Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Developing a test environment for a railway demonstrator

Master's Thesis

Author
Gergő Ecsedi

Advisor
dr. Zoltán Micskei

December 16, 2018

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Ecsedi Gergő*, szigorló hallgató kijelentem, hogy ezt a diplomaterv meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2018. december 16.

Ecsedi Gergő
hallgató

Kivonat

Egy többrétegű, hardver és szoftver elemekből álló összetett alkalmazás esetén a meghibásodások számos komponensből eredhetnek. Ráadásul egy ilyen rendszerben a hiba okának feltárása és a hiba elhárítása is komoly kihívás lehet, amely erőforrás is időigényes. A biztonságkritikus rendszerekben mindenképpen meg kell előzni a végzetes, akár emberi életekbe kerülő hibák előfordulását. Az ilyen tulajdonságokkal rendelkező rendszerek során manapság elterjedt ellenőrzési technika a tesztelés.

A Méréstechnika és Információs Rendszerek Tanszéken folyamatosan fejlesztés alatt van egy modellvasútra épülő, többrétegű biztonságkritikus demonstrátor rendszer. A projekt célja a modell vezérelt és verifikációs technológiák bemutatása. Az eddigi teszt megvalósítások a keretrendszer gyors fejlődése miatt hamar elavultak és nehezen karbantarthatóak lettek. A rendszerben számos egyedi megvalósítású hardver és szoftver elem van kapcsolatban egymással. Ahhoz hogy megbizonyosodjunk minden réteg hibamentes és biztonságos működéséről részletesen megtervezett tesztelési stratégiára van szükségünk.

Diplomamunkám során elsődleges célom egy szisztematikus, részletes és könnyen karbantartható teszt keretrendszer és dokumentáció megalkotása a demonstrátor ellenőrzésére. Elsősorban meg kell határozni a rendszer szintű követelményeket és megismerni modellvasút és egyedileg készített elemek tulajdonságait. A következő lépés a tesztelési lehetőségek felderítése a hasonlóan komplex rendszerek esetén. Végezetül egy a hibadetektálására alkalmas teszt keretrendszer megvalósítása és tesztek végrehajtása a feladatom.

A demonstrátor rendszer felderítése során elkerülhetetlen az egyes mikrokontrollerek megismerése (Rapsberry PI, BeagleBone Black, Arduino), valamint számos különböző megoldásokat alkalmazó szoftver technológia használata. Ebből kifolyólag a tesztelési folyamatot is több módszerrel és különböző rétegekben kell végrehajtani. Egy szabványos tesztelési dokumentációt követve, megvizsgáltam és megvalósítottam a szoftver komponensek tesztelési stratégiáit mint egység, integrációs és rendszer szinten egyaránt.

A megvalósított tesztek a demonstrátor rendszerbe is integrálva lettek, mely során egyes szoftver komponenst is módosítani kellett a tesztelhetőség érdekében. A demonstrátor biztonságkritikus funkcióinak ellenőrzéseként rendszerszintű tesztek lettek meghatározva, amelyek a későbbiekben is használhatóak az egyes bemutatók előtt.

Abstract

A multi-layered complex system, where hardware and software elements interact with each other, many component can cause system failures. In addition finding the root cause in these systems can be difficult and time-consuming. However these failures are unacceptable in a safety-critical system, like in a railway system, where human life can be in a risk. Nowadays the most common verification method to avoid these situations is testing.

In the department of Measurement and Information Systems, the students are developing a railway demonstrator system, which simulates a safety-critical railway system. Aim of this project is to demonstrate the model driven development and model verification techniques. Because of the frequently changed software and hardware components all the test implementations became outdated and were hardly maintainable. In addition there are several custom hardware and software units in the system, which need to be tested. Therefore a systematic, detailed test design is required to verify the demonstrator.

The aim of my thesis is to create a test framework to detect failures and bugs in the demonstrator system, which is maintainable and covers the safety-critical aspects. First the system requirements have to be defined and then the demonstrator architecture must be assigned in details. Next step is to analyze the test design possibilities and approaches for such a multi-layered system. The final step is to create a test framework which is capable of safety-critical error detection in the demonstrator.

During the demonstrator architecture investigation, it is inevitable to get to know several hardware (like Rapsberry PI, BeagleBone Black, Arduino) and software technologies. Therefore the test strategy must cover different aspects in several layers to verify each component functionalities separately. Following a standard test documentation approach, I have designed and implemented test cases for custom software component for unit, integration and system level also.

Finally the implemented tests were integrated into the demonstrator system with some code restructuring of several components. Furthermore the system-level test cases were defined and can be a useful strategy before any system demonstration.

Chapter 1

Introduction

Nowadays the usage of a multi-layered applications have increased, however these can be very complex systems. They are often used in a safety-critical environment, therefore the need to create a reliable application is important. In this area a software or hardware failure can cause unsafe events with high risks. In addition finding a problem's root cause can be time-taking in the latter development phases. To avoid such a situation the most common verification method is testing. Several failures and errors can be found by a detailed and well-designed testing framework.

The need for a maintainable testing structure have also came up in the department of Measurement and Information Systems for a safety-critical project. The Model-based Demonstrator for Smart and Safety Systems (in abbreviation: MoDeS^3) relies on a model railway system and have been extended with hardware and software elements to control the track and trains on it. Its purpose is also to demonstrate the model-based design and verification techniques. To satisfy these objectives, off-the-shelf and custom hardware elements have been attached to the railway track. These products required also custom-made software components, which have several technological dependencies making the demonstrator system a complex multi-layered application. The track consists of 24 sections and 6 turnouts, for which 6 BeagleBone Black microcontrollers have been configured with custom hardware extensions to control them. An Arduino microcontroller have been added to the system to sense the availability (shows that for example a train is on a segment or not) of each segment In addition a Raspberry PI provides a graphical interface to control the track elements separately and supplies the necessary technical environment for the other microcontrollers. A picture of the whole track is shown in Figure 2.1 from top view.

During the development of MoDeS^3 system, there wasn't a maintainable and well-designed testing approach, which is able to follow the frequently changing software and hardware implementations. Right now the demonstrator system basic functionalities become more stable, but the need for a testing framework is still remained. In my thesis I will investigate the possible test approaches for creating a suitable test environment for the railway system. In order to construct a maintainable and applicable test framework for the system, all parts must be examined which we are aiming to test. The first aspect is to detail and verify the safety-critical functionalities with requirements. Then a systematic and detailed test design can be created for the project.

To start with I have defined the requirement for the demonstrator system and detailed the safety-critical aspects, which should be covered by tests. A systematic and detailed test design have been constructed for testing purposes aligned with the ISO 29119 standard [1]. This documentation was helpful for creating a maintainable and systematic test docu-

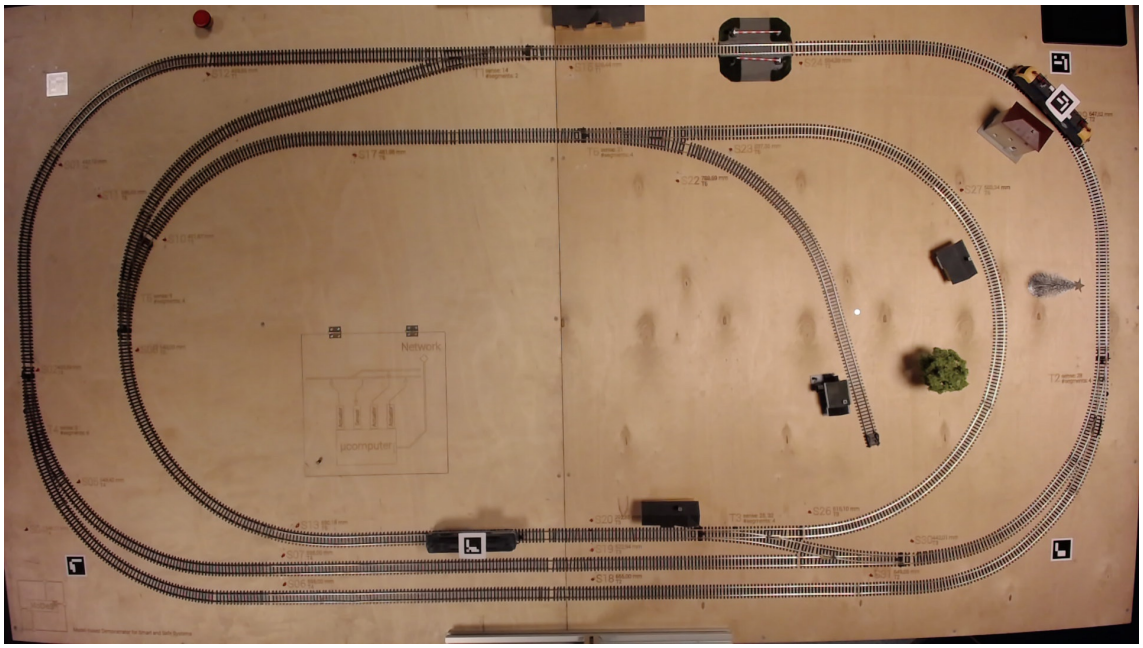

Figure 1.1: Railway system top overview

mentation for such a complex system. The MoDeS^3 demonstrator railway system requires a wide knowledge in the model driven and low level (close to hardware) technologies also. On the microcontrollers Java executables are deployed, except the Arudino component for which a C++ unit is installed. Therefore the test plans have to be designed to support implementations in multiple languages. Additionally high-level model-based techniques are used to avoid safety-critical situations on the track, like Viatra, Yakindu with Gamma, which can generate Java classes representing their models. Throughout the implementation, Eclipse was used with Xtend plugin, which generates the source code into Java classes. Therefore I have used the JUnit 5 library along with Mockito and PowerMock frameworks for test case implementation.

In my thesis I describe the architecture of the MoDeS^3 railway demonstrator system in Chapter 2, where after the custom hardware and software components, the safety-critical scenarios and functions are detailed, aligned with the system requirements. In the next Chapter 3, the possible test design techniques are detailed with the overview of the test documentation standard. After the description of system architecture and the general test documentation chapters, the Chapter 4 follows with the design decisions and details of the test documentation for MoDeS^3 system. The test implementation key points and the status of test execution reports are described in Chapter 5. Finally the Chapter 6 summarizes my thesis work.

Chapter 2

Railway demonstrator system architecture

First of all this section describes the railway demonstrator system architecture as it is the system under test.(A top overview of the demonstrator system is shown in Figure 2.1.) This system's purpose is to simulate a real-life safety critical railway system, with basic functionalities and with a safety logic. The demonstrator is based on a railway model stub which is extended with custom and off-the-shelf hardware and software components. Naturally the system is also capable of moving and stopping the trains on them, nevertheless with the safety logic it can avoid safety-critical scenarios like train collision or train derail. In the following chapter I will further discuss the system architecture with their components and describe the requirements for them.

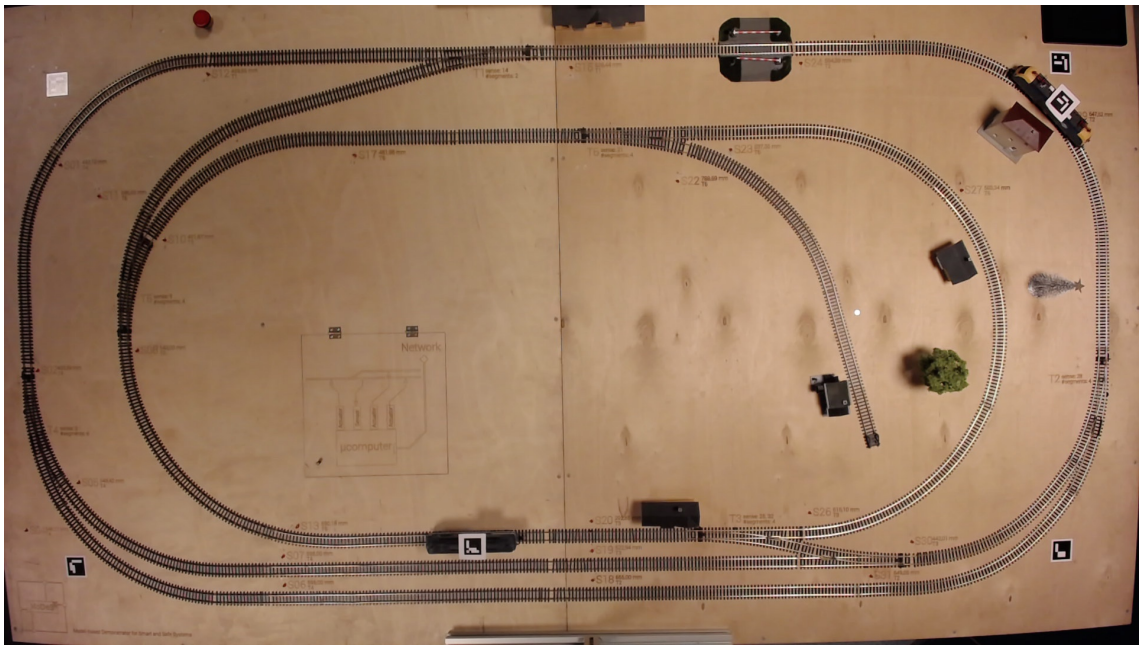

Figure 2.1: Railway system top view

2.1 Railway system basic components

I will introduce the physical components and the basic process of the railway systems stub in the following section. Basically there are 31 sections, with one blind track and 7 turnouts. Figure 2.2 shows the layout of railway elements with the corresponding segment ids. In the following, the abbreviation of S is stands for sections and T is for turnouts.

Create more visible figure about layout?

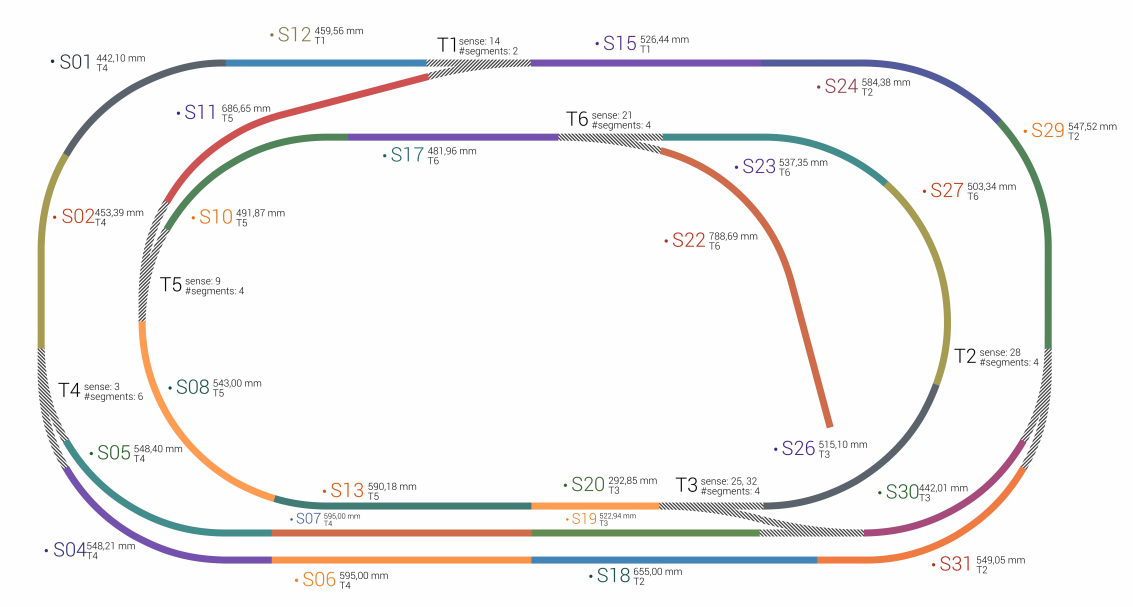

Figure 2.2: Railway layout

Section First principal element is a 30-65 cm long railroad. Each section is connected to 2 other segments (except the S22 blind section) and wired to a command station, which gives them sufficient power source for moving the trains on them. On Figure 2.2, these sections are identified by *SXX* strings, where *XX* is two unique digits for this layout. Furthermore these numerals determine which bit shows this section's occupancy in the occupancy vector (see section 2.3.1 for more information). After the identifier each section lengths are shown with responsible BeagleBone Black ids.

