Radio Communication Management
ERTMS Subset-026-3.5

Cécile Braunstein
University of Bremen
cecile@informatik.uni-bremen.de

April 22, 2013

Abstract

This document describes the model of the radio communication management The model has be made from the specification description of the ERTMS subset-026-3.5 baseline 3.

The model is composed of two packages : the *system* and the *requirement*. The system package consists of the SUT model and the TE model and a set of constants and enumeration types. The requirement package contains all the requirement and a requirement diagram. package. FIXME: cite SYSML FIXME

1 Model description

The radio communication management module (MoRC) is the user application interacting with Euroradio protocol. In the following model only the general behaviors is presented, the actual details of how messages are transported are not relevant for this description.

1.1 Interfaces

The radio communication management module (MoRC) is the user application interacting with Euroradio protocol. In this section we present the general behavior only, the actual details of how messages are transported are not relevant for this description.

The MoRC is the part of the EVC (European vital computer) responsible for the management of radio communication. According to the specification this module interacts directly with the following on-board modules, as shown in figure 1:

- (DMI) driver module interface : receives/displays information from/to the driver,
- (RTM) radio transmission module : receives/gives commands from/to the radio network,
- (BTM) balise transmission module : sends transmission request from a balise group,
- (JRU) Juridical recorder unit : records part of the data exchange for
- (OBU tasks) other on-board functions (SUBSET-026-4.5) used by the MoRC such as:
  - track conditions management functions (SUBSET-026-3.12.1) that determine for instance if the ETCS system of the track is compatible with the system on-board.
  - Mode determination (SUBSET-026-3.12.4, SUBSET-026-4.6).

The orders to initiate or terminates a radio communication may come from the DMI, the BTM, and the RBC (radio block centre) through the RTM. The others OBU tasks may also order a radio communication (see section1.2 for more details).

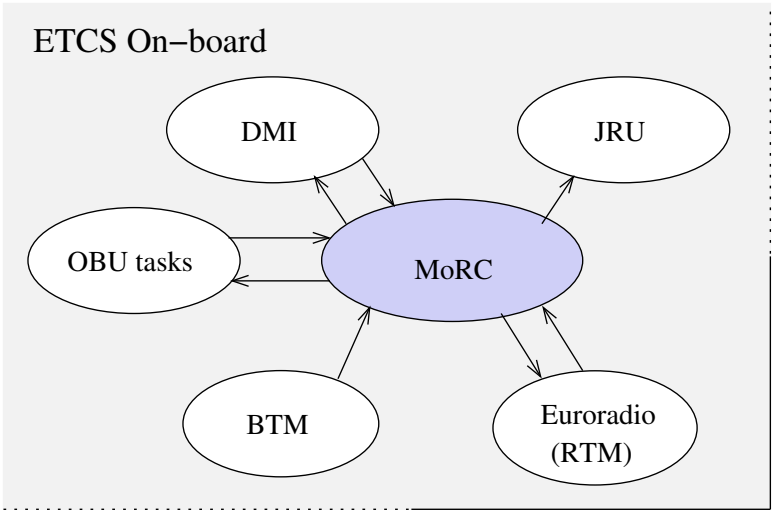

Figure 1: Interactions between the MoRC and others On-board modules

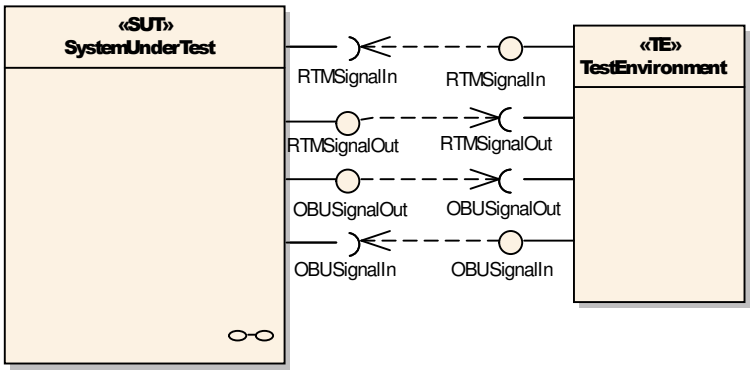

Figure 2: Model Overview

In figure 2 the MoRC model - *our test model* - is composed with a the test environment (TE). The inputs and outputs interfaces are explained in detail in section 1.3. The test environment abstracts away all the others functions or blocks which the SUT may interact with.

