Simulation of Relay Interlocking Systems

Louise Elmose Eriksen
Boe Pedersen

Kongens Lyngby 2007
IMM-BSC-2007-52

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

All railway networks implement special systems for securing the travel of trains. These systems are called interlocking systems and prevent trains from colliding and derailing. Different interlocking systems are currently implemented in the Danish railway networks. This project is concerned with so-called relay interlocking systems.

A detailed description of the domain for the project, i.e. relay interlocking systems is explained on order to be able to perform an analysis of the system. The main part of an interlocking system is a complex electrical circuit and this circuit is only documented on static paper diagrams. The electrical circuit can be in numerous states according to the states of the electrical components in the circuit. To analyze the diagrams that depict the circuit, propagation of current and the state of each component must continuously be verified. Performing analysis of relay interlocking systems is thus somewhat cumbersome, time-consuming and quite vulnerable in relation to human errors.

The purpose of this project has been to develop an application for simulation of relay interlocking systems. The application should reflect different states of a circuit and thus function as a sort of state machine. Error detection, analysis and validation of an interlocking system should be eased remarkably by the application.

In the analysis the requirement of simulating propagation of current is broken down, and potential sub-problems are identified. The application must e.g. be able to display the propagation of current in intermediate steps, and hence an indirect requirement is to develop a graphical user interface.

Designing the application leads among other things to a definition of how intermediate steps of propagation of current should be performed. The intermediate steps is a way to "pause" the current in its propagation through the circuit, to be able to see every change of the circuit and thereby allowing the relay interlocking system to be "debugged".

Thoroughly testing an application used for simulation is very important since the slightest error renders the simulation completely unreliable. Extensive testing has been performed on different layers of the application architecture giving a very good test coverage. Identified defects have been corrected and the application thus has no known defects.

A functional application for analysis of relay interlocking systems were successfully developed. Besides meeting the initial requirements additional functionality such as simulating a train and simulating defective components has been implemented and improved the usability of the application.

Preface

This thesis was prepared at the institute for Informatics and Mathematical Modelling, at the Technical University of Denmark as part of the requirements for acquiring the B.Sc. degree in software technology.

The project is made in collaboration with Banedanmark. The objective is to make a tool which makes it easier for Banedanmark to simulate the propagation of current in the electrical circuits that are the backbones on most Danish railway stations.

The main focus is on understanding the domain and developing the tool for Banedanmark.

The final version application among other relevant files can be found on the attached CD. The source code is extracted from the printed version of the thesis but is accessible on the CD. A list of the content of the CD is found in appendix L.

Lyngby, June 2007

Louise Elmose Eriksen
Boe Pedersen

Acknowledgements

First of all we would like to thank our supervisors Anne Haxthausen and Hubert Baumeister for their support, their great interest in and commitment to the project and for the beneficial discussions and meetings.

Additionally we thank Kirsten Mark Hansen – our contact at Banedanmark – for her interest in the project and willingness to help us understand the domain throughout the project period by explaining and answering all of our questions.

Towards the end of the project period we have had Jakob Dam Jensen read the thesis to give us feedback on the technical details and the formulation of the thesis. Thank you for the useful comments and your time.

Contents

Chapter 1

Introduction

At railway stations special systems are used to secure the travel of trains. These systems are highly safety-critical and prevent trains from colliding and derailing. The main part of the system is a complex electrical circuit which is documented on diagrams. The diagrams display one state of the circuit and are currently the only ”tool” available for analysis of the system. Numerous electrical components are displayed on the diagrams and each component can be in different states in relation to whether the component is conductive and/or live. To analyze such diagrams one must be in full control of the states of multiple components and their connections which is rather unmanageable. The objective of this project is to develop an application which allows an easier, more flexible and less time-consuming analysis through simulation.

The diagrams contain a lot of compact information. Understanding such diagrams requires some background knowledge about the system as a whole. The first chapter in this thesis will provide information about the system that secures the travel of trains, called an *interlocking system*, and furthermore explain how a diagram is to be read.

When the domain has been explained, the main requirement, i.e. developing an application that simulates interlocking systems, will be analyzed and split into sub-requirements. After the analysis the various possible solutions will be discussed resulting in a detailed description of the chosen design. Details of the implementation of the application will be explained next and after that the test strategy and the results from the tests will be examined.

Finally possible extensions to the application will be described and the results of the project will be summarized with a conclusion.

A terminology can be seen in appendix A. Since a lot of technical terms are presented in relation to the domain an English-Danish dictionary is available in appendix B.

Chapter 2

Domain description

2.1 Introduction to Interlocking Systems

When several trains are in use on a railway network it is necessary to secure the travel of each train, in order to prevent collisions and derailings.

Security in a railway network is ensured with an *interlocking system* which controls signals and points.

As technology has evolved so has the type of interlocking systems and hence different technologies have been used.

2.1.1 Different Technologies

At the current time, four major types of interlocking systems have been implemented in the Danish railway network, namely the *mechanical interlocking system*, the *electro mechanic interlocking systems*, the *relay interlocking system*, and the *electronic interlocking system*. The different types of systems will be described briefly in the following sections.

2.1.2 Mechanical interlocking systems

Towards the end of the 19^{th} century mechanical interlocking systems were used. The system created inter-dependency between signals and points using wire pulls. At a given time the operations that would lead to a collision were mechanically blocked.

2.1.3 Electro mechanic interlocking systems

Electro mechanic interlocking systems were implemented in the railway network from around 1915 until 1950. The interlocking system is a mix of the mechanical interlocking system and the relay interlocking system (see section 2.1.4). External components such as points and signals were now controlled by relays instead of wire pulls. The part of the interlocking system that prevented trains from colliding, were still mechanically controlled. The control was now made by a mechanical registry consisting of steel beams with holes in.

2.1.4 Relay interlocking systems

In the middle of the 1990ies, relay interlocking systems of the type DSB 1953 and 1954 were introduced. Wire pulls were replaced by pushbuttons on an operators panel and, among other things, electrical monitoring was possible using track sections.

2.1.5 Electronic interlocking systems

The newest technology is an interlocking system controlled by microprocessors. Instead of an operators panel the monitoring takes place on a computer screen.

2.2 Relay Interlocking Systems

In this thesis the focus will be on relay interlocking systems. From this point on a "relay interlocking system" will be referred to as an interlocking system. An interlocking system is a quite extensive system and thus a complete description is considered outside the scope of this project. For this reason it is only the most essential part of the interlocking system that is described in this section.

One type of interlocking system covers the open track between stations, while another covers the station itself. The systems at the stations are of the most interest since they are by far the most complicated. The goal of an interlocking system is, as previously mentioned, to secure the travel of a train. This is obtained by creating a complex circuit where each component in the circuit translates into a logic condition, e.g. "if there is a train at the station, let the entry signal be red" (this condition requires several components though). The most important component in the circuit and thus in the interlocking system, is an electrical component called a *relay*. Besides relays a station is composed by *track sections*, *points*, *buttons*, *fuses*, *resistors*, *signals* and *lamps*.

In the following section the mechanical functionality of a relay will be explained. Next, the previously mentioned components on a station will be introduced, whereafter the structure in a circuit will be defined. After that, abstract concepts such as *train route* and *train route table* will be clarified. Finally the process of translating abstract conditions into a circuit of electrical components will be explained and a typical procedure for a train entering a station will be examined.

Further domain specifications can be found in appendix C.

2.2.1 Relays

There are two mechanical types of relays in a circuit, namely *regular relays* and *steel core relays*.

First, the functionality of a regular relay will be explained and thereafter the distinctive functionality of the steel core relay will be examined.

2.2.1.1 Regular relays

A regular relay consists of a coil, an electromagnet, an armature, a pole with horizontal conductive bars and a number of contacts, typically 6, 10 or 20. The electromagnet is placed inside the coil and each end of the coiling is connected to a pin. When no current is applied to these pins, the electromagnet is demagnetized and the state of the relay will be as on figure 2.1. Each contact consists of two pins, to which wires can be connected. The lower contacts on figure 2.1

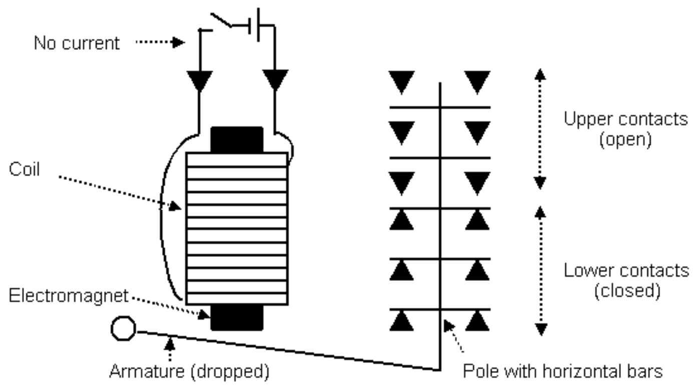


Figure 2.1: *The relay only allows current to pass through the lower contacts, given that the armature is dropped*

are said to be *closed*, since current can pass from one pin on the contact to the other pin on the contact, via the horizontal bar. The upper contacts of the relay are said to be *open*, since the horizontal bar through which the current can pass, is not in contact with the pins.

When current is applied to the coil pins the electromagnet will carry current and magnetize. The magnetized electromagnet draws the armature which in turn pushes the pole upwards. This will invert the state of the contacts so that the upper contacts are closed and the lower contacts are open, as seen on figure 2.2. When no more current is applied, the electromagnet will demagnetize, making the armature, and thus the pole, drop. The relay has now returned to the original state on figure 2.1.

The pins are the only externally accessible parts on the relay since the other components are protected from dust and wear by a black box as seen on figure 2.3. The pins on a relay can be numbered in one of two ways. The coil pins are the uppermost pins as seen on figure 2.4. In general it is the relay that is said to be *dropped* and *drawn*, even though it is in fact the armature and pole that are dropped and drawn respectively.

2.2.1.2 Steel core relays

Steel core relays mechanically differ from regular relays in that there instead of an electromagnetic core, is a core of heat-treated steel. The heat-treated steel core causes the core to remain magnetized, even when the supply of cur-

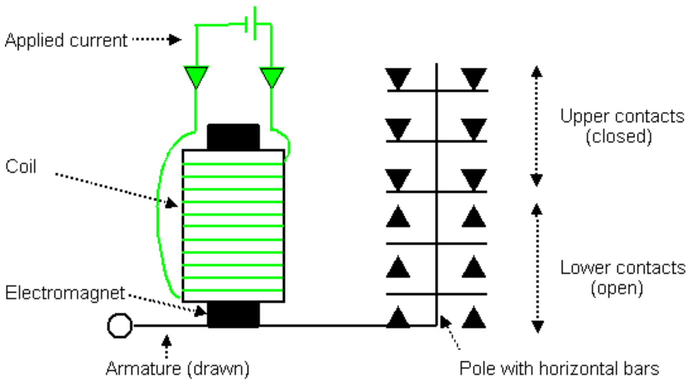


Figure 2.2: *The relay only allows current to pass through the upper contacts, given that the armature is drawn (green illustrates current).*

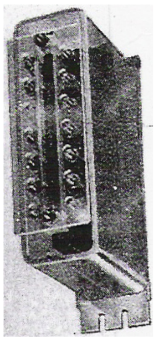


Figure 2.3: *The coil, armature, electromagnet and the pole with horizontal bars are protected from dust and wear by a black box.*

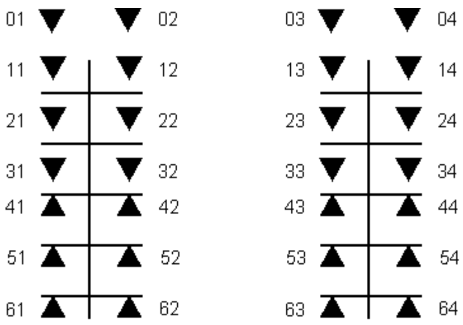


Figure 2.4: *The pins on the relay are numbered in one of two ways.*

rent is stopped. The coil in which the steel core is placed, has two coilings; a magnetizing coiling and a demagnetizing coiling. Initially the steel core relay is magnetized. When current is applied to the demagnetizing coiling, the steel core will demagnetize and remain demagnetized until current is applied to the magnetizing coiling. When the steel core is magnetized, it will remain magnetized until current is applied to the demagnetizing coiling. The pin with ID 12 or 14 (see the pin's positions on the relay in figure 2.4) is not externally connectable, since it is connected internally as regards the coilings.