Turnout The second principal element is the turnout, which can differentiate 2 paths on the track. The train which is going through a turnout can reach different sections depending on the state of the exact turnout. On the Figure 2.3 *T1* turnout illustrates a regular turnout and *T3* exemplifies an English turnout. On the layout figure (2.2) each of these elements are visible as gray dashed sections with the IDs. Each turnout ID starts with T and ends with a numeric (1..6). These IDs determines, which bit identifies the turnout's occupancy in the occupancy vector) and *#segments* as number of supervised sections by the corresponding BeagleBone Black (see section 2.3.1 for more information about occupancy vector).

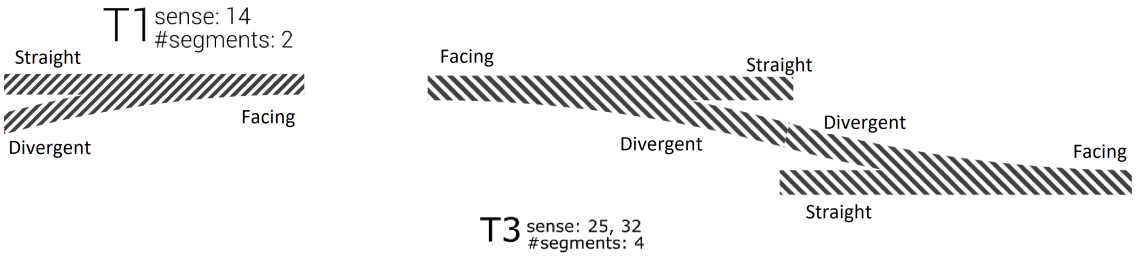

Figure 2.3: Turnout directions

Train Any model train can be used with the demonstrator system, which can be moved on the track elements. With a supplementary element a train's speed and length can be measured.

Command Station Supply power source for the sections which provides tension for the trains on the track. Although an off-the-shelf microcontroller consumes 5V DC, the command station supplies 12V DC. Therefore we have to convert down the power source to match the microcontroller's needs. (For more details see Section 2.2.2)

Controller In connection with the *Command Station* an XPressNet protocol [1] based controller is attached to the system. This component's purpose is setting the direction and speed for each train on the track.

footnote is ok? move it out to the bibs?

2.2 Hardware extensions

The basic hardware environment is not sufficient for controlling and analyzing purposes, therefore additional hardware elements have been designed to satisfy these requirements. In this section these platforms will be described in details. (The A.1 appendix contains pictures about the elements.) For modeling purposes I have used MagicDraw with Sysml plugin [6].

2.2.1 Data processing units

BeagleBone Black (BBB) An industrial microcontroller platform which provides 4GB 8-bit eMMC on-board flash storage and 2x PRU 32-bit microcontrollers, which could satisfy the function for parallel monitoring. There are 6 BBB on the track connected to the railway, used for controlling turnouts and enabling/disabling each section.

Rapsberry Pi 3 A Rapsberry Pi microcontroller is dedicated to handle most of the software components related to the Railway demonstrator system. It has twice as large computing capacity in RAM and also in CPU as BBB.

[1] More details about XPressNet Protocol `http://www.lenzusa.com/1newsite1/Manuals/xpressnet.pdf`

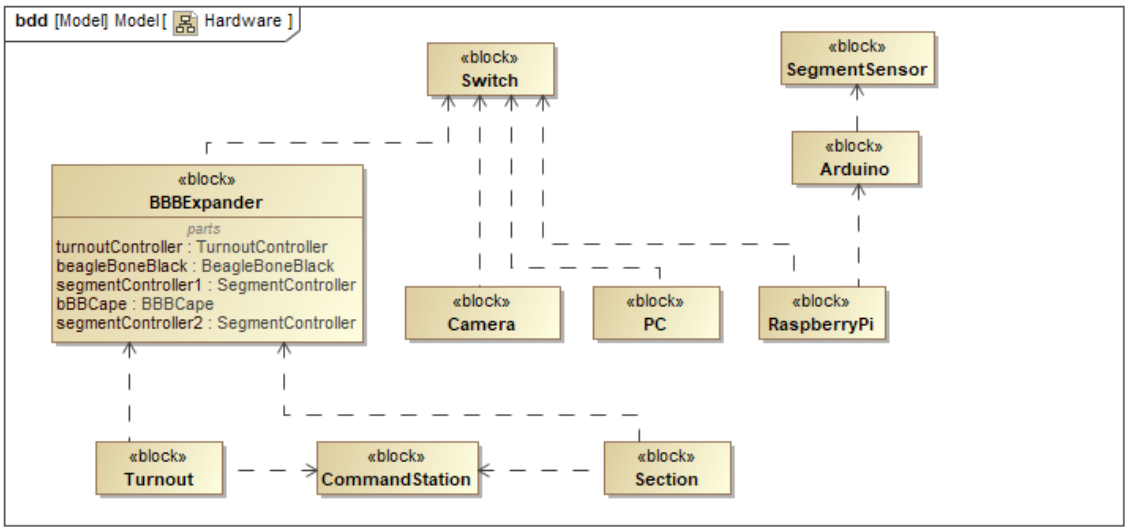

Figure 2.4: Hardware block definition diagram

Arduino Dedicated hardware element for reading the 6 DigiSens-8-S88 output data through S88 protocol (see 2.2.2 section for details about this component). This communication layer requires proper timing conditions which the Arduino platform can satisfy.

Camera A web camera is placed above the demonstrator table, so it gets a top overview of the table.

2.2.2 Custom hardware extensions

BeagleBone Black cape and expanders The BeagleBone Black components expect 5VDC power source instead of 12VDC which our power station supplies. Because of that reason a so called cape have been created for each controller. Additionally the need for easy-to-use ports to attach additional circuits to the main board also have come up. The expanders could be used to extend the functionality of one BeagleBone unit, which is on the figure 2.5.

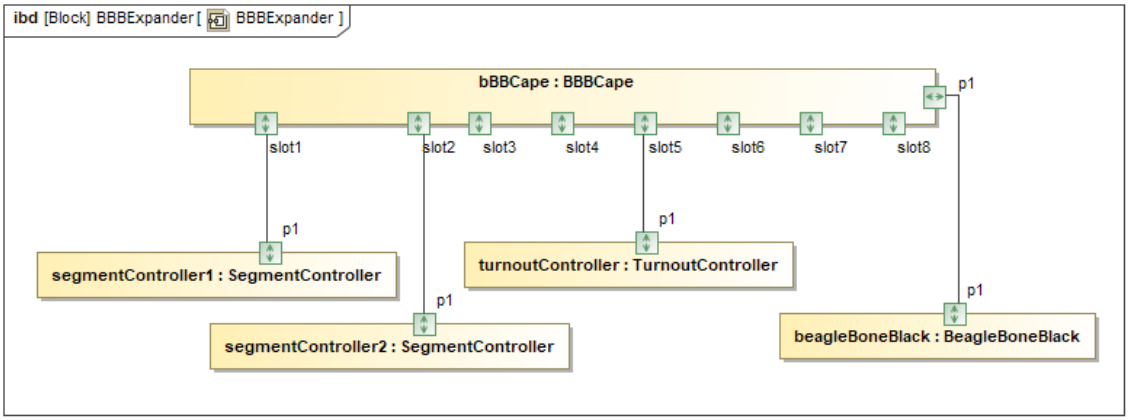

Figure 2.5: Layout and current attachment of cape and expander

Each cape have 8 general purpose expander slot, for which the pin layout is expressed in the following 2.1 table.

Table 2.1: Pin layout

pin3	pin2	pin1	pin0
3V3	5V	Gnd	12V

The upper row of each connector is dedicated for GPIO connections. Two of the GPIO pins connected to the application processor and the remaining two GPIO pins are connected to the PRU unit. With this setup, the PRU and the application processor can cooperate on hardware level.

Segment sensor The DigiSens-8-S88 component is an off-the-shelf product, which can detect the occupancy for 8 segments. [2].

Segment actuator Segment Actuator expanders are designed to stop a train on the corresponding segment. The concept behind this expander based on the Lenz Asymmetrical DCC and Automatic Brake Control functionality of train-decoders. Every Segment Actuator expander has two slots (A or B) and can enable and disable two segments. Each segment can be enabled setting two GPIOs to HIGH level. One GPIO is connected to the PRU and the other to the application processor.

Turnout actuator Turnout Actuator expanders can switch turnouts on the table between their states. Previously a Commercial off-the-shelf (COTS) units was user for this purpose, but in that case we were not able to query the position of a switch programmatically. The Turnout Actuator expander gives the ability for both, switching the turnout and sensing its state.

The concept behind this unit is based on the fact, that turnout mechanism is working as a wire between the common (COM) pole and an other pole (straight or divergent) when switched in one position, therefore we can sense its state.

The electronic characteristics of the BeagleBone unit could not satisfy the switching process electrically, thus we had to use an ATmega328 micro-controller unit (Arduino). Additionally the state-sensing process is based on Analog to Digital Converters, which are also integrated into the Arduino.

For usage, the Arduino has 2 input and 2 output pins connected to the expander connector as shown in the Section 2.2.2. The turnout switching information is input for the Arduino and the turnout state sensing is an output information for that, because it supplies the turnout actuator elements with these details.

Pin 0	Pin 1	Pin 2	Pin 3
Straight	Divergent	Straight	Divergent
Turnout switching		Turnout state sensing	
Input for the Arduino		Output for the Arduino	

[2]More information about the product can be found here:`http://www.digitools.hu/termekek/erzekelok/digisens-8-s88`

2.3 Software components

In general the following software components were implemented in java and c++ languages and deploying separation is shown in Figure 2.6. Mainly each BBB microcontroller have TrackElementController deployed involving the GPIO, PhysicalSegmentController and PhysicalTurnoutController components. The Rapsberry Pi serves the Dashboard, Touchboard, Messaging and OccupancyQuery components and the Arduino microcontroller is dedicated for SectionOccupancyQuery functionality.

delete physical controllers? add xpressnet controller? update figure to be nicer and add componentlevel SL

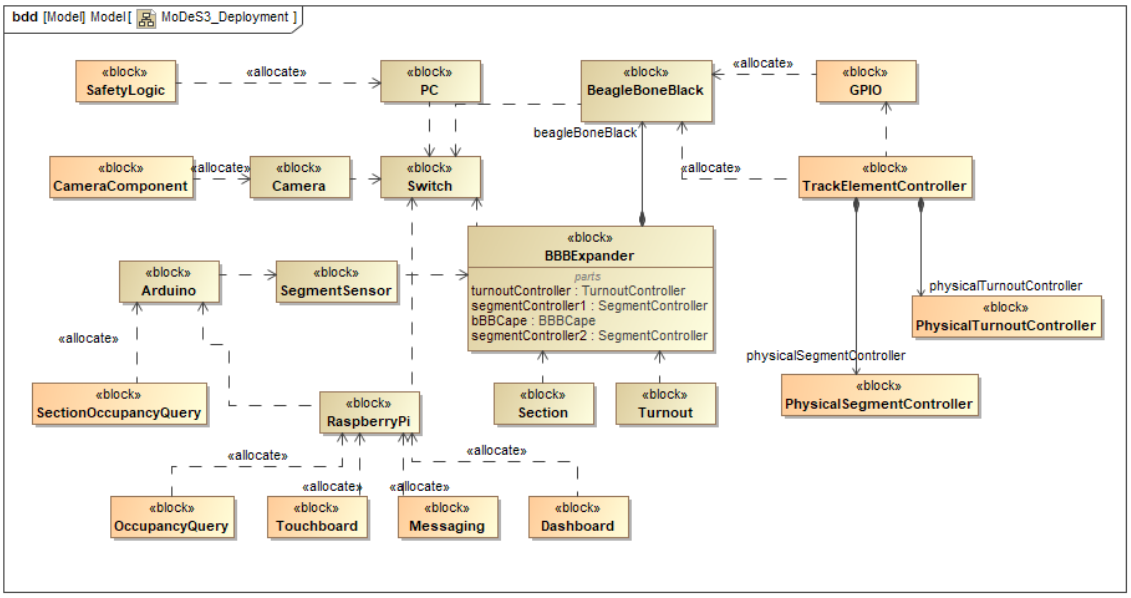

Figure 2.6: Software components deployment to hardware elements

2.3.1 Occupancy detection elements

Section Occupancy Query Responsible for debouncing the 32bit long occupancy vector with proper timing conditions regarding S88 protocol and forward this 32bit to *Occupancy Query* through usb connection. Computed data contains the occupancy information for each track element (section or turnout) per one bit.

Occupancy Query In connection with the *Section Occupancy Query*, this component is storing the occupancy states for the whole track. Only if the state has changed for one segment, it sends an occupancy change message to the dedicated MQTT topic with the track element id and the track's new occupancy state.

2.3.2 Track segment control elements

GPIO Handles the GPIO pin changes and commands for each extension point of the BBB cape (see 2.2.2 section for details about BBB cape and expanders).

Track Element Controller On each BeagleBone Black microcontroller there is a controller, which executes the turnout and segment operations for the supervised sections. To satisfy this functionality this controller forwards commands to the corresponding Physical Segment or Turnout Controllers, which communicates with GPIO managers. Consequently with this software component, we are able to enable or disable sections and set turnout directions.

2.3.3 Track control and supervisor elements

XPressNet controller Provides a protocol for sending messages to the *Command Station* component, therefore we can extend the communication form for controlling the track. In the current layout there is a controller and a web-based opportunity for controlling also.

Dashboard Model railway track dashboard implementation, where we can manipulate the track elements. In Addition this component is instantiated only once, and up for the whole time while the track is in use. Consequently we can reach one common dashboard from the web and it contains the actual occupancy and track element status. It is deployed on the Raspberry Pi microcontroller, which starts the service automatically at startup.

Touch board Dashboard for the model railway track, with focus on touchable elements, that can be controlled. This touch board is attached to the demonstrator table itself and start automatically with the Dashboard on the Raspberry Pi microcontroller.