The SUT block consists here of unique block that is the MoRC. The MoRC block contains the internal variables used for the protocol implementation and the state-charts describing its behavior. For the moment the constants used in this chapter and defined in SUBSET-026.3-A.3.1 are not part of the MoRC block. At this stage of modelisation it is not defined where they should be declared and recorded to keep track of possible change or extension. I think they should be part of a special package regrouping all this kind of information. In the current model they belong to the system package.

1.2 Abstraction

The MoRC model has been made by direct translation of the specification described in ERTMS subset-026-3.5 In order to keep the complexity low, some abstractions have been made. The model's behavior would be refined in a next step of the model's design process. In a first step we focus on the communication protocol between the MoRC and the RTM. This leads us to abstract some behaviors.

First, our representation will not consider the interfaces with the JRU since it only recorded existing signals and is not relevant for tests generation Secondly, the orders coming from BTM, DMI, EVC will be abstracted as only one message from the on-board. This message will indicate that a communication session should be started or be ended. We will not distinguished between the different events that may occurred since they follow the same connection protocol. Moreover the discrimination between these events is not well defined in the specification, we will assume that the decision is taken by another task of the EVC and that the MoRC task only starts or terminates a radio communication Finally, the output messages from MoRC to DMI are not considered for the first version.

1.3 Inputs and outputs messages

Figure 3 shows the messages exchanged at the interface of the radio communication management module. The numbers in brackets represent the maximal value the message may have. Note that in a first version the communication will only be set up with an RBC, communication with a RIU (radio in-fill unit) are not taken into account.

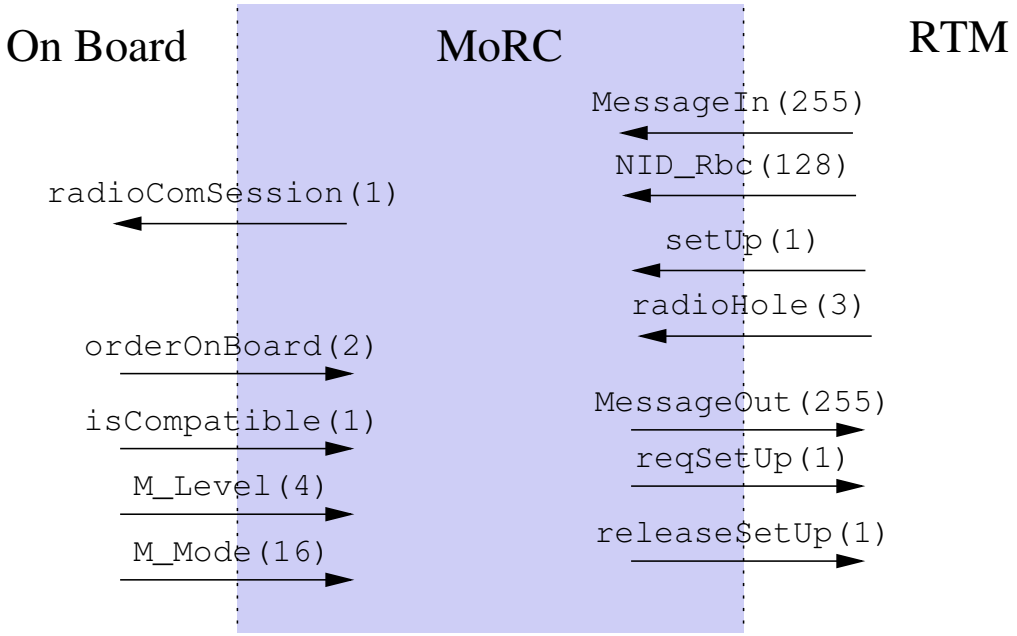

Figure 3: Radio control manager interfaces.

