A Versatile Test Component Based on the EULYNX Subsystem Point

Daniel Schwencke

With the advent of standardized modular signalling architectures, the external validation and verification of corresponding subsystems of signalling suppliers becomes increasingly important, e.g. to prove conformity to the standard and interoperability with other subsystems. This can be done by testing, using different test set-ups where the same subsystem will take different roles. This insight lead to the idea of a „versatile test component“. The article reports on a realization covering the point controller and point machine subsystems, based on the EULYNX Subsystem Point specification.

1. Introduction

1.1 Demand for Testing Induced by Standardized Modular Architectures

For several years, European railway initiatives have been developing standardized modular architectures: EULYNX [1] focuses on the interfaces (IF) of an interlocking, decoupling e.g. field elements from the interlocking core; the Reference CCS Architecture (RCA) [2] was devoted to the wider scope of the – mostly trackside – command control and signalling (CCS) system, layering the information flow from traffic management systems down to devices in the field and back; and the Open CCS On-board Reference Architecture (OCORA) [3] cares about unified IF of networked on-board CCS components. Moreover, Europe’s Rail’s System Pillar is concerned with the integration of these architectures in a „System Pillar Architecture“ [4], completing the architecture derivation process towards the topmost level of operational analysis of the railway system.

Whereas previously modularization, the design of IF and the detailed behaviour of CCS subsystems largely was a supplier-internal matter, with the new architectures they are now prescribed to a greater extent. This leads to exchangeable subsystems, potentially provided by many different suppliers. This in turn induces an increased need for verification and validation, to prove – to the external system integrator and to authorities – that the subsystems will actually interact towards realization of the intended system of systems behaviour. In the first place, conformity to the standardized specification is to be verified, and ideally, this would already imply all kinds of desirable properties built-in to the specification. However, in practice this is usually not sufficient; also safety, robustness, the successful integration of subsystems, etc. will require further verification. More and more, the standardization initiatives care about standardized verification: With Baseline 4 Release 1, EULYNX has started to provide “certification test cases” for its “subsystems”, i.e. for those standardized interlocking IF which also include standardized behaviour of the connected field element controllers. EULYNX is looking into formal verification as well [5, 6], but for now testing seems to be the envisaged means of verification [7].

1.2 The Idea of a Versatile Test Component

Consider Figure 1 – it shows a system of interconnected subsystems A to H (blue boxes and connections). For testing, depending on purpose and scope of the test, each subsystem will be assigned a role (consider the red boxes for an example):

- either part of the system under test (SUT),
- or part of the test environment,
- or not used in the test set-up.

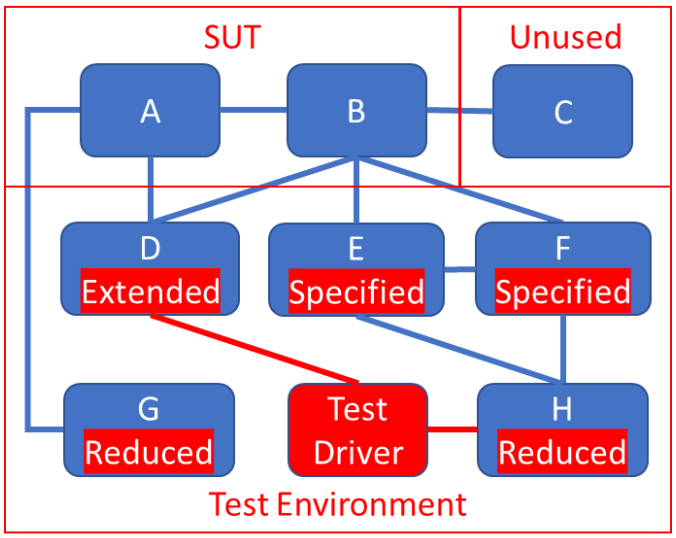

Figure 1: Example set-up for testing (in red) in a system-of-systems context (in blue)

Moreover, each subsystem that is part of the test environment will be required to realize functionality to a certain extent (text with red background as an example):

- either reduced functionality (acting e.g. as an adapter or simplified simulator),
- or specified functionality (acting like a subsystem implementation),
- or extended functionality (acting e.g. as failed, defective or manipulated subsystem).

Finally, some test environment subsystems will be connected to the test driver (red connections), which controls test execution according to a given test case; but others may not.

It is not difficult to imagine that there is a multitude of possible uses of subsystems in testing; and the need to test all system parts as well as the considerations about different verification purposes in Section 1.1 indicate that several of these uses will actually be required in test campaigns. Implementing new versions for each use of the same subsystem would be laborious and involve redundant work, as parts of the subsystem IF and functionalities will stay the same. The idea of the test component (TC) presented in this article is to avoid that by creating a software that can be flexibly configured, connected and controlled, thus serving many different test set-ups.