Safety Logic In the MODES^3 safety-critical project we want to avoid the collision of model trains and the turnout derail scenarios, therefore the safety logic software component detects these critical scenarios by Viatra Query [5] patterns and act the necessary action (for example disable a section or the whole track). The component level safety logic is a light-weight implementation with Gamma Statechart Composition Framework [**?**], that also serves this purpose and deployed to each BBB separately.

make bib references for viatra, yakindu

2.3.4 Communication topics

Each software component share informations about the railroad system through the messaging software element, which is based on protobuf messages and provides high-level designed API with MQTT connection for this purpose.

Separated topics are distinguished for different information flows, for which any component can subscribe. Basically all topic is change based, therefore every component must send a signal message to the dedicated topic with the proper information of the change like their state have been changed. For example, if a specific section became free, then the *OccupancyQuery* component sends an OccupancyStateChanged message to the Segment Occupancy topic with free state and with the specific section's ID. Although there is a *SendAllStatusCommand* which asks every component to send their actual status with their unique identifier.

In the following paragraphs , I will list the specific MQTT topics and the connected software components. So basically what kind of information is shared on those topics.

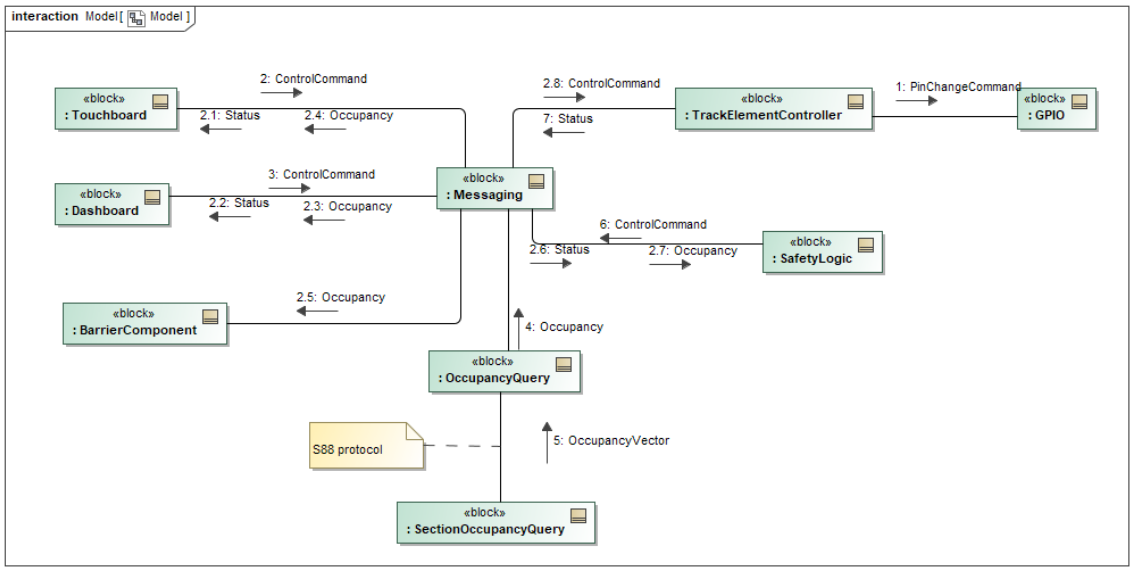

Figure 2.7: Communication between software components

Segment Occupancy topic On this topic the *Occupancy Query* component sends information whether an occupancy's state is changed on a segment. (Notice that a segment can also be a section or a turnout too.) Most of the components are subscribed for this topic as they want to have information which segment is occupied. These components are the *Touch board*, *DashBoard*, *Barrier* (only for the supervised segments), *Track Element Controller* and *Safety Logic*.

Segment and Turnout Command topic Basically the *Track Element Controller* process and accomplish these commands, and *Touch board*, *Dashboard* and *Safety Logic* components can give these instructions.

Segment and Turnout Status topic For every command the *Track Element Controller* will give a status acknowledgment message with the new state of the turnout or section. In this way the *Touch board*, *Dashboard* and *Safety Logic* elements will be informed about current/new state.

Computer Vision topic The camera notification is communicated on that topic, which is received only by the system level *Safety Logic*.

All topic For this specific topic all the components in the system are subscribed, and information about train controlling and *Command Station* related messages are shared.

2.3.5 Communication messages

The following table (Table 2.2) describes the possible message types with their information details also, which is used on all the communication topics.

Table 2.2: Message types

Message Name	Details
	Command messages
SendAllStatus	Every receiver should send back their information about everything they store
DCCOperationCommand	sent via XPressNet protocol by the digital command control (DCC)
SegmentCommand	to enable or disable a specific segment
TrainReferenceSpeedCommand	to set the speed and direction for a specific train with DCC
TurnoutCommand	to set a turnout into straight or divergent state
	Status message
ComputerVisionObjectPositions	position details of a physical object on the track
DCCOperationState	actual status information about DCC operation
SegmentOccupancy	occupied or free status information about a segment
SegmentState	enabled or disabled status information about a segment
TrainReferenceSpeed	train reference speed status information
TurnoutReferenceState	straight or divergent status information about a turnout sent by train controller
TurnoutState	straight or divergent status information about a turnout

2.3.6 Complementary elements

Barrier Handle commands to open/close the barrier via the network.

Computer vision Calculates the coordinates of each elements which have a marker like trains. The position information can be used by the *Safet Logic* to avoid safety-critical scenarios.

2.4 Safety-critical functionalities

Maybe a figure of the Sw-Hw sysml for each paragraph

Occupancy detection In order to know where are the trains on the track, we first must know which track element (section or turnout) is occupied. This attribute can be

determined whether the specific section has power consumption or not, because of a train . The actual detection is made by the *DigiSens-8-S88* sensing element. The demonstration railway system have four sensing elements, and they are connected to an *Arduino* through S88 port. This microcontroller computes basic calculations by *Section Occupancy Query* C++ software component and forwards the 32-bit occupancy vector information (current state of every track element) to the *Occupancy Query* via USB connection. The *Occupancy Query* Java component compares the new occupancy states with the previous one and sends a *SegmentOccupancy* message to the *Segment Occupancy* topic about the change.

Track element controlling For safety-critical purposes, any collision scenario can be avoided by stopping the trains. For this purpose it is a good manner to disable a track element, which is affected in the critical scenario. To make this switch possible, we have to cut the electric circle between the segment and *Command Station*. The *Section Controller* hardware element have been developed for changing a section's state, the *Turnout Controller* is responsible for changing a turnout's state. These hardware elements are attached to a *BBB cape*, which is designed for extending the ports of BBB. In software point of view, through GPIO pins (specific file writing), we can give impulses from the BBB to the section or turnout. Therefore a *GPIO manager* component is responsible for that in connection with the *Track Element Controller* component. Both of them are Java components and deployed to the BBB microcontrollers.

Safety critical verification Because of the network communication, it is easy to connect a *Safety Logic* into the system. In addition a camera (which has a top overview of the table) can read the position of a train on the track. If the *Safety Logic* detects an unsafe scenario with the occupancy detection or camera, it switches off the affected segments or the whole track.

In the demonstrator architecture we can distinguish two different safety logic implementations. First is a component-based approach, which derives each component's scope to the supervised sections of one BBB (as defined in Figure 2.2). Consequently these elements are deployed to every BBB. The other implementation is a system level safety logic and uses Viatra [5] queries on an Eclipse Modeling Framework (EMF) model to avoid safety-critical scenarios. This model is updated with communication messages thus it observes every topic on the network.

Safety Logic: Turnout derail test coverage items (FS-4/2.0)

FS-5/2.0-1 Train is moving through a turnout from top to divergent, while the turnout is in straight state

FS-5/2.0-2 Train is moving through a turnout from top to straight, while the turnout is in divergent state

2.5 Safety-critical scenarios

redraw?

Train collision scenarios To avoid any train collision on a track, it is necessary to measure the distance between the trains. In the demonstrator railway system, we can

measure the position so as well the distance between to trains by using the segment occupation information or with the *Computer Vision* component.

Let's consider an example case (shown in Figure 2.8), where *Train 1* is going to the section where *Train 2* is staying. Notice that no matter in which direction the *Train 2* is moving this is an unsafe situation.

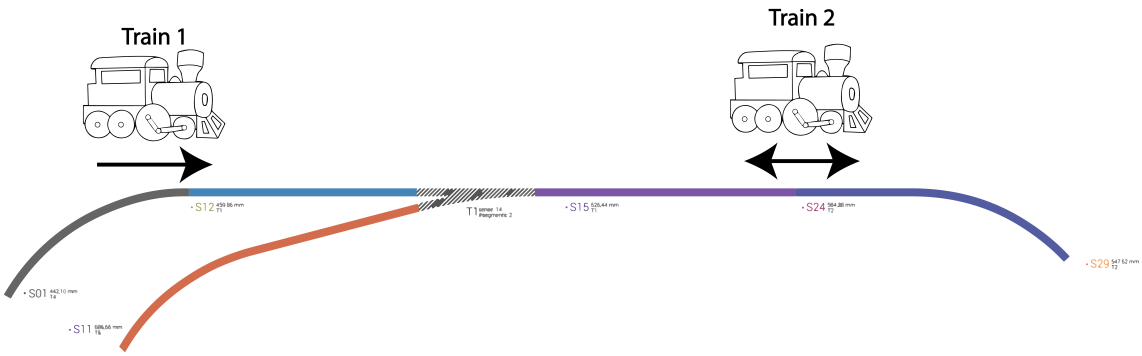

Figure 2.8: Turnout 1 collision

Turnout derail scenarios It is an unsafe situation, if train is approaching a turnout from the straight/divergent edge, but the turnout is set in the other (divergent/straight) state. On the Figure 2.9, the 2 possible derail situation is shown. In the upper case, the turnout is in state direction and any train from the straight branch will go off the rails. In addition the second case is the opposite of the previous one, that any train from the divergent branch will cut the turnout which is in straight state.

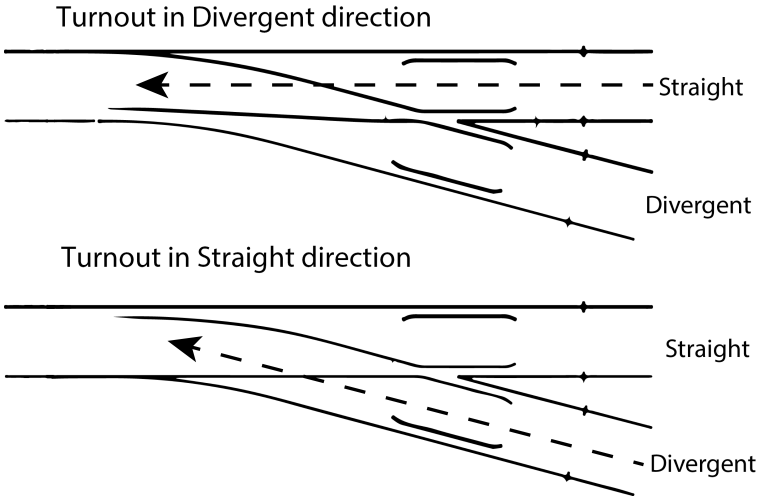

Figure 2.9: Turnout derail

2.6 MoDeS^3 Requirements

In this section I want to collect the requirements for the above described software and hardware extension elements in the system, not considering market products. I have categorized them into functional and component groups.

Create sysml diagrams for better overview

Hardware extensions

REQ-BBB-1 The BeagleBone Black cape must supply 5VDC power source.

REQ-BBB-2 The BeagleBone Black expander must provide further options to attach additional hardware elements. The purpose is to access application processor and PRU unit pins of BeagleBone Black with more elements.

REQ-SA-1 The Segment actuator must stop the train on the given section with the built in *Command Station* (see section 2.1).

REQ-TA-1 The Turnout actuator must switch the corresponding turnout between divergent and straight states.

REQ-TA-2 The Turnout actuator must sense the actual state of the connected turnout.

Occupancy detection software elements

REQ-SOQ-1 The Section Occupancy Query must collect and store the occupancy information about all segments on the track.

REQ-OCQ-1 The Occupancy Query software element must determine whether a train is on a specific segment (results a free occupancy state) or not (results an occupied occupancy state) for each segment.

REQ-OCQ-2 The Occupancy Query software element must send a message when an occupancy state changed for any segment. The component must send all the other current section's occupancy states in this message.

Track segment controller software elements

REQ-GPIO-1 The GPIO software element must be able to change the GPIO pins between their input and output states.

REQ-GPIO-2 The GPIO software element must be able to read the connected GPIO pins.

REQ-TEC-1 The Track Element Controller must be able to enable and disable each section with a specific command.

REQ-TEC-2 The Track Element Controller must be able to change each turnout's direction to straight or divergent state.

Track control and supervisor software elements

REQ-DB-1 The Dashboard must observe the track element states throughout the whole life-cycle.

REQ-DB-2 The Dashboard must be able to change all turnout states.

REQ-DB-3 The Dashboard must be able to change each turnout state separately.

REQ-DB-4 The Dashboard must be able to enable and disable all segments on the track.

REQ-DB-5 The Dashboard must be able to enable and disable each segment on the track, separately.

REQ-SL-1 The Safety Logic software must stop the whole track itself, when 2 trains are in a distance of 3 segments or less.

REQ-SL-2 The Safety Logic must ensure that a train must not go on a disabled segment.

REQ-SL-3 The Safety Logic must ensure that a train must stop in the straight direction of the turnout, when the turnout is in divergent state.

REQ-SL-4 The Safety Logic must ensure that a train must stop in the divergent direction of the turnout, when the turnout is in straight state.

Chapter 3

Test design and documentation

3.1 Test design techniques

The MoDeS^3 is a multi-layered, safety-critical application, which consist of off-the-self and custom made components also. As a complex system it is crucial to have a well-designed testing process and documentation which helps determine any problem in the system. For this purpose in the following chapter I will examine the possible and most suitable test approaches.