Table 1: Messages exchange during the Management of Radio Communication

Message Name	Id	Packets	Description
TERM_SESSION_TRACK	24	Packet 42 ; Q_RBC = 0	The RBC orders the EVC to **terminate** a communication with RBC
INIT_SESSION_ORDER	24	Packet 42 ; Q_RBC = 1	The RBC **orders the initiation** of a communication
SYS_VERSION	32		The RBC **acknowledge the initiation** of a communication and gives its system version
INIT_SESSION_TRACK	38		The RBC **initiates** a communication
TERM_ACK	39		The RBC **acknowledge** the termination of a communication
TRANS_OVER_ORDER	131		The RBC orders a **transition order** over another RBC
SYS_NO_COMP	154		The EVC **acknowledge the establishment with error**.
INIT_SESSION	155		The EVC **initiates** a communication
TERM_SESSION	156		The EVC **terminates** a communication
SESSION_ESTABLISHED	159		The EVC **acknowledge the establishment** of a communication
NO_MESSAGE	255		No messages are send.

1.3.1 RTM interface

`MessageOut`, `MessageIn` are the Euroradio messages, their possible values are defined in the subset-026-8.4 and the subset-026-7.4. The test cases define by the subset-076 are performed by analyzing the recording of these messages. In our model we have consider only the relevant messages. Furthermore, messages are decomposed in variable and packets, these are not taken into account by our modeling. Our model considers messages only as a number corresponding to an action. Table 1 summarizes the considered messages, the message name are those used in the model, Id and Packets are those defined in Subset-026-7 and Subset-026-8.

`NID_Rbc` identifies the RBC it joins the `NID_RBC` and `NID_C` (Subset026-3).

`setUp`,`reqSetUp`,`releaseSetUp` messages are used for setting or releasing a safe radio communication.

`radioHole` may have the value `BEGIN`, `INSIDE`, `END` or `NONE` regarding if the train enters, leaves or is in a announced radioHole are compatible.

1.3.2 Interface with others on board functions

The different cases to initiate or terminate a communication have been abstract by a single signal. Since the behavior is the same regardless the different events, we assume that others tasks of the EVC will activate the signal wen needed.

`orderOnBoard` represents the order from the on-board EVC to initiate or terminate a communication. The possible values are the following ones :

- `NONE`: no order;
- `INIT` represents one of these cases :
  - Start of mission procedure,
  - Report a mode change,

  - Driver change level to 2 or 3,
  - End of a radio hole,
  - The balise group orders a radio communication.
- `TERM` represents one of these cases
  - End of mission procedure,
  - Driver closes the desk,
  - Error condition detected on-board.
  - The balise group orders to end up a radio communication.

`isCompatible` is set to 1 if a the track and the on-board systems (function system version management)

The radio management module should take some decision with respect to internal on-board variables. This variable are listed blow. The variable definition are detailed in subset-026-7.

- `M_LEVEL` $\in [0..4]$ represents the levels 0,1,2,3 or NTC.
- `M_ mode` $\in [0..15]$ represents the on-board operating mode computed by mode function (SUBSET-026.4).

1.4 Internal variables

We define a set of variables use for the radio communication management computation itself. Figure 4 shows the block element.

- `countSetUp`, `countMsg`, are used to count the number of try for establishing a radio communication, or to wait for the acknowledgment message (SUBSET-026.3.A).
- `msgRecorded`: Keep track of `MessageIn`
- `safeRadio` $\in \{$`NOCOM`, `COM`, `LOST`$\}$ indicates if a safe radio communication is on.
- `radoComSession` $\in \{$`TERMINATED`, `ESTABLISHED`$\}$: indicates if a radio session is established with the track.
- `time` used for timeout evaluation

Note that the `safeRadio` and `radoComSession` may be used for other OBU functions like the messages given to the driver or mode computation

The constants or the enumeration type such as Modes name we had used in the MorC description are not part of the model itself, they should belong to a special package.

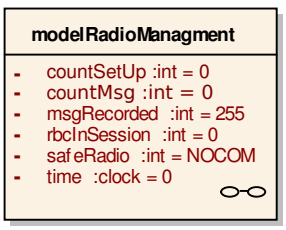

Figure 4: The MoRC class

1.5 Behaviour

The behavior is described figure 5. A classical transaction starts with an order to established a communication with an RBC. In this model we assume that the RBC belongs to the RBC accepting list. Note that we have abstracted the different ways to contact an RBC (last known number, number entered by the driver ...). Secondly, the MoRC sets up a safe radio connection, then it initiates a radio communication with the RBC. An order to terminate a radio communication session may occurred, in this case, the MoRC sends a termination message to the RBC waits for the acknowledgment and then releases the safe radio communication. Our model does not manage the consistency of the successive order, we do not impose any constraints to when the orders may occur. This may be done by external tasks.