1.3 Structure of the Article

The following Chapter 2 introduces purpose and scope of the TC, including an overview of the specifications used. Chapter 3 presents some design considerations for important aspects of the TC, and Chapter 4 quickly sketches its implementation in hardware and software. Chapter 0 describes set-up, scope and results of a basic TC validation by testing. Finally, Chapter 6 summarizes the work, provides conclusions and discusses future use and development of the TC.

2. Purpose and Scope of the Test Component

2.1 Use Cases and Requirements

The most basic requirement for the TC has been to (ideally completely) implement the possible behaviour of the subsystem(s). For standardized behaviour this means “as specified”; for non-standardized behaviour this means “adding a simulation of realistic input/output (I/O) behaviour”. It may include behaviour in degraded situations; and it includes to make any external and possibly open internal IF, as well as important status information, accessible from the outside. This already allows to realize a number of use cases for the TC:

- Use as SUT (when no implementation is available yet or does not provide sufficient information/insight for debugging purposes). This may refer to the complete SUT, e.g. during building up a test lab, or to part of the SUT, e.g. for early integration testing.
- Use as reference (SUT and reference both have the same scope and receive the same inputs, and SUT outputs are compared to reference outputs).
- Use – possibly multiple instances – as dummy in larger test set-ups (because real products are not yet ready or considered too expensive / too laborious to deploy for testing), e.g. during (software) system testing.

As the identity and functionality of signalling systems is usually configurable, the TC is required to be fully configurable (according to its specification or to the simulation options available). This ensures that all configurable functions can actually be tested or are available as part of the dummy, and that the TC can be adapted for use in given environments.

In case more than one subsystem is covered by the TC, an option to switch off subsystems – making internal IF of them external – should be considered. This allows to alternatingly or gradually replace subsystems of the TC by real subsystems, supporting testing of all kinds of integration stages.

Furthermore, impossible behaviour of the subsystems should be possible to provoke by the tester, going beyond specified or realistic I/O behaviour. This may include manipulation of functions (failure injection) or of non-functional properties (e.g. insertion of delays). This enables additional use cases like testing the SUT reaction to a faulty environment, or to check that test cases actually fail in case of a faulty SUT.

2.2 Scope and Features

The scope of the TC has been chosen to comprise two subsystems (see darker blue boxes in Figure 2):

1. A point controller, with IF and functionality as specified by EULYNX (see Section 2.3 below), except that the special “4-wire” variant of the P3 IF, the option to use an IF to the EULYNX “Maintenance and Data Management” subsystem and the “Standard Diagnostic IF” (not affecting the controller functions) have not been realized. The main controller functions are to relay a point position commanded by the interlocking to the point machine (PM) and to report an aggregated point position status and failed point movements back to the interlocking.
2. A PM simulation, with the IF to the point controller inherited from the EULYNX specification, but otherwise with self-specified functionality. The main PM simulation functions are to simulate the movement of the PM and to report its status – position and possibly ability to move – to the point controller.