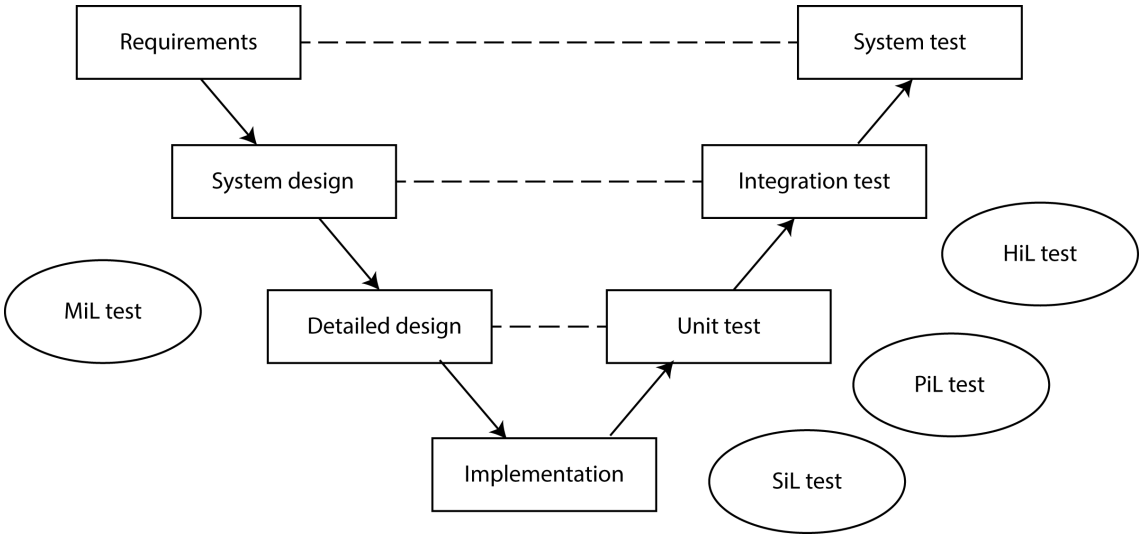

Figure 3.1: V-model levels overview. Adapted from [**?**]

3.1.1 V-model and testing levels

Nowadays one of the most popular software development methodology is the V-model [7] (shown on figure 3.1), which defines four stages during the development and four verification steps accordingly (each step is shown with rectangles on the figure). In addition for making the development more effective for error-detection, there are four more steps defined for verifying our system's correctness, called test levels [2] (shown with ellipses). While the development starts with an abstract **requirement analysis**, which declares the aim of our application, later on each step contains more detailed information. The second phase is about understanding the abstract user requirements and define the sys-

tem's **functional specification** by the developers. During this phase our verification is based on **Model-in-the-Loop (MiL) testing**. The third phase contains all the low-level design and specific information about the application. From these design decisions, the **implementation** should be quite straight-forward or even it can be generated. During the **Software-in-the-Loop (SiL) testing**, the implementation will be verified for a subset of functionalities, which is depending on the system's purpose and requirements. As of now our implementation is done and our design decisions are verified, we make sure with **unit tests** that our functions are serving their exact purposes. **Processor-in-the-Loop (PiL)** testing ensures that the computing processor at the bottom of the V-model works as expected. Then moving up in the V-model with **integration tests**, we are focusing on more abstract and complex components. In this phase we consider software and hardware elements also, completed with **Hardware-in-the-Loop (HiL)** testing. The last step is about verifying the complete system in real environment and check the high-level requirements.

3.1.2 Specification-based testing techniques

During the test design process and implementation phases, the conception can be a quite complex and difficult task. Nevertheless the test design techniques can be useful in these situations. The commonly used approaches are described in this section, which are used during the MoDeS^3 test design process, based on [1]. These methods can be divided into 3 branches, which are specification-based, structure-based and experience-based techniques. The first is deriving the tests from the system specification details. In opposite to that the structure-based approach is based on the system architecture (in software systems the implementation architecture). The third branch is a way to test a system, which is lack of structure and specification documentation.

The following paragraphs are details 3 design technique, which are belongs to the specification-based area.

Scenario Testing The first technique is consists of specifying the interactions between the test items and the intervened system, which is usually a user. Therefore this technique is also called use-case testing.

Equivalence Class Partitioning The key point in the equivalence class partitioning is to categorize the test item inputs and outputs into classes, which are similar in these aspects. In addition these partitions can cover valid and invalid inputs or outputs also.

Decision Table Testing If the test items can be identified with a logical relationship between the input and output values where these values can have only 2 partitions, then during the test design the decision table testing can be used.

3.2 Test documentation

During the test design phases of the MoDeS^3 I will follow a subset of the ISO 291119 standard [1]. First this section describes the documents in general, which will be later used in the MoDeS^3 test documentation process in section Chapter 4. As of now the organizational and role details are irrelevant for this project, I will just give a short example

for those. In the following sections, I will subtract the documentations into 3 phases, called Organizational, Test Management and Dynamic Test Documentation.

redraw figure to something similar

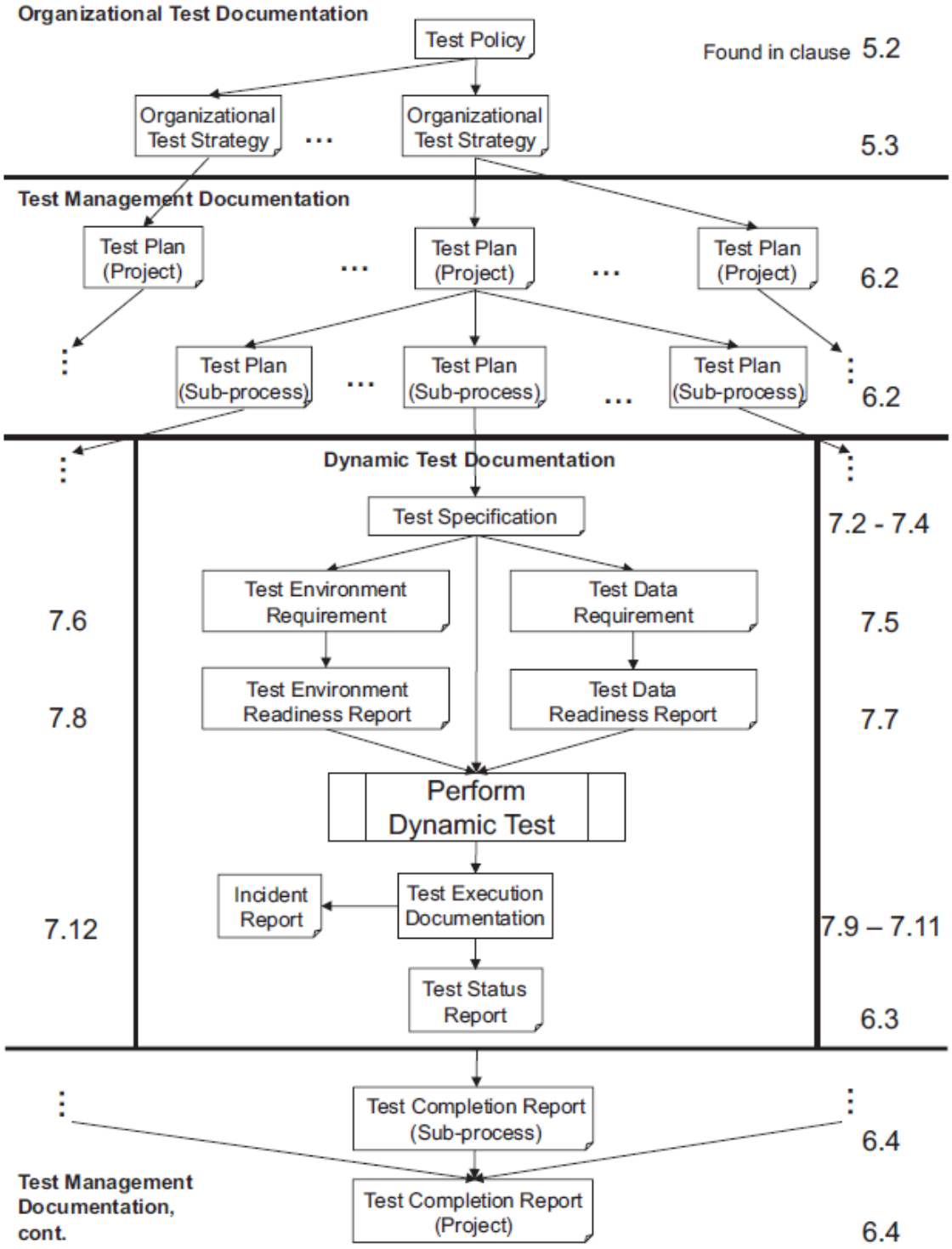

Figure 3.2: Test Documentation Overview. Adapted from [1]

3.2.1 Organizational test documentation

These general documents must be distributed organizational-wise and must be the same for every project within the organization. In according to this, during the test documentation the organization will be the MoDeS^3 team and the only project will be the above detailed railway system.

Test Policy Throughout the test development a Test Policy must contain the test principles and objectives for the whole organization. It clarifies what should be tested during development, but not how testing must be implemented. Furthermore it must define the provisions to be used for establishing, improving, maintaining and reviewing the test Policy.

Test Strategy A Test Strategy must define a proper guideline on how testing should be performed for all projects in the organization. In our case one Test Strategy is sufficient, but for an organization which have projects in different technical areas, can have a Test Strategy for each area. In latter case, the organization can define separate Test sub-processes to satisfy any special need for an area or project.

3.2.2 Test Management Documentation

The Test Management Documentation is focusing on one project's planning and test evaluation. This group of documentation provides guidelines for Test Plan, Test Status Report and Test Completion Report.

Test Plan The collection of the test planning and test management documents is the Test Plan. It's scope be mapped to multiple projects, a single project or to sub-projects, where each Test Plan is dedicated for a sub-process (For example system test plan, integration software test plan, sub-system test plan, or unit software test plan). In a complex design structure it is advised to have a mapping tree document, which shows the relations between these Test Plans.

Test Status Report A report from the currently ongoing test process in a given reporting period is the Test Status Report. For example in an agile project it can be delivered at the end of the iteration. Although it is not necessarily a written document.

Test Completion Report As a summary of our test execution, a Test Completion Report can be provided for every test sub-process or for the whole project itself.

3.2.3 Dynamic Test Documentation

During the Dynamic Test Documentation phase the following documents are prepared:

- Test Specification with sub-documents of:
  - Test Design Specification
  - Test Case Specification
  - Test Procedure Specification

- Test Data Requirements
- Test Environment Requirements
- Test Data Readiness Report
- Test Environment Readiness Report
- Test Execution Documentation with sub-documents of:
  - Actual Result
  - Test Results
  - Test Execution Log
  - Incident Report

In the following sections, I will give a brief summary for each documentation.

Test Design Specification The features and test conditions (derived from test basis) are collected in the Test Design Specification with the specific test cases and test procedures. These detailed test cases will be executed during testing process.

Here we can create groups called *feature sets* for those features which are connected together, then we can handle them independently. A *test condition* is aligned for each feature set and can be verified by a test case. Technically a test condition means a concrete state or representation of a feature. In addition on or more requirement can be referenced by a test case. In addition, to develop a more maintainable test document we must take care for the references from test cases to each requirement.

Test Case Specification One or more feature set with the defined test conditions and coverage items, constitutes a Test Case Specification.

Test coverage items are obtained from a set of test conditions and specific test design technique. These items can be summarized in a list (describing test coverage items, test conditions and feature sets with proper priorities). From each test coverage item a *Test case* can be derived, which verifies the correct implementation of a function. Regarding the possible number of test cases it can be collected into a list, table or database. Each test case should have a proper definition of dependencies. These are collected in Table 3.1.

Test Procedure Specification Describes the precondition setup and test cases execution in the proper order with possible post evaluation activities.

Those test cases, which have the same purpose in the test item's property (for example precondition, test basis or even in the identified risk) can be collected into a *Test set*. Typically this will reflect to one or more feature sets. For each *Test set* the proper execution order is defined in the *Test procedure* considering test case dependencies, preconditions, postconditions and other environment requirements.

Test Data Requirements Defines the required properties of the test data, which is used during test procedure execution. These requirements can be summarized with a list. In addition, this detailed test data requirements makes it more maintainable and traceable.

Table 3.1: Details of a test case

Aspect	Description
Priority	Defines the execution order and importance between test cases. Higher priority test cases should be examined earlier then lower priorities
Traceability	Reference the parent test coverage, condition item and the feature requirement
Preconditions	Describes special states that must exists to execute the test case (Can be other test cases also)
Inputs	A specific action which sets the test item into a state, where the actual and expected results can be compared
Expected result	Specifies the expected output and behavior (with tolerances) of the test item which was in the precondition state and was modified with the proper inputs
Actual result, test result	A description of test item outputs and a comparison between actual and expected results

Test Data Readiness Report Gives a list of Test Data Requirements determining whether they are satisfied or not.

Test Environment Requirements Describes the required test environment properties, which are setting the test items into the expected test environment. These requirements can also be grouped by environment types like hardware, middleware, software, tools or security and may vary for each test case.

Test Environment Readiness Report Lists the fulfillment of Test Environment Requirements.

Actual Results After executing the selected test procedures, each test case's actual result (output state or behavior after the input actions) should be recorded. It may require full recording of the process with an automated tool also.

Test Result Determines if the actual results are corresponding to the expected results with the specified deviation or not. It is usually recorded as passed or failed, along with the actual results.

Test Execution Log Detailed documentation of one or more test procedure's execution. Can be represented in a list or table with measuring times and events of the test execution.

Test Incident Reporting If there is any deviation between the actual and expected results after each test procedure execution, we should create a dedicated Incident Report

detailing the problems. These reports could contain a description of the problem with severity, date and possible risk information.

Chapter 4

Test Planning for MoDeS^3 project

4.1 Test Policy for MoDeS^3 organization

Scope: The following Test Policy must be followed by the MoDeS^3 team in the Department of Measurement and Information Systems.

Introduction: The railway system project have started several years back, and from time to time the team faced smaller and bigger problems with the implemented product. Presumably a high percentage of them could be prevented with a detailed test documentation and process. To avoid these situations in the future I will now introduce a test documentation approach for the demonstrator railway system.

Objectives of testing: The objective of testing is to measure and improve the software quality and to avoid safety-critical failures in the system.

Standards: This test documentation will follow the ISO/IEC/IEEE standard 291119-3 "Test Documentation" part [1].

Test improvement: Every student or a group of them who participated in the test implementation should give an overview about the test results and their execution aligned with the test documentation approach. Any further developed component should be covered by test cases with the corresponding test documentation. Consequently the requirements, the test plan sub-processes (unit, integration and system test plans) and the design specifications must be extended to cover the newly attached or modified components.

Test evaluation: Each extension should be handled as a github pull request, which must be reviewed by 2 of the team members and additionally must be checked by the student's supervisor.

4.2 Test Strategy for MoDeS^3 projects

Scope: The following test strategy is applicable for the demonstrator railway system project (described in Chapter 2). This strategy's aim is to support testing implementation

and maintainability at the software development phases in component, integration and system level also.

Risk management: A risk management must be handled for test plans separately. The detailed format must be a traceable list or table.

Test selection prioritization: The test cases and test procedures will follow a bottom-up strategy order, aligned with safety-critical risk levels. Consequently a test element with lower dependencies and more independent functionalities (also more hardware related) got higher priority than a complex test element, which relies on multiple sub components.

Test document and reporting: The test process and documentation must be well-separated and clear that a new team member can easily understand the structure.

Test automation and tools: All tools, which is used during the test process and documentation must be available for a university student (meaning education license or freeware).

Incident management: All detected defects must be created as a github issue. [1]

Test sub-processes: All test projects must provide the following test levels: unit test plan, integration test plan, system test plan.

4.3 Master Test Plan for Railway System project

Scope: Test plan's scope is to provide the necessary test framework for executing tests for the demonstrator railway system of MoDeS^3 team. This document identifies a way for planning, executing and maintaining tests in a multi-layered project.