States `INIT_COM` and `TERM_COM` are decomposed as state automaton handling the maximal number of try and the time out of requests.

Sate `COM` is decomposed as an automaton, it handles the lost of safe radio.

2 Requirements Modeling

The model describes in section 1 is a direct translation of the SUBSET026-chap 3.5. This section describes how the model is related to the specification. SysML offers a modeling constructs to represent text-based requirements and relationship with other modeling elements. The requirements in EA diagram may be described in a graphical or tree structured format. The requirement may also appear directly other diagram. The diagram requirements intent to may be imported and exported from other tools in csv or XMI format.

In this particular experiment, a table document has been made in csv format from the SRS. Each requirement is represented as a number that corresponds to the numbers and bullet points of the specification document. From this list of requirements, one can directly find the corresponding paragraph in the documents. We had also added the complete text in a text attribute for helping the reader.

The requirement list is then automatically translate to a requirement package in SysML where each requirement is a *requirement* SysML element. Our requirement contains the following attributes :

- ID : the corresponding name from the SRS
- Body: the corresponding text from the SRS

Figure 6 shows an example of the requirement representation.

Most of requirement refers to the behaviors model as transition of the state-chart representation. The link has been made such that each transition satisfy at least one requirement. Note that in EA, it is not possible to directly link a transition element to a requirement element, a invariant constraint true has been added on the transition to make the link.

Others requirement may describe a set of transitions or a more general behavior property. In these cases the requirement may be translated as one or more constraints that must be satisfied by the model. In our model, constraints are invariants expressed in LTL[5]. SysML does not imposed any language, one can choose other constraints language such as OCL [2]. The following example shows the translation of two requirements into a LTL properties. The reader should refer to section **??** for variables and values details.

REQ-3.5.3.1 "It shall be possible for ERTMS/ETCS on-board equipment and RBC to initiate a communication session".

```
Finally ((orderOnBoard == 1 || MessageIn == INIT:_SESSION_ORDER ||
MessageIn == INIT_SESSION_TRACK  ) &&
Finally (MessageOut == SESSION_ESTABLISHED))
```