The difference from the regular relay to the steel core relay is thus that the state of the regular relay depends on whether current is applied or not and the state of the steel core relay depends on, to which coiling current was last applied.

2.2.2 Track Sections

A track has the ability to carry current. A *track section* is a piece of a track that is isolated so that the current does not spread from one track section to another. This means that track sections can carry current independently of each other. Each track section is connected to a relay so that the relay is drawn when the track section is free, see figure 2.5. The wheels and the axles of the

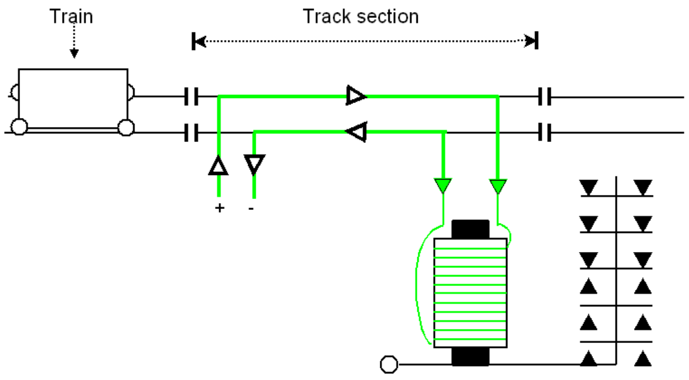


Figure 2.5: *The relay is drawn, giving the track section is free.*

train are conductive. This means, that when the wheels of the train come in contact with the track section, the circuit shorts out, see figure 2.6. The only external influences on the track section that will affact the state of the relay is a train or other conductive components. On the operators panel track sections are identified by a short text. E.g. in figure 2.8 the leftmost track section is denoted by "01" just beneath the track section.

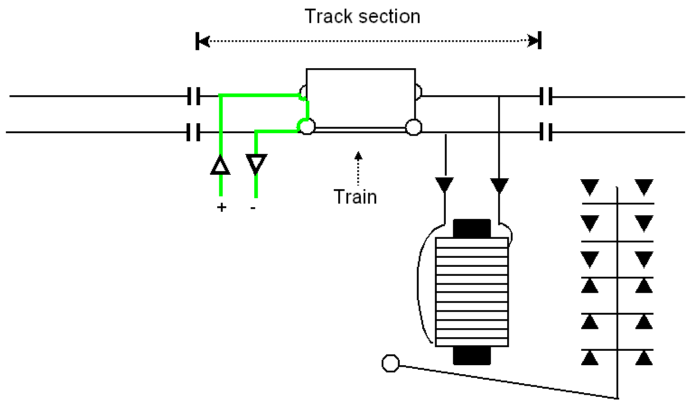


Figure 2.6: *The relay is dropped, giving the track section is occupied.*

2.2.3 Points

Some track sections contain a branching of the track. These track sections are called points.

Besides the functionality of a track section, the point has the functionality of being switched to one of two directions. A point can thus be in one of three states; switched to the *plus direction*, switched to the *minus direction*, or switched in an intermediate state. The intermediate state corresponds to the situation where the point has not been completely switched, i.e. the situation where the point is neither switched to the plus direction nor the minus direction. The plus direction is the direction to the right and the minus direction is the direction to the left, as seen on figure 2.7. Besides being connected to the relay

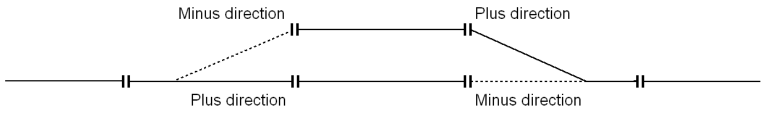


Figure 2.7: *The plus direction is to the right and the minus direction to the left.*

specified in the previous section, a point is connected to two relays: A plus and a minus relay. These relays indicate which position the track is switched to. When the point is switched to the plus direction the plus relay is drawn and minus relay is dropped, when the point is switched to the minus direction the minus relay is drawn and the plus relay is dropped. And finally when the point is in the intermediate state both relays are dropped. This means that only one of the plus and minus relays can be drawn at the same time.

In addition to the ID specified in the previous section the point has another ID used to distinguish between the different points.

2.2.4 Buttons

Communication between an operator and a circuit takes place with buttons. When an operator wants to initiate certain actions he/she uses the buttons on an operators panel.

There are typically five types of buttons which are all placed on an operators panel and electrically connected in the circuit. The button types are:

entry buttons indicating at which track section a train is going to enter the station,

exit buttons indicating at which track section a train is going to exit the station,

plus buttons indicating that the specific point should be switched to the plus direction,

minus buttons indicating that the specific point should be switched to the minus direction,

track buttons indicating the destination track for the train.

When a button is pushed by an operator, it is conductive and when it is released it again becomes non-conducting.

The buttons are typically placed as seen on figure 2.8. If e.g. a train is approaching the station from direction A and should stop at track section 04, the minus button on point **pa** would initially be pushed to switch the point, whereafter button I on track section 01 and button T on track section 04 would be pushed simultaneously.

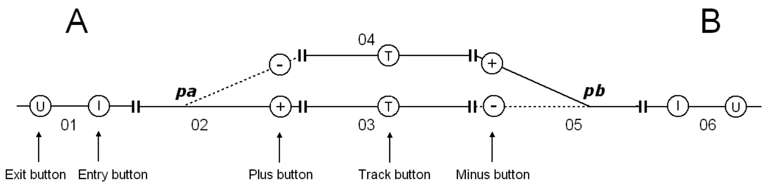


Figure 2.8: *Schematic layout of an operators panel. Switching the relevant point as necessary and afterwards pushing an entry/exit button simultaneously with a track button tells the interlocking system where the operator wants the train to go.*

2.2.5 Fuses

Fuses protect a circuit from over-current. A metal wire in the fuse melts at a certain current, protecting the rest of the circuit from over-current. When the metal wire is intact the fuse is conductive, and when the metal wire is melted, the fuse is non-conducting.

2.2.6 Resistors

Resistors are used to control the path of current, that is to make sure only a known safe amount of current flows through the circuit. A resistor is initially conductive, but if it is damaged e.g. due to over-current, the resistor is non-conducting.

2.2.7 Signals and Lamps

A signal is similar to a traffic signal, in that it is a set of lamps. On a station there are different types of signals for instance entry, exit and platform signals.

A lamp consists of one or two filaments of which the second filament is a spare filament. Usually each filament is connected to one relay, so that when the filament is live the relay is drawn, and when the filament is not live, the relay is dropped. The filament is conductive when it is intact and non-conducting if it is burned out.

Signals are the final indication of an interlocking system, in the sense that the interlocking system interprets the current state of the station and uses the signals to indicate to the driver which actions he/she should take. If all conditions for letting a specific train enter/exit the station at its current state are fulfilled the relevant signals will show ”go”, indicating to the driver that it is safe to enter/exit the station.

2.2.8 Circuit structure

The structure in a circuit is created by connecting *terminals* with wires. ”Terminal” is a common term for the ”power in” and ”power out” components and the pins on the electrical components. The ”power in” and ”power out” components on the power source will from now on be referred to as the *positive pole* and the *negative pole.*

The following statements apply for a functional circuit:

- Connections are made between terminals.
- A pin can be connected to two terminals at the most.
- There is no upper limit on how many pins the positive and negative poles can be connected to.
- There exists at least one *path* from the positive pole to the negative pole in a circuit. A path consists of terminals and wires.
- Each component has a certain amount of internal resistance. If there are several branching paths with different internal resistance, most of the current will only go through the branching with the least internal resistance. If e.g. a circuit has two branches, one with coil pins and one with a contact the contact will have the least amount of internal resistance. Most of the current will pass through the contact and too little current will pass through the coil pins to make the relay draw.

2.2.9 Train Routes

A train route is an abstract concept describing a set of assembled track sections. There is one set of train routes for entering a station and one set of train routes for leaving a station.

A simple station on a single tracked stretch will typically have 4 train routes per track on the station, since there is an enter and exit train route per driving direction through the station. On the schematic layout of the operators panel in figure 2.8 the enter train routes would be:

1. Entering at A using plus direction: $\{01, 02, 03\}$
2. Entering at A using minus direction: $\{01, 02, 04\}$
3. Entering at B using plus direction: $\{06, 05, 04\}$
4. Entering at B using minus direction: $\{06, 05, 03\}$

and the exit train routes would be:

1. Exiting towards A from 04: $\{04, 02, 01\}$
2. Exiting towards A from 03: $\{03, 02, 01\}$
3. Exiting towards B from 04: $\{04, 05, 06\}$
4. Exiting towards B from 03: $\{03, 05, 06\}$

On larger stations or stations on double tracked stretches there are several train routes.

When an operator wants to allow a train to enter the station, he/she will initiate a *locking of a train route*. When a specific train route, e.g. for entering the station is locked, the track sections in the train route are reserved for the train and points are switched correctly. This means the train can safely enter the station. The locking of the train route prevents train routes that might lead to collisions (so-called *conflicting train routes*) from being locked.

Once the train route is locked and the train is at a certain place on the train route (specified in the *train route table*) there will be a *release of the train route*. The release of the train route cancels the reservation of the train route allowing train routes that were previously conflicting to be locked.

2.2.10 Train Route Tables

A train route table is a description of the required functionality of an interlocking system. A train route table is a description of the state of track sections, points and signals. The description contains the following information for each train route:

1. *Track sections*, which track sections should be free, so that a train can safely use them?
2. *Points*, which positions should they be in, so that the correct track sections will be used?
3. *Signals*, what should they show at different places on the station when the train route is locked?

Furthermore the table contains information about when the train route should begin to release and when the train route should be completely released.

2.2.11 From Train Route Table to Circuit

As previously mentioned an interlocking system secures the travel of trains through a station by defining train route tables. But how does one go from an abstract train route table to electrical components in a circuit? This procedure will be described through a simple example.

A station S has a single track section ts, and a signal A with a green and a red lamp. The train route table TRT is described as:

TRT: To enter the station S through the route ts, signal A must be green.

As mentioned in section 2.2.2, the relay for the track section will be drawn when the track section is free. Thus, we can use one of the upper contacts from the relay to make sure that the green lamp is only turned on, when the track section is free, as seen on figure 2.9. When the wheels of the train enter the track section, the current shorts out as previously mentioned. The lack of current causes the relay to drop and the upper contact will prevent the current from going to the green lamp as seen on figure 2.10; instead the red lamp is turned on. The train route table TRT has thus been turned into a small circuit. Banedanmark depicts the situations in the figures much more compactly though, which will be described in section 2.3.