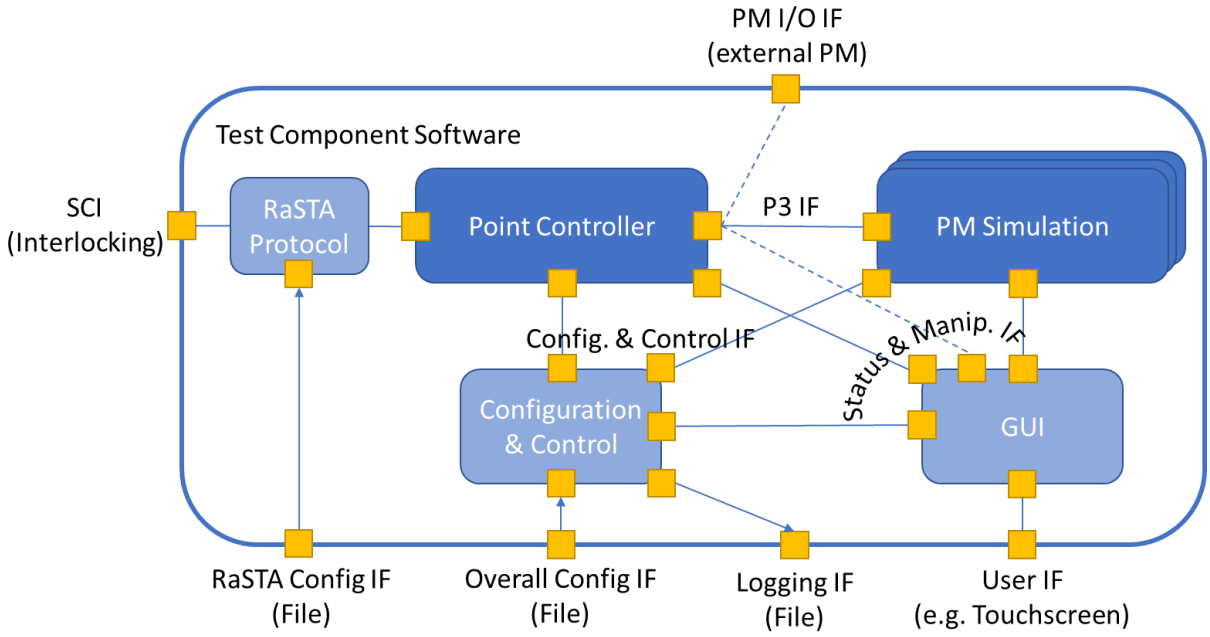

Figure 2: Overview of the TC's structure and IF

This scope has the advantage of being self-contained in the sense that there is only one mandatory external IF (the standard communication interface, SCI, of the point controller) to another subsystem (interlocking) that is important for functional testing, which moreover is completely standardized. Note that to achieve the latter, a (pre-existing) DLR implementation of the necessary RaSTA (Rail Safe Transport Application) protocol [8] has been integrated into the TC software (see leftmost lighter blue box in Figure 2). A general advantage of choosing a EULYNX subsystem is the existing detailed specification not only of IF, but also of the subsystem behaviour, and the existing (first version of) certification test cases.

For setting up and running the TC, configuration and control functionality has been added, as well as a graphical user IF (GUI) for interaction with a tester (see lower lighter blue boxes). The former e.g. cares for a fair processing of the point controller, all PM simulations and the in- and outputs of the SCI and PM I/O IF, and stores all log entries to a file upon finishing a TC run. The latter has four tabs corresponding to the four internal status and manipulation IF shown in Figure 2; a picture of the first tab is shown in Figure 3.

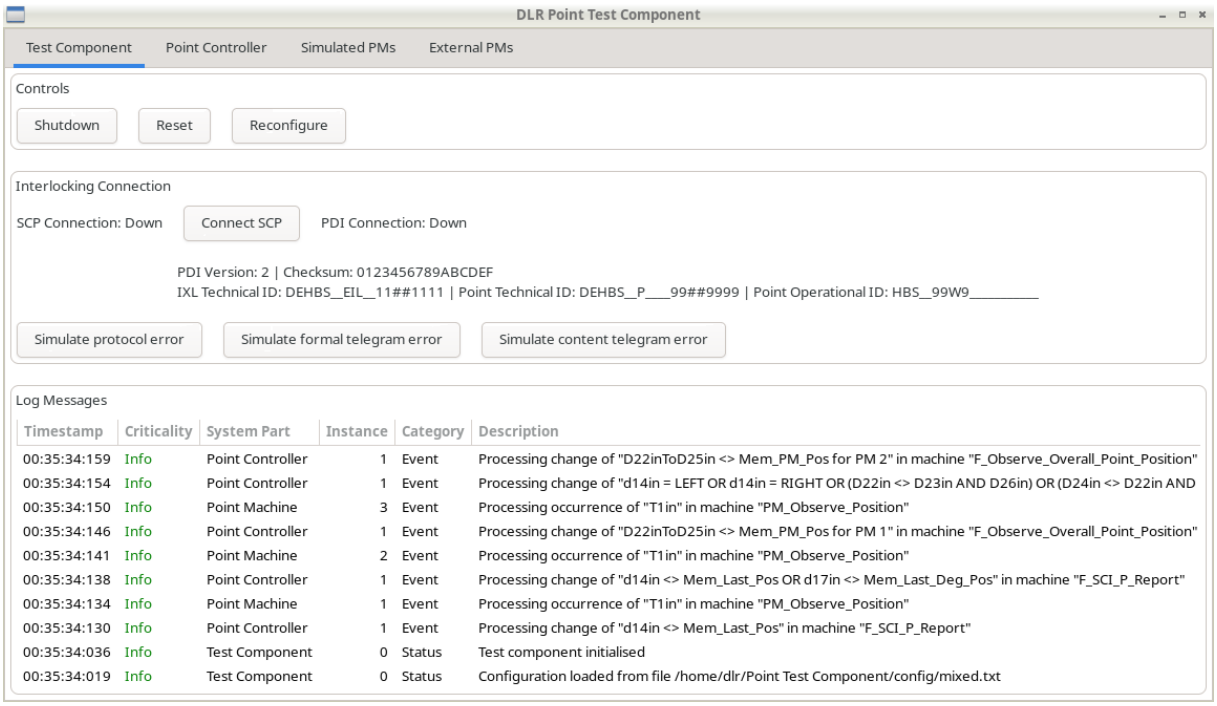

Figure 3: Screenshot of the first tab of the TC's GUI for overall monitoring and control

