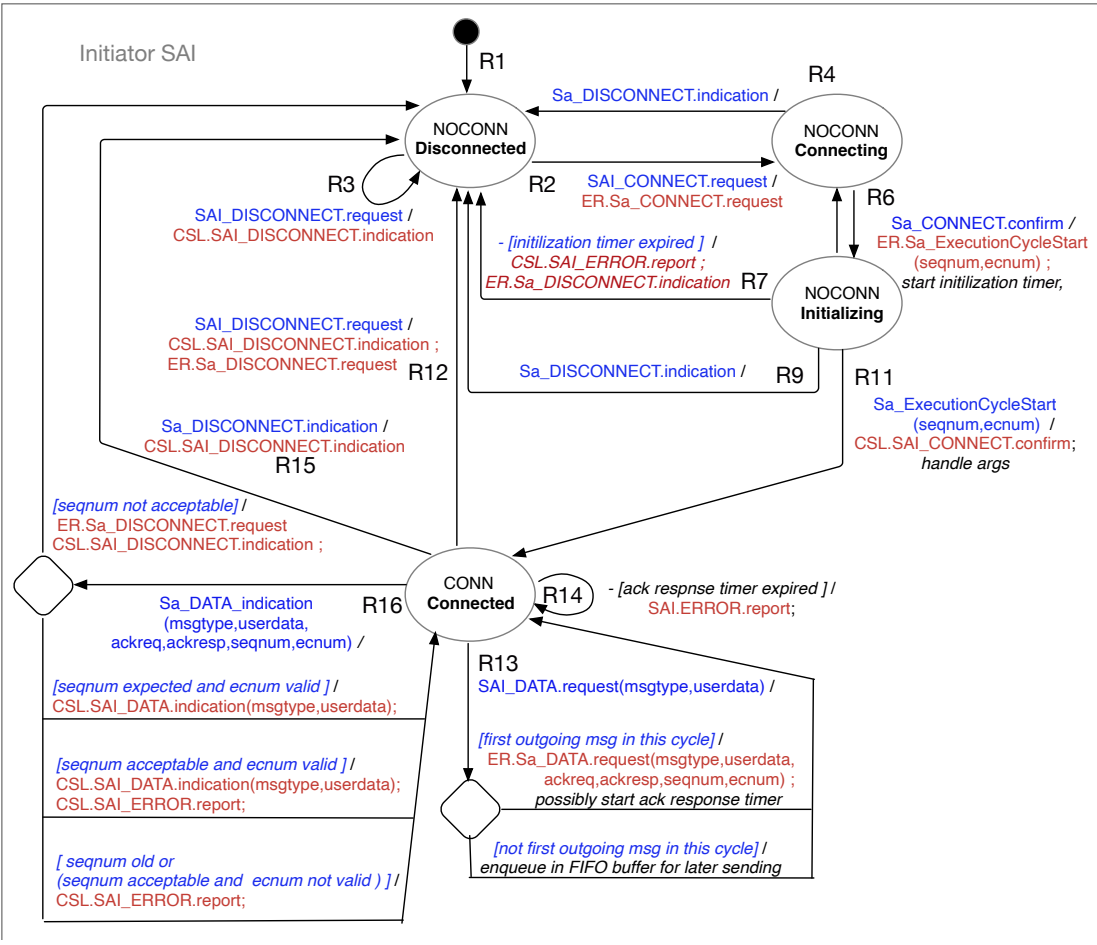
**RD3:** When in *Initializing* state, the *SAI* discards any message except for, *Sa\_DISCONNECT.indication* and *Sa\_DATA.indication* from the *ER-SL*;



**RD4:** When in *Connected* state, the SAI discards any *Sa\_ExecutionCycleStart* message from the ER-SL.

### 3 Graphical semiformal UML state machine designs









## 4 Executable UML Model

```
-----  
-- VERSION 53 26-07-2021  
-----  
--  
-- ERnice_irbcdata_V53.umc  
--  
-----  
Class I_CSL is  
-----  
  
Signals  
  -- from RBC  
  IRBC_User_Data_request(arg1: int);  
  
  -- from I_SAI  
  ISAI_CONNECT_confirm;  
  ISAI_DISCONNECT_indication;  
  ISAI_Error_report;  
  ISAI_DATA_indication(arg1: Token, arg2: int);  
  
  -- from Timer  
  icsl_tick;  
  
  -- outgoing to RBC  
  -- IRBC_User_Connect_indication;  
  -- IRBC_User_Disconnect_indication;  
  -- IRBC_User_Data_indication(arg1);  
  
  -- outgoing to SAI  
  -- ISAI_CONNECT_request;  
  -- ISAI_DISCONNECT_request;  
  -- ISAI_DATA_request(arg1,arg2);  
  
  -- outgoing to Timer  
  -- ok_icsl  
  
Vars  
  ----- PORTS  
  RBC_User: I_RBC;  
  SAI: I_SAI;  
  -----  
  
  ----- CONFIGURATION PARAMS  
  max_receiveTimer: int; -- CONFIGURATION PARAM  
  max_sendTimer: int; -- CONFIGURATION PARAM  
  max_connectTimer: int; -- CONFIGURATION PARAM  
  -----  
  
  ----- LOCAL VARS  
  receiveTimer: int := 0;  
  sendTimer: int := 0;  
  connectTimer: int := 0;  
  -----
```

Behaviour

-----  
-- R1: At startup, the CSL is in Disconnected state  
-----

```
R1_ICSL:  
initial -> NOCOMMSready
```

-----  
-- R2: When in Disconnected state, the CSL immediately  
-- sends a SAI\_CONNECT.request to the SAI component,  
-- starts a connTimer, and moves to Connecting state  
-----

```
R2_ICSL_connecting:  
  NOCOMMSready -> NOCOMMSconnecting  
  {- /  
  SAI.ISAI_CONNECT_request;  
  connectTimer := 0;}
```

-----  
-- R3: When in Connecting state the connTimer expires, the  
-- CSL moves to Disconnected state  
-----

```
RTa_ICSL_okicsl_incr:  
  NOCOMMSconnecting -> NOCOMMSconnecting  
  {icsl_tick [connectTimer < max_connectTimer ] /  
  Timer.ok_icsl;  
  connectTimer := connectTimer +1}
```

-- while connecting in case of timeout become ready to retry

```
R3_ICSL_okicsl_connect:  
  NOCOMMSconnecting -> NOCOMMSready  
  {icsl_tick [connectTimer = max_connectTimer ] /  
  Timer.ok_icsl}
```

-----  
-- R4: When in Connecting state is received a SAI\_CONNECT.confirm from  
-- the SAI component, the CSL sends a RBC\_User\_Connect.indication  
-- to the RBC component, starts both the sendTimer and the recTimer,  
-- and moves to Connected state  
-----

```
R4_ICSL_userconnind:  
  NOCOMMSconnecting -> COMMS  
  {ISAI_CONNECT_confirm /  
  RBC_User.IRBC_User_Connect_indication;  
  connectTimer := max_connectTimer;  
  receiveTimer := 0;  
  sendTimer := max_sendTimer}
```