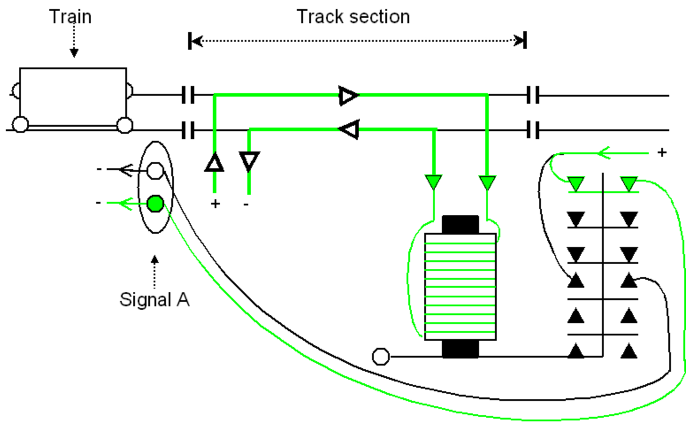


Figure 2.9: *The signal is green, given that the track section is free.*

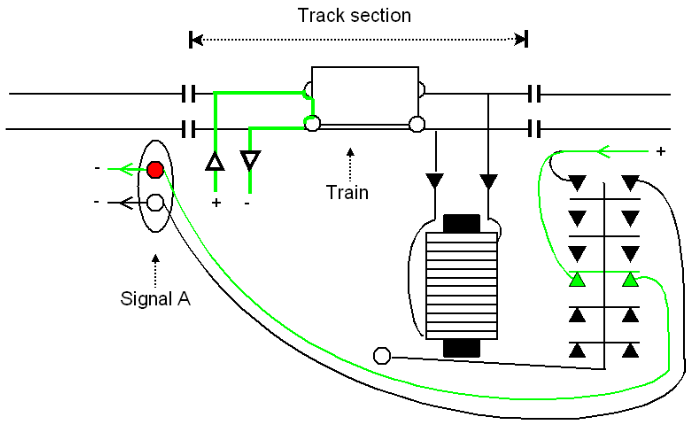


Figure 2.10: *The signal is red, given that the track section is occupied.*

2.2.12 Typical Scenario

In this section a typical scenario for a train entering a station will be described.

The procedure for letting a train enter a station is initiated when the operator acknowledges the train approaching the station from the open track. The procedure described below is for a successful scenario, where all conditions are met.

1. The operator chooses, according to the train schedule, which track on the station the train must be directed to, by switching one or more points using relevant plus and/or minus buttons.
2. The operator simultaneously pushes the entry button pursuant to the direction in which the train approaches the station and the button for the track which the train should be directed to.
3. The relevant train route is locked by the interlocking system.
4. The interlocking system switches the entry signal to green.
5. When the train is at a certain place on the train route (specified in the train route table), the interlocking system will begin to release the train route.
6. When the train route is released, train routes that were conflicting to this train route can be locked.

As mentioned this scenario is when everything goes well, i.e. every point is switched correctly and so forth. Of course, an interlocking system must take all possible scenarios into consideration and only make the signals turn green when all conditions for secure train travel are fulfilled.

2.3 Diagrams

As previously mentioned every station, using the interlocking system, has got several diagrams documenting the electrical circuit.

All of the components described in section 2.2 are displayed in a unique way on the diagrams, just like the diagrams precisely state which components are connected and which pins on these components the wires are actually connected to. This is fundamental to know when reading the diagrams and trying to understand how the electrical circuit is assembled.

In the following section the form of notation used on Banedanmark's diagrams is described whereafter an example using the notation is given.

2.3.1 Notations

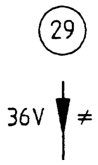
All diagrams display the *normal state* of the circuit. The normal state is defined as a state where:

- current is applied to the system,
- all points are switched to the plus direction,
- no trains are at the station and
- no train route is locked.

Among other things the diagrams display contacts and coil pins. To make it easier to understand why these components are placed and connected as they are, they are depicted in a way that reflects the use of the associated relay. Although depicted differently the relays are all regular relays i.e. mechanically alike with one exception; the "Train route locking relay" is a steel core relay. In appendix D these different symbols are described.

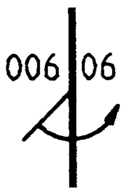
In the following sections it is described how every component is displayed on the diagrams and how they are to be read.

2.3.1.1 Power supplies



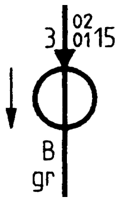
Every electrical circuit needs a power source. The $\neq$ sign shows that the input is a DC power source (36V in this figure). This is only shown when the power input on the sub circuit is connected directly to the main power source. At each power input is shown in which diagram the power source is displayed. This power source can be found on diagram no 29.

2.3.1.2 Buttons



When an operator at a station wants to interact with the electrical circuit he/she pushes one or more buttons on the operators panel. These buttons are illustrated on the diagrams as seen on the figure. The two numbers are the coordinates of the button on the operators panel. The leftmost number is the x-coordinate and the rightmost is the y-coordinate.

2.3.1.3 Regular relay coil pins



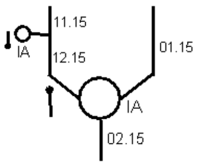
This figure illustrates the regular coil pins.

The arrow on the left hand side of the figure tells which state the relay is in; pointing down when the relay is dropped and pointing up when the relay is drawn.

The numbers on the figure show where the relay is placed in the rack of relays. This relay is placed in field 15 and level 3. On some figures only the rightmost number is written. This is because the relay is placed in a rack that uses consecutive numbers as identifiers (instead of field and level). The two smaller numbers are the IDs of the pins. These pins are identified on the relay as pin 01 and pin 02.

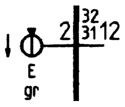
The text is a short description of the relay, added to make it easier to understand the use of the specific relay. The relay displayed on this figure is reflecting the state of the green lamp at signal B.

2.3.1.4 Steel core relay coil pins



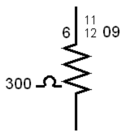
This figure illustrates the four coil pins on a steel core relay. The four lines of numbers depict the four pins. The number after the dots is the consecutive number which identifies the relay in the rack of relays. The numbers in front of the dots are the IDs of the pins. In the upper left corner the pins 11 and 12 are displayed as a regular contact on which pin 12 is connected internally to pin 02 and therefore cannot be connected to a wire (described in section 2.2.1.2). The only pins that are connectable are pin 11 in the upper left corner, pin 01 in the upper right corner and pin 02 in the lower part of the figure. Just like the previously mentioned symbol (regular relay coil pins), this symbol displays a short description and the state of the relay. The description of this relay is "IA". The "arrows" on the left hand side of the figure never points in different directions. When they point upwards the relay is magnetized and when they point downwards the relay is demagnetized.

2.3.1.5 Contacts



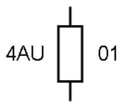
When wires are connected to a contact on a relay, the contact is shown like on this figure. The larger numbers show that the relay is located at field 12 and level 2 in the rack of relays. The two smaller numbers are the numbers of the pins. These pins are identified on the relay as pin 31 and pin 32. The arrow on the left hand side of the figure indicates the state of the relay. The horizontal line is used to indicate whether the contact is open or closed. When the line is on both sides of the vertical line, the contact is open (as on this figure) and when the line is only on the left hand side of the vertical line, the contact is closed.

2.3.1.6 Resistors



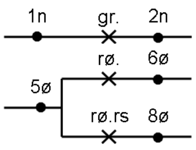
This is the symbol of a resistor. Two numbers are written to the left of the resistor. The lowermost tells how large the internal resistance is. The uppermost specifies (along with the larger number on the right hand side of the resistor) where in the rack of relays the resistor is placed. This resistor is placed at field 9, level 6. The two smaller numbers are the IDs of the pins. Pin 11 is above and pin 12 is below the resistor.

2.3.1.7 Fuses



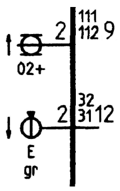
The number written to the right of the fuse (as displayed on the figure) is a consecutive number used to identify the specific fuse. The size and type of the fuse is written to the left of the fuse.

2.3.1.8 Lamps

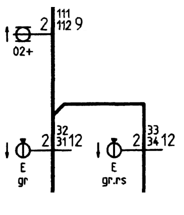


A lamp consists of either one or two filaments. Both examples are displayed on the figures to the left. The lamp on the upper figure only has the main filament and the lamp on the lower figure has been supplied with a spare filament. Both of the filaments on the lower lamp share the pin to the left (5ø) but the other ends of the filaments are connected to two different pins (6ø and 8ø).

2.3.1.9 Wires



The wires are the lines which connect the components described above. At this figure two contacts are connected. The order in which the pins are written on the symbols, is used to find out which pins on the two symbols that actually are connected to the wire. Since the wire is connected to the lowermost part of the uppermost contact, one has to look at the lowermost pin number. This tells that the wire is connected to pin 112 on the relay that is placed at field 9 and level 2 in the rack of relays. The same technique is used to find out which pin the other end of the wire is connected to. The wire is connected to the uppermost part of the lowermost contact therefore the wire must be connected to pin 32 on the relay which is placed at field 12 and level 2 in the rack of relays.



Sometimes more than one wire is connected to a single pin. This will result in a branch like in this figure. One end of the horizontal wire is connected at pin 33 on the relay in field 12, level 2. The other end of the wire bends just before it hits the vertical wire to the left. This indicates which of the two pins on the vertical wire it is connected to. The wire bends downwards which means the wire is connected to pin 32 on the relay in field 12, level 2 (the lowermost contact).

This means that the wires from the uppermost contact and the rightmost contact are both connected to the same pin and the current can thus run from the uppermost contact to both of the lowermost contacts.

2.3.2 Example

This section will give a brief example of how a circuit is drawn in a diagram using Banedanmark's notations described in the previous sections.

In section 2.2.11 it was described how a train route table could be converted into an electrical circuit. Using diagram notation the electrical circuit in figure 2.9 will be displayed partly as in figure 2.11. The diagram does not display all components described in section 2.2.11. For instance the connection between the coil pins and the track section is not displayed. All components on figure

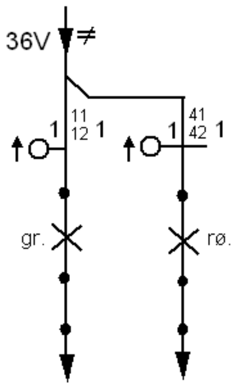


Figure 2.11: *A subset of figure 2.9 in Banedanmark diagram notation.*

2.11 are drawn in their normal state where the current can pass from the positive pole through the leftmost contact and the green lamp to the negative pole. The current cannot pass through the red lamp, since the rightmost contact is open. As soon as the associated track section gets occupied, the relay is dropped and hence the leftmost contact is open and the rightmost contact is closed. This turn the green lamp off and the red lamp on.

CHAPTER 3

Analysis

In this section the requirements to the application will be analyzed, whereafter issues that might lead to potential problems in the implementation of the interlocking system will be identified. Finally the results of the analysis will be structured and clarified through use cases.

3.1 Requirements

The initial requirement to the application is as follows:

The application must automate analysis of interlocking systems.

Since the main part of an interlocking system is the associated circuit, it is the behaviour of the circuit that should be automated. This is done by simulating the propagation of current and the state of components in the circuit. A sub-requirement is that the simulation should be executable in two different ways:

1. It should be possible to do a ”full simulation” of a circuit, such that the final state of the circuit will be displayed, according to the state the circuit was originally in.

2. It should somehow be possible to display intermediate states of the full simulation, such that e.g. fault location is easier to perform, i.e. the propagation of current should be visualized in intermediate "steps".

To be able to satisfy these sub-requirements, an overall analysis of an interlocking system must be performed. There is also an indirect requirement to the application, i.e. that it must incorporate a GUI (Graphical User Interface) so the various states of a circuit can be displayed to the user.