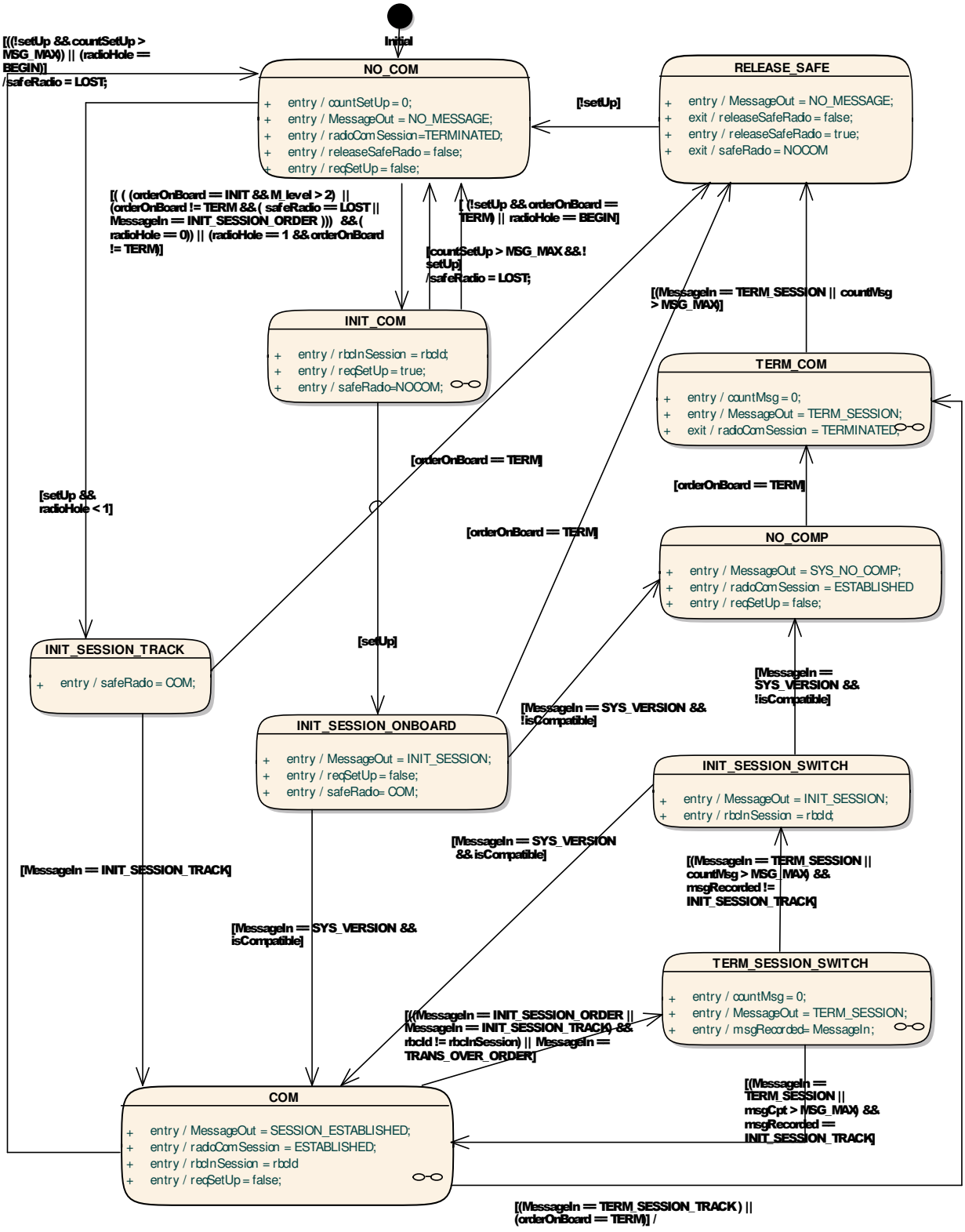

Figure 5: Automaton of the radio communication management

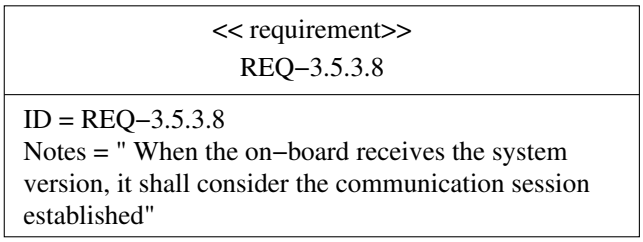

Figure 6: Example of modeling a requirement

REQ-3.5.3.10 "If the establishment of a communication session is initiated by the RBC, it shall be performed according to the following steps ..."

```
Finally (MessageIn == INIT_SESSION_TRACK && setUp == 1 ->
Next (MessageOut == SESSION_ESTABLISHED && radioComSession == 1ESTABLISHED)
```

The requirement diagram only represent the "satisfy" relations between the requirements and the model. One could refine this diagram and add derive dependency or containment relationship. Note that in practice we could show the "satisfy" relations directly on the state-chart view. The separate representation is more readable.

3 Test Generation

3.1 Tool overview

The RT-Tester test automation tool, made by Verified [6], performs automatic test generation, test execution and real-time test evaluation. It supports different testing approach such as unit testing, software integration testing for component, hardware/software integration testing and system integration testing. The RT-Tester version we had used, follows the model-based testing approach [3, 4] and it provides the following features :

- Automated Test Case Generation
- Automated Test Data Generation
- Automated Test Procedure Generation
- Automated Requirement Tracing
- Test Management system

Starting from a test model design with UML/SYML, the RT-tester fully automatically generates test cases. They are then specified as test data (sequences of stimuli with timing constraints) and used to stimulate the SUT and run concurently with the generated test oracles. The test procedure is the combination of the test oracles and the SUT that can be compiled and executed.