Plan context: The railway system architecture (detailed in Chapter 2) consist of off-the-shelf and custom hardware and software elements. According to this, the *test items* are only the custom software elements, which were made by the MoDeS^3 team. Details about test items can be found in Section 2.3. Other product testing and the custom hardware extension verifications are not part of this Test Documentation. Furthermore not all the software component have been involved in this testing plan, because not safety-critical components have been skipped. *Barrier* is irrelevant regarding the train-collision detection. An additional way to set the speed and directions of the trains is not safety-critical either, so *LeapMotion* and *XPressNet* components are not safety-critical (Note that an other components are capable to detect train positions on the railway). *Touchboard* is a component which have a reduced functionality considered to *Dashboard*, so we will skip that also in this test plan.

[1] MoDeS^3 github page is available at the following page: `https://github.com/FTSRG/BME-MODES3`

Risk register: On the following 2 tables the possible product (Table 4.1) and project risks (Table 4.2) are described. The following assessment will cover the risk management for all the test sub-plans in the MoDeS^3 project.

Table 4.1: Product risks

ID	Risk details	Mitigation activities	Level
		Occupancy detection	
1	Wrong or no occupancy detected	Review of appropriate hardware and software elements considering network connection Track element controlling	High
2	Not controllable turnout or segment	Review of hardware and software components considering standard railway elements also. Safety critical verification	Above middle
3	Wrong collision avoidance decision made	Review of design and safety algorithm. Extra test cases to cover incorrect scenarios	High
4	Train collision	Review of design and safety algorithm.	High

Test strategy: For the railway system project, the following test sub-processes are defined:

- Unit test (see in Section 4.3.1)
- Integration test (see in Section 4.3.2)
- System test (see in Section 4.3.3)

In addition for all sub-processes the following documents must be delivered:

- Test sub-process plan
- Test specification
- Test log
- Test sub-process completion or status report

The test design techniques, test completion criteria and test data must be applied accordingly for every test sub-process. We use the following test environments in general for every test sub-process:

- Eclipse Photon where applicable
- Visual Studio Code where applicable
- JUnit Jupiter for Java base components
- Mockito with PowerMockito for components written in Java

Table 4.2: Project risks

ID	Risk	Mitigation activities	Level
1	Busy students and estimation	For a student every semester is different and not always predictable how much time will the student have for the project. Therefore it must be consider during project estimation and planning accordingly.	Middle
2	New students and planning	This is a university project, so we must calculate with the often changing student and knowledge transfer about the existing system.	High
3	Cutting-edge tools	Unpredictable cutting-edge technologies can make a huge rule during project planning. The compatibility of tools and their reliability can differ during few month also, which must be adapted during estimation.	Above middle

- GTest for components written in C++
- Railway track elements

Test activities and estimates: These are confirmed personally and aligned with university studies.

Stuffing: The actual team and roles are highly dependent with the MoDeS^3 team in every semester.

Retesting and regression testing: This is specified for each test sub-process.

4.3.1 Unit Test Plan

Scope: The aim for this Test Plan is to verify each software component's functionality in the system. This plan is derived from a Master Test Plan (detailed the Section 4.3 for Master Test Plan).

Plan context: Unit Test Plan context is the same as for Master Test Plan (see Section 4.3), because all self developed software component must be tested in a standalone environment also.

Risk register: Same as Master Test Plan.

Test strategy: The deliverables are this test plan, the unit test specification and the test status report. For every software component the code coverage measurement must reach at least 80%. This means that for each component the implemented unit tests must execute greater or equal then 80% of the component's code.

4.3.2 Integration Test Plan

Scope: This test plan is stands for the integration level testing examination. This approach is one abstraction layer above from the unit tests, because the following test cases verifies the correct functionalities between one or more components. This test plan is also derived from a Master Test Plan (detailed the Section 4.3 for Master Test Plan).

Plan context: In this plan we aim to test the system divided into several groups, consequently the Integration Test Plan's context is the same as for Master Test Plan (see Section 4.3) and for Unit Test Plan.

Risk register: Same as Master Test Plan.

Test strategy: The deliverables are this test plan, the integration test specification and the test status report. It is required to cover every existing connection between 2 components with at least 1 parameter of a message type. (Component communication is described in Figure 2.7.)

4.3.3 System Test Plan

Scope: The third test plan is designed to test the demonstrator functions at system level. These tests are verifies the system's commonly demonstrated features. This test plan is also derived from a Master Test Plan (detailed the Section 4.3 for Master Test Plan).

Plan context: This plan's focus is the whole railway system, therefore the context is the same as for Master Test Plan (see Section 4.3).

Risk register: Same as Master Test Plan.

Test strategy: The deliverables are this test plan, the system test specification and the test status report. The test cases must cover the safety-critical functionalities (see details in Section 2.4) of the demonstrator system and greater than 50% of the system requirements (described in Section 2.6) for every component.

4.4 Test Design Specification for Unit Test Plan

Purpose: The purpose of this test specification is to give a guideline for executing unit tests for the demonstrator railway system components.

References: The railway system related requirements can be found in Section 2.6.

4.4.1 Feature Sets

We can easily separate the system's functionalities into feature sets following the referenced system requirement's structure, which are shown in Table 4.3.

Table 4.3: Feature sets

Feature Set		Scope	Priority	Approach	Traceability
ID	Name				
FS-1	GPIO handling	To test GPIO pin handling	Am	Scenario Testing with Equivalence Class Partitioning	REQ-GPIO-1, REQ-GPIO-2
FS-2	Occupancy detection	To test the occupancy related functionalities in the system, including state and change detection	H	Decision Table Testing	REQ-OCQ-1, REQ-OCQ-2, REQ-SOQ-1
FS-3	Track element controller	To test segment availability and turnout direction setting	Am	Scenario Testing	REQ-TEC-1, REQ-TEC-2
FS-4	Safety Logic	To test railway system's safety logic	H	Scenario Testing	REQ-SL-1, REQ-SL-2 , REQ-SL-3, REQ-SL-4
FS-5	Dashboard	To test Dashboard capabilities	L	Scenario Testing	REQ-DB-1, REQ-DB-2, REQ-DB-3, REQ-DB-4, REQ-DB-5

4.4.2 Test Conditions

In the following section I will describe the test conditions for each feature set. Most of the feature sets in the Unit Test Plan are simple low level functions so the test conditions also will describe the test coverage items.

GPIO handling (FS-1) Considering the GPIO's functionality the following test conditions are test coverage items also. With the usage of use case based design approach, the

GPIO can be used for input and output communication. The proper setup and operation cases are detailed in Table 4.4.

Table 4.4: GPIO handling test condition and coverage items

Test condition	Direction	Phase	Configuration file	Setting
FS-1/1.0	Input	Initialization	edge	both
FS-1/1.1		Operation	value	low
FS-1/1.2				high
FS-1/2.0	Output	Initialization	value	low
FS-1/2.1		Operation		low
FS-1/2.2				high

Occupancy detection (FS-2) The 2 test condition items are also considered as test coverage items, which are shown in Table 4.5). The detection is made periodically in the demonstrator system life-cycle and the decision can be that the section is either free or occupied.

Table 4.5: Occupancy detection test condition and coverage items

Test condition	Value	Comment
FS-2/1.0	Free	Specific segment is free
FS-2/1.1	Occupied	Specific segment is occupied

Track element controller (FS-3) A track element controller can handle segment and turnout specific statements also, consequently we must distinguish our test condition and coverage items for these segment types. The merged test condition and coverage items are shown in Table 4.6. The specific use cases for a segment, that it can be enabled or disabled by the controller whether a train can drive on it or not. Consequently the use case for the turnout to change its state between straight and divergent.

Table 4.6: Track element controller test condition and coverage items

Test condition	Affected element	State
FS-3/1.0	Segment	Enabled
FS-3/1.1		Disabled
FS-3/2.0	Turnout	Straight
FS-3/2.1		Divergent

Safety Logic (FS-4) The safety-critical aspects are shown in Table 4.7, which should be handled by the Safety Logic.

Table 4.7: Safety Logic test condition items

Test condition	Safety Level	Safety type
FS-4/1.0	System	Train collision
FS-4/2.0		Turnout derail

Dashboard (FS-5) A full use case coverage is aimed for the Dashboard which can be defined by the following test condition and coverage items, shown in Table 4.8.

Table 4.8: Dashboard test conditions

Test condition	Scope	Section type	State
FS-5/1.0	All	Turnout	Straight
FS-5/1.1			Divergent
FS-5/1.2		Segment	Enabled
FS-5/1.3			Disabled
FS-5/2.0	Each	Turnout	Straight
FS-5/2.1			Divergent
FS-5/2.2		Segment	Enabled
FS-5/2.3			Disabled

4.4.3 Test coverage items

create diagrams for each test coverage item

Safety Logic: Train collision test coverage items (FS-4/1.0)

FS-5/1.0-1 1 Train is moving to a segment where there is an other train

FS-5/1.0-2 1 Train is moving to a segment where the other train is in 1 distance away on any the path

FS-5/1.0-3 1 Train is moving to a segment where the other train is in 2 distance away on any the path

Safety Logic: Turnout derail test coverage items (FS-4/2.0)

FS-5/2.0-1 Train is moving through a turnout from top to divergent, while the turnout is in straight state

FS-5/2.0-2 Train is moving through a turnout from top to straight, while the turnout is in divergent state

4.4.4 Test cases

In this section I will describe the test cases defined by test conditions and coverage items for each feature set. In addition for all the feature sets just one test case will be mentioned and additional test cases can be found in the A.2 appendix.

Gpio Handling (FS-1) test cases In order to work in production with GPIO component, the following configuration file must exists: sys/gpio/gpioPIN/export, sys/gpio/gpioPIN/value and sys/gpio/gpioPIN/edge. This need is also propagate to all test cases regarding GPIO handling, so files must be accessible or mocked.

Table 4.9: Test case 1-1

Test case ID: 1-1	Purpose: to test the GPIO initialization in input direction. Priority: am Tracing: FS-1/1.0
Precondition	The GPIO's necessary files are available.
Input	Initialize the GPIO itself with input direction.
Expected result	The "both" string have been written to "edge" configuration file.

Occupancy detection (FS-2) test cases Occupancy detection components cannot be easily separated into subcomponents for testing purpose, because the communication between their parts is based on a serial port connection. There is no fake serial port

connection available on the market for windows operating system, so we must test them together with real hardware connection or modify our implementation.

Table 4.10: Test case 2-1

Test case ID: 2-1	Purpose: to test the detection of segment occupancy (the train power consumption) through section occupancy query, when the specific segment is free Priority: above middle Tracing: (FS-2/1.0)
Precondition	S88 serial port connection and available Arduino hardware element
Input	Unclosed circuit between the specific segment's hardware elements elements
Expected result	Occupancy components have queried free occupancy state

Track element controller (FS-3) test cases This component can further be divided into segment and turnout controllers like on hardware level. Although in this test plan, it does not detailed into further test cases for the specific controllers, because in software point of view, there is no need to distinguish them.

Table 4.11: Test case 3-1

Test case ID: 3-1	Purpose: to test the track element controller's segment state setting as enabled Priority: above middle Tracing: (FS-3/1.0)
Precondition	Observable GPIO components
Input	Call the track element controller set segment state function with enabled parameter
Expected result	All GPIO levels are in "HIGH" state, which are related to the specific segment

Safety logic (FS-4) test cases The following test cases are related to system level safety logic feature set.

Table 4.12: Test case 4-1

Test case ID: 4-1	Purpose: to test the safety logic awareness, when a train is moving on a path where the next section in the direction already occupied by an other train Priority: high Tracing: (FS-4/1.0)
Precondition	None
Input	Insert a train to a specific segment and move an other train to the adjacent segment
Expected result	Safety Logic sent a segment disable command with the id of the specific segment

Dashboard (FS-5) test cases The following test cases are Dashboard feature set related items. This component can control and visualize the current track element states, consequently the unit test cases are checking the control network messages.

Table 4.13: Test case 5-1

Test case ID: 5-1	Purpose: to test the dashboard's set all turnout to straight functionality Priority: am Tracing: FS-5/1.0
Precondition	All turnout must be in divergent state
Input	Simulate a button press to the change all turnout direction function
Expected result	Message have been prepared to send with straight and a turnout id parameter for all turnouts

4.4.5 Test Procedure Specification

In this Unit Test Plan, each feature set is derived from a custom software component in the demonstrator railway system. Regarding this property each test set can involve a group of test cases which are related for that feature set. A unit test must be a fast, isolated, repeatable, self-validated execution, therefore test set's ordering should not influence the test results. Consequently we can execute them independently and parallel but we can also give an order, for example as the test cases ordered in the Test case definition section (described in Section 4.4.4) and one test procedure can contain one test set.

4.5 Test Design Specification for Integration Test Plan

Purpose: The aim of the following test specification is to give instructions about setting up and executing integration tests. These tests will verify two or more component's behavior together.

References: The railway system related requirements can be found in Section 2.6.

4.5.1 Feature Sets

In the (4.14) table the components of the demonstrator railway system are grouped into integration feature sets separated by functional purposes.

Table 4.14: Feature sets for Integration Test Plan

Feature Set		Scope	Priority	Approach	Traceability
ID	Name				
FSI-1	Occupancy message	To test the occupancy network messages sent out by the occupancy components	Am	Scenario Testing	REQ-OCQ-1, REQ-OCQ-2, REQ-SOQ-1
FSI-2	Track element controller instructions	To test Track element controller and GPIO handling functionalities, manipulating by network command messages	Am	Scenario Testing	REQ-TEC-1 REQ-TEC-2 REQ-GPIO-1 REQ-GPIO-2
FSI-3	Safety logic interaction	To test the occupancy detections with safety logic reactions and their network command messages	H	Scenario Testing	REQ-SL-1 REQ-SL-2 REQ-SL-3 REQ-SL-4 REQ-OCQ-1, REQ-OCQ-2, REQ-SOQ-1
FSI-4	Safety logic intervention	To test the safety logic intervention through Track element controller by network command messages	H	Scenario Testing	REQ-SL-1 REQ-SL-2 REQ-SL-3 REQ-SL-4 REQ-TEC-1 REQ-TEC-2

4.5.2 Test Conditions and coverage items

The test condition and coverage items for the integration tests can be defined with the same aspects as the specific component in the unit test plan. The following enumeration is showing a mapping between the integration and unit test feature sets.

1. Occupancy message (FSI-1) - Occupancy detection (FS-3) (4.5)
2. Track element controller instructions (FSI-2) - Track element controller(4.6)
3. Safety Logic interaction (FSI-3) - Occupancy detection (FS-3) (4.5)
4. Safety logic intervention (FSI-4) - Safety Logic (FS-4) (4.7)

4.5.3 Test cases

The following test cases are a subset of all integration test cases, which are fully detail in Section A.3.

Occupancy message (FSI-1) text cases The occupancy related test cases are verifying the network message sent by the Occupancy Query software element.

Table 4.15: Integration test case 1-1

Integration test case ID: 1-1	Purpose: to test the detection of segment occupancy when a segment is free and to verify the propagated network occupancy message Priority: am Tracing: FS-2/1.0
Precondition	There must be an MQTT server connection available and an connected with serial port
Input	Unclosed circuit between the specific segment's hardware elements
Expected result	A new segment occupancy message must be send to the network with free segment state and the specific segment id

Track element controller instructions (FSI-2) test cases The main focus in the following test cases is to send segment and turnout command messages to the Track element controller. We can further observer the file output for the supervised GPIOs.