In the following sections entities and functions in an interlocking system will be identified with use of the domain description. Next an analysis will be made to identify possible entities and functions not identified in the domain description. Finally it must be considered which entities and functions are necessary to satisfy the requirements, and whether additional functionality could be included in the application.

3.1.1 Entities of the domain

To be able to understand and translate the domain (with regards to future design and implementation) the entities of the domain need to be identified. The atomic entities identified are:

- Positive pole of the power source
- Negative pole of the power source
- Pin
- Wire
- Track section
- Point
- Train
- Operator

Besides these a number of composite entities are identified. In parentheses is described of which components the composite entity is composed.

- Circuit (A power source, any number of relays, wires, fuses, resistors, lamps and buttons)

- Diagram (Any number of wires, regular coil pins, steel core coil pins , upper and lower contacts, buttons, fuses, resistors, lamps and positive and negative power sources)
- Regular relay (2 pins and either 6, 10 or 20 contacts)
- Steel core relay (4 pins and either 5, 9 or 19 contacts)
- Upper contact (2 pins)
- Lower contact (2 pins)
- Button (3 pins)
- Fuse (2 pins)
- Resistor (2 pins)
- Filament (2 pins)
- Lamp (1 or 2 filaments. In case of 2 filaments the total number of pins is 3)
- Signal (1 or more lamps)
- Operators panel (Any number of track sections, points and buttons)
- Train Route (1 or more track sections and any number of points)
- Train Route Table (Any number of lamps, track sections, points and train routes)
- Interlocking System (A power source and any number of track sections, points, relays, buttons, fuses, resistors, lamps, signals, wires and train routes)
- Station (1 operators panel, 1 interlocking system, 1 train route table and any number of signals)

3.1.2 Functions of the domain

The entities identified in the previous section are used when identifying the functions of the domain. The following functions define the ways the state of the system can be changed by external events or actions. That is for instance when the operator interacts with the system by using the operators panel.

- Create a rack of relays

- Place a relay in the rack of relays
- Move a relay in the rack of relays
- Remove a relay from the rack of relays
- Add the spare filament to a lamp
- Remove the spare filament to a lamp
- Connect two components using a wire (power sources, buttons, fuses, resistors, lamps, contacts or relay pins)
- Disconnect two connected components
- Assemble track sections
- Apply current to the circuit
- Cut current from the circuit
- Push a button
- Release a button
- Switch a point
- A train approaches the station
- The front of a train moves forward and hence occupies a track section
- The rear of a train moves forward and hence frees the rearmost track section previously occupied by the train
- The coil pins on a relay are not conductive
- A regular relay stays drawn even though no current is applied to it
- A regular relay stays dropped even though current is applied to it
- A steel core relay stays magnitezed even though current is applied to the demagnetezing coiling
- A steel core relay stays demagnetized even though current is applied to the magnetezing coiling
- A contact is conductive when it should be non-conducting
- A contact is non-conducting when it should be conductive
- A fuse is exposed to over-current and hence it gets non-conducting

- A resistor is exposed to over-current and hence it gets non-conducting
- A filament in a lamp is exposed to over-current and hence it gets non-conducting
- A broken component is repaired

In addition to this there are a number of functions that are used when creating the diagrams. These functions are in the use cases in section 3.3.

3.1.3 Analysis of Interlocking Systems

In this section an analysis of the interlocking system will be made, disclosing any entities not recognized in the description of the domain.

3.1.3.1 Interface Relays

In the domain description it was explained how relays change state as a result of current being applied or being removed. In this section it is recognized that there is a distinction between the events that lead to current being applied or removed.

Often current is being applied to or removed from relays as a consequence of the change of another relays state. This can be perceived as a chain reaction that is initiated by e.g. push a button. The pushed button allows current to propagate to a relay, the relay changes state, which allows current to propagate through the upper contacts of the relay to other relays and so on.

Not all relays change state as a part of a chain reaction though. The relays unaffected by chain reactions will be referred to as *interface relays*, since their state is purely influenced by sources external to the circuit. The interface can be a part of a chain reaction, but then it will be the first initiating part in the chain reaction. It cannot be predicted when an interface relay changes state, whereas a regular relay always changes state as a result of certain conditions in the circuit. It is important to recognize that the mechanical functionality of an interface relay is the same as for a regular relay, but the conditions under which the relays change state are different. The relays connected to track sections and points are in this way all interface relay.

The conceptual difference between an interface relay and a regular relay means that an interface relay also qualifies as an entity in the system.

3.1.3.2 Ends

When the user creates the structure of a station with track sections and points, it is necessary to specify how the track sections and points are assembled to each other. The user should be able to define exactly which ends should be assembled, and thus the entity *End* is defined. The definition of End leads to additional functions: assemble two ends and disassemble two ends. The function of disassembling two ends is delimited though since there is only limited time for the project as previously mentioned. By this abstraction the entities track section and point are made composite instead of atomic. They are composed of 2 and 3 ends, respectively.

3.1.4 Scope of application

The main requirements for the application are formulated quite vaguely. There is no specification of which components are required to be a part of the simulation.

The scope of the application is thus defined by delimiting the identified entities and functions. Ideally all entities would be included in the application, but since there is only a limited period of time the following entities are delimited:

1. Operator. The operator entity is only necessary if there is a need to differentiate between the operators on a station, or if the operator was a composite entity. The simulation of the interlocking system is only dependent on the actions of an operator, not of the operator himself/herself, and therefore the operator entity is excluded.

2. Signal. The signal entity is a composition of lamps that already exist in the circuit. The state of each lamp can thus be read in the circuit, which reduces the functionality of the signal, to merely being a different way of displaying the lamps. For this reason the signal entity is delimited.

3. Train route table. The train route table is an abstract description of the required functionality of the interlocking system. The train route table entity is thus useful for validating the circuit. The functionality of validating a circuit against a predefined train route table is quite complex though, and considered out of the scope of this project.

4. Train route. The only entity that uses the train route entity is the train route table. Since the train route table is delimited, so is the train route entity.

As mentioned in the domain description, the track sections and points are connected to the electrical circuit to detect when they are occupied by a train. The part of the circuit where this connection is made is also delimited, that is the track section and point are not perceived as electrical components, instead an interpretation of this connection will be made in the design.

Since the main requirement to the application is focused on *simulating* an interlocking system, focus will be on implementing as much functionality as possible. The more functionality implemented the more accurate a translation from paper diagrams to application diagrams will be. The more precise this translation is, the more correct the simulation will be. Other areas such as usability is thus rated as a low priority.

The scope of the application has now been outlined, and will be further clarified through use cases as described in section 3.3.

3.2 Issues

In this section areas in the interlocking system that introduce problems which need further consideration when developing the application will be analyzed.

3.2.1 Embedded cycles in circuit

Besides being a structure of electrical components, a circuit can be perceived as a collection of logic statements. Each relay has a statement associated. A relay for a track section could e.g. have the statement "The track section is free". The statement is true when the relay is drawn, and false when the relay is dropped. With this perception the upper contacts on the relay have the same truth value as the relay, and the lower contacts have the negated truth value of the relay.

Circuit compositions that are equivalent to expressions such as $A \vee B$ and $A \wedge B$ are frequently used in circuits for interlocking systems. Examples of these expressions are "The point must be switched to the plus *or* minus direction", and "The track sections in the train route must be free *and* the train route must be locked". If the components are the coil pins or contacts, the statements A, B indicates the state of the relay. The expression of statements is true if a path of current is created from the plus to the minus pole. An example of the mentioned statements can be seen in figure 3.1 and 3.2. Both expressions are true, since a path of current is created. In figure 3.2 it is important to notice

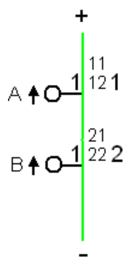


Figure 3.1: *A and B both must be conductive to create a live path in the circuit.*

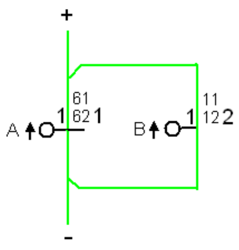


Figure 3.2: *A or B must be conductive to create a live path in the circuit.*

that there only is one path of current. The contact with pins 61-62 is open and current passes for this reason not through the contact. The path of current is thus $\{+, 61, 11, 12, 62, -\}$. The $A \vee B$ composition could be problematic since the connected components form a cycle. If both A and B are true, the path of current also forms a cycle as seen in figure 3.3. The path of current might never

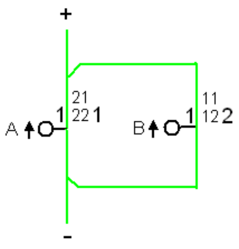


Figure 3.3: *The path of current forms an embedded cycle in the circuit since both A and B are true.*

encounter the negative pole, the path could e.g. be $\{+, 61, 11, 12, 62, 61, \cdots\}$ and so on. When the search algorithm for the circuit is designed, considerations must be made to prevent infinite loops due to these embedded cycles.

3.2.2 Cyclic Circuits

A full simulation is completed without detecting a cyclic circuit when a final state of a circuit is found. The final state of a circuit is the state where no electrical component changes state. Since relays are the only electrical components that change state without external influences, the final state of the circuit is the state where no relays are going to change state.

It has to be considered that this final state in some cases does not exist. If the final state does not exist, it can only be because there always are relays that change state. The circuit consists of a finite amount of components, so it must be the same relays that change state. If it is the same relays that change state, the circuit or a subset of the circuit must be a cycle, as seen in figure 3.4. When the full simulation is designed it has to be determined how potential cyclic constructions are detected in a circuit.

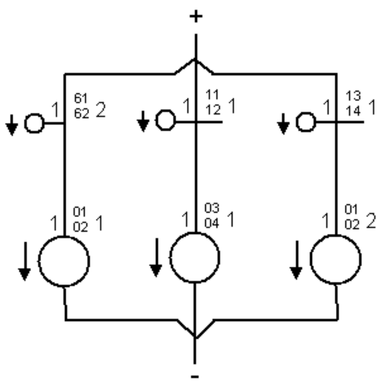


Figure 3.4: *A cyclic circuit. When current passes through the leftmost coil pins the associated relay is drawn and the contact displayed in the middle gets closed. This allows the current to propagate to the coil pins in the middle of the figure and thereby this associated relay is drawn and the rightmost contact is closed. When the current changes the state of the relay associated with the rightmost coil pins, the leftmost contact gets open and cuts the current to the leftmost coil pins.*

3.3 Use cases

The purpose of creating use cases is to identify in more detail the functionality required of the application, and how this functionality is used to solve specific tasks. All actions in the application have been described which has resulted in 52 use cases. The use cases are divided into 4 areas: circuit, operators panel, train and administration as listed in table 3.1 to 3.4.

The details for each use case can be seen in appendix E.

Several fields generally used in use cases have been delimited from the use case template used in this thesis. The reason for excluding the fields is either because the value of the field is the same for all use cases, or simply because the field is evaluated as having no relevance pursuant to this project. The more detailed reasons are as follows:

Scope The scope field describes which system is considered black box. Since all use cases are started by a user's actions, and no other systems or actors are involved, the scope would always be the application it self.