-----  
-- R5: When in Waiting state is received a SAI\_DISCONNECT.indication  
-- from the SAI component, the CSL moves to Disconnected state  
-----

```
RTb_ICSL_okicsl_incr:  
  NOCOMMSwait -> NOCOMMSwait  
  {icsl_tick /  
  Timer.ok_icsl}
```

```
R5_ICSL_becomeready:  
  NOCOMMSwait -> NOCOMMSready  
  {ISAI_DISCONNECT_indication}
```

```
-----  
-- R6: When in Connected state the recTimer expires, the CSL sends a  
--   SAI_DISCONNECT.request to the SAI component, a  
--   RBC_User_Disconnect.indication to the RBC and moves to Waiting state  
-----
```

```
RTc_ICSL_okicsl:  
  COMMS -> COMMS  
    {icsl_tick [(receiveTimer < max_receiveTimer)  
               and (sendTimer < max_sendTimer)] /  
    Timer.ok_icsl;  
    sendTimer := sendTimer +1;  
    receiveTimer := receiveTimer+1}
```

```
R6_ICSL_okicsl:  
  COMMS -> tbcr6  
    {icsl_tick [receiveTimer = max_receiveTimer] /  
    Timer.ok_icsl}
```

```
R6_ICSL_saidisconnreq:  
  tbcr6 -> tbcr6a  
    { - /  
  SAI.ISAI_DISCONNECT_request;  
  receiveTimer := 0;  
  sendTimer := 0}
```

```
R6_ICSL_userdisconnind:  
  tbcr6a -> NOCOMMSwait  
    { - /  
    RBC_User.IRBC_User_Disconnect_indication}
```

```
-----  
-- R7: Each time that in Connected state the sendTimer expires,  
--   the CSL sends a SAI_DATA.request with a life_sign to the  
--   SAI component  
-----
```

```
R7_ICSL_okicsl:  
  COMMS -> tbcr7  
    {icsl_tick [(receiveTimer < max_receiveTimer)  
               and (sendTimer = max_sendTimer)] /  
    Timer.ok_icsl;  
    sendTimer := 0;  
    receiveTimer := receiveTimer+1}
```

```
R7_ICSL_saidatareq:  
  tbcr7 -> COMMS  
    { - /  
    SAI.ISAI_DATA_request(LifeSign,0)}
```

```
-----  
-- R8: When in Connected state is received a RBC_User_Data.request  
--   with RBC_data from the RBC component, the CSL sends a  
--   SAI_DATA.request with such RBC_data to the SAI component  
-----
```

```
R8_ICSL_saidatareq:  
  COMMS -> COMMS  
    {IRBC_User_Data_request(arg1) /  
    SAI.ISAI_DATA_request(RBCdata, arg1);  
    sendTimer := 0}
```

```
-----  
-- R9: When in Connected state is received a SAI_DATA.indication with SAI_data
```

```
--      from the SAI component, the CSL sends a RBC_User_Data.indication with
--      such SAI_data to the RBC component and restarts the recTimer
-----
```

```
R9_ICSL_userdataind:
  COMMS -> COMMS
    {ISAI_DATA_indication(arg1, arg2)
      [arg1 /= LifeSign] /
    RBC_User.IRBC_User_Data_indication(arg2);
  receiveTimer := 0}
-----
```

```
-- R10: When in Connected state is received a SAI_DATA.indication with a
--      life_sign from the SAI component, the CSL restarts the recTimer
-----
```

```
R10_ICSL_handlelifesign:
  COMMS -> COMMS
    {ISAI_DATA_indication(arg1, arg2)
      [arg1 = LifeSign] /
  receiveTimer := 0}
-----
```

```
-- R11: When in Connected state is received a SAI_DISCONNECT.indication from
--      the SAI component, the CSL sends a RBC_User_Disconnect.indication
--      to the RBC component and moves to Disconnected state
-----
```

```
R11_ICSL_userdisconnind:
  COMMS -> NOCOMMSready
    {ISAI_DISCONNECT_indication /
    RBC_User.IRBC_User_Disconnect_indication;
  receiveTimer := 0;
  sendTimer := 0}
-----
```

```
-- RD1: When in Disconnected state the CSL does not accept any
--      kind of message
--
```

```
-- RD2: When in Connecting state, the CSL discards any message
--      except for SAI_CONNECT.confirm from the SAI component
--
```

```
-- RD3: When in Waiting state, the CSL discards any message except
--      for SAI_DISCONNECT.indication from the SAI component
--
```

```
-- RD4: When in Connected state, the CSL component discards only
--      SAI_CONNECT.confirm and SAI_ERROR.report messages from
--      the SAI component
-----
```

```
RD2a_ICSL_discuserdata:
  NOCOMMSconnecting -> NOCOMMSconnecting
    {IRBC_User_Data_request(arg1)}
-----
```

```
RD2b_ICSL_discdisconnind:
  NOCOMMSconnecting -> NOCOMMSconnecting
    {ISAI_DISCONNECT_indication}
-----
```

```
RD2c_ICSL_discerrorreport:
  NOCOMMSconnecting -> NOCOMMSconnecting
    {ISAI_Error_report}
-----
```

```
RD2d_ICSL_discdataind:
  NOCOMMSconnecting -> NOCOMMSconnecting
-----
```



```
        {ISAI_DATA_indication(arg1, arg2)}  
RD3a_ICSL_discuserdata:  
    NOCOMMSwait -> NOCOMMSwait  
        {IRBC_User_Data_request(arg1)}  
RD3b_ICSL_discerrorreport:  
    NOCOMMSwait -> NOCOMMSwait  
        {ISAI_Error_report}  
RD3c_ICSL_discdataind:  
    NOCOMMSwait -> NOCOMMSwait  
        {ISAI_DATA_indication(arg1, arg2)}  
RD3d_ICSL_disconfirm:  
    NOCOMMSwait -> NOCOMMSwait  
        {ISAI_CONNECT_confirm}  
RD4a_ICSL_discommconfirm:  
    COMMS -> COMMS  
        {ISAI_CONNECT_confirm}  
RD4b_ICSL_usererror:  
    COMMS -> COMMS  
        {ISAI_Error_report}  
end I_CSL;
```

```
-----  
-----  
Class I_SAI is  
-----  
-----
```

Signals

```
-- from CSL  
ISAI_CONNECT_request;  
ISAI_DISCONNECT_request;  
ISAI_DATA_request(arg1:int,arg2:int);  
-- arg1 = message type, arg2 = RBC data value  
  
-- outgoing to CSL  
-- ISAI_DISCONNECT_indication;  
-- ISAI_CONNECT_confirm;  
-- ISAI_DATA_indication(arg1:int,arg2:int);  
-- arg1 = message type, arg2 = RBC data value  
-- ISAI_Error_report;  
  
-- from ER  
ISAI_SA_Connect_confirm;  
ISAI_SA_Disconnect_indication;  
ISAI_SA_Data_indication  
    (arg1:int,arg2:int,arg3:int,arg4:int,arg5:int,arg6:int);  
-- arg1 = message type, arg2 = data value,  
-- arg3 = ack request, arg4 = ack response,  
-- arg5 = sequence number, arg6 = EC number  
ISAI_SA_Execution_Cycle_Start(arg1:int,arg2:int);  
  
-- outgoing to ER  
-- ISAI_SA_Connect_request;  
-- ISAI_SA_Disconnect_request;  
-- ISAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6);  
-- arg1 = message type, arg2 = data value,  
-- arg3 = ack request, arg4 = ack response,  
-- arg5 = sequence number, arg6 = EC number  
-- ISAI_SA_Execution_Cycle(arg1:int,arg2:int);  
-- arg1 = sequence number, arg2 = EC counter  
  
-- from Timer  
isai_tick;  
  
-- outgoing to Timer  
-- ok_isai;
```

Vars

```
----- PORTS  
CSL: I_CSL;  
ER: EuroRadio;  
-----  
  
----- CONFIGURATION PARAMS  
K: int; -- ec delay limit  
max_initTimer: int; -- init timeout  
N: int; -- msg loss limit  
M: int; -- seq num limit  
max_ack_requestTimer: int; -- ack request period  
max_ack_responseTimer: int; -- ack response timeout  
Mec: int; -- ec counter limit  
-----
```

```

----- LOCAL VARS
sarg1:          int := 0;
sarg2:          int := 0;
sarg4:          int := 0;
-----
-- for ECS procedure --
-----
initTimer:      int := 0;
OFFSET:         int := 0;
EC_expected:    int := 0;
DELTA:          int := 0;
currentEC:      int := 0;
-----
-- for sequence number --
-----
last_in: int := 0;
next_out: int := 0;
dist: int := 0;
-----
-- for ACK procedure --
-----
ack_requestTimer: int := 0;
ack_responseTimer: int := 0;
ack_reply: int := 0;
ack_request: int := 0;
-----
-- for DATA storing
outdatabuff: int[] := [];
waitnextcycle: bool := False;

```

#### Behaviour

```

-----
-- R1: At startup the SAI is in Disconnected state
-----

```

```

R1_ISAI:
  initial -> Disconnected

```

```

-----
-- R2: When in Disconnected state is received a SAI_CONNECT.request
--      from the CSL component, the SAI sends a Sa_CONNECT.request
--      to the ER-SL and moves to Connecting state
-----

```

```

R2_ISAI_connreq:
  Disconnected -> Connecting
    {ISAI_CONNECT_request /
    ER.ISAI_SA_Connect_request}

```

```

-----
-- R3: When in Disconnected state is received a SAI_DISCONNECT.request
--      from the CSL component, the SAI replies with a
--      SAI_DISCONNECT.indication to the CSL component.
--      This is necessary to avoid deadlocks in the CSL component, when the CSL is
--      NOCOMM_waiting state.
-----

```

```

R3_ISAI_disconndiscard:
  Disconnected -> Disconnected
    {ISAI_DISCONNECT_request /
    CSL.ISAI_DISCONNECT_indication}

```



-----  
-- R4: When in Connecting state is received a Sa\_DISCONNECT.indication  
-- from the ER-SL, the SAI moves to Disconnected state  
-----

```
RTg_ISAI_okisai:  
    Connecting -> Connecting  
        {isai_tick /  
         Timer.ok_isai}
```

```
R4_ISAI_abortconn:  
    Connecting -> Disconnected  
        {ISAI_SA_Disconnect_indication}
```

-----  
-- R5: When in Connecting state is received a SAI\_DISCONNECT.request  
-- the message is discarded (it should never happen)  
-----

```
R5_ISAI_CSLdisconnind:  
    Connecting -> Connecting  
        {ISAI_DISCONNECT_request}
```

-----  
-- R6: When in Connecting state is received a Sa\_CONNECT.confirm from the ER-SL,  
-- the SAI replies with a Sa\_ExecutionCycle(seqnum, ecnum) to the ER-SL and  
-- moves to the Initializing state, waiting for a Sa\_ExecutionCycleStart  
-- message from the ER-SL within a maximum initialization delay.  
-- For the management of the Sa\_ExecutionCycleStart parameters see the  
-- SEQ\_NUMs and REQ\_ECNUMs Management sections  
-----

```
R6_ISAI_confinitproc:  
    Connecting -> Initializing  
        {ISAI_SA_Connect_confirm /  
         ER.ISAI_SA_Execution_Cycle(next_out,currentEC);  
         next_out := (next_out + 1) mod M;  
         initTimer := 0}
```