Table 4.16: Integration test case 2-1

Integration test case ID: 2-1	Purpose: to test the track element controller, that it enables its supervised segment's state Priority: am Tracing: FS-3/1.0
Precondition	There must be an MQTT server connection available
Input	Send a SegmentCommand message with enabled state and a segment id which is supervised by the track element controller component
Expected result	All related GPIO (pru and app) has the writer with value "1" and targetFile "value"

4.5.4 Test procedure

In the integration test plan all the FSI-1 feature set related test cases requires hardware components during test execution. Apart from that all other test cases are purely software component tests, therefore they can be executed separately.

4.6 Test Design Specification for System Test Plan

Purpose: The system test plan is describes a road map to verify the high-level requirements of the railway system.

References: The related requirements are previously described in Section 2.6.

4.6.1 Feature Sets

Table 4.17: System feature sets

Feature Set		Scope	Priority	Approach	Traceability
ID	Name				
FSS-1	Track element availability verification	To test the railway system track element's proper operations	Am	Scenario Testing	REQ-DB-1 REQ-DB-2 REQ-DB-3 REQ-DB-4 REQ-DB-5
FSS-2	Safety logic verification	To test the safety logic functionalities	H	Scenario Testing	REQ-SL-1 REQ-SL-2 REQ-SL-3 REQ-SL-4

4.6.2 Test condition and coverage items

From the previously described (4.17) feature sets the test condition and coverage items can be derived, which are detailed in the following sections.

Track element availability verification (FSS-1) The purpose of this test condition is to verify that all segments and turnouts are available with a fast functional test.

Table 4.18: Train detection test conditions

Test condition	Scope	Section type
FSS-1/1	All	Turnout
FSS-1/2		Segment

Safety logic verification (FSS-2) The system level *safety logic verification* the test conditions are the same as it was previously shown for unit tests in Table 4.7. To summarize that, the conditions can be divided into turnout derail and train collision scenarios.

4.6.3 Test cases

All test cases details can be found in Section A.4.

Track element availability verification (FSS-1) The following test cases will use network segment and turnout command messages as previously described in Section 4.5.3 integration test case.

Table 4.19: System test case 1-1

System test case ID: 1-1	Purpose: to test all turnout controllability Priority: am Tracing: FS-6/1.0
Precondition	None
Input	Send a switch turnout command to all turnouts twice
Expected result	All turnout state have been changed to straight from divergent and the other way

Safety logic verification (FSS-2) The below detailed safety logic test cases instead of checking the whole table, will just verify the safety logic decisions in the most problematic scenarios.

Mention good figures about the layout here

Table 4.20: System test case 2-1

System test case ID: 2-1	Purpose: to test the safety logic for turnout derail scenario Priority: am Tracing: FS-6/1.2
Precondition	Turnout T5, T1 is in straight state and a train is on the segment S13
Input	Move the train to segment S15 from segment S13 through the path of S13, S8, T5, S11, T1, S15.
Expected result	Before T1 turnout the whole railway system is disabled by the safety logic to avoid turnout derail

4.6.4 Test procedure

System test procedure for FSS-1 The following procedure is focusing on verifying the track element controller functionalities.

1. Objective: FSS-1
2. Priority: high
3. Start up: all the track element controllers should be started properly
4. Stop and wrap up: all track element controllers on the components must be stopped

Table 4.21: System test procedure for FSS-1

Test case name	Actual results	Test result
1-1: change all turnout		
1-2: disable all segment		
1-3: enable all segment		

System test procedure for FSS-2

1. Objective: FSS-2
2. Priority: high
3. Start up: place 2 trains on the sections of S13 and S15
4. Stop and wrap up: restart the disabled sections or the whole track

Table 4.22: System test procedure for FSS-2

Test case name	Actual results	Test result
2-1: turnout derail		
2-2: train collision		

4.7 Test Environment Readiness Requirement

Hardware The demonstrator railway system hardware elements are fixed to 2 tables, which have all the necessary layout and electronic dependencies already set up. Before starting the demonstrator table, the 2 parts must be properly connected.

Software To build the Java language elements of the code base, you have to install Gradle and Java 8 SDK. For C++ it is advised to use minGW with Visual Studio. The deployment prerequisite is the Ansible tool, which only available for unix-based system.

4.8 Test Incident Report

Any problem in the MoDeS^3 project, must be addressed as a github issue. A template for this purpose is shown below on Table 4.23.

Table 4.23: System test result for procedure FSS-1

Issue title	Severity	How to reproduce	Description
Brief summary for the issue	Low / Medium / High	Give a few steps, when and how the issue is appearing	Long description and assumptions about the root cause if there is any

Chapter 5

Test Implementation and Execution for MoDeS^3 project

5.1 Test Environment Readiness Report

Hardware The entire table have been properly set up in a university laboratory.

Software The Eclipse Photon (4.8.0) have been set up with the necessary dependencies (Gradle, Java, Viatra, Xtend, Xtext, E(fx)clipse) and for deployment purposes a linux-subsystem for windows have been accepted.

5.2 Test Implementation details

This section describes the technical details of test implementation and execution for each test plans of MoDeS^3 project.

Deploying For implementation and testing purposes it is not recommended to use the real microcontrollers (BBB, Pi or Arduino itself). During my thesis work, I have used Windows 10 operating system with Eclipse and Visual Studio Code (which are available also for Unix and Mac systems), but for deploying purposes the MoDeS^3 team uses Ansible[1], which is not available for Windows systems. Fortunately there is an option to create unix subsystem inside a Windows operating system, where from I have managed to deploy the necessary components to the BBB.

5.2.1 Unit tests

Considering the unit test feature sets (Table 4.3), the GPIO Manager, Occupancy Query, Track Element Controller, Safety Logic and Dashboard components are written in Java or Xtend[2] languages and the Section Occupancy Query is implemented in C++ for the Arduino.

[1]For more info visit `https://www.ansible.com/` site.

[2]Xtend is a general purpose programming language, from which java code is generate. More information is available here: `https://www.eclipse.org/xtend/documentation/index.html`

The first task is to set up the proper environment in Eclipse with Gradle and JUnit 5 [4] plugins to test Java components. To create additional objects for verification and faking purposes (like avoid using file operations during testing), Mockito [8] and PowerMock [3] was added to the environment. The first extension is inheriting the objects, which should be faked in the test and replacing it in the background to a fake object. These objects will behave as the test case requires them, like returning a specific value for the exact parameters. Additionally these objects can also be verified that how many times and with which parameters they have been called. The PowerMock extends these functionalities with faking for example static and private methods, which can not be done with an inheritance (so with Mockito framework). Unfortunately the JUnit team have not yet implemented the possibility to use PowerMock in JUnit 5 tests, but the framework supports running JUnit 4 tests. To summarize that, with the setup of JUnit 5 and PowerMock for Mockito we can test static methods (which is commonly used for Singleton pattern). Each component must reach an 80% of code coverage defined by the Unit Test Plan. Although the measurement could not be run with jacoco plugin, because it does not applicable for Xtend programming language. One alternative tool is the EclEmma[3] Eclipse plugin.

For the C++ language based Section Occupancy Query component it is advised to use Visual Studio Code with PlatformIO extension[4]. The tests can be implemented for this component in the Google Test framework[5].

FS-1: GPIO Handling The handling of GPIO pins is based on the GPIO sysfs interface[6], which provides a file based GPIO controlling with */sys/class/gpio/* root folder for unix systems. Therefore the GPIO Manager Java component must handle file operations in the specific folders. In order to test the logic regarding GPIO handling, the file writing and reading operations must be extracted into separate classes.

On Figure 5.1, a class diagram shows the re-factored structure of the component. This structure gives an advantage that the file operation dependencies are extracted into interfaces, which we can be easily mocked. The following code snippet shows how a fake object can be created and verified.

```
private static final String GPIOFOLDER = "/sys/class/gpio/";
// Create a fake class which behaves as any ICommandWriter class
@Mock
private ICommandWriter writer = Mockito.mock(ICommandWriter.class);
// Verify that the 'writer' fake object have been called
// with 1 time during the execution with these exact parameters
Mockito.verify(writer, Mockito.times(1)).executeCommand(String.valueOf(67), GPIOFOLDER + "export");
```

As explained before the static methods can only be mocked with PowerMock, thus the following example shows how to make the GpioProvider to return a fake Gpio instance when the *getGpioInstance* method is called with parameters: 86, Gpio.Direction.IN. Therefore in any further tests we can examine the mGpio instance as explained before.

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(GpioProvider.class)
public class GpioManagerTest{
  @Mock
  private Gpio mGpio = Mockito.mock(Gpio.class);

  @Before
  public void initEnv(){
    // Setup powermockito for static GpioProvider mocking
```

[3]The plugin and their usage is described here: https://www.eclemma.org/
[4]Detailes about the framework are available here: https://platformio.org/
[5]Introduction can be found here:https://github.com/google/googletest
[6]Interface is explained here in details: https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

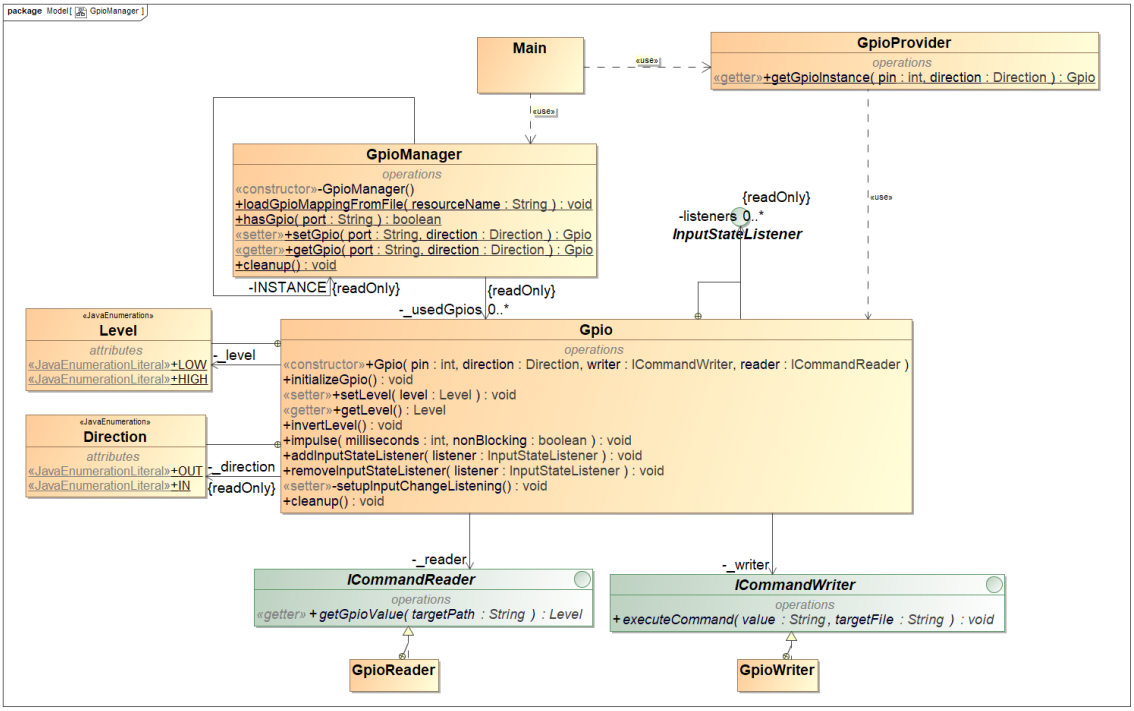

Figure 5.1: Gpio manager class diagram

```
    PowerMockito.mockStatic(GpioProvider.class);
    // For these specific parameters, return the mGpio parameter
    PowerMockito.when(GpioProvider.getGpioInstance(86, Gpio.Direction.IN)).thenReturn(mGpio);
  }
}
```

Naturally a unit test should cover only one class, consequently if we consider the previously described structure (shown in Figure 5.1), 6 unit test classes should be created (note that Level and Direction objects are only enumerables). Although there is no need to test basic file operation functions, provided by the Java framework in the GpioWriter and GpioReader classes and in addition there is no advantage to test the Main and GpioProvider classes, because there is no such a complex logic which should be verified by a test.

FS-2: Occupancy detection The occupancy information handling components are the Section Occupancy Query (in C++ language) and the Occupancy Query elements (written in Xtend). The information flow between these components is based on S88 serial port connection. In this case this test implementation requires an S88 connection to the Arduino hardware element, because there is no virtual serial port simulators for Windows available on the market. Otherwise the C++ component can be separately verified that, it is collecting all the informations from the sections with Google Test. The Occupancy Query component runs on Java Virtual Machine, so JUnit 5 can be used without a problem.

The class structure of the Occupancy Query component is shown in Figure 5.2, which describes the separation of a OccupancyQueryBridge, OccupancyQuery and Main class with their interfaces. The advantage of this architecture is the Occupancy calculation logic is well-separated from the message handling (in the OccupancyQueryBridge) logic. The obstacle to implement hardware independent test cases is the serial port dependency in the OccupancyQuery, for which the component must be re-factored. In addition the Occu-

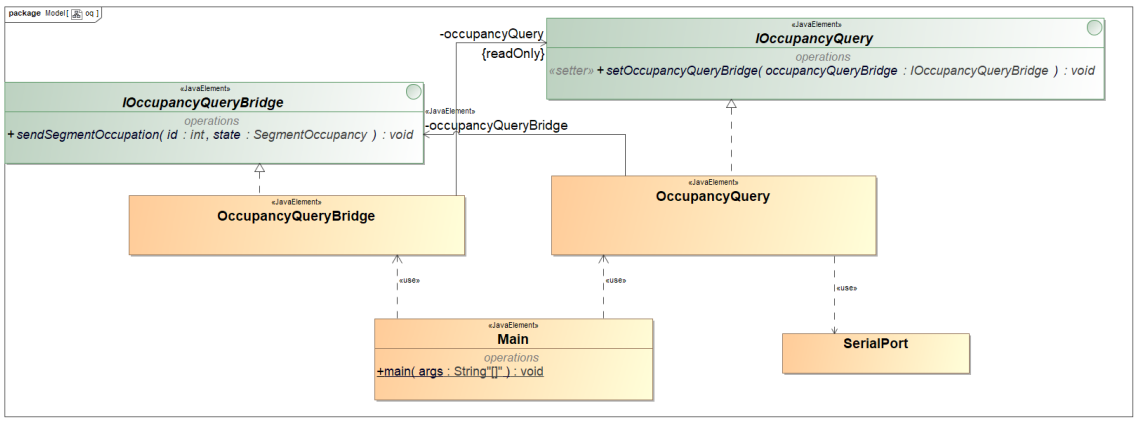

Figure 5.2: Occupancy Query class diagram

pancyQuery have dependencies to the SegmentOccupancy message type and for message handling elements, which should be also faked during the test executions.

FS-3: Track element controller The next component have additional dependencies to the Turnout and SegmentState messages, to be able to perceive any state change on the track. In order to supervise the sections, the Track element controller must rely on the messaging service component aligned with the track configuration properties. The currently applicable version from the PowerMock is still in beta state, but until this point it was working properly. Unfortunately I have faced an issue, when I tried to mock a static method with a parameter of a static instance. The PowerMock already assigned the issue, but there is no bugfix implementation yet for the problem. Apart from that the architecture can also be reviewed to avoid static classes where it is not necessary.

FS-4: Safety Logic The component level implementation of the Safety Logic consists of high level model based validators, from which a Java code was generated. In addition there is no need to test the glue code for these generated models. The system level Safety Logic also relies on generated Java code from the EMF model, nevertheless the decision-making logic is written in Xtend. This implementation also follows the previously described application and communication bridge architecture, but have a purely separated track refreshing algorithm. There are several shutdown strategies, which can be tested with this algorithm together. In addition this component also have dependencies to the SendAllStatus, SegmentOccupancy, TurnoutState, SegmentState and ComputerVisionObjectPositions messages with the communication services also.

FS-5: DashBoard This feature set and component is responsible for controlling and visualizing the status of the track elements. Therefore it is handling mainly message communication and there is no safety-critical responsibility.

5.2.2 Integration tests

Through the messaging service every component can be controlled and observed, therefore in an integration test cases sending a specific message to the topic (to which the observed component is subscribed) can give an input to the observed component. For example to create an input for the Barrier component (which moves a physical barrier up or down on

the track, if the supervised sections are occupied), a SegmentOccupancyChanged message must be sent to the topic of the supervised sections (15th or 24th). The component must implement ISegmentOccupancyChangedListener interface to get notification about a segment occupancy change event and handle every action. The following code shows the interface for a segment occupancy change and and example for subscribing mechanism.