Configurability of the TC already starts at the overall TC level: induced by a configuration parameter of the EULYNX point controller that allows an arbitrary[1] number (>=1) of controlled PM, it is possible to use several instances of the PM simulation (indicated by multiple boxes for PM Simulation in Figure 2). At the same time, it is possible to configure, for each PM separately, to switch off the built-in simulation, leaving the PM behaviour to an external actor (e.g. a connected physical PM, or the user of the TC interacting with its GUI – see the dashed lines in Figure 2).

2.3 Specification Analysis/Creation

It was decided to implement the point controller subsystem according to the latest EULYNX specifications from Baseline 4 Release 2; this includes in particular the requirement specifications [9, 10, 11], the IF specifications [12, 13] of the SCI, and the architecture specification [14]. Generally, the tabular (SCI telegrams) and model-based (subsystem logical structure and behaviour) specifications define the subsystem very precisely and consistently; also, a list of configuration options is already included. Overall, the specifications comprise 14 telegrams, 10 interconnected SysML (Systems Modeling Language [15]) blocks containing one statechart each, and 15 configuration options relevant for the point controller. However, there remain specification gaps, in particular since SysML (version 1.6) leaves the execution order of events in statecharts, as well as few other details, up to implementation, and EULYNX seems not to close these gaps either. In addition, during analysis of the specifications (by review), besides some formal issues, behavioural ambiguities in the concrete model were detected, reported to and discussed with the EULYNX cluster in charge. To proceed, some working assumptions were made to resolve these issues.

For the PM simulation subsystem, no standardized specification was known. Anyway, for this relatively small subsystem it has not been too difficult to come up with an own specification in the same style as the EULYNX specifications, building on the already defined P3 IF (see Figure 2) and on

[1] [17] mentions a limit of 52 PM, but without further justification or reference.

some imagination of the physical processes. This amounted to 3 interconnected SysML blocks containing one statechart each (one of which is shown in Figure 4), and 2 configuration options.

3. Design of the Test Component

3.1 Point Controller

Having the structured (interconnected SysML blocks) and detailed EULYNX Subsystem Point specification at hand, and considering that performance optimization is not urgent for controllers which have timing requirements in the order of 100 milliseconds, it was decided that it is easiest to stick with the design given by the specification. However, a few parts are left open intentionally by the specification, modelled as open ports of SysML blocks. Mostly, simply corresponding GUI controls were designed for direct input by the tester (e.g. buttons for power on/off, detection of a safety critical fault, controller-internal ability to move point). For some others, explicit connections to controller events were designed (if the tester activates “auto-finish booting”, booting is considered finished if a RaSTA connection has been established successfully; inputs for RaSTA disconnect and for telegram errors at the SCI are triggered by corresponding detection mechanisms).

3.2 PM Simulation

For the PM simulation design, besides the P3 IF defined by EULYNX, further external IF for power supply and to the physical point were defined which can be controlled by the tester via the GUI (e.g. buttons for local point operation to left/right, and for forcing unintended position representing an event like trailing of the point). Furthermore, the time it takes the PM to move from one end position to the other can be changed via the GUI.

The PM simulation subsystem has been structured into three logical components, each of which is responsible for a certain function of the PM simulation, defined by a statechart:

- The main function is the simulation of the PM movement, based on inputs from the P3 IF, the physical point IF, on drive power availability and the PM movement time. Internally, it maintains a precise position between the end positions, but its output is just one of the three states left, right or no end position. For its design, it was necessary to define what happens in special situations like conflicting inputs (e.g. move left AND move right) or change of the movement time during an ongoing movement. In case a PM is configured as “no drive capability” (i.e. being a point detector only), the logical component responsible for the PM movement will not be instantiated.
- The second function is the detection of the PM position (see the corresponding state machine in Figure 4), based on the output of the PM movement simulation, forced unintended position and detection power availability. Its output is left, right or no end position or unintended position. For the special case of detector only PM a design was chosen which derives position information (d70in input) from the neighbouring PM with drive capability.
- The third function is the detection of the PM’s ability to move, based on drive power availability and on whether the PM is configured to detect unability to move at all. Its output is able/unable to move or not used. Its design was straight forward.