-----  
-- R7: When in Initializing state the maximum initialization delay expires,  
-- the SAI sends an SAI\_ERROR.report to the CSL component, a  
-- Sa\_DISCONNECT.request to the ER-SL and moves to Disconnected state  
-----

```
RTa_ISAI_confinitprocwait:  
    Initializing -> tbcrt  
        {isai_tick /  
         Timer.ok_isai;  
         initTimer := initTimer + 1;  
         currentEC := (currentEC + 1) mod Mec}
```

```
R7_ISAI_confinitprocwaiterrorr:  
    tbcrt -> tbcrt7a  
        { - [initTimer = max_initTimer] /  
         CSL.ISAI_Error_report}
```

```
R7_ISAI_confinitprocwaitdisconn:  
    tbcrt7a -> Disconnected  
        { - / ER.ISAI_SA_Disconnect_request;  
         initTimer := 0;  
         currentEC := 0}
```

```
RTb_ISAI_connreqdiscard:
```

```
tbcrt -> Initializing
      { - [initTimer < max_initTimer]}

-----
-- R8: When in Initializing state is received a SAI_DISCONNECT.request from the
--     CSL component, the message is discarded (it should never happen).
-----

R8_ISAI_CSLdisconnind:
  Initializing -> Initializing
    {ISAI_DISCONNECT_request / }

-----
-- R9: When in Initializing state is received a Sa_DISCONNECT.indication
--     from the ER-SL the SAI moves to Disconnected state
-----

R9_ISAI_abortconn:
  Initializing -> Disconnected
    {ISAI_SA_Disconnect_indication /
     initTimer := 0;
     currentEC := 0}

-----
-- R10: When in Initializing state is received a SAI_CONNECT.request from
--       the CSL component, the message is discarded
-----

R10_ISAI_connreqdiscard:
  Initializing -> Initializing
    {ISAI_CONNECT_request}

-----
-- R11: When in Initializing state is received a
--     Sa_ExecutionCycleStart(seqnum, ecnum) from the ER-SL, the SAI sends
--     a SAI_CONNECT.confirm to the CSL component and moves to Connected state.
--     The received seqnum is accepted as initial remote sequence number and the
--     ecnum is accepted as initial value of the remote execution cycle counter.
--     The execution_cycle_OFFSET variable is set as the difference between the
--     current execution cycle counter and the received execution cycle counter.
--     While the last_received_sequence_number variable is set to the received
--     sequence number
-----

R11_ISAI_confinitproc:
  Initializing -> Connected
    {ISAI_SA_Execution_Cycle_Start(arg1,arg2) /
     CSL.ISAI_CONNECT_confirm;
     OFFSET := currentEC - arg2;
     initTimer := 0;
     ack_requestTimer := 0;
     ack_responseTimer := max_ack_responseTimer + 1;
     ack_reply := 0;
     last_in := arg1}

-----
-- R12: When in Connected state is received a SAI_DISCONNECT.request from the
--     CSL component, the SAI replies with a SAI_DISCONNECT.indication to the
--     CSL component, sends a Sa_DISCONNECT.request to the ER-SL, and moves
--     to Disconnected state
-----

R12_ISAI_CSLdisconnind:
  Connected -> tbcrt12
```

```
{ISAI_DISCONNECT_request /
CSL.ISAI_DISCONNECT_indication}
```

```
R12_ISAI_CSLdisconnreq:
  tbcrl2 -> Disconnected
  { - /
  ER.ISAI_SA_Disconnect_request;
  ack_requestTimer := max_ack_requestTimer;
  ack_responseTimer := 0;
  ack_reply := 0;
  currentEC := 0;
  outdatabuff := [];
  ack_request := 0;
  waitnextcycle := False}
```

```
-----
-- R13a: When in Connected state is received a SAI_DATA_request(msgtype, data)
-- from the CSL component, and yet no other data message has been sent in
-- this cycle, the SAI sends a
-- Sa_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)
-- to the ER-SL. The ackreq and ackresp parameters are set according to
-- REQ_ACKs.
-- The seqnum parameter is set according to SEQ_NUMs and the ecnum parameter
-- is set according to REQ_ECNUMs Management
-----
```

```
R13a_ISAI_datareqforward:
  Connected -> Connected
  {ISAI_DATA_request(arg1,arg2)
  [waitnextcycle = False]/

  ER.ISAI_SA_Data_request(arg1,arg2,ack_request,ack_reply,next_out,currentEC);
  next_out := (next_out + 1) mod M;
  if (ack_request = 1)
  {ack_request := 0;
  ack_requestTimer := 0;
  ack_responseTimer := 0};
  ack_reply := 0;
  waitnextcycle := True}
```

```
-----
-- R13b: When in Connected state is received a SAI_DATA.request(msgtype, data)
-- from the CSL component, but another data message has already been
-- sent in this cycle, the SAI_DATA.request is saved in a FIFO dataout
-- buffer (see also the REQ_OUTDATABUFFs Management)
-----
```

```
R13b_ISAI_datareqstore:
  Connected -> Connected
  {ISAI_DATA_request(arg1,arg2)
  [waitnextcycle = True]/
  outdatabuff := outdatabuff + [arg1,arg2]}
```

```
-----
-- R14: When in Connected state is received a Sa_DISCONNECT.indication from
-- the ER-SL, the SAI sends a SAI_DISCONNECT.indication to the CSL
-- component and moves to Disconnected state
-----
```

```
R14_ISAI_abortconn:
  Connected -> Disconnected
  {ISAI_SA_Disconnect_indication /
  CSL.ISAI_DISCONNECT_indication;
  ack_requestTimer := max_ack_requestTimer;
```

```

ack_responseTimer := 0;
ack_reply := 0;
outdatabuff := [];
ack_request := 0;
waitnextcycle := False}

```

```

-----
-- R15: When in Connected state is received a
-- Sa_DATA.indication(msgtype, data, ackreq, ackresp, seqnum, ecnum) from
-- the ER-SL we can have four cases, depending on the received seqnum
-- and ecnum values (see SEQ_NUMS and REC_ECNUMS Management): *
-----

```

```

R15_ISAI_datareceive:
  Connected -> tbc15
    {ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6) /
    dist := arg5 - last_in;
    if (dist < -M/2) then {dist := dist + M };
    if (dist > M/2) then {dist := dist - M };
    if ((dist > 0) and (dist <= N)) then
      {last_in := arg5};
    sarg1 := arg1;
    sarg2 := arg2;
    ack_reply := ack_reply + arg3 - ack_reply*arg3;
    sarg4 := arg4;
    EC_expected := (currentEC + Mec - OFFSET) mod Mec;
    DELTA := EC_expected - arg6;
    if (DELTA < -Mec/2) then {DELTA := DELTA + Mec };
    if (DELTA > Mec/2) then {DELTA := DELTA - Mec };
    EC_expected := 0}

```

```

-----
-- Case 1
-- * The seqnum is the one EXPECTED and ecnum is VALID: In this case the
-- SAI sends a SAI_DATA.indication(msgtype, data) to the CSL component
-----

```

```

R15a_ISAI_dataindforward:
  tbc15 -> Connected
    { - [dist = 1 and DELTA < K] /
    CSL.ISAI_DATA_indication(sarg1,sarg2);
    if((sarg4 = 1) and (ack_responseTimer < max_ack_responseTimer))
      {ack_responseTimer := max_ack_responseTimer + 1};
    sarg1 := 0;
    sarg2 := 0;
    sarg4 := 0;
    dist := 0;
    DELTA := 0}

```

```

-----
-- Case 2
-- * The seqnum is ACCEPTABLE and the ecnum is VALID: in this case the
-- SAI sends a SAI_DATA.indication(msgtype, data) and a SAI_ERROR.report
-- to the CSL component
-----

```

```

R15b_ISAI_dataindforward:
  tbc15 -> tbc15b
    { - [dist > 1 and dist <= N and DELTA < K] /
    CSL.ISAI_DATA_indication(sarg1,sarg2);
    if((sarg4 = 1) and (ack_responseTimer < max_ack_responseTimer))
      {ack_responseTimer := max_ack_responseTimer + 1};
    sarg1 := 0;
    sarg2 := 0;