```
enum SegmentOccupancy {
  FREE,
  OCCUPIED
}

interface ISegmentOccupancyChangeListener {
  def void onSegmentOccupancyChange(int id, SegmentOccupancy oldValue, SegmentOccupancy newValue)
}

@Data
class SegmentOccupancyMessage extends InternalMessage {
  int segmentId
  SegmentOccupancy state
}

// subscribing to the 15th and 24th segment topics
val supervisedSections = #{15, 24}
val occupancyTopics = TopicFactory::createSegmentTopics(supervisedSections,
     #{SegmentOccupancyMessage}).toSet
```

A messaging dispatcher component, called TrackCommunicationServiceLocator provides an opportunity to send a specific message to the predefined topic which can be used by any component with the inheritance of AbstractCommunicationComponent abstract class. For example it is possible to send a SegmentOccupancy message to the subscribers through the trackElementCommander field, which is shown in the next code snippet.

```
abstract class AbstractCommunicationComponent implements Runnable {
  protected val TrackCommunicationServiceLocator locator
}

class TrackCommunicationServiceLocator {
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrackElementStateSender trackElementStateSender
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrackElementCommander trackElementCommander
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrainCommander trainCommander
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val IDccCommander dccCommander

  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrackElementCommandCallback
     trackElementCommandCallback
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrackElementStateRegistry trackElementStateRegistry
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ITrainSpeedStateRegistry trainSpeedStateRegistry
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val ISendAllStatusCommandCallback sendAllStatusCallback
  @Accessors(PUBLIC_GETTER, PRIVATE_SETTER) val IComputerVisionCallback computerVisionCallback
}

class TrackElementCommander implements ITrackElementCommander {
  var protected MessagingService mms

  /**
  * Send a command to a segment, denoted by its ID.
  */
  override sendSegmentCommand(int id, SegmentState state) {
    mms.sendMessage(new SegmentCommand(id, state))
  }
}