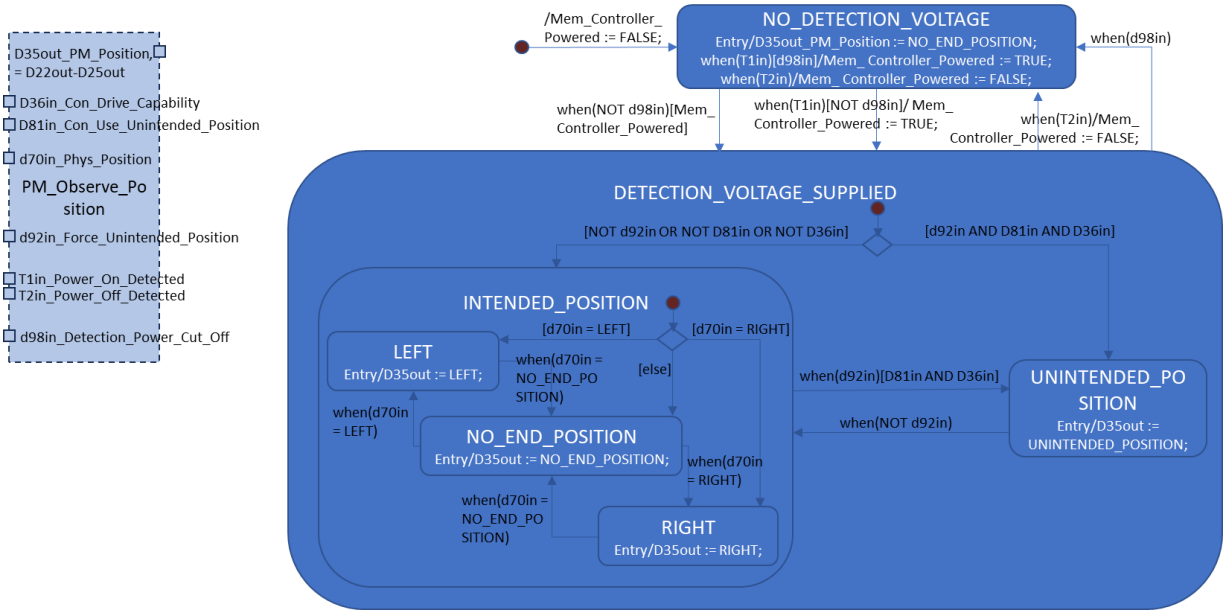

Figure 4: Logical component (left) and statechart (right) for observing the PM position as part of the PM simulation specification

3.3 Connecting External PM

For connection of external PM (alternatively to using the PM simulation), it is foreseen by the design that the P3 IF can be exposed as an external IF of the TC. More precisely, it is exposed in parallel both for connecting real PM and on the GUI – see the two dashed lines in Figure 2. The GUI serves for displaying the status of the IF to the tester, but at the same time allows the tester to change inputs (all input changes from the two sources will be processed, i.e. relayed to point controller and GUI). The latter has been allowed to (a) enable simulation of literally any PM scenario directly by a tester, without the need to connect a real device, and (b) enable to force or resolve special situations through the tester (e.g. force unintended position without damaging a connected real point). Currently, it is the tester's responsibility to use the GUI input feature with care and not to disturb/overwrite outputs of a real PM.

3.4 Configuration and Manipulation

A simple text-based parameter/value configuration file format has been designed to allow for a flexible use of different configurations. Firstly, it defines fixed-value data which cannot be changed during a test run: 15+2 parameters for point controller and PM configuration, such as for enabling redrive functionality of the controller or for the initial position of a PM. In addition, there are configuration parameters to choose between internal simulation or external use of a PM and between auto-powering the subsystems at test-component start-up or not. Secondly, it defines initial values for simulation parameters that can be changed during a test run via the GUI: 8+12 parameters for point controller and PM such as the initial internal ability to move for the controller, or whether left movement is initially blocked for a PM. Thirdly, it defines 9+9 initially displayed failure injection options for the GUI (see paragraph below). All parameters related to PM need to be specified for each configured PM.

A default configuration has been designed which allows ad-hoc usage of the TC. Without shutting down the TC, it is possible to reset (restart without changing the configuration) or reconfigure (restart choosing the default or a user-defined configuration) it. To enable successful RaSTA

connections with any devices taking the role of an EULYNX interlocking, a separate RaSTA configuration XML file is used (see Figure 2).

Manifold manipulation capabilities have been planned (not yet fully implemented) for both the behaviour of the point controller and the PM simulation. For the controller, those functions with timing requirements specified by EULYNX (e.g. starting redriving) can be artificially delayed. And for both subsystems, outputs, event processing and statechart transitions can be manipulated by means of suppressing, inserting, exchanging, duplicating, swapping or delaying them. The desired fault can be selected and injected via the GUI.