```

```
sarg4 := 0;
dist := 0;
DELTA := 0}
```

```
R15b_ISAI_datainderror:
  tbc15b -> Connected
  { - /
  CSL.ISAI_Error_report;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  dist := 0;
  DELTA := 0}
```

```
-----
-- Case 3
-- * The seqnum is OLD or (the seqnum is ACCEPTABLE and the eenum is VALID):
-- In this case the SAI sends a SAI_ERROR.report to the CSL component and
-- discards the Sa_DATA.indication message
-----
```

```
R15c_ISAI_datainvalid_or_old:
  tbc15 -> Connected
  { - [dist < 1 or (dist <= N and DELTA >= K)] /
  CSL.ISAI_Error_report;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  dist := 0;
  DELTA := 0}
```

```
-----
-- Case 4
-- * The seqnum is NOT_ACCEPTABLE: In this case the SAI component sends a
-- Sa_DISCONNECT.request to ER-SL and a SAI_DISCONNECT.indication to the CSL
-- component, and then moves to Disconnected state
-----
```

```
R15d_ISAI_abortconn:
  tbc15 -> tbc15d
  { - [dist > N] /
  ER.ISAI_SA_Disconnect_request}
```

```
R15d_ISAI_notifyfatal:
  tbc15d -> Disconnected
  { - /
  CSL.ISAI_DISCONNECT_indication;
  ack_responseTimer := 0;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  next_out := 0;
  dist := 0;
  DELTA := 0;
  outdatabuff := [];
  ack_request := 0;
  waitnextcycle := False}
```

```
RTf_ISAI_okisai:
  Disconnected -> Disconnected
  { isai_tick /
  Timer.ok_isai}
```

```
RD1a_ISAI_datadiscard:
```

```
Disconnected -> Disconnected
    {ISAI_DATA_request(arg1,arg2)}

RD1b_ISAI_connconfdiscard:
    Disconnected -> Disconnected
    {ISAI_SA_Connect_confirm}

RD1c_ISAI_discinddiscard:
    Disconnected -> Disconnected
    {ISAI_SA_Disconnect_indication}

RD1d_ISAI_datainddiscard:
    Disconnected -> Disconnected
    {ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6)}

RD1e_ISAI_disconndiscard:
    Disconnected -> Disconnected
    {ISAI_SA_Execution_Cycle_Start(arg1,arg2)}

RD2a_ISAI_datainddiscard:
    Connecting -> Connecting
    {ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6)}

RD2b_ISAI_datareqdiscard:
    Connecting -> Connecting
    {ISAI_DATA_request(arg1,arg2)}

RD2c_ISAI_connreqdiscard:
    Connecting -> Connecting
    {ISAI_CONNECT_request}

RD2d_ISAI_ecsdiscard:
    Connecting -> Connecting
    {ISAI_SA_Execution_Cycle_Start(arg1,arg2)}

RD3a_ISAI_datainddiscard:
    Initializing -> Initializing
    {ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6)}

RD3b_ISAI_datareqdiscard:
    Initializing -> Initializing
    {ISAI_DATA_request(arg1,arg2)}

RD3c_ISAI_connconfdiscard:
    Initializing -> Initializing
    {ISAI_SA_Connect_confirm}

RD4a_ISAI_connconfdiscard:
    Connected -> Connected
    {ISAI_SA_Connect_confirm}

RD4b_ISAI_connreqdiscard:
    Connected -> Connected
    {ISAI_CONNECT_request}

RD4c_ISAI_ecsdiscard:
    Connected -> Connected
    {ISAI_SA_Execution_Cycle_Start(arg1,arg2)}
```

```
-----
-- Each time that in Connected state the set_ack_response expires,
-- the SAI sends a SAI_ERROR.report to the CSL component
-- (for further details see REQ_ACKs)
-----
```

```
RTc_ISAI_okisai:
  Connected -> tbcrtta
    {isai_tick /
    Timer.ok_isai;
    ----- ack timing management
    if (ack_responseTimer < max_ack_responseTimer)
      {ack_responseTimer := ack_responseTimer + 1};
    if (ack_requestTimer < max_ack_requestTimer)
      {ack_requestTimer := ack_requestTimer + 1};
    if (ack_requestTimer = max_ack_requestTimer and
        ack_responseTimer >= max_ack_responseTimer)
      {ack_request := 1};
    ----- ec timing management
    currentEC := (currentEC + 1) mod Mec;
    -----
    waitnextcycle := False}

RACK2_ISAI_ackresponserror:
  tbcrtta -> tbcrtack
    { - [ack_responseTimer = max_ack_responseTimer] /
      CSL.ISAI_Error_report;
      ack_responseTimer := max_ack_responseTimer + 1}

ROUTDATABUFF_ISAI_sendbuffered:
  tbcrtack -> Connected
    { - [ outdatabuff /= [] ] /
    ER.ISAI_SA_Data_request(
      outdatabuff.head, outdatabuff.tail.head,
      ack_request,ack_reply,next_out,currentEC);
    outdatabuff := outdatabuff.tail.tail;
    waitnextcycle := True;
    next_out := (next_out + 1) mod M;
    if (ack_request = 1)
      {ack_request := 0;
      ack_requestTimer := 0;
      ack_responseTimer := 0};
    ack_reply := 0;
    }

RTd_ISAI_continue:
  tbcrtta -> tbcrtack
    { - [ack_responseTimer /= max_ack_responseTimer] }

RTe_ISAI_okisai:
  tbcrtack -> Connected
    { - [ outdatabuff = [] ]}

end I_SAI;
```

```
-----  
-----  
Class C_SAI is  
-----  
-----  
Signals  
  
-- from ER  
CSAI_SA_Connect_indication;  
CSAI_SA_Disconnect_indication;  
CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6:int);  
-- arg1 = message type, arg2 = data value,  
-- arg3 = ack request, arg4 = ack response,  
-- arg5 = sequence number, arg6 = EC number  
  
CSAI_SA_Execution_Cycle_Start(arg1:int,arg2:int);  
-- arg1 = sequence number, arg2 = EC counter  
  
-- from CSL  
CSAI_DISCONNECT_request;  
CSAI_DATA_request(arg1:int,arg2:int);  
-- arg1 = message type, arg2 = RBC data value  
  
-- from Timer  
csai_tick;  
  
-- outgoing to ER  
-- CSAI_SA_Connect_response;  
-- CSAI_SA_Disconnect_request;  
-- CSAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6);  
-- arg1 = message type, arg2 = data value,  
-- arg3 = ack request, arg4 = ack response,  
-- arg5 = sequence number, arg6 = EC number  
-- CSAI_SA_Execution_Cycle(arg1,arg2);  
-- arg1 = sequence number, arg2 = EC counter  
  
-- outgoing to CSL  
-- CSAI_CONNECT_indication;  
-- CSAI_DISCONNECT_indication;  
-- CSAI_DATA_indication(arg1:int,arg2:int);  
-- arg1 = message type, arg2 = RBC data value  
-- CSAI_Error_report;  
  
-- outgoing to Timer  
-- ok_csai  
  
Vars  
----- PORTS  
CSL: C_CSL;  
ER: EuroRadio;  
-----  
  
----- CONFIGURATION PARAMS  
K: int; -- ec delay limit  
max_initTimer: int; -- init timeout  
N: int; -- msg loss limit  
M: int; -- seq num limit  
max_ack_requestTimer: int; -- ack request period  
max_ack_responseTimer: int; -- ack response timeout  
Mec: int; -- ec counter limit  
-----  
  
----- LOCAL VARS  
sarg1: int := 0;
```



```

sarg2:          int := 0;
sarg3:          int := 0;
sarg4:          int := 0;
-----
-- for ECS procedure --
-----
initTimer:     int := 0;
OFFSET:        int := 0;
EC_expected:   int := 0;
DELTA:         int := 0;
currentEC:     int := 0;
-----
-- for sequence number --
-----
last_in:       int := 0;
next_out:      int := 0;
dist:         int := 0;
-----
-- for ACK procedure --
-----
ack_requestTimer: int := 0;
ack_responseTimer: int := 0;
ack_reply:      int := 0;
ack_request:    int := 0;
-----
-- for DATA storing
outdatabuff:   int[] := [];
waitnextcycle: bool := False;

```

#### Behaviour

```

-----
-- R1: At startup, the SAI is in Disconnected state
-----
R1_CSAI_startup:
initial -> Disconnected

-----
-- R2: When in Disconnected state is received a Sa_CONNECT.indication from
--      the ER-SL, the SAI replies with a Sa_CONNECT.response to the ER-SL
--      and moves to Connecting state
-----
RTf_CSAI_okcsai:
  Disconnected -> Disconnected
  { csai_tick /
    Timer.ok_csai}

R2_CSAI_connreq:
  Disconnected -> Connecting
  {CSAI_SA_Connect_indication /
   ER.CSAI_SA_Connect_response}

-----
-- R3: When in Connecting state is received a Sa_DISCONNECT.indication from
--      the ER-SL, the SAI moves to Disconnected state
-----
RTg_CSAI_okcsai:
  Connecting -> Connecting
  {csai_tick /
   Timer.ok_csai}

R3_CSAI_abortconn:

```

```

Connecting -> Disconnected
    {CSAI_SA_Disconnect_indication }

-----
-- R4: When in Connecting state is received a SAI_DISCONNECT.request from the
--     CSL component, the message is discarded (it should never happen).
-----

R4_CSAI_CSLdisconnind:
    Connecting -> Connecting
    {CSAI_DISCONNECT_request}

-----
-- R2b: When in Connecting state is received a SAI_SA_Connect.request from the
--     ER sublayer, the connection process is restarted.
-----

R2b_CSAI_restartconnect:
    Connecting -> Connecting
    {CSAI_SA_Connect_indication /
    ER.CSAI_SA_Connect_response}

-----
-- R5: When in Connecting state is received a
--     Sa_ExecutionCycleStart(seqnum, ecnum)
--     from the ER-SL, the SAI replies with a Sa_ExecutionCycle(seqnum, ecnum)
--     to the ER-SL, starts an initTimer set to the maximum initialization
--     delay, and moves to Initializing state. For the management of the
--     Sa_ExecutionCycleStart parameters see the following Sequence Numbers
--     Management and Execution Cycle Counters Management sections
-----

R5_CSAI_confinitproc:
    Connecting -> Initializing
    {CSAI_SA_Execution_Cycle_Start(arg1,arg2) /
    ER.CSAI_SA_Execution_Cycle(next_out,currentEC);
    OFFSET := currentEC - arg2;
    initTimer := 0;
    last_in := arg1;
    next_out := (next_out +1) mod M}

-----
-- R6: When in Initializing state the maximum initialization delay expires, the
--     SAI sends a SAI_ERROR.report to the CSL component, Sa_Disconnect_request
--     to ER, and moves to Disconnected state
-----

RTa_CSAI_confinitprocwait:
    Initializing -> tbcrt_a
    {csai_tick /
    Timer.ok_csai;
    initTimer := initTimer + 1;
    currentEC := (currentEC + 1) mod Mec}

R6_CSAI_confinitfail:
    tbcrt_a -> tbcrt_6
    { - [initTimer = max_initTimer] /
    ER.CSAI_SA_Disconnect_request }

R6_CSAI_confinitprocwait:
    tbcrt_6 -> Disconnected
    { - /
    CSL.CSAI_Error_report;
    ack_requestTimer := max_ack_requestTimer;
    ack_responseTimer := 0;