Post condition The field "Post condition" has been replaced by the field "Suc-

Use case #	Name
1	Create relay
2	Add coil pins
3	Add contact
4	Add button
5	Add fuse
6	Add resistor
7	Add lamp
8	Add spare filament
9	Add power supply
10	Remove component
11	Remove spare filament
12	Connect terminals
13	Disconnect terminals
14	Push button
15	Release button
16	Draw relay permanently
17	Drop relay permanently
18	Set relay non-conducting
19	Set contact non-conducting
20	Set contact conductive
21	Set fuse non-conducting
22	Set resistor non-conducting
23	Set lamp non-conducting
24	Set spare filament non-conducting
25	Repair relay
26	Repair contact
27	Repair fuse
28	Repair resistor
29	Repair lamp
30	Repair spar filament
31	View normal state
32	Step simulation
33	Full simulation

Table 3.1: *Use cases for the circuit.*

Use case #	Name
34	Add track section
35	Add point
36	Assemble track sections/points
37	Occupy track section
38	Free track section
39	Occupy point
40	Free point
41	Switch point

Table 3.2: *Use cases for the operators panel.*

Use case #	Name
42	Start train
43	Occupy next track section
44	Free rear track section
45	Cancel train

Table 3.3: *Use cases for the train.*

Use case #	Name
46	Create project
47	Create diagram
48	Create operators panel
49	Save project
50	Load project
51	Reset station
52	Rename diagram

Table 3.4: *Use cases for application administration.*

cess end condition", since they are evaluated as having the same meaning.

Failed end condition The field "Failed end condition" has not been included in the template since a failed end condition always is the inverse of "Success end condition". Furthermore the negative scenarios are described in detail under the "Extensions"-description in the use case template.

Primary actors The field is excluded since there is only one primary actor and hence one primary user of the application, i.e. the personnel at Banedanmark.

Secondary actors Secondary actors specify other systems required for the specific use case. Since the application does not interact with other systems, the field would be empty.

Trigger Since the usage pattern is quite simple, it will always be the user's wish to perform an action that triggers a use case.

Performance The performance field describes how time-consuming a task of a use case is on a system. All tasks described in the use cases are expected to be performed "at once".

Frequency The frequency field describes how often an action is expected to be performed. The frequency of the use cases is not significantly different and hence the field is excluded.

Chapter 4

Design

In the following sections, the design of the application will be described. First general design issues will be resolved, whereafter the different parts of the application will be designed separately.

4.1 General issues

In the analysis the entities and functions of the domain were found. The entities and functions can be used to describe the functionality of the interlocking system. This is an important issue when deciding which type of programming language to use for the application. Entities can be mapped into objects and functions into methods in an object-oriented programming language. For this reason we have chosen to use an object-oriented programming language for the application.

4.1.1 Application architecture

As mentioned in the analysis it is a requirement that the application includes a GUI through which the state of the electrical circuit and the rest of the interlocking system can be observed and the operators panel can be operated. Furthermore the application is not meant to be used across a network but in an environment comprised by a single computer. This sets the state for choosing the model-view-controller (MVC) architectural pattern which is graphically illustrated in figure 4.1. The idea of this pattern is that the *model* layer only

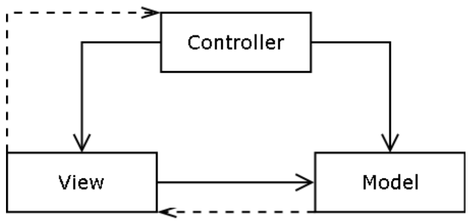


Figure 4.1: *The composition of the model-view-controller architectural pattern.*

knows about its own existence and takes care of the calculations and data storage of the application. In addition to this is a *view* layer that knows about the model layer. This layer is used to graphically display the information stored in the model and to collect events from the model and the user interacting with the application. The final layer is the *controller* layer which processes and responds to events created either by the user or internally in the application. These events can cause the controller to invoke changes in the model and/or view layers.

We have chosen to use a slightly different pattern which joins the view and controller layers into one; the *presentation* layer. This layer then takes care of displaying the information given in the model, listening for events created by the user or model and responding to these events.

In this application it is necessary for the presentation layer to always reflect the current state of the model, i.e. when a relay is being drawn it should immediately be displayed. A way for this to happen is to use the *observer pattern*. The idea behind this is that the GUI – or every GUI, in the case that more GUIs are used on the same model – is registered within the model as an observer. When the model experiences an internal change it invokes some predefined methods on all registered observers.

Again, in this application only one GUI is used and all changes made in the model are caused by actions triggered by the GUI. This makes the models knowledge of the GUI redundant. Instead the GUI just needs to look for changes in the model every time a method in the model is invoked by the GUI.

Another aspect is what should happen when errors are encountered within the model as a result of faulty input from the GUI or user. When not using the observer pattern it can be handled by using return values and/or throwing exceptions to the GUI. This option has been chosen for the application.

In general the model is designed without considering how the presentation layer is functioning. It is an advantage that the presentation layer can be replaced without having to change the core functionality.

Likewise it has to be determined whether the creation of model layer components (relays, buttons, fuses, lamps, resistors, points and track sections) should occur only within the model or anywhere in the application. The advantage of creating the objects in the presentation is that whereever possible only objects are passed on and not simple variables. But a draw bag is that it is harder for the model layer to control when different objects are initialized and how they are treated before being added to the model layer – if ever. By always initializing the components in the model, it is not possible for the presentation layer to blindly use a component that is unknown to the model layer. For these reasons all components are created within the model layer and afterwards passed on to the presentation layer as desired.

4.1.2 Division of entities

There are several entities in the domain. A structuring of these entities will give an overview and a better design. The entities can be divided into different groups sorted by a number of criteria, the question is which criteria will give a reasonable design. The design chosen here is made in accordance with the real world. There seems to be two different areas, the circuit consisting of connected electrical components, and the physical components on a station, accessible to the outside world. When the entities are divided in accordance with this perception, we get the following groups:

1. Circuit
    (a) Diagram
    (b) Positive pole
    (c) Negative pole
    (d) Regular relay
    (e) Steel core relay
    (f) Regular coil pins
    (g) Steel core coil pins

  (h) Contact
  (i) Button
  (j) Fuse
  (k) Resistor
  (l) Filament
  (m) Lamp
  (n) Pin
  (o) Wire

2. Operators panel
   (a) Track section
   (b) Point
   (c) Train
   (d) End

According to the domain the button is placed on the operators panel and connected in the circuit. In the division above the button only belongs to the circuit. Displaying the button on the operators panel is delimited since the functionality of the button is covered in the circuit. Instead of pushing buttons on the operators panel the buttons can be pushed directly at their location in the circuit.

The functionality of the operators panel is extended to include a train. The reason for this is that the operators panel already contains the track sections and points of the station and their assemblings. The train will initially only be visible by looking at the states of track sections and points. In reality small lamps placed by the track sections and points on the operators panel indicates whether the relevant track section / point is occupied so the design is quite similar to the domain. Station is the common term for the circuit and the operators panel, and thus not in the above division.

4.2 Model

The division of entities in the previous section has lead to three main areas in the application: a circuit, an operators panel and a train. In the following the general structure of the model is described and next the design of the circuit, operators panel and train is examined.

4.2.1 Structure

The model is in general developed without concern to the needs of the presentation layer. The station object works as an interface between the model and presentation though, and for this reason the methods in the station object are adapted to the needs of the presentation.

In the following the design structure of the model will be described by examinating which objects should be created. Finally it will be discussed how certain parameters and methods for satisfying the required functionality of the application are identified.

4.2.1.1 Non-Abstract Classes

Each entity listed in section 4.1.2 is with one exception turned into an object. The filament entity is contained in the lamp object since two filaments share the same pin as mentioned in the analysis. There is thus no need to create the entity "filament" as a separate object since a spare filament cannot exist independently of the original filament. The diagram and wire objects are only used in the presentation layer since their functionality is purely visual.

In general the model is quite similar to the domain. Relations between components are almost exactly as in the domain and an object exists for almost every entity identified. The structure of the model and the detailed relations between the objects are documented with a UML diagram, see appendix F.1.

The only abstraction made in the design in relation to the analysis is that a button is defined by two pins instead of three. The reason for this is that having additional components such as e.g. the lamp were prioritized higher than having additional functionality for the button. The key functionality of a button i.e. being conductive or non-conducting is still obtained with two pins.

4.2.1.2 Abstract Classes

Since some of the objects share common features the design of the application can be improved with use of abstract classes. When components have common features, abstract classes support reuse of code. 6 abstract classes are used in the model to structure the design. The purpose of each abstract class is described below:

Component gives each component a name and a description.

ElectricalComponent gives each electrical component a unique ID.

Terminal defines whether the specific terminal is live or not, and which terminals it is connected to in the circuit.

PinGroup defines a collection of pins where the connection between the pins is dependent of the state of the specific pin group. Thus a pin group can be either conductive or non-conducting.

Contact associates a pin group with a relay.

Relay defines the parameters and methods common for regular, steel core and interface relays.

4.2.1.3 Parameters and methods

In this section considerations are made as to which layer of abstraction certain parameters and methods should be placed.

There are two key parameters concerning the functionality of all electrical components, i.e. whether the component is conductive and whether it is live. Current only passes between pins in a pin group when the pin group is conductive. The parameter determining whether a component is conductive must thus be placed on the pin group abstraction level or in a level above e.g. the electrical component. There is no reason for putting the parameter at the electrical component level though since terminals are considered as being permanently conductive. For this reason the parameter determining whether a pin group is conductive is placed in the pin group object. Even though the terminal object does not have a parameter determining whether it is conductive the object has a method for determining this. Given two terminals the method determines whether their connection is conductive or not.

The parameter defining whether a component is live is placed in the terminal object since a terminal in a pingroup e.g. one of the pins in a contact can be live even though the connection between the pins is non-conducting as seen on figure 4.2. A method determining whether a component is live is placed at the pin group level even though a pin group has not got the live parameter. A pin group e.g. a contact is live if both terminals in the pin group is live.

Coil pins are always conductive but in addition to this the relay can be in one of two state: dropped or drawn. This is unique for the relay component and thus the parameter determining the state of the relay is placed at the relay level.

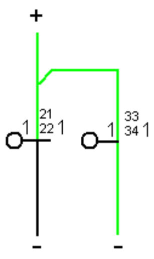


Figure 4.2: *Pin 21 is live even though contact 21-22 is non-conducting.*

The different levels of abstraction achieved with the 6 abstract classes make it easy to determine exactly where a parameter or a method belongs. Using the knowledge obtained from the domain and the levels of abstraction makes it somewhat straightforward to determine at which level the remaining parameters and methods are used to the optimum effect.

4.2.2 Circuit

In the analysis the phrase chain reaction was used to describe how an external action, e.g pushing a button causes current to reach relays. These relays change state, allowing current to pass through the upper pins on the relay to other relays and so forth. The propagation of current in these chain reactions must be simulated in two different ways. At any time the user should be able to progress a chain reaction to the final state where no more relays change state. This simulation is referred to as a full simulation. In addition intermediate ”steps” of a chain reaction should be simulated to make e.g. error detection easier. A full simulation thus corresponds to a simulation of a certain amount of steps until the circuit reaches a final state.

A step in a circuit state is thus the core in a simulation of a circuit. In reality it is of course not possible to propagate current in intermediate steps and hence a definition of an intermediate step is needed. In the following a definition of ”a step” will be determined. When the behaviour of a step has been determined the two types of simulation will be discussed in more detail in the following sections.

A *path of current* exists in a circuit if there is a ”free” path from the positive pole to the negative pole. A path is free if all components allow the current to pass, i.e. if buttons are pushed and contacts are closed and so forth.

In one step, giving the present state of the components in a circuit, all paths of current between the positive pole and negative pole should be found. The components on a path of current should then change from being "not live" to being "live".

If a component on a path is a relay, the passing of current should at some point result in a change of the relay's state. This change of state can happen in the same step as the current passes through the relay, or in the following step. For this decision to be made the domain is considered. When comparing the time it takes a relay to drop or draw, and the time it takes current to propagate through wires, it is assumed the latter is by far the least time-consuming scenario. This means, that in step s_i the relay becomes live, and in step s_{i+1} the live coil in the relay makes the relay draw, hence changes the state of all the contacts on the relay.