4. Implementation of the Test Component

4.1 Hardware

Important requirements on the TC hardware are

- support for connecting the SCI, which is normally done via ethernet cable;
- some IF for displaying the GUI and operating the TC through it;
- some IF for connecting external PM;
- easy transportability (as a versatile TC); and
- low cost (to keep testing cost moderate even if many such TC are used).

It was decided to use a Raspberry Pi microcomputer (model 4B) running Raspberry Pi OS (version "Bookworm", based on Debian 12), inside a solid case and together with a transportable 10.1'' touchscreen connected via HDMI. This clearly fulfils the above requirements (using its GPIO IF for external PM) and provides even more possibilities (e.g. using WiFi as an alternative to Ethernet cable).

4.2 Software

The TC software is mainly handwritten in C++, incorporating the pre-existing RaSTA implementation in C. The logical components of the subsystems and their statecharts have been transformed into C++ classes in a structured way. In particular, member functions were created for the evaluation of changed inputs or timers, which may generate events; further member functions were created for the processing of these change or timeout events (see Figure 5 for an example transition which translates to the C++ code shown in Program Code 1). The event scheduling is managed by a central control class, based on one queue for each subsystem; also relaying of information at internal and external IF is done by this control class. For creation of the GUI the gtkmm framework [16] in version 4.0 has been used. gcc has been used on the Raspberry Pi to compile the software.

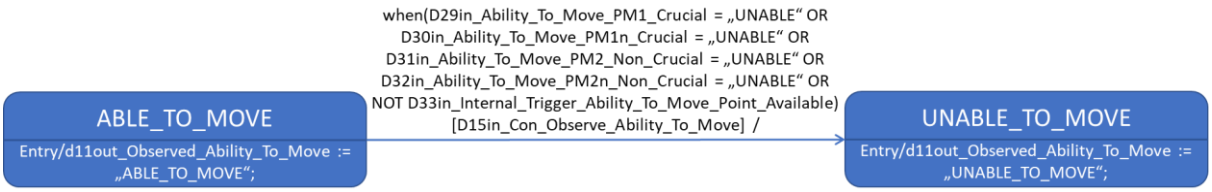

Figure 5: Example statechart transition, redrawn from [8]

```
// Process change event involving D29in to D33in ports
```

```
void F_Observe_Ability_To_Move::processChangeEvent_D29inToD33in(int event)
{
  if (currentState == ABLE_TO_MOVE) {
    // Transition ABLE_TO_MOVE -> UNABLE_TO_MOVE
    if (event == 0) {
      if (Con_Observe_Ability_To_Move) {
        […]
        enterUnableToMove(detectionTime);
        currentState = UNABLE_TO_MOVE;
      }
    }
  }
  […]
}
```

Program Code 1: C++ code for the statechart transition from Figure 5, where event 0 stands for the change event "when(D29in_Ability…Point_Available)" from the transition

5. Validation of the Test Component

5.1 Test Environment

For a basic validation of the TC, it has been tested using the DLR RailSiTe® lab. The core of the latter is a modular lab software for simulation and testing of different signalling subsystems. In the current context, it has been used to execute test cases and to act as the interlocking side of the SCI. The detailed definition of SCI telegrams (SCI-P version 4) to be generated by the lab and those expected back from the point controller has been implemented in the lab (telegram structure) and has been given as part of the test cases (telegram values). The RailSiTe® includes a RaSTA protocol implementation and runs on a computer with ethernet port, so that a connection with the TC can be easily established after defining some connection parameters (IP address, UDP ports, RaSTA IDs) on both sides.

5.2 Test Cases and Test Execution

With EULYNX Baseline 4 Release 1, a set of certification test cases for EULYNX point controllers has been delivered. However, this is neither complete (see [17]) nor fully compatible with Baseline 4 Release 2 point controllers (e.g. using telegrams in SCI-P version 3), so existing test cases have been adapted and completed for the purpose of TC validation. Five main test cases have been defined which can be used with the default configuration of the TC:

1. Successfully commanding the point from left to right end position
2. Commanding the point from right to left end position, taking longer than allowed
3. Reporting unability to move due to PM output "unable to move"
4. Reporting ability to move (again) due to PM output „ability to move not used"
5. Reporting degraded point position due to non-crucial PM output „no end position"

These test cases have been arranged into two test sequences which are executable by the lab, including additional prefix test cases for establishing a connection at the SCI. During test execution, the TC GUI has been used by the tester to manipulate PM simulations so that the PM outputs were delayed (test case 2) or changed (test cases 3-5) as required. The log messages recorded by the TC for both test sequences have been saved (automatically) into a text file (see Table 1) for detailed evaluation of the test runs.