```

```
ack_reply := 0;
ack_request := 0;
currentEC := 0;
initTimer := 0;
OFFSET := 0;
-- EC_expected := 0;
DELTA := 0;
dist := 0;
last_in := 0;
next_out := 0;
outdatabuff := [];
waitnextcycle := False}

-- when initializing wait first data
RTb_CSAI_waitdata:
  tbcrt -> Initializing
  { - [initTimer < max_initTimer] }

-----
-- R2c: When in Connecting state is received a SAI_SA_Connect.request from the
-- ER sublayer, the connection process is restarted.
-----
R2c_CSAI_restartconnect:
  Initializing -> Connecting
  {CSAI_SA_Connect_indication /
  ER.CSAI_SA_Connect_response;
  ack_requestTimer := max_ack_requestTimer;
  ack_responseTimer := 0;
  ack_reply := 0;
  ack_request := 0;
  currentEC := 0;
  initTimer := 0;
  OFFSET := 0;
  -- EC_expected := 0;
  DELTA := 0;
  dist := 0;
  last_in := 0;
  next_out := 0}

-----
-- R7: When in Initializing state is received a SAI_DISCONNECT.request from the
-- CSL component, the message is discarded (it should never happen)
-----
R7_CSAI_CSLdisconnind:
  Initializing -> Initializing
  {CSAI_DISCONNECT_request }

-----
-- R8: When in Initializing state is received a Sa_DISCONNECT.indication from
-- the ER-SL, the SAI moves to the Disconnected state
-----
R8_CSAI_abortconn:
  Initializing -> Disconnected
  {CSAI_SA_Disconnect_indication /
  ack_requestTimer := max_ack_requestTimer;
  ack_responseTimer := 0;
  ack_reply := 0;
  ack_request := 0;
  currentEC := 0;
  initTimer := 0;
```

```

OFFSET := 0;
-- EC_expected := 0;
DELTA := 0;
dist := 0;
last_in := 0;
next_out := 0;
outdatabuff := [];
waitnextcycle := False}

```

```

-----
-- R9: When in Initializing state is received a Sa_DATA.indication
-- from the ER-SL may have four cases, depending on the received seqnum
-- and ecnum values (see REQ_SEQ_NUMs and REC_ECNUMs): *
-----

```

```

R9_CSAI_confinitproc:
  Initializing -> tbc9
  {CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6) /
  dist := arg5 - last_in;
  if (dist < -M/2) then {dist := dist + M };
  if (dist > M/2) then {dist := dist - M };
  if ((dist > 0) and (dist <= N)) then
    {last_in := arg5};
  sarg1 := arg1;
  sarg2 := arg2;
  sarg3 := arg3;
  EC_expected := (currentEC + Mec - OFFSET) mod Mec;
  DELTA := EC_expected - arg6;
  if (DELTA < -Mec/2) then {DELTA := DELTA + Mec };
  if (DELTA > Mec/2) then {DELTA := DELTA - Mec };
  EC_expected := 0}

```

```

-----
-- Case 1
-- The seqnum is the one EXPECTED and ecnum is VALID : In this case the SAI
-- moves to Connected state and sends both a SAI_CONNECT.indication and a
-- SAI_DATA.indication(msgtype, data) to the CSL component
-----

```

```

R9a_CSAI_connecting:
  tbc9 -> tbc9a
  { - [dist = 1 and DELTA < K] /
  CSL.CSAI_CONNECT_indication;
  ack_reply := sarg3;
  sarg3 := 0;
  initTimer := 0 ;
  dist := 0;
  DELTA := 0}

R9a_CSAI_forwarding:
  tbc9a -> Connected
  { - /
  CSL.CSAI_DATA_indication(sarg1,sarg2);
  ack_requestTimer := 0;
  ack_responseTimer := max_ack_responseTimer + 1;
  sarg1 := 0;
  sarg2 := 0}

```

```

-----
-- Case 2
-- The seqnum is ACCEPTABLE and the ecnum is VALID: in this case the SAI moves
-- to Connected state and sends a SAI_CONNECT.indication, a
-- SAI_DATA.indication