This interpretation of a "step" is specified with an example in figures 4.3 to 4.7.

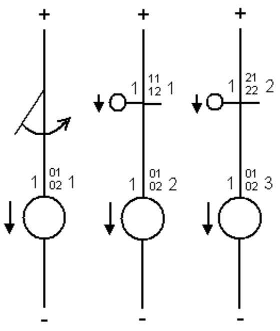


Figure 4.3: *Step 1: The button is not pushed and for this reason there are no paths of current.*

Banedanmark uses a compact form of notation called *functional diagrams* to describe states of a circuit. A functional diagram notes the state of each relay in a circuit using arrows to indicate whether a relay is drawn (↑) or dropped (↓). The definition of "a step" as written above, is equivalent to one row in Banedanmark's functional diagrams. The functional diagram for figures 4.3 to 4.7 can be seen in table 4.1. In the table the ID of a relay is written compactly as {level, field, position} so {1,3,r} denotes a relay at level 1 field 3 in the **r**ight position.

To reflect that a change of a relay state does not happen immediately, it has to be considered exactly which relays should change state in a step.

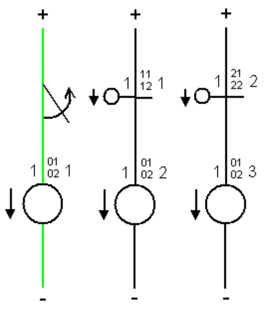


Figure 4.4: *Step 2: The button is pushed, allowing current through relay 1. The relay is still dropped.*

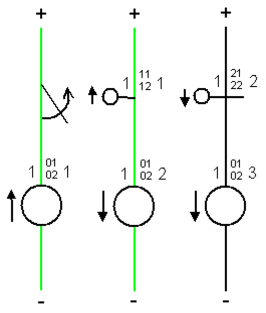


Figure 4.5: *Step 3: Relay 1 changes state, hence contact 11-12 allows current to pass.*

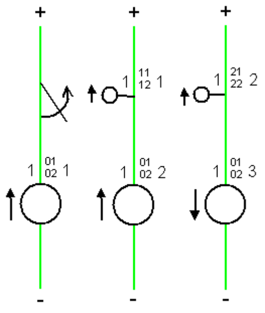


Figure 4.6: *Step 4: Relay 2 changes state, hence contact 21-22 allows current to pass.*

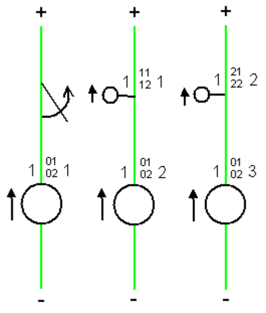


Figure 4.7: *Step 5: Finally relay 3 changes state.*

Step	Button	{1, 1, l}	{1, 2, l}	{1, 3, l}
1	Open	↓	↓	↓
2	Closed	↓	↓	↓
3	Closed	↑	↓	↓
4	Closed	↑	↑	↓
5	Closed	↑	↑	↑

Table 4.1: *Functional diagram for figures 4.3 to 4.7*

If relays are connected in parallel, parameters such as wire length, resistance, and the size of the individual relays (mass of the coil), influence the order in which the relays change state. In theory, the order in which the relays change state can be different from time to time. For this reason an abstraction is made from these parameters, such that the simulation considers the current to reach both relays at the same time, and thus the relays change state simultaneously, as seen on figure 4.8.

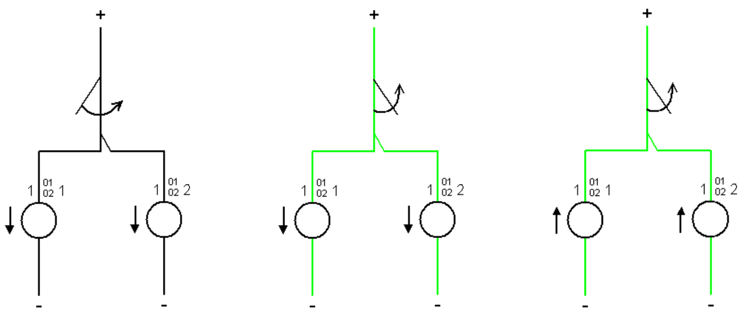


Figure 4.8: *When the button is pushed current propagates to the relays. The step after the relays become live, the relays draw simultaineously.*

Step	Button	{1, 1, l}	{1, 2, l}
1	Open	↓	↓
2	Closed	↓	↓
3	Closed	↑	↑

Table 4.2: *Functional diagram for figure 4.8*

Since current is perceived as propagating through a circuit instantly, current is applied to all relays in a series connection at the same time. For this reason the relays in a series connection will begin to change their state at the same time as seen on figure 4.9.

Step	Button	{1, 1, l}	{1, 2, l}
1	Open	↓	↓
2	Closed	↓	↓
3	Closed	↑	↓
4	Closed	↑	↑

Table 4.3: *Functional diagram for figure 4.9*

The change of state from drawn to dropped will proceed as the change of state

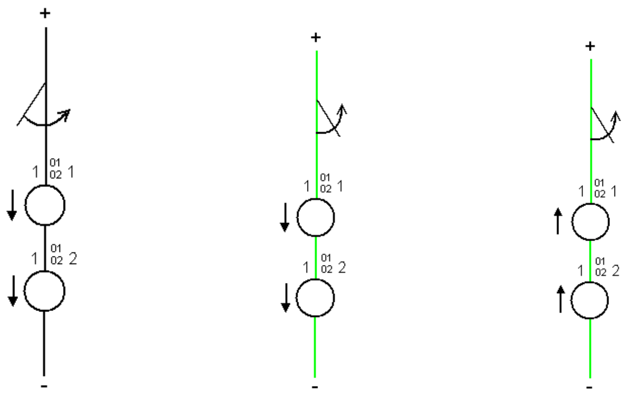


Figure 4.9: *When the button is pushed current propagates and reaches the relays approximately at the same time. The step after the relays become live, the relays draw.*

from dropped til drawn. If e.g. the buttons in figure 4.8 and 4.9 are released, all the relays will drop simultaneously in the following step.

As a definition has been made to the interpretation of a step the following sections will examine the two types of simulation, i.e. intermediate step simulation and full simulation.

4.2.2.1 Intermediate Step Simulation

In relation to the definition of a step, a step consists of two parts, i.e. searching a circuit determining which terminals are live and establishing which relays should change state for each step. These areas of the intermediate step simulation will be examined in the following sections.

Locating live paths in circuits
In this section an algorithm for locating live paths in a circuit will be designed. First a data structure will be chosen, whereafter an overall algorithm will be chosen for further development. Next a pseudo algorithm will be provided and the running time for this is analyzed.

Data structure
There are two important factors to consider when determining which data structure to use: a terminal can have from zero to many connections, and a circuit

can contain embedded cycles. The fact that a terminal can have multiple terminals connected makes data structures such as stacks, queues, hash tables and linked lists highly inappropriate. This roughly leaves trees and graphs. At first sight trees seems well suited, since the fact that a terminal can have several terminals connected is in accordance with the parent-child design. When embedded cycles are considered though the tree structure is not as useful as first assumed. The problem arises because a single terminal can have several parents as seen in figure 4.10. More than one parent per terminal is counter-intuitive

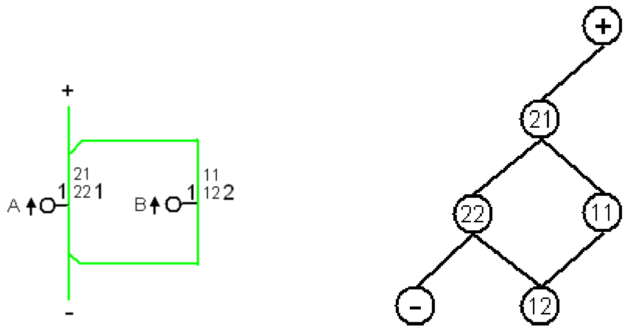


Figure 4.10: *The embedded cycle to the left will result in a tree structure where a terminal (pin 12 on the figure) has two parents.*

and conceptually in conflict with the idea of a tree structure.

Finally a graph is considered as a data structure. The structure of a graph allows an unlimited amount of connections per terminal, and cycles does not disrupt the concept of the graph structure. Every pin has a maximum of two wires connected, and an internal connection to a pin partner (e.g. the other pin in a contact). For this reason there are a maximum of three connections for each pin, that is a maximum of three connections C per terminal T. Since $3C << T^2$ the graph is categorized as being sparse, i.e. there are significantly more vertices than edges [1]. This means that it will be an advantage to use a type of adjacency-list over an adjacency-matrix. This is supported by the fact that all adjacency vertices *must* be searched, and hence being able to quickly e.g. determine whether a certain edge exists is not useful. The graph should be undirected in accordance with the behaviour of current. If the graph was directed, special attention would be needed when connecting the circuit, which would lead to an unnecessary source of error.

Search algorithm

Next a suitable algorithm for searching a graph has to be found. There are two main searching algorithms for graphs to consider; breadth-first search and depth-first search. Before this decision is made, conditions for stopping the search is considered, since these conditions might have an influence on which search algorithm to use.

The current is perceived as going from the positive pole through a number of terminals to the negative pole. A rough approach for finding the paths of current in a circuit would be to start the search from the positive pole, only stopping the search when the negative pole or a non-conducting connection is found.

Since the search algorithm is the core in the simulation and is likely to be used extensively, it is important that it is as efficient as possible. For this reason it is necessary to assess when a search of a specific path can be stopped. When a terminal a is being investigated, an individual investigation needs to take place for each terminal a is connected to. For each connected terminal b_j there are different scenarios to take into consideration. The decision to be made depends on b_j and the relationship between b_j and a. The connected terminal can be either "live" or "not live", while the relationship between a and the connected terminal can be "conductive" or "non-conducting". If the connection between a and b_j is non-conducting there is no need to continue the search. If the connection is conductive though, the decision whether to continue the search rely on whether b_j is i) the negative pole, ii) live or iii) not live. If b_j is not live, the search continues. If b_j is live or the negative pole, the path searched so far is set live. This is done under the assumption, that if b_j is live, it must be because a path from the positive pole to the negative pole exists, in which b_j is included. Besides the characteristics of the individual terminal and the connection it also has to be taken into consideration that a terminal under investigation could create a cycle. A path $\{t_0 \rightarrow t_1, \rightarrow \cdots \rightarrow t_k\}$ in a circuit forms a cycle if $t_k \in \{t_0, t_1, \cdots, t_{k-1}\}$. There is no need to continue the search since the terminal creating a cycle will not lead to connected terminals that have not yet been searched and the search would result in an infinite loop. Thus the possible scenarios for a connected terminal b_j are as follows:

1. The connection between a and b_j is non-conducting. The search is stopped.

2. The connection between a and b_j is conductive and b_j is live or the negative pole. The path searched so far is set live, and the search is stopped.

3. The connection between a and b_j is conductive and b_j is not live. The search is continued.

4. The connection between a and b_j is conductive, b_j is not live but adding b_j to the current path will form a cycle. The search is stopped.

The terms of stopping a search of a specific path has now been defined, and the overall search algorithm can be chosen. The search of a path should be stopped when a live terminal is found among other things. The search algorithm can be optimized if live components are found as quick as possible. A depth-first search could thus be more efficient for a circuit than a breadth-first search, since the negative pole usually is the vertex the farthest/deepest from the positive pole. If internal resistance should have been simulated a breadth-first search would propably have been a better choice.