class SampleOccupancySender extends AbstractCommunicationComponent {
  override sendSegmentOccupation(int id, SegmentOccupancy state) {
    locator.trackElementStateSender.sendSegmentOccupation(id, state)
  }
}
```

5.2.3 System tests

In order to execute system tests the demonstrator table must be started properly with the safety logic also. To start when the table is switched on, the BBB, PI and Arduino hardware elements are configured to start automatically with the basic services. These are the SectionOccupancyQuery, OccupancyQuery, TrackElementController and Dashboard components. The Safety Logic implementations prerequisites are to have these services already running on the network. Therefore after every automatically starting service is up, the component level safety logic service can be started on every BBB.

5.3 Test Results

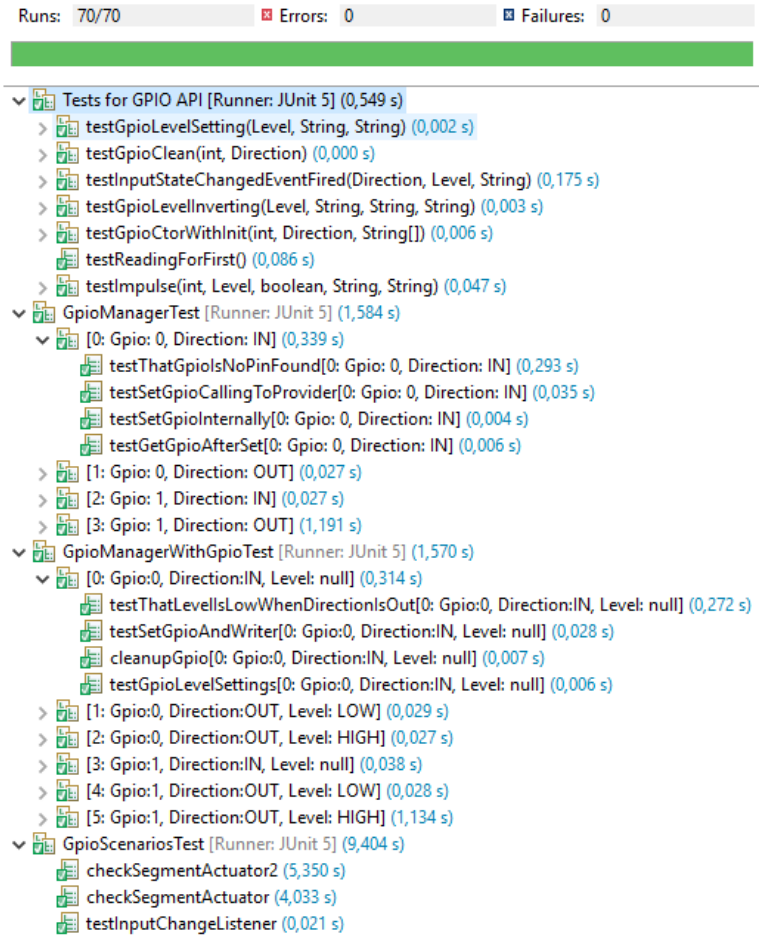

Figure 5.3: Implemented test cases for GPIO Manager component

Unit test results Thus I have implemented unit test classes for Gpio and GpioManager separately (shown as "Tests for GPIO API" and "GpioManagerTest" groups on Figure 5.3), then test cases for verifying the GpioManager with Gpio together. The GpioScenariosTest is stands for running an initialization test with all the components together in the final environment (in Debian operating system on the BBB). All the previous test classes are parameterized with more than one GPIO pin and all (in and out) directions.

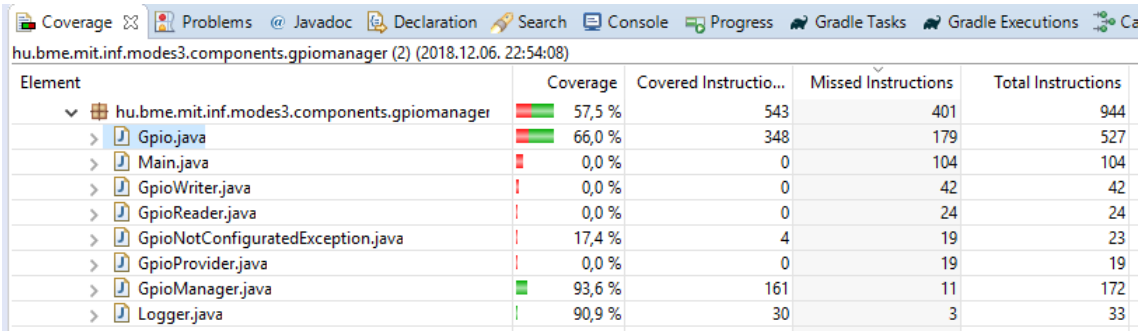

Figure 5.4: Code coverage measurement for GPIO Manager component

For each component the requirement was to achieve an 80% code coverage. It is unnecessary to test classes without any business logic, so the Main, GpioWriter, GpioReader, GpioNotConfiguredException, GpioProvider and Logger classes have been skipped. The GpioManager code coverage is 93.6% which is acceptable, but for the Gpio class it is 66.0% which is below the required percentage. The root cause for the low rate is that there are hardly reachable error handling branches in the algorithm, which is not covered by any test.

System test results The first attempt was failed, because of the wrong initialization of 2 segments. An issue is assigned with the title of: "Failed initialization during startup for section 8, 15".

Table 5.1: System test result for procedure FSS-1 (1)

Test case name	Actual results	Test result
1-1: change all turnout	It was traceable with the DashBoard functionalities	Passed
1-2: disable all segment	S8 and S5 segments were remained disabled in the initialization phase already	Failed
1-3: enable all segment	S8 and S5 segments were remained disabled in the initialization phase already	Failed

The second attempt was successful without any issue, therefore changing all turnout states from the dashboard was successful as well as segment availability change. In case of a disabled segment, a train have stopped on that exact segment, verifying that the change was effective and there was no power supply on the segment.

Table 5.2: System test result for test procedure FSS-1 (2)

Test case name	Actual results	Test result
1-1: change all turnout	It was traceable with the DashBoard functionalities	Passed
1-2: disable all segment	All segment status were changed to disabled	Passed
1-3: enable all segment	All segment status were changed to enabled	Passed

Table 5.3: System test result for test procedure FSS-2

Test case name	Actual results	Test result
2-1: turnout derail	Segment was disabled	Passed
2-2: train collision	Segment was disabled	Passed

5.4 Incident report

One problem was found during the system test, which is assigned by the following Table 5.4

Table 5.4: System test result for procedure FSS-1

Issue title	Severity	How to reproduce	Description
Failed initialization during startup for section 8, 15	Low	The first startup of the demonstrator table ends in a state of 2 failed section initialization.	A possible workaround is to restart the whole table in less then a few minute. Because in this case it is initialized correctly.

Chapter 6

Summary

Rewrite

In my first semester of thesis work, I have examined the Demonstrator railway system with all of its components. I have created architectural overview with basic functional and component details in order to give an overall picture. I have searched and inquired into test strategies for this multi layered application. I have created test scenarios in different test levels and decided which component have serious affect in safety critical point of view.

Further ideas are implementing and evaluating the test scenarios with different techniques if needed. Finally I will collect the experiences and summarize the result of test approaches.

List of Figures

List of Tables

Bibliography

[1] The gamma statechart composition framework. 2018. Design, Verification and Code Generation for Component-Based Reactive Systems.

[2] Robert Göransson David Bergström. Model- and hardware-in-the-loop testing in a model-based design workflow. http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8776530&fileOId=8776533 (2016).

[3] Arthur Zagretdinov Johan Haleby, Jan Kronquiest. Powermock with mockito. https://github.com/powermock/powermock/wiki/Mockito.

[4] JUnit team. The new major version of the programmer-friendly testing framework for java. https://junit.org/junit5/docs/current/user-guide/.

[5] IncQueryLabs Kft. Viatra query patterns. https://www.eclipse.org/viatra/.

[6] Inc (2015) No Magic. Sysml plugin 18.1. https://www.nomagic.com/files/manuals/SysML%20Plugin%20UserGuide.pdf.

[7] Andrew Powell-Morse. V-model: What is it and how do you use it? https://airbrake.io/blog/sdlc/v-model.

[8] Mockito team. Tasty mocking framework for unit tests in java. https://site.mockito.org/.

Appendix

A.1 Pictures of Demonstrator railway system

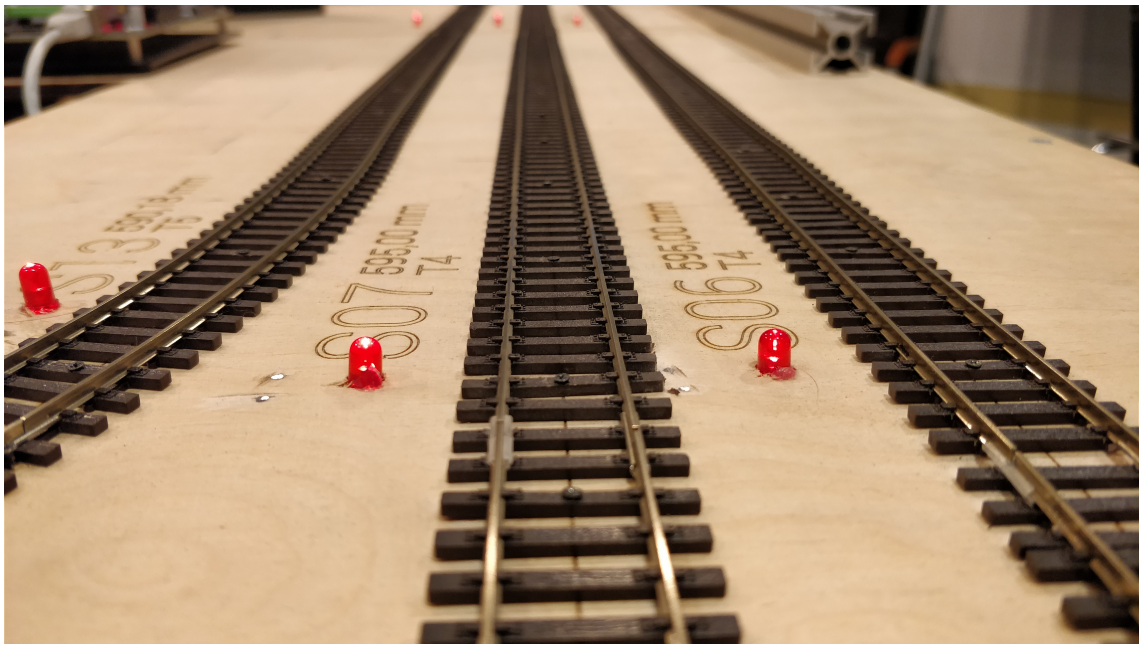

Figure A.1.1: Railway Section

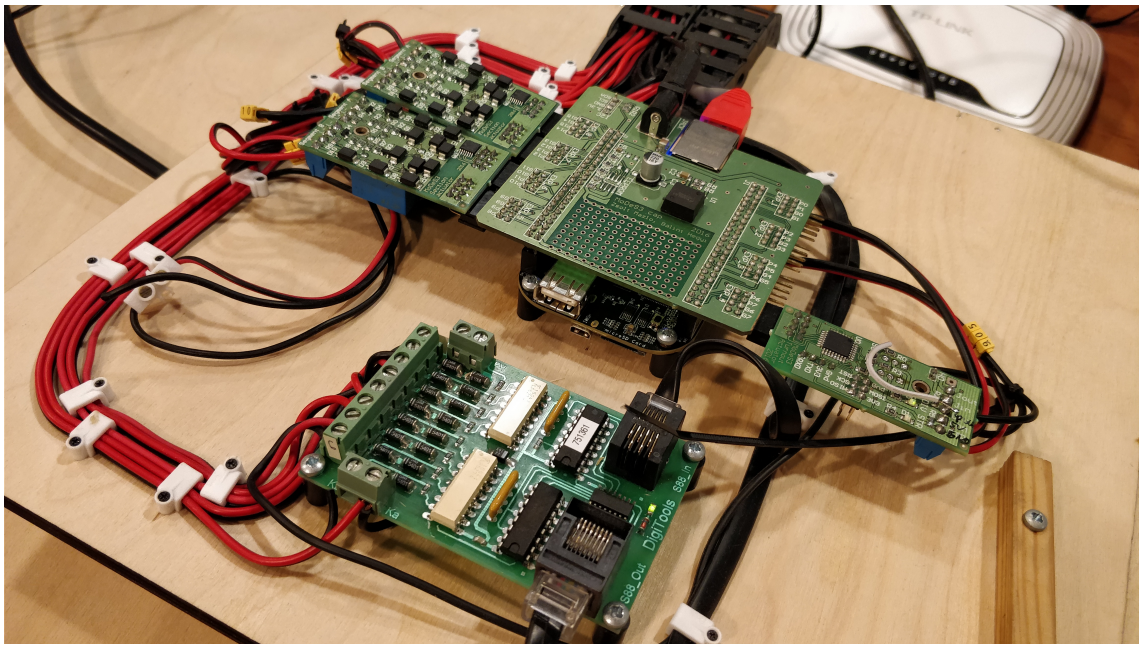

Figure A.1.2: Cape, segment and turnout actuator attached to BBB and connection to segment sensor

A.2 Test cases for MoDeS^3 Unit Test Plan

A.2.1 GPIO manager

Table A.1: Test case 1-1

Test case ID: 1-1	Purpose: to test the GPIO initialization in input direction. Priority: am Tracing: FS-1/1.0
Precondition	The GPIO's necessary files are available.
Input	Initialize the GPIO itself with input direction.
Expected result	The "both" string have been written to "edge" configuration file.

Table A.2: Test case 1-2

Test case ID: 1-2	Purpose: to test the GPIO pin input change listener while direction is input and the value is "0". Priority: am Tracing: FS-1/1.1
Precondition	The GPIO's necessary files are available.
Input	The value file have been written to "0", considered as LOW.
Expected result	GPIO noticed the change and read the "value" configuration file content as LOW level.

Table A.3: Test case 1-3

Test case ID: 1-3	Purpose: to test the GPIO pin change listener while direction is input and the value is "1". Priority: am Tracing: FS-1/1.2
Precondition	The GPIO's necessary files are available.
Input	The value file have been written to "1" considered as HIGH.
Expected result	GPIO noticed the change and read the "value" configuration file content as HIGH level.

Table A.4: Test case 1-4

Test case ID: 1-4	Purpose: to test the GPIO initialization in output direction. Priority: am Tracing: FS-1/2.0
Precondition	The GPIO's necessary files are available.
Input	Initialize the GPIO itself with output direction.
Expected result	The "0" string have been written to "value" configuration file.

Table A.5: Test case 1-5

Test case ID: 1-5	Purpose: to test the GPIO pin's level setting to LOW. Priority: am Tracing: FS-1/2.1
Precondition	The GPIO's necessary files are available.
Input	Set the GPIO's level to LOW.
Expected result	The "value" file has been modified with value "0"

Table A.6: Test case 1-6

Test case ID: 1-6	Purpose: to test the GPIO pin's level setting to HIGH. Priority: am Tracing: FS-1/2.2
Precondition	The GPIO's necessary files are available.
Input	Set the GPIO's level to HIGH.
Expected result	The "value" file has been modified with value "1"

A.2.2 Occupancy detection

Table A.7: Test case 2-1

Test case ID: 2-1	Purpose: to test the detection of segment occupancy (the train power consumption) through section occupancy query, when the specific segment is free Priority: above middle Tracing: (FS-2/1.0)
Precondition	S88 serial port connection and available Arduino hardware element
Input	Unclosed circuit between the specific segment's hardware elements elements
Expected result	Occupancy components have queried free occupancy state

Table A.8: Test case 2-2

Test case ID: 2-1	Purpose: to test detection of occupancy components, when the specific segment is occupied Priority: above middle Tracing: (FS-2/1.1)
Precondition	S88 serial port connection and available Arduino hardware element
Input	Closed circuit between the specific segment's hardware elements
Expected result	Occupancy components have queried occupied occupancy state

A.2.3 Track Element Controller

Table A.9: Test case 3-1

Test case ID: 3-1	Purpose: to test the track element controller's segment state setting as enabled Priority: above middle Tracing: (FS-3/1.0)
Precondition	Observable GPIO components
Input	Call the track element controller set segment state function with enabled parameter
Expected result	All GPIO levels are in "HIGH" state, which are related to the specific segment

Table A.10: Test case 3-2

Test case ID: 3-2	Purpose: to test the track element controller's segment state setting as disabled Priority: above middle Tracing: (FS-3/1.1)
Precondition	Observable GPIO components
Input	Call the track element controller set segment state function with disabled parameter
Expected result	All GPIO levels are in "LOW" state, which are related to the specific segment

Table A.11: Test case 3-3

Test case ID: 3-3	Purpose: to test the track element controller's turnout changing to straight state Priority: above middle Tracing: (FS-3/2.0)
Precondition	Observable GPIO components
Input	Call the track element controller set turnout state function with straight parameter
Expected result	The GPIO, which is controlling the straight branch, sent an impulse sign (inverting the current level twice with a specific time shift)

Table A.12: Test case 3-4

Test case ID: 3-4	Purpose: to test the track element controller's turnout changing to divergent state Priority: above middle Tracing: (FS-3/2.1)
Precondition	Observable GPIO components
Input	Call the track element controller set turnout state function with divergent parameter
Expected result	The GPIO, which is controlling the divergent branch, sent an impulse sign (inverting the current level twice with a specific time shift)

A.2.4 Safety Logic

Table A.13: Test case 4-1

Test case ID: 4-1	Purpose: to test the safety logic awareness, when a train is moving on a path where the next section in the direction already occupied by an other train Priority: high Tracing: (FS-4/1.0)
Precondition	None
Input	Insert a train to a specific segment and move an other train to the adjacent segment
Expected result	Safety Logic sent a segment disable command with the id of the specific segment

Table A.14: Test case 4-2

Test case ID: 4-2	Purpose: to test the safety logic awareness, when a train is moving and the 2nd section the path is already occupied by an other train Priority: high Tracing: (FS-4/1.0-1)
Precondition	None
Input	Insert a train to a specific segment and move an other train there from a 2 distance away segment
Expected result	Safety Logic sent a segment disable command with the id of the specific segment

Table A.15: Test case 4-3

Test case ID: 4-2	Purpose: to test the safety logic awareness, when a train is moving and the 3rd section in the path is already occupied by an other train Priority: high Tracing: (FS-4/1.0-2)
Precondition	None
Input	Insert a train to a specific segment and move an other train there from a 3 distance away segment
Expected result	Safety Logic sent a segment disable command with the id of the specific segment

Table A.16: Test case 4-4

Test case ID: 4-4	Purpose: to test the safety logic awareness, when a train is going through a turnout from top to straight, but the turnout is in divergent state Priority: high Tracing: (FS-4/2.0-1)
Precondition	Set the specific turnout into divergent state
Input	Set a train to go through the specific turnout from top branch to straight branch
Expected result	Safety Logic sent a turnout disable command with the id of the specific turnout

Table A.17: Test case 4-5

Test case ID: 4-5	Purpose: to test the safety logic awareness, when a train is going through a turnout from top to divergent, but the turnout is in straight state Priority: high Tracing: (FS-4/2.0-2)
Precondition	Set the specific turnout into straight state
Input	Set a train to go through the specific turnout from top branch to divergent branch
Expected result	Safety Logic sent a turnout disable command with the id of the specific turnout

A.2.5 DashBoard

Table A.18: Test case 5-1

Integration test case ID: 5-1	Purpose: to test the dashboard's set all turnout to straight functionality Priority: am Tracing: FS-5/1.0
Precondition	All turnout must be in divergent state
Input	Simulate a button press to the change all turnout direction function
Expected result	Message have been prepared to send with straight and a turnout id parameter for all turnouts

Table A.19: Test case 5-2

Integration test case ID: 5-2	Purpose: to test the dashboard's set all turnout to divergent functionality Priority: am Tracing: FS-5/1.1
Precondition	All turnout must be in straight state
Input	Simulate a button press to the change all turnout direction function
Expected result	Message have been prepared to send with divergent and a turnout id parameter for all turnouts

Table A.20: Test case 5-3

Integration test case ID: 5-3	Purpose: to test the dashboard's set all segment to enabled functionality Priority: am Tracing: FS-5/1.2
Precondition	None
Input	Simulate a button press to the set all segments to enabled state function
Expected result	Segment command message have been prepared to send with enable parameter for all segments

Table A.21: Test case 5-4

Integration test case ID: 5-4	Purpose: to test the dashboard's set all segment to disabled functionality Priority: am Tracing: FS-5/1.2
Precondition	None
Input	Simulate a button press to the set all segments to disabled state function
Expected result	Segment command message have been prepared to send with disable parameter for all segments

Table A.22: Test case 5-5

Integration test case ID: 5-5	Purpose: to test the dashboard's set turnout to straight functionality Priority: am Tracing: FS-5/2.0
Precondition	A specific turnout must be in divergent state
Input	Simulate a button press to the specific turnout
Expected result	Turnout command message have been prepared to send with straight and with turnout id parameter

Table A.23: Test case 5-6

Integration test case ID: 5-6	Purpose: to test the dashboard's set turnout to divergent functionality Priority: am Tracing: FS-5/2.1
Precondition	A specific turnout must be in straight state
Input	Simulate a button press to the specific turnout
Expected result	Turnout command message have been prepared to send with divergent and with turnout id parameter

Table A.24: Test case 5-7

Integration test case ID: 5-7	Purpose: to test the dashboard's functionality of enable a specific segment Priority: am Tracing: FS-5/2.2
Precondition	None
Input	Simulate a button press to the specific segment
Expected result	Segment command message have been prepared to send with enable and segment id parameter

Table A.25: Test case 5-8

Integration test case ID: 5-8	Purpose: to test the dashboard's functionality of disable a specific segment Priority: am Tracing: FS-5/2.3
Precondition	None
Input	Simulate a button press to the specific segment
Expected result	Segment command message have been prepared to send with disable and segment id parameter

A.3 Test cases for MoDeS^3 Integration Test Plan

A.3.1 Occupancy message (FSI-1) text cases

Table A.26: Integration test case 1-1

Integration test case ID: 1-1	Purpose: to test the detection of segment occupancy when a segment is free and to verify the propagated network occupancy message Priority: am Tracing: FS-2/1.0
Precondition	There must be an MQTT server connection available and an connected with serial port
Input	Unclosed circuit between the specific segment's hardware elements
Expected result	A new segment occupancy message must be send to the network with free segment state and the specific segment id

Table A.27: Integration test case 1-2

Integration test case ID: 1-2	Purpose: to test the detection of segment occupancy when a segment is occupied and verify the network occupancy message Priority: am Tracing: FS-2/1.1
Precondition	There must be an MQTT server connection available and an Arduino with S88 serial port connected
Input	Closed circuit between the specific segment's hardware elements
Expected result	A new segment occupancy message must be send to the network with occupied segment state and the specific segment id

A.3.2 Track element controller instructions (FSI-2) test cases

Table A.28: Integration test case 2-1

Integration test case ID: 2-1	Purpose: to test the track element controller, that it enables its supervised segment's state Priority: am Tracing: FS-3/1.0
Precondition	There must be an MQTT server connection available
Input	Send a SegmentCommand message with enabled state and a segment id which is supervised by the track element controller component
Expected result	All related GPIO (pru and app) has the writer with value "1" and targetFile "value"

Table A.29: Integration test case 2-2

Integration test case ID: 2-2	Purpose: to test the track element controller, that it disables its supervised segment's state Priority: am Tracing: FS-3/1.1
Precondition	There must be an MQTT server connection available
Input	Send a SegmentCommand message with disable state and a segment id which is supervised by the track element controller component
Expected result	All related GPIO (pru and app) has the writer with value "0" and targetFile "value"

Table A.30: Integration test case 2-3

Integration test case ID: 2-3	Purpose: to test the track element controller, that it sets its turnout to straight state Priority: am Tracing: FS-3/2.0
Precondition	There must be an MQTT server connection available
Input	Send a TurnoutCommand message with straight state and the turnout id which is controller by the track element controller component
Expected result	Verify that the straight GPIO handle of the specific turnout have written with values: "1", "0", "1" in this specific order and the "value" targetfile parameter

Table A.31: Integration test case 2-4

Integration test case ID: 2-4	Purpose: to test the track element controller, that it sets its turnout to divergent state Priority: am Tracing: FS-3/2.1
Precondition	There must be an MQTT server connection available
Input	Send a TurnoutCommand message with divergent state and the turnout id which is controller by the track element controller component
Expected result	Verify that the divergent GPIO handle of the specific turnout have written with values: "1", "0", "1" in this specific order and to the "value" targetfile

A.4 Test cases for MoDeS^3 System Test Plan

A.4.1 Track element availability verification (FSS-1)

Table A.32: System test case 1-1

System test case ID: 1-1	Purpose: to test all turnout controllability Priority: am Tracing: FS-6/1.0
Precondition	None
Input	Send a switch turnout command to all turnouts twice
Expected result	All turnout state have been changed to straight from divergent and the other way

Table A.33: System test case 1-2

System test case ID: 1-2	Purpose: to test all segment controllability Priority: am Tracing: FS-6/1.2
Precondition	None
Input	Send a segment disable command to all segments
Expected result	All segment have been disabled

Table A.34: System test case 1-3

System test case ID: 1-3	Purpose: to test all segment controllability Priority: am Tracing: FS-6/1.2
Precondition	None
Input	Send a segment enable command to all segments
Expected result	All segment have been enabled

A.4.2 Safety Logic verification

Table A.35: System test case 2-1

System test case ID: 2-1	Purpose: to test the safety logic for turnout derail scenario Priority: am Tracing: FS-6/1.2
Precondition	Turnout T5, T1 is in straight state and a train is on the segment S13
Input	Move the train to segment S15 from segment S13 through the path of S13, S8, T5, S11, T1, S15.
Expected result	Before T1 turnout the whole railway system is disabled by the safety logic to avoid turnout derail

Table A.36: System test case 2-2

System test case ID: 2-2	Purpose: to test the safety logic for train collision scenario Priority: am Tracing: FS-6/1.2
Precondition	Turnout T5 is in straight state, turnout T1 is in divergent state and 2 trains are on the segments of S13 and S15
Input	Move the first train to segment S15 from segment S13 through the path of S13, S8, T5, S11, T1, S15.
Expected result	Before segment S15 the whole railway system is disabled by the safety logic to avoid train collision