```

```
-- and a SAI_ERROR.report to the CSL component
```

```
-----  
R9b_CSAI_connecting:  
  tbcr9 -> tbcr9b  
  { - [dist > 1 and dist <= N and DELTA < K] /  
    CSL.CSAI_CONNECT_indication;  
    ack_reply := sarg3;  
    sarg3 := 0;  
    initTimer := 0 ;  
    dist := 0;  
    DELTA := 0}
```

```
R9b_CSAI_forwarding:  
  tbcr9b -> tbcr9ba  
  { - /  
    CSL.CSAI_DATA_indication(sarg1,sarg2);  
    sarg1 := 0;  
    sarg2 := 0}
```

```
R9b_CSAI_reportingding:  
  tbcr9ba -> Connected  
  { - /  
    CSL.CSAI_Error_report;  
    ack_requestTimer := 0;  
    ack_responseTimer := max_ack_responseTimer + 1}
```

```
-----  
-- Case 3  
-- The seqnum is OLD or (the seqnum is ACCEPTABLE and the ecnum is VALID):  
-- In this case the SAI sends a SAI_ERROR.report to the CSL component and  
-- discards the Sa_DATA.indication message
```

```
-----  
R9c_CSAI_discarding:  
  tbcr9 -> Initializing  
  { - [dist < 1 or (dist <= N and DELTA >= K)] /  
    CSL.CSAI_Error_report;  
    sarg1 := 0;  
    sarg2 := 0;  
    sarg3 := 0;  
    dist := 0;  
    DELTA := 0}
```

```
-----  
-- Case 4  
-- * The seqnum is NOT_ACCEPTABLE: in this case the SAI component sends a  
-- Sa_DISCONNECT.request to ER-SL and moves to Disconnected state
```

```
-----  
R9d_CSAI_disconnecting:  
  tbcr9 -> Disconnected  
  { - [dist > N] /  
    ER.CSAI_SA_Disconnect_request;  
    sarg1 := 0;  
    sarg2 := 0;  
    sarg3 := 0;  
    ack_requestTimer := max_ack_requestTimer;  
    ack_responseTimer := 0;  
    ack_reply := 0;  
    ack_request := 0;  
    currentEC := 0;  
    initTimer := 0;  
    OFFSET := 0;
```

```
-- EC_expected := 0;  
DELTA := 0;  
dist := 0;  
last_in := 0;  
next_out := 0;  
outdatabuff := [];  
waitnextcycle := False}
```

```
-----  
-- R10: When in Connected state is received a SAI_DISCONNECT.request from the  
-- CSL component, the SAI replies with a SAI_DISCONNECT.indication to the  
-- CSL component, sends a Sa_DISCONNECT.request to the ER-SL, and moves to  
-- Disconnected state  
-----
```

```
R10_CSAI_CSLdisconnind:  
  Connected -> tbcrl0  
  {CSAI_DISCONNECT_request /  
   CSL.CSAI_DISCONNECT_indication}  
  
R10_CSAI_CSLdisconnreq:  
  tbcrl0 -> Disconnected  
  { - /  
   ER.CSAI_SA_Disconnect_request;  
   ack_requestTimer := max_ack_requestTimer;  
   ack_responseTimer := 0;  
   ack_reply := 0;  
   ack_request := 0;  
   currentEC := 0;  
   initTimer := 0;  
   OFFSET := 0;  
   -- EC_expected := 0;  
   DELTA := 0;  
   dist := 0;  
   last_in := 0;  
   next_out := 0;  
   outdatabuff := [];  
   waitnextcycle := False}
```

```
-----  
-- R11: When in Connected state is received a Sa_DISCONNECT.indication from the  
-- ER-SL, the SAI sends a SAI_DISCONNECT.indication to the CSL component  
-- and moves to Disconnected state  
-----
```

```
R11_CSAI_abortconn:  
  Connected -> Disconnected  
  {CSAI_SA_Disconnect_indication /  
   CSL.CSAI_DISCONNECT_indication;  
   ack_requestTimer := max_ack_requestTimer;  
   ack_responseTimer := 0;  
   ack_reply := 0;  
   ack_request := 0;  
   currentEC := 0;  
   initTimer := 0;  
   OFFSET := 0;  
   -- EC_expected := 0;  
   DELTA := 0;  
   dist := 0;  
   last_in := 0;  
   next_out := 0;  
   outdatabuff := [];  
   waitnextcycle := False}
```

```
-----
-- R12: When in Connected state is received a Sa_CONNECT.indication from the
-- ER-SL, the SAI replies with a Sa_CONNECT.response to the ER-SL, sends a
-- SAI_DISCONNECT.indication to the CSL component, and moves to Connecting
-- state
-----
```

```
R12_CSAI_connconfresp:
  Connected -> tbcrl2
    {CSAI_SA_Connect_indication /
     ER.CSAI_SA_Connect_response}
```

```
R12_CSAI_connconfdiscard:
  tbcrl2 -> Connecting
    { - /
     CSL.CSAI_DISCONNECT_indication;
     ack_requestTimer := max_ack_requestTimer;
     ack_responseTimer := 0;
     ack_reply := 0;
     ack_request := 0;
     currentEC := 0;
     initTimer := 0;
     OFFSET := 0;
     -- EC_expected := 0;
     DELTA := 0;
     last_in := 0;
     next_out := 0;
     dist := 0;
     outdatabuff := [];
     waitnextcycle := False;}
```

```
-----
-- R13a: When in Connected state is received a SAI_DATA.request(msgtype, data)
-- from the CSL component, and yet no other data message has been sent in
-- this cycle, the SAI sends a
-- Sa_DATA.request(msgtype, data, ackreq, ackresp, seqnum, ecnum)
-- to the ER-SL.
-- The ackreq and ackresp parameters are set according to REQ_ACKs.
-- The seqnum parameter is set according to SEQ_NUMs and the ecnum parameter
-- is set according to REQ_ECNUMs
-----
```

```
R13a_CSAI_datareqforward:
  Connected -> Connected
    {CSAI_DATA_request(arg1,arg2)
     [waitnextcycle = False]/
     ER.CSAI_SA_Data_request(arg1,arg2,1,ack_reply,next_out,currentEC);
     next_out := (next_out + 1) mod M;
     if (ack_request = 1)
       {ack_request := 0;
        ack_requestTimer := 0;
        ack_responseTimer := 0 };
     ack_reply := 0;
     waitnextcycle := True}
```

```
-----
-- R13b: When in Connected state is received a SAI_DATA.request(msgtype, data)
-- from the CSL component, but another data message has already been sent in
-- this cycle, the SAI_DATA.request is saved in a FIFO dataout buffer
-- (see also REQ_OUTDATABUFF)
-----
```

```
R13b_CSAI_datareqstore:
  Connected -> Connected
```

```
{CSAI_DATA_request(arg1,arg2)
  [waitnextcycle = True]/
  outdatabuff := outdatabuff +[arg1,arg2]}
```

```
-----
-- R14: When in Connected state is received a Sa_DATA.indication from the ER-SL
-- we can have four cases, depending on the received seqnum and ecnum values
-- (see SEQ_NUMS and REC_ECNUMS):
-----
```

```
R14_CSAI_datareceive:
  Connected -> tbcrl4
  {CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6) /
  dist := arg5 - last_in;
  if (dist < -M/2) then {dist := dist + M };
  if (dist > M/2) then {dist := dist - M };
  if ((dist > 0) and (dist <= N)) then
    {last_in := arg5};
  sarg1 := arg1;
  sarg2 := arg2;
  ack_reply := ack_reply + arg3 - ack_reply*arg3;
  sarg4 := arg4;
  EC_expected := (currentEC + Mec - OFFSET) mod Mec;
  DELTA := EC_expected - arg6;
  if (DELTA < -Mec/2) then {DELTA := DELTA + Mec };
  if (DELTA > Mec/2) then {DELTA := DELTA - Mec };
  EC_expected := 0}
```

```
-----
-- Case 1
-- * The seqnum is the one EXPECTED and ecnum is VALID: In this case the SAI
-- sends a SAI_DATA.indication(msgtype, data) to the CSL component.
-- Depending on the received values of the ackreq and ackresp parameters,
-- appropriate actions are performed (see REQ_ACKS)
-----
```

```
R14a_CSAI_dataindforward:
  tbcrl4 -> Connected
  { - [dist = 1 and DELTA < K] /
  CSL.CSAI_DATA_indication(sarg1,sarg2);
  if((sarg4 = 1) and (ack_responseTimer < max_ack_responseTimer))
    {ack_responseTimer := max_ack_responseTimer + 1};
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  dist := 0;
  DELTA := 0}
```

```
-----
-- Case 2
-- * The seqnum is ACCEPTABLE and the ecnum is VALID: in this case the SAI
-- sends a SAI_DATA.indication(msgtype, data) and a SAI_ERROR.report to the
-- CSL component.
-- Depending on the received values of the ackreq and ackresp parameters,
-- appropriate actions are performed (see REQ_ACKS)
-----
```

```
R14b_CSAI_dataindforward:
  tbcrl4 -> tbcrl4b
  { - [dist > 1 and dist <= N and DELTA < K] /
  CSL.CSAI_DATA_indication(sarg1,sarg2);
  if((sarg4 = 1) and (ack_responseTimer < max_ack_responseTimer))
    {ack_responseTimer := max_ack_responseTimer + 1};
  sarg1 := 0;
```



```

    sarg2 := 0;
    sarg4 := 0;
    dist := 0;
    DELTA := 0}

R14b_CSAI_dataindreport:
  tbcr14b -> Connected
  { - /
  CSL.CSAI_Error_report;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  dist := 0;
  DELTA := 0}

-----
-- Case 3
-- * The seqnum is OLD or (the seqnum is ACCEPTABLE and the ecnum is VALID):
-- In this case the SAI sends a SAI_ERROR.report to the CSL component and
-- discards the Sa_DATA.indication message
-----

R14c_CSAI_datainvalid_or_old:
  tbcr14 -> Connected
  { - [dist < 1 or (dist <= N and DELTA >= K)] /
  CSL.CSAI_Error_report;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  dist := 0;
  DELTA := 0}

-----
-- Case 4
-- * The seqnum is NOT_ACCEPTABLE: In this case the SAI component sends a
-- Sa_DISCONNECT.request to ER-SL, a SAI_DISCONNECT.indication to the CSL
-- component, and then moves to Disconnected state
-----

R14d_CSAI_abortconn:
  tbcr14 -> tbcr14d
  { - [dist > N] /
  ER.CSAI_SA_Disconnect_request;}

R14d_CSAI_notifydisconn:
  tbcr14d -> Disconnected
  { - /
  CSL.CSAI_DISCONNECT_indication;
  sarg1 := 0;
  sarg2 := 0;
  sarg4 := 0;
  ack_requestTimer := max_ack_requestTimer;
  ack_responseTimer := 0;
  ack_reply := 0;
  ack_request := 0;
  currentEC := 0;
  initTimer := 0;
  OFFSET := 0;
  -- EC_expected := 0;
  DELTA := 0;
  last_in := 0;
  next_out := 0;
  dist := 0;
  outdatabuff := [];