Pseudo code

The pseudo code for the search algorithm is given below. The method is recursive and is placed in the terminal object class such that `this` refers to the terminal currently being searched. The algorithm could also have been placed in a central object e.g. the circuit object. The algorithm must behave in two different ways though, depending on whether the terminal is the negative pole or not. As described in the design the search should be stopped whenever the negative pole is met. Having the algorithm in a central class would thus require validating the type of terminal being searched. This is avoided using polymorphism and hence placing the search method in the terminal object and override it in the negative pole object.

Listing 4.1: *Pseudocode for the searching algorithm in the terminal object.*

```
findLiveComponents(Terminal from, List currentPath){
  List connectedTerminals = copy(this.adjencyList)
  connectedTerminals.remove(from)
  currentPath.add(this)

  for each b in connectedTerminals
    if conductive(this, b)
      if b is live
        for each c in currentPath
          c.setLive(true)
      else
        if b does not create embedded cycle
          newPath = copy(currentPath)
          findLiveComponents(this, currentPath)
}
```

The terminal being searched is referred to as a and the terminals connected to a are called b_j. The terminals in the path currently being searched are called c_l.

In line 2 in the algorithm a copy is made of the terminals connected to a. The terminal that was searched before a, called `from` in the algorithm, is removed from the copied list. Since `from` was previously searched there is no need to search it again. In line 4 the terminal being searched is added to the current path.

The `for`-loop in line 6 examines each connected terminal in the copied list. In accordance with the principles described in the design the search of the current path stops if the connection between a and b_j is non-conducting, if b_j is live or if b_j forms a cycle. The order in which these conditions are examined is made to make the algorithm as efficient as possible. For example the algorithm checks whether a component forming a cycle is live, before determining which actions to take. A recursive call is only made if the connection is conductive, b_j is not live and b_j does not form a cycle.

The negative pole overrides the `findLiveComponents` method, since the search should stop when the negative pole is found.

Listing 4.2: *Pseudocode for the searching algorithm in the negative pole object.*

```
1 findLiveComponents(Terminal from, List currentPath){
2   currentPath.add(this)
3   for each c in currentPath
4     c.setLive(true)
5 }
```

In line 2 in listing 4.2 the negative pole is added to the current path whereafter the current path is set live.

Running time

In this section `findLiveComponents(Terminal from, List currentPath)` in listing 4.1 is analyzed with regards to the running time. The terminals in the circuit is denoted T and the connections are denoted C.

The code in line 6 and 9 seems to be the most complicated constructions in the code and thus these lines will be examined first.

An initial guess on the running time for line 6 is $\Theta(C)$ since $\sum |Adj[\texttt{from}]| = \Theta(C)$ [1]. In the `findLiveComponents` algorithm there is occasionally a need for searching the same connections more than once though. This need arises when a circuit contains an embedded cycle. The order in which paths are discovered determine whether a connection should be searched more than once. In the following sections the best- and worst-case scenario for a circuit with an embedded cycle is examined.

Best-case discovery of paths
To give an example of best- and worst-case scenarios consider figure 4.11. The circles annotated a, b, c and d are individual pins, and the figure is thus more detailed than Banedanmark's diagrams. a and b constitute a contact, the line

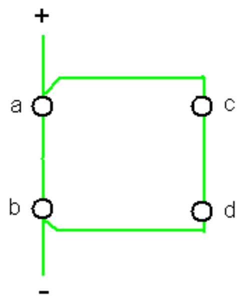


Figure 4.11: *A circuit structure containing an embedded cycle.*

drawn between a and b is the horizontal pole mentioned in the domain description, see section 2.2.1. c and d constitutes a contact likewise. In this way the figure represents a graph since the terminals are vertices and the connections between them are edges.

There are 4 possible paths in the circuit on figure 4.11:

$$\begin{aligned}
P_1 &= \{+, a, b, -\} \\
P_2 &= \{+, a, b, d, c, a\} \\
P_3 &= \{+, a, c, d, b, -\} \\
P_4 &= \{+, a, c, d, b, a\}
\end{aligned}$$

As the base of the algorithm is a depth-first search, P_1 and P_2 will be executed consecutively, and so will P_3 and P_4.

The best-case scenario is when the paths are discovered in the order starting with P_1 or P_3. The reason for this is that these paths make components live, which is a criteria for stopping the search of another subpath. In table 4.4 a best-case scenario is described where path P_1 is the first path to be discovered. For any circuit the overall best-case scenario is of course if many or all connections are non-conducing, since the amount of recursive calls then will be $<< C$. Circuit structures where all connections are conductive are highly unlikely since so-called "or-constructions" as on figure 3.2 are frequently used, e.g. "both points must be switched to track 1 *or* track 2".

Step	Path	Terminals discovered	Result of search algorithm
1	$\{+, a, b, -\}$	$\{+, a, b, -\}$	Negative pole is found, stop search and set current path live.
2	$\{+, a, b, d, c, a\}$	$\{d, c\}$	a is live, stop search and set current path-live.
3	$\{+, a, c\}$	$\{c, d\}$	Connected is live, stop and live

Table 4.4: *The best-case discovery of paths in the circuit on figure 4.11*

Worst-case discovery of paths

The worst-case scenario for the circuit in figure 4.11 is when P_2 or P_4 is searched first. One would think the worst-case scenario is when all connections in a circuit are conductive and the circuit is cycle-free, since this according to the structure of the algorithm will lead to the highest amount of recursive calls. This is not the case though, a connection can be searched more than once as seen in table 4.5.

Step	Path	Terminals discovered	Result of search algorithm
1	$\{+, a, b, d, c, a\}$	$\{+, a, b, d, c, a\}$	Cycle, search is stopped.
2	$\{+, a, b, -\}$	$\{-\}$	Negative pole, search is stopped and path is set live.
3	$\{+, a, c, d, b\}$	$\{c, d\}$	b is live, search is stopped and the path is set live.

Table 4.5: *The worst-case discovery of paths in the circuit on figure 4.11*

c and d are searched twice, i.e. in step 1 and step 3. The search in line 3 is not redundant though, since the search results in setting c and d live. c and d are searched twice due to the order in which the paths in the circuit are discovered.

The order in which paths are found described in table 4.5 supports the fact that the graph should be undirected. If the edge between a and c was directed not all pins would become live in a search of the circuit.

To avoid the worst-case scenario a path to the negative pole should be found as quick as possible. When a terminal is connected to the negative pole, the negative pole should thus be the first terminal in the adjency list, since this will guarantee that the path to the negative pole is searched first. When terminals are connected the negative pole could be placed ahead of all other connected terminals in the adjency list, and the path to the negative pole would be searched first. The problem could still occur though if the path branches two or more steps before the negative pole, as seen on figure 4.12.

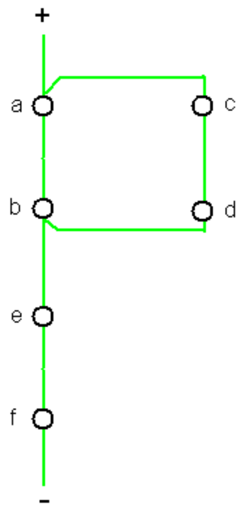


Figure 4.12: *Pin c and d will still be searched twice if the first pin in b's adjency list is d.*

To make sure that e is searched before d the search algorithm will have to know beforehand which path leads to the negative pole which was the goal of the search in the first place. For this reason it is accepted that some terminals can be searched twice.

An upper-bound on how many times the algorithm is called is still $O(C)$ since each pin has three connections at the most i.e. $O(3C) = O(C)$.

In line 9 the current path being searched is set live. A rough guess of the running time is $O(T)$ since the current path contains the greatest amount of terminals when all terminals are in the same path. If a path branches though, the terminals the two branches have in common will be set live twice. The running time is not $O(T^2)$ though since this would require a current path with all terminals, end each terminal should be set live one by one. For this reason the running time is $O(c_1T)$ where c_1 is an unknown constant.

The worst-case running time for the remaining lines can be seen in table 4.6

Line number	Worst case running time	Scenario
2	$T - 2 = O(T)$	The positive pole is connected to all terminals except itself and the negative pole.
14	$O(T)$	The current path contains all terminals in the circuit.

Table 4.6: *Running time analysis of the search algorithm.*

The code in the lines not mentioned in table 4.6 runs in constant time.

The total running time for `findLiveComponents` can now be determined. A rough estimate would be to multiply the running time for line 6 and 9 which would lead to $O(T \cdot C)$. Instead aggregate analysis is used to get a more detailed analysis [1]. Lines 2 and 14 have a combined running time $O(2T) = O(T)$ and as previously mentioned the `for`-loop in line 6 is called $O(3C) = O(C)$ times. The loop in line 9 was called $O(c_1T) = O(T)$ times and thus the total running time for the algorithm is $O(2 \cdot O(T) + O(C)) = O(C + T)$.

Changing relay states
In any step current is applied to or removed from a certain amount of relays and at some point the relays are going to change state. A relay should change state in the scenarios described in section 4.2.2.2. Due to the definition of a step relays should change state in the beginning of a step. Before the step is completed, it should be determined whether any relays are going to change state in the forthcoming step, such that it e.g. can be determined whether or not a circuit has a final state.

Before each step is executed the terminals in a circuit should be set to "not-live". If the terminals are not reset, terminals that should become live might not become live and vice versa. A possible solution to this could be to associate a boolean parameter with each terminal, indicating whether the terminal is live or not. Before each step this parameter should be reset to "not-live". This solution seems a bit rough though since each terminal in the circuit should be reset for each step making the operation somewhat heavy. A more efficient solution is to associate every terminal with an integer which will specify whether the terminal were live or not. A terminal is live if the "live number" is equal to the step number and not live otherwise. The ”live number” is initialized to a value that is not equal to the step so every terminal initially is not live. When a path of current is set to be live, the ”live number” is set to the value of the current step. When the next step is performed the step number will increase leaving the live number of previously live terminals an integer smaller. In this way the terminals in the circuit are indirectly reset to ”not live” in every step.

Pseudo code

The pseudo-code for the simulation algorithm is a combination of changing relay states and finding live paths in the circuit:

Listing 4.3: *Pseudocode for the step algorithm.*

```
 1  int step = 0
 2
 3  boolean nextStep()
 4  for each rtc in relaysToChange
 5    if rtc is still connected in circuit
 6        rtc.changeState
 7  positivePole.setStep(step++)
 8  positivePole.findLiveComponents(null, new List())
 9  rtc = new List()
10  for each r in relays
11    if r should change
12        relaysToChange.add(r)
13  return !relaysToChange.isEmpty()
```

The algorithm is placed in the circuit class. The step number is initialized to 0, and is incremented for each call of `nextStep()` as seen in line 7. In line 7 the static parameter `step` in the terminal object is set.

In lines 9-12 a list of relays that should change `rtc` is made, such that $rtc \subseteq r$. The list is created using the definitions for when a relay should change as mentioned in the design section 4.2.2.2. The `rtc` list is used in the return statement to determine whether the next call to `nextStep()` will change state of any relays in the circuit.

In line 4-6 the relays that should change before the search of the circuit is initiated are changed. The `rtc` list made in the ending of the previous step is reused to avoid searching all the relays in the circuit again. To be able to reuse the list it is checked that the relays in `rtc` has not been deleted in between the steps, as seen in line 5. The search is performed with a call to `findLiveComponents` in line 8.