Table 1: Log excerpt from a test run, to be read bottom-up; IXL = interlocking. The central block of five log entries shows a test sequence where the second PM's drive voltage becomes insufficient (encoded as Boolean input d99in, see bottom step of block). The point controller is notified that the PM is unable to move (the following step upwards, compare the transition from Figure 5), processes that change and finally sends a telegram informing the interlocking that the point is considered unable to move (topmost step of block).

Timestamp	Criticality	System Part	Instance	Category	Description
03:05:49:126	Info	Test Component	0	Status	Test component finished
03:05:44:114	Info	IXL - Controller	0	Status	SCP connection closed
[...]					
03:04:01:805	Info	IXL - Controller	0	Data <-	Sent "Ability To Move Point(ability=UNABLE)" telegram
03:04:01:790	Info	Point Controller	1	Event	Processing change of "d11in = UNABLE_TO_MOVE" in machine "F_Control_Point"
03:04:01:776	Info	Point Controller	1	Event	Processing change of "d11in = UNABLE_TO_MOVE" in machine "F_SCI_P_Report"
03:04:01:761	Info	Point Controller	1	Event	Processing change of "D29in = UNABLE OR D30in = UNABLE OR D31in = UNABLE OR D32in = UNABLE OR NOT D33in" in machine "F_Observe_Ability_To_Move"
03:04:01:747	Info	Point Machine	2	Event	Processing change of "NOT d99in" in machine "PM_Observe_Ability_To_Move"
[...]					
03:03:49:202	Info	IXL - Controller	0	Status	SCP connection established successfully
[...]					
03:03:48:821	Info	Test Component	0	Status	Test component initialised
03:03:03:895	Info	Test Component	0	Status	Default configuration loaded

5.3 Test Results

The test sequences have been repeatedly executed, while issues found in the TC, but also in the test cases and in the lab, were removed successively; finally, they could be executed completely with all test cases succeeding. "Issues found in the lab" refers to an initially incomplete adaptation to the SCI-P version 4 telegrams and initially wrong lab configuration values for test case interpretation; "issues found in the test cases" refers to inappropriate timing and incomplete adaptation of existing test cases (regarding telegram formalization, scoping of test case steps and step descriptions). They can be attributed to low experience of the tester with lab usage and test case format. More interestingly, "issues found in the TC" refers to

- a copy-and-paste error;
- a forgotten line of code;

- wrong checksum de-/encoding algorithm;
- two wrongly structured conditional (if…then…else) statements (same error).

For a handwritten software with roughly 10.000 lines of code (including empty lines, excluding the RaSTA implementation) this is extremely few errors, of which only the last bullet point refers to statechart code parts. Some explanations for that are that (a) the five test cases only cover the TC code very partially, (b) the model-based specification (SysML blocks and statecharts) could be transferred to code in a very systematic manner, and (c) a couple of GUI errors and of statechart initialization errors (visible in the log) had been identified visually and fixed in advance of testing. Nevertheless, the test results – besides validating that important functions work as expected – give rise to the assumption that the TC code has high quality.

Furthermore, an issue with the EULYNX specifications has been detected: They seem not to constrain the scheduling of events for statecharts (intentionally) left open by SysML (cf. Section 2.3), which can lead to presumably undesired sequences of telegrams at the SCI observed during testing.

For the evaluation of the test runs the recorded log entries were used; a direct observation of the behaviour is sometimes not possible during the test runs due to the quick sequence of events (e.g. during connection establishment). The interpretation of the logs was feasible, but sometimes a bit arduous because no cause-effect relation between the log entries is recorded and consequently needs to be re-established by hand.

6. Summary, Conclusions and Future Work

Summarizing, a versatile TC has been designed, implemented and validated. It comprises a point controller as specified by EULYNX and a PM simulation, and ample possibilities for configuration and interaction. Based on a Raspberry Pi microcomputer, the TC is easily transportable and connectable via ethernet cable at the SCI of the point controller. It can be used as part of many different test set-ups: as SUT or as part of the test environment, with internally simulated or externally connected/simulated PM, with behaviour as specified or with manipulated behaviour (not yet fully implemented).

Conclusions from the work include the following:

- The detailed and clear design of the TC has been an important precondition to arrive at a high-quality software. This included maintaining complete lists of IF parameters, configuration options, GUI controls and manipulation options for not losing the overview; it also included a detailed description of the PM behaviour via state machines and additional text, to cope with each possible configuration and combination of inputs.
- Transforming many state machines into C++ code is possible on the basis of a clear transformation scheme, but laborious. It is estimated that if more than one EULYNX subsystem specification is to be implemented, implementing an automatic transformation first and applying it to exported XMI (XML Metadata Exchange) files of the specification models provided by EULYNX would be more efficient.
- Depending on the particular kind of manipulation, this can be integrated easily into the state machine based code structure (reading/setting information which is stored with or passed between state machines, or manipulating telegrams or events) or requires major local

(manipulation of transitions) or global (passing additional delay values together with information) changes.