The tool supports test cases/data generation for structural testing. It automatically generates reach statement coverage, branch coverage and modified condition/decision coverage (MC/DC) as far as this is possible. The test cases may all be linked to requirements ensuring a complete requirement traceability. Additionally RT-tester may produce test cases/data from a LTL formula, since a LTL formula describes a possible run of the model.

Finally the tool may produce the documentation of tests for certification purposes. For each test cases the following document are produced :

- *Test procedure*: that specifies how one test case can be executed, its associated test data produced and how the SUT reactions are evaluated against the expected results.
- *Test report*: that summarizes all relevant information about the test execution.

In [1], a general approach on how to qualify model-based testing tool according to the standard ISO 26262 ad RTCA DO178C has been proposed and applied with success to the RT-tester tool. Following the same approach compatibility with the CENELEC EN50128 may be easily done.

3.2 Test generation

RT-Tester is able to automatically generate test cases from the test model. The first step may be to generate the test cases that cover all the model artifacts : a test to reach all transitions, control state and perform MC/DC coverage of the model. The test cases are related to their attached requirement. This is automatically derived from the requirement diagram.

The test campaign may then be guided by requirement coverage, one can select the test cases associated to a requirement, or by test cases coverage, one select the tests to be covered.

Finally, it is possible to add user define test cases. This has been done to import test cases part from the SUBSET-076. his has been made manually by translating the test cases description from the word document as an LTL formula. For example the test case 1 of feature 177, tests the requirement : "Establishing a radio communication initiated by RBC". It checks that after receiving a connection indication of the safe radio from the RTM, the OBU shall confirm the establishment and shall consider the session established following the process describes in the requirements 3.5.3. This has been translated into the following formula:

```
// Initial state NO_COM
[
setUp == 0 &&
radioComSesssionEstablished == 0 &&
M_Level >= 3 &&
M_Mode != 3 &&
M_Mode != 4 &&
M_Mode != 5 &&
M_Mode != 9 &&
M_Mode != 10 &&
radioHole == NONE &&
orderOnBoard == NONE  &&
messageIn == NO_MESSAGE
] &&
[
setUp == 0 &&
radioComSesssionEstablished == 0 &&
M_Level >= 3 &&
M_Mode != 3 &&
M_Mode != 4 &&
M_Mode != 5 &&
M_Mode != 9 &&
M_Mode != 10 &&
radioHole == NONE &&
orderOnBoard == NONE  &&
messageIn == NO_MESSAGE
]
Until
(// Safe radio set up
[
  setUp == 1 &&
  IMR.messageFromRBC == 38
] &&
Finally
// Message session established send
// session considered as established
[
  IMR.messageFromRBC == 38 &&
  IMR.messageToRBC == 159 &&
  IMR.radioComSesssionEstablished == 1
])
```

Table 2 summarizes the set of test cases generated and the related requirements. The line LTL represents the requirement that has been rewrite with a LTL formula. The test 177 represents the translation as LTL formula of the two test cases described in SUBSET 076. This set of test cases covers 36 requirements from SUBSET-026-chap 3.5.

References

[1] Jörg Brauer, Jan Peleska, and Uwe Schulze. Efficient and trustworthy tool qualification for model-based testing tools. In Brian Nielsen and Carsten Weise, editors, *Testing Software and Systems*, volume 7641 of *Lecture Notes in Computer Science*, pages 8–23. Springer Berlin Heidelberg, 2012.

Test cases covered	# tests generated
Basic state	11
Transition	28
MC/DC	63
LTL	4
Test 177	2
Total Tests	108

Total Requirement cover 36

Table 2: Test cases generation summary

[2] Object Management Group (OMG). OMG Object Constraint Language (OCL), 2012.

[3] Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated test case generation with smt-solving and abstract interpretation. In Mihaela Bobaru, Klaus Havelund, GerardJ. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 298–312. Springer Berlin Heidelberg, 2011.

[4] Jan Peleska, Elena Vorobev, Florian Lapschies, and Cornelia Zahlten. Automated model-based testing with RT-Tester. Technical report, Universität Bremen, 2011.

[5] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, 31 1977-Nov. 2.

[6] Verified Systems International GmbH. Verified :: Products. http://www.verified.de/en/products.