```

```
waitnextcycle := False}

RD1a_CSAI_disconndiscard:
  Disconnected -> Disconnected
  {CSAI_DISCONNECT_request}

RD1b_CSAI_datadiscard:
  Disconnected -> Disconnected
  {CSAI_DATA_request(arg1,arg2)}

RD1c_CSAI_discinddiscard:
  Disconnected -> Disconnected
  {CSAI_SA_Disconnect_indication}

RD1d_CSAI_datainddiscard:
  Disconnected -> Disconnected
  {CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6)}

RD1e_CSAI_disconndiscard:
  Disconnected -> Disconnected
  {CSAI_SA_Execution_Cycle_Start(arg1,arg2)}

RD2a_CSAI_datainddiscard:
  Connecting -> Connecting
  {CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6)}

RD2b_CSAI_datareqdiscard:
  Connecting -> Connecting
  {CSAI_DATA_request(arg1,arg2)}

RD3a_CSAI_datareqdiscard:
  Initializing -> Initializing
  {CSAI_DATA_request(arg1,arg2)}

RD3c_CSAI_execycstardiscard:
  Initializing -> Initializing
  {CSAI_SA_Execution_Cycle_Start(arg1,arg2)}

RD4a_CSAI_ecsdiscard:
  Connected -> Connected
  {CSAI_SA_Execution_Cycle_Start(arg1,arg2)}

-----
-- Each time that in Connected state the set_ack_response expires,
-- the SAI sends a SAI_ERROR.report to the CSL component
-- (for further details see REQ_ACKs)
-----

RTc_CSAI_okcsai:
  Connected -> tbcrtc
  {csai_tick /
  Timer.ok_csai;
  ----- ack timing management
  if (ack_responseTimer < max_ack_responseTimer)
    {ack_responseTimer := ack_responseTimer + 1};
  if (ack_requestTimer < max_ack_requestTimer)
    {ack_requestTimer := ack_requestTimer + 1};
  if (ack_requestTimer = max_ack_requestTimer and
      ack_responseTimer >= max_ack_responseTimer)
    {ack_request := 1};
  ----- ec timing management
  currentEC := (currentEC + 1) mod Mec;
  -----
```

```
waitnextcycle := False}

RACK2_CSAI_ackresponsererror:
  tbcrtc -> tbcrack
  { - [ack_responseTimer = max_ack_responseTimer] /
    CSL.CSAI_Error_report;
    ack_responseTimer := max_ack_responseTimer + 1}

ROUTDATABUFF_CSAI_sendbuffered:
  tbcrack -> Connected
  { - [ outdatabuff /= [] ] /
    ER.CSAI_SA_Data_request(
      outdatabuff.head, outdatabuff.tail.head,
      ack_request,ack_reply,next_out,currentEC);
    outdatabuff := outdatabuff.tail.tail;
    waitnextcycle := True;
    next_out := (next_out + 1) mod M;
    if (ack_request = 1)
      {ack_request := 0;
       ack_requestTimer := 0;
       ack_responseTimer := 0};
    ack_reply := 0;
  }

RTd_CSAI_continue:
  tbcrtc -> tbcrack
  { - [ack_responseTimer /= max_ack_responseTimer]}

RTe_CSAI_okisai:
  tbcrack -> Connected
  { - [ outdatabuff = [] ]}

end C_SAI;
```

```
-----  
-----  
Class C_CSL is  
-----  
-----  
Signals  
  
  -- from C_RBC  
  CRBC_User_Data_request(arg1: int);  
  
  -- from C_SAI  
  CSAI_CONNECT_indication;  
  CSAI_DISCONNECT_indication;  
  CSAI_DATA_indication(arg1: Token, arg2: int);  
  CSAI_Error_report;  
  
  -- from Timer  
  ccsl_tick;  
  
-- outgoing to RBC  
--   RBC_User_Connect_indication;  
-- RBC_User_Disconnect_indication;  
-- RBC_User_Data_indication(arg1);  
  
-- outgoing to C_SAI  
-- CSAI_CONNECT_request;  
-- CSAI_DISCONNECT_request;  
-- CSAI_DATA_request(arg1,arg2);  
  
-- outgoing to Timer  
-- ok_ccsl  
  
Vars  
----- PORTS  
RBC_User: C_RBC;  
SAI: C_SAI;  
-----  
  
----- CONFIGURATION PARAMS  
max_receiveTimer: int; -- CONFIGURATION PARAM  
max_sendTimer:   int; -- CONFIGURATION PARAM  
-----  
  
----- LOCAL VARS  
receiveTimer:   int := 0;  
sendTimer:     int := 0;  
-----  
  
Behaviour  
  
-----  
-- R1: At startup, the CSL is in Disconnected state  
-----  
  
R1_CCSL:  
  initial -> NOCOMMS  
  
-----  
-- R2: When in Disconnected state is received a SAI_CONNECT.indication  
--   from the SAI component, the CSL sends a RBC_User_Connect_indication  
--   to the RBC component, starts both the sendTimer and the recTimer,  
--   and moves to Connected state  
-----
```

```
RTa_CCSL_okccsl:
  NOCOMMS -> NOCOMMS
    {ccsl_tick /
      Timer.ok_ccsl}
```

```
R2_CCSL_userconnind:
  NOCOMMS -> COMMS
    {CSAI_CONNECT_indication /
      RBC_User.CRBC_User_Connect_indication;
      receiveTimer := 0;
      sendTimer :=max_sendTimer}
```

```
-----
-- R3: When in Connected state is received a SAI_CONNECT.indication from
--      the SAI component (this is not supposed to occur), the message is
--      discarded.
-----
```

```
RTb_CCSL_okccsl:
  COMMS -> COMMS
    {ccsl_tick [(receiveTimer < max_receiveTimer)
      and (sendTimer < max_sendTimer)] /
      Timer.ok_ccsl;
      sendTimer := sendTimer +1;
      receiveTimer := receiveTimer+1; }
```

```
R3_CCSL_userconnind:
  COMMS -> COMMS
    {CSAI_CONNECT_indication}
```

```
-----
-- R4: When in Connected state is received a RBC_User_Data.request with
--      RBC_data from the RBC component, the CSL sends a SAI_DATA.request
--      with such RBC_data to the SAI component
-----
```

```
R4_CCSL_saidatareq:
  COMMS -> COMMS
    {CRBC_User_Data_request(arg1) /
      SAI.CSAI_DATA_request(RBCdata,arg1);
      sendTimer := 0}
```

```
-----
-- R5: Each time that in Connected state the sendTimer expires, the CSL
--      sends a SAI_DATA.request with a life_sign to the SAI component
-----
```

```
R5_CCSL_okccsl:
  COMMS -> tbc5
    {ccsl_tick [(receiveTimer < max_receiveTimer)
      and (sendTimer = max_sendTimer)] /
      Timer.ok_ccsl;
      sendTimer := 0;
      receiveTimer := receiveTimer+1}
```

```
R5_CCSL_saidatareq:
  tbc5 -> COMMS
    {-/
      SAI.CSAI_DATA_request(LifeSign,0)}
```

```
-----
-- R6: When in Connected state is received a SAI_DATA.indication with a
--      life_sign from the SAI component, the CSL restarts the recTimer
-----
```

```
R6_CCSL_handlelifesign:
  COMMS -> COMMS
  {CSAI_DATA_indication(arg1,arg2)
   [arg1 = LifeSign] /
   receiveTimer := 0}

-----
-- R7: When in Connected state is received a SAI_DATA.indication with
-- SAI_data from the SAI component, the CSL sends a
-- RBC_User_Data.indication with such SAI_data to the RBC component
-- and restarts the recTimer
-----

R7_CCSL_userdataind:
  COMMS -> COMMS
  {CSAI_DATA_indication(arg1,arg2)
   [arg1 /= LifeSign] /
   RBC_User.CRBC_User_Data_indication(arg2);
   receiveTimer := 0}

-----
-- R8: When in Connected state is received a SAI_DISCONNECT.indication
-- from the SAI component, the CSL sends a RBC_User_Disconnect.indication
-- to the RBC component and moves to Disconnected state
-----

R8_CCSL_userdisconnind:
  COMMS -> NOCOMMS
  {CSAI_DISCONNECT_indication /
   RBC_User.CRBC_User_Disconnect_indication;
   receiveTimer := 0;
   sendTimer := 0}

-----
-- R9: When in Connected state the recTimer expires, the CSL sends a
-- SAI_DISCONNECT.request to the SAI component, a RBC_User_Disconnect.indication
-- to the RBC component and moves to Disconnected state
-----

R9_CCSL_okccsl:
  COMMS -> tbcr9
  {ccsl_tick [receiveTimer = max_receiveTimer] /
   Timer.ok_ccsl}

R9_CCSL_saidisconnreq:
  tbcr9 -> tbcr9a
  {- /
   SAI.CSAI_DISCONNECT_request;
   receiveTimer := 0;
   sendTimer := 0}

R9_CCSL_userdisconnind:
  tbcr9a -> NOCOMMS
  {- /
   RBC_User.CRBC_User_Disconnect_indication}

-----
-- RD1: When in Disconnected state the CSL does not accept any kind of message
-- except for SAI_CONNECT.indication from the SAI component
--
-- RD2: When in Connected state the CSL discards only SAI_ERROR.report messages
-- from the SAI component
-----
```



```
RD1a_CCSL_discuserdata:  
  NOCOMMS -> NOCOMMS  
    {CRBC_User_Data_request(arg1) }  
  
RD1b_CCSL_discdisconnind:  
  NOCOMMS -> NOCOMMS  
    {CSAI_DISCONNECT_indication }  
  
RD1c_CCSL_discerrorreport:  
  NOCOMMS -> NOCOMMS  
    {CSAI_Error_report }  
  
RD1d_CCSL_discdataind:  
  NOCOMMS -> NOCOMMS  
    {CSAI_DATA_indication(arg1,arg2) }  
  
RD2a_CCSL_usererror:  
  COMMS -> COMMS  
    {CSAI_Error_report }  
  
end C_CSL;  
-----
```

-----  
Class I\_RBC is  
-----

Signals

```
-- incoming from CSL
IRBC_User_Connect_indication;
IRBC_User_Disconnect_indication;
IRBC_User_Data_indication(arg1: int);
-- incoming from Timer
irbc_tick
-- outgoing to CSL
-- IRBC_User_Data_request(arg1)
-- outgoing to timer
-- ok_irbc
```

Vars

```
CSL:obj;    -- CSL port
n: int := 1;
nmax: int;  -- initialized at configuration time
```

Behaviour

R1a\_IRBCS\_tick:

```
wait -> wait
  {irbc_tick /
   Timer.ok_irbc;}
```

R1d\_IRBC:

```
wait -> connected
  {IRBC_User_Connect_indication}
```

R2a\_IRBC:

```
connected -> sending
  {irbc_tick /
   Timer.ok_irbc;}
```

R2b\_IRBC:

```
connected -> connected
  { IRBC_User_Data_indication(arg1)}
```

R2c\_IRBC:

```
sending -> connected
  { - [n <= nmax] /
   CSL.IRBC_User_Data_request(n);
   n := n+1; }
```

R2f\_IRBC:

```
sending -> connected
  { - [n > nmax] }
```

R3\_IRBC:

```
connected -> wait
  {IRBC_User_Disconnect_indication }
```

end I\_RBC;

-----  
Class C\_RBC is  
-----

```
-- Scenario SC1
-- CRBC not sending NRBC messages, just receiving indications
```

-----  
Signals



```

    -- incoming from CSL
    CRBC_User_Connect_indication;
    CRBC_User_Disconnect_indication;
    CRBC_User_Data_indication(arg1: int);
    crbc_tick
    --
    -- outgoing to CSL
    -- CRBC_User_Data_request(arg1)
Vars
    CSL: C_CSL; -- CSL port

Behaviour

-- increment S counter at every time slop
R1_CRBCS_tick:
    wait -> wait
        {crbc_tick /
         Timer.ok_crbc;}

-- do not handle connect indications
R4_CRBC_discard_CI:
    wait -> wait
        {CRBC_User_Connect_indication }

-- do not handle disconnect indications
R5_CRBC_discard_DI:
    wait -> wait
        {CRBC_User_Disconnect_indication }

-- do not handle data indications
R6a_CRBC_discard_UD:
    wait -> wait
        {CRBC_User_Data_indication(arg1)}

-- never triggered , used just for CADP compatibility.
R7_CRBC_justforsync_DR:
    sending -> wait
        {- /
         CSL.CRBC_User_Data_request(0); }

end C_RBC;
-----

-----
-----
Class Clock is
-----
-----
Signals
    ok_irbc, ok_icsl, ok_isai, ok_eur, ok_csai, ok_ccsl, ok_crbc

Vars
-- Priority := 0;

Behaviour

R0_Timer_IRBC:
    s0 -> s1
        { - /
         IRBC.irbc_tick;}

R1_Timer_ICSL:

```

```
s1 -> s2
    { ok_irbc /
      ICSL.icsl_tick;}

R2_Timer_ISAI:
s2 -> s3
    {ok_icsl /
      ISAI.isai_tick; }

R3_Timer_EUR:
s3 -> s4
    { ok_isai /
      EUR.EUR_tick; }

R4_Timer_ISAI:
s4 -> s5
    {ok_eur /
      CSAI.csai_tick; }

R5_Timer_ISAI:
s5 -> s6
    {ok_csai /
      CCSL.ccsl_tick; }

R6_Timer_ISAI:
s6 -> s7
    {ok_ccsl /
      CRBC.crbc_tick; }

R7_Timer_ISAI:
s7 -> s1
    {ok_crbc /
      IRBC.irbc_tick; }

end Clock;
-----

-----

-----
Class EuroRadio is
-----
-----

Signals

-- incoming from ISAI
ISAI_SA_Connect_request;
ISAI_SA_Disconnect_request;
ISAI_SA_Data_request(arg1:int,arg2:int,arg3:int,arg4:int,arg5:int,arg6:int);
-- arg1 = message type,    arg2 = data value,
-- arg3 = ack request,    arg4 = ack response,
-- arg5 = sequence number, arg6 = EC number
ISAI_SA_Execution_Cycle(arg1:int,arg2:int);
-- arg1 = sequence number,    arg2 = EC counter

-- incoming from CSAI
CSAI_SA_Disconnect_request;
CSAI_SA_Data_request(arg1:int,arg2:int,arg3:int,arg4:int,arg5:int,arg6:int);
-- arg1 = message type,    arg2 = data value,
-- arg3 = ack request,    arg4 = ack response,
-- arg5 = sequence number, arg6 = EC number
```

```
CSAI_SA_Connect_response;
CSAI_SA_Execution_Cycle(arg1:int,arg2:int);
-- arg1 = sequence number,   arg2 = EC counter

-- incoming from Timer
EUR_tick;

-- outgoing to ISAI
-- ISAI_SA_Disconnect_indication;
-- ISAI_SA_Connect_confirm;
-- ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6);
--   arg1 = message type,   arg2 = data value,
--   arg3 = ack request,   arg4 = ack response,
--   arg5 = sequence number, arg6 = EC number
-- ISAI_SA_Execution_Cycle_Start(arg1,arg2);
--   arg1 = sequence number,   arg2 = EC counter

-- outgoing to CSAI
-- CSAI_SA_Disconnect_indication;
-- CSAI_SA_Connect_indication;
-- CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6);
--   arg1 = message type,   arg2 = data value,
--   arg3 = ack request,   arg4 = ack response,
--   arg5 = sequence number, arg6 = EC number
-- CSAI_SA_Execution_Cycle_Start(arg1,arg2);
--   arg1 = sequence number,   arg2 = EC counter

-- outgoing to Timer
-- ok_eur

Vars
--   Priority := 0;
--   Iside: I_SAI;
--   Cside: C_SAI;

Behaviour

-----
-- handle time slots
-----

R1_EUR_okier:
  s0 -> s0
  { EUR_tick /
    Timer.ok_eur}

-----
-- handle inoming ISAI requests
-----

R2i_EUR_newconnection:
  s0 -> s0
  {ISAI_SA_Connect_request /
   Cside.CSAI_SA_Connect_indication;
  }

R3i_EUR_disconnection:
  s0-> s0
  {ISAI_SA_Disconnect_request /
   Cside.CSAI_SA_Disconnect_indication;
  }

R4i_EUR_acceptdata:
  s0 -> s0
```

```
{ ISAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6) /
Cside.CSAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6);
}

-- R5i_EUR_losedata:
-- s0 -> s0
-- { ISAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6) }

-----
-- handle incoming CSAI requests
-----
--
R2c_EUR_confirm:
s0 -> s0
{CSAI_SA_Connect_response /
Iside.ISAI_SA_Connect_confirm;
}
--
R3c_EUR_disconnection:
s0-> s0
{CSAI_SA_Disconnect_request /
Iside.ISAI_SA_Disconnect_indication;
}
--
R4c_EUR_acceptdata:
s0 -> s0
{ CSAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6) /
Iside.ISAI_SA_Data_indication(arg1,arg2,arg3,arg4,arg5,arg6);
}

-- R5c_EUR_losedata:
-- s0 -> s0
-- { CSAI_SA_Data_request(arg1,arg2,arg3,arg4,arg5,arg6) }

-- when ISAI responding with execution cycle start
R6a_EUR_execcycstart_call:
s0 -> s0
{ISAI_SA_Execution_Cycle(arg1,arg2) /
Cside.CSAI_SA_Execution_Cycle_Start(arg1,arg2)}

-- when CSAI performing execution cycle start
R6b_EUR_execcycstart_response:
s0 -> s0
{CSAI_SA_Execution_Cycle(arg1,arg2) /
Iside.ISAI_SA_Execution_Cycle_Start(arg1,arg2)}

end EuroRadio;
-----
-----
-----
-----
SYSTEM CONFIGURATION
-----
-----
Objects:
LifeSign, RBCdata: Token;
Timer: Clock;
IRBC: I_RBC
(CSL -> ICSL, nmax -> 5);
Project Acronym – GA 881775
```



```

ICSL: I_CSL
  (RBC_User -> IRBC, SAI -> ISAI,
   max_receiveTimer -> 20, -- timeout for receiving data
   max_sendTimer -> 10, -- timeout for sending data
   max_connectTimer -> 20) -- timeout for connection phase

ISAI: I_SAI
  (CSL -> ICSL, ER -> EUR,
   -- EC numbers management
   max_initTimer -> 20, -- initialiation phase timeout
   Mec -> 7, -- limit for EC numbers
   K -> 3, -- delay limit for incoming messages
   --acks management
   max_ack_requestTimer -> 20, -- ack request period
   max_ack_responseTimer -> 20, -- max delay before ack response
   -- sequence numbers management
   N -> 1, -- limit for consecutive loss of
messages
   M -> 3) -- limit for sequence numbers

EUR: EuroRadio
  (Iside -> ISAI, Cside -> CSAI);

CCSL: C_CSL
  (RBC_User -> CRBC, SAI -> CSAI,
   max_receiveTimer -> 20, -- timeout for receiving data
   max_sendTimer -> 10); -- timeout for sending data

CSAI: C_SAI
  (CSL -> CCSL, ER -> EUR,
   -- EC numbers management
   max_initTimer -> 10, -- initialiation phase timeout
   Mec -> 7, -- limit for EC numbers
   K -> 3, -- delay limit for incoming messages
   --acks management
   max_ack_requestTimer -> 20, -- ack request period
   max_ack_responseTimer -> 20, -- ack response timeout
   -- sequence numbers management
   N -> 1, -- limit for consecutive loss of messages
   M -> 3) -- limit for sequence numbers

CRBC: C_RBC (CSL -> CCSL);

```

```

Abstractions {
  TLABELS
  -- DEBUG
  Action lostevent($1) -> lostevent($1)
  Action $1($*) -> $1($*)
}

```

```

-----
-- UMC ON THE FLY VERIFICATION:
-----
-- umc -300 -x ERnice_irbcddata_V53_Mec7K3.umc
-- ** REC=20, SEND=10, CONN=20, INIT=20, MEC=7, K=3, AckRq=20, AckRs=20, N=1, M=3 **
--
-- A[ {not CRBC_User_Data_indication} U { CRBC_User_Connect_indication}]
--
-- AF {CRBC_User_Connect_indication}

```

```
-- A[ {not CRBC_User_Data_indication} U { CRBC_User_Data_indication(1)}]
(2.473719)
--
-- A[true {not CSAI_Error_report} U {CRBC_User_Data_indication(1) } true]
(2.473719)
--
-- A[true {not CSAI_Error_report} U {CRBC_User_Data_indication(2) } ]
(7.334053)
--
-- AF { CRBC_User_Data_indication(1)}
-- A[ {not CRBC_User_Data_indication} U { CRBC_User_Data_indication(2)}]
(7.334053)
--
-- A[true {not CRBC_User_Disconnect_indication} U {CRBC_User_Data_indication(3)}]
(11.990690)
--
-- AF {CRBC_User_Data_indication(2)}
-- A[true {not CRBC_User_Data_indication} U {CRBC_User_Data_indication(3)}]
(11.990690)
--
-- AF {CRBC_User_Data_indication(3)}
-- A[true {not CRBC_User_Data_indication} U {CRBC_User_Data_indication(4)}]
(15.109362)
--
-- AF {CRBC_User_Data_indication(4)}
-- A[true {not CRBC_User_Data_indication} U {CRBC_User_Data_indication(5)}]
(16.522994,)
```

-----



### Statechart for Class Clock

Zoom Out   Zoom In