- The detailed model-based specification leads to surprisingly high ad-hoc quality of implementations as witnessed by the results of basic testing of the TC. However, the tests also showed that it is not trivial to come up with an appropriate solution for parts (event scheduling) which remain unspecified by EULYNX and SysML.

A first application of the TC as SUT is planned in connection with a new EULYNX test bed currently developed by DB within the European R2DATO project. Another application as part of a test environment may be the integration into DLR's RailSiTe® lab as point simulation.

Besides completion of the implementation, ideas for further development may be to enable standalone PM simulations, to include missing features (the 4-wire point controller option, EULYNX standard maintenance/diagnostic IF) and to care about flexible integration of future versions of the EULYNX Subsystem Point standards.

Acknowledgement and Disclaimer

Most of the works reported have been conducted in the course of the FP2 - R2DATO project. The project is funded by the European Union through the Europe's Rail Joint Undertaking (JU) under Grant Agreement No. 101102001. A first precursor software (point controller state machine C++ encoding according to EULYNX Baseline 4 Release 1) has been created during an internship of Mr. Tsogtbaatar Mendbayar at DLR in summer 2022.

Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the Europe's Rail JU. Neither the European Union nor the JU can be held responsible for them.

Literature

[1] EULYNX Website, https://eulynx.eu, accessed on 03/05/2024 2:14 p.m.

[2] EUG and EULYNX partners (2022): RCA Architecture Poster, Preliminary issue, With OCORA contribution. RCA.Doc.40, version 0.4 (0.A) from 26/04/2022, published in Baseline 0 Release 4.

[3] OCORA public Github repository, https://github.com/OCORA-Public/Publications, accessed on 03/05/2024 4:55 p.m.

[4] Europe's Rail System Pillar website, https://rail-research.europa.eu/system_pillar/system-pillar-architecture/, accessed on 03/05/2024 4:58 p.m.

[5] Salunkhe, S., Berglehner, R., Rasheeq, A. (2021): Automatic Transformation of SysML Model to Event-B Model for Railway CCS Application. In: Raschke, A., Méry, D. (eds) Rigorous State-Based Methods. ABZ 2021. Lecture Notes in Computer Science, vol 12709. Springer, Cham. https://doi.org/10.1007/978-3-030-77543-8_14

[6] Mark Bouwman, Djurre van der Wal, Bas Luttik, Mariëlle Stoelinga, and Arend Rensink (2023): A Case in Point: Verification and Testing of a EULYNX Interface. Form. Asp. Comput. 35, 1, Article 2. https://doi.org/10.1145/3528207

[7] EULYNX (2022): Certification plan. Eu.Proc.7, version 2B from 22/11/2022, published in Baseline 4 Release 1.

[8] DIN VDE V 0831-200 (2015-06): Electric signalling systems for railways - Part 200: Safe transmission protocol according to DIN EN 50159 (VDE 0831-159).

[9] EULYNX (2023): Requirements specification for subsystem Point. Eu.Doc.36, version 4.3 (0.A) from 28/06/2023, published in Baseline 4 Release 2.

[10] EULYNX (2023): Generic interface and subsystem requirements. Eu.Doc.20, version 4.0 (3.A) from 27/06/2023, published in Baseline 4 Release 2.

[11] EULYNX (2023): Generic interface and subsystem requirements for SCI. Eu.Doc.119, version 1.0 (3.A) from 27/06/2023, published in Baseline 4 Release 2.

[12] EULYNX (2023): Interface specification SCI Generic. Eu.Doc.93, version 3.2 (0.A) from 28/06/2023, published in Baseline 4 Release 2.

[13] EULYNX (2023): Interface specification SCI-P. Eu.Doc.38, version 4.2 (0.A) from 27/06/2023, published in Baseline 4 Release 2.

[14] EULYNX (2023): EULYNX System architecture specification. Eu.Doc.16, version 2.2. (0.A) from 27/06/2023, published in Baseline 4 Release 2.

[15] OMG SysML Website, http://www.omgsysml.org/, accessed on 15/05/2024 11:51 a.m.

[16] gtkmm Reference Manual Website, https://gnome.pages.gitlab.gnome.org/gtkmm/, accessed on 15/05/2024 12:47 a.m.

[17] EULYNX (2022): Scope and coverage justification for EULYNX Certification test cases. Version 1.A from 18/11/2022, published in Baseline 4 Release 1.