Running time
A detailed analysis of the `nextStep` algorithm is listed in table 4.7. The relays in the circuit are denoted by R, and a relay in R is denoted r. The `relaysToChange` list is called RTC, and an element in RTC is called rtc. The code in the lines

Line number	Worst case running time	Scenario
4	$RTC = O(R)$	All the relays in the circuit change state.
8	$O(T + C)$	Cf. the running time analysis for findLiveComponents in section 4.2.2.1.
10	$R = O(R)$	

Table 4.7: *Running time analysis of the search algorithm.*

not mentioned in the table takes constant time to execute due to hashtables. The running time for `nextStep` is thus $O(2R + T + C) = O(R + T + C)$.

4.2.2.2 Full simulation

In the following sections the term "final state" will be examined in more detail, and possible problems concerning executing a full simulation will be treated.

Detecting cyclic circuits
A full simulation corresponds to continuously executing a step, until a possible final state of the circuit is reached. The final state is as previously mentioned the state where no more relays are going to change state. When this state occurs the state of the circuit will remain in the same state until the user performs an action e.g. pushes a button. The problem of identifying the final state of the circuit is thus equivalent to determining whether any relays are going to change state as a result of the step that has just been executed. When a random step s_i has been executed the relays that are going to change state in step s_{i+1} are:

1. Regular relays that are live, dropped and functional.
2. Regular relays that are not live, drawn and functional.
3. Steel core relays where the demagnetizing coiling is live, the relay is magnetized and functional.

4. Steel core relays where the magnetizing coiling is live, the relay is demagnetized and functional.

If there are one or more of the above relays in the circuit the current state of the circuit is not the final state. If there are zero of the relays in the above list the current state of the circuit is the final state. If a final state is detected in step s the state of the circuit in the following step s_{i+1} will be identical to the state of the circuit in step s.

A circuit is a cyclic circuit if there does not exist a final state. In relation to the above description a circuit is not a cyclic circuit if there are one or more relays in the circuit in one of the above described states. At some point the search for a final state should be cancelled and the circuit should be branded as a cyclic circuit.

When the `nextStep` algorithm returns `false` for step s_i it means that no relays are going to change state in step s_{i+1}. When no relays are going to change state the final state of a circuit is found. When a circuit has no final state and thus is a cyclic circuit, the `nextStep` algorithm will return `true` infinitely.

To avoid an infinite loop it needs to be determined at what point the search for a final state should stop. The number of calls to `nextStep` should somehow be related to the number of relays in the circuit, since the number of relays in a circuit determines the number of possible states the circuit can be in. Electrical components such as buttons, lamps and so forth are in this case considered as having only one state, i.e. the state the components are in when a call to the full simulation is executed. The reason for this is, that it is not possible to change the states of these components, until the call to the full simulation is completed. The possible number of states a circuit can be in must thus be determined solely by the number of relays. If r denotes the number of relays in the circuit, the possible states of the circuit are 2^r. Due to the behaviour of a circuit it is only $2r$ states that can be achieved though.

The structure of the cyclic circuit in figure 3.4 at page 32 has $2r$ states. The functional diagram in table 4.8 for the circuit shows the $2r$ states.

Pseudocode

The pseudocode for the full simulation algorithm can be seen in listing 4.4.

Listing 4.4: *Pseudocode for the full simulation algorithm.*

```
boolean fullSimulation()
int maxStep = step + 2*r
```

Step	{1,1,l}	{1,1,r}	{1,2,l}
1	↓	↓	↓
2	↑	↓	↓
3	↑	↑	↓
4	↑	↑	↑
5	↓	↑	↑
6	↓	↓	↑

Table 4.8: *Functional diagram for the circuit on figure 3.4, page 32*

```
3 while (nextStep() and step<maxStep){
4 }
5 if (nextStep)
6   return false;
7 return true;
```

At first the maximum step number is calculated by adding the current step number to $2r$. Alternatively the step number could be reset to 0 after the simulation executed. The problem with this though is that terminals that were previously live in step s_i will incorrectly become live again when the step number has been incremented to s_i.

In line 3 `nextStep` is continuously called until `nextStep` returns `false` or until it has returned `true` $2r$ times. In line 4 it is determined which scenario caused the `while` loop to terminate. If the circuit reaches a state where no relays change state i.e. `nextStep` returns `false`, the next call to `nextStep` will also return `false`. If a circuit is a cyclic circuit on the other hand, all $2r$ calls to `nextStep` returns `true`. Since a circuit only has $2r$ possible states, the following states of the circuit will be identical to the states that already have been detected. Thus the next calls to `nextStep` will return `true`.

Running time

In the worst-case scenario $2R+1$ calls are made to `nextStep`. Since the running time for `nextStep` is $O(R+T+V)$ the total running time for `fullSimulation` is thus $O(R \cdot R+T+V)$.

4.2.3 Operators panel

In this section the design of the operators panel will be explained. As described in the analysis the electrical connection between the relays and the track section/points is delimited so an abstract interpretation of the connection needs to be made.

In the following sections the abstract connection between a track section and the associated relay will be designed. Next the abstract connection between the plus and minus direction on the point and the associated relays is designed.

4.2.3.1 Track sections

Somehow a connection between the track section and the relay has to be modelled. One way of doing this would be to create a relay in the circuit and manually switching the state of the relay by adding and removing current to the coil pins. When a track section should be simulated as being free the user must apply current to the coil pins. When the track section is simulated as being occupied current should be removed from the coil pins e.g. by deleting one of the wires connected from a pole to the relay. This solution is not particularly user-friendly since the user must remember which relays depict relays for track sections. Removing and adding wires to simulate a relay associated with a track section additionally seems rather awkward. In addition to this there would be no visualization of the track section itself.

The implemented solution associates an interface relay to each track section and point respectively. When operations such as "free track section" and "occupy track section" are carried out, the track section will change the state of the interface relay. The coil pins will thus not be connected with wires in the circuit, the state of the interface relay only changes through operations on the track section.

4.2.3.2 Points

As described in the domain the plus and minus relays on a point are electrically connected to the point. For the same reasons as in section 4.2.3.1 there is made an abstraction from the actual wire-connection between the plus and minus relays and the point itself. Instead a point has three associated interface relays; one interface relay with the functionality described in section 4.2.3.1, and two interface relays indicating the connection between the point and the plus and minus relays respectively. When the operation for switching the point is carried out, it is the point that changes the state of the relays. The coil pins will thus not be connected with wires in the circuit, the state of the interface relays only change through operations directly on the point. In fact connecting the coil pins in the circuit is *disallowed*, since an interface relay only should change state as a result of external events, not as a result of the propagation of the current in the circuit.

Disallowing the coil pins to be connected in the circuit removes the possibility of setting conditions for when the point should be able to switch. In reality a point cannot e.g. be switched when a train route is locked but this functionality is lost with the abstraction made above.

The actual physical switching of a point will take a certain amount of time. In this period of time neither the plus interface relay nor the minus interface relay will be drawn as described in the domain. This intermediate position of the point should also be modelled.

A possible solution is that the switch of a point is initiated by the user, bringing the point in the intermediate position. When the next step is executed the switch is completed to the relevant direction. The solution builds on the assumption that the time it takes to drop or draw a relay is equal to the time it takes to switch a point. The relation between the two periods of time is highly unlikely, and the times should thus not be compared. The chosen solution allows the user to define how much time it takes to switch a point by defining when a point should be switched to the intermediate position, and afterwards when the switch should be completed.

4.2.4 Train

A train can enter a station and stop at a platform, exit a station, or drive through a station. One way to simulate a train could be to synchronize the train in accordance to the steps of the circuit simulation. When the user starts the train, the train could move e.g. one track section forward each time a step is executed. This solution assumes that the time it takes a relay to change state is the same as the time it takes for a train to move a track section. The two time periods cannot be compared, and the solution furthermore puts bounds on the size and speed of the train.

The chosen design lets the user define the size and speed of the train. The user starts the simulation of a train by specifying the track section where the train should enter the station, allowing simulation of a train from different directions.

The user can define how many track section should be occupied, and how many track sections should be released. The speed of the train is simulated by the fact that in between steps it is up to the user how many track sections should be occupied and released by the train at a time. When the train should move to a point validation is performed as to whether the point is switched to the direction from which the train enters it, and that the point is not in an intermediate switch.

Track sections can be occupied with two different scenarios in mind; it can either be occupied by a train or by some other conductive object. Occupying a track section with e.g. a utility pole allows error detection to be made in relation to the train. If e.g. a utility pole lies across a track section the entry signal on a station should not be able to turn green at any time until the utility pole is removed.

Since a starting track section or point should be identified it has to be decided how this should be done. Since the operators panel already displays track sections and points the operators panel is used to define from where a train should enter a station. This solution is also user-friendly since actions concerning track sections and points are gathered in the same place. The starting track section or point should have exactly one end that is not assembled with another end. The reason for this is that the disassembled end indicates the open track and this information is used to determine in which direction the train should proceed.

4.3 Presentation

As for the model the presentation layer has to be designed to have the state of the model displayed as optimal as possible to the user. Besides this the user has to be able to create, display and interact with the components in the model by interacting with the application. This is done through the GUI.

First the structure of the presentation will be introduced and then a design will be made as to how components should be created and edited. Next it will be described how the user interacts with the application and then considerations are made as to how the different aspects of the application should be displayed. Finally it will be examined how a project should be saved and loaded.

4.3.1 Structure

The main purposes of the presentation layer is to allow the user to set up an interlocking system, have the electrical circuit displayed as diagrams, have the operators panel displayed and finally interact with the system. The diagrams that each documents a separate part of a circuit all belong to the same *project*. This means that a project contains all the documentation and information for an entire station.

For the user to be able to create a circuit by adding relays, contacts, buttons, etc. and connecting them, it should be possible to choose between a set of tools that each would allow the user to perform these actions. When a circuit is constructed the tools should be easy of access to facilitate the user's method of working. The tools can potentially be accessed through either:

- A separate window
- A menu bar
- A toolbar
- The right-click menu

The application is going to contain a large amount of tools so to make it easy for the user to locate and use a specific tool, they have to be placed as optimal and compact as possible. Since the diagrams will need as much space on the screen as possible, the tools must only take up a very little area of the screen. This indicates that the first of the four options is not suitable for this application. A menu bar can be annoying to use if the user has to access buttons within sub-menus or even within subsub-menus, however the menu bar allows access to many tools without taking up too much screen area. A toolbar is almost like a menu bar, except that the user does not have to open a menu to be able to select a tool. This makes a toolbar very useful for a few tools, that are frequently used. The right-click menu also allows the user to access tools very quickly, but if to many items are added to the menu, it is difficult for the user to find a specific tool.

Given the above considerations the user will be able to access all tools through a horizontal menu bar, that is placed in the top of the main window. The tools that will be used the most are also accessible through a horizontal toolbar placed just below the menu bar. The button icons on this toolbar must clearly indicate to the user what tool is selected when clicking the individual buttons.

Some tools, like when a component needs to be simulated as broken or repaired, will be gained access to in a pop-up menu by right-hand clicking on the specific component.

The structure of the model layer is based on the entities and functions identified in the analysis. As previously mentioned a part of the presentation is to display the state of every component in the model layer. This makes it obvious to reuse the structure on the model layer in the presentation layer and have it put "on top" of the model. In this way e.g. a button in the domain will be represented in the application with a button object in the model layer and a button object in the presentation layer. The model object keeps track of when the button is pushed or released and whether it is live or not, and the second object reads the state of the button in the model object and displays it on a diagram for the user to read.

The reused structure from the model layer has to be slightly modified, to fulfil the visual requirements to the application.

Two regular relays, that has got the exact same functionality in the model might need to be displayed differently on the diagram, depending on their use in the circuit. This leads to an extension of the regular relay object in the model structure. The presentation layer must have an object for each use of the relay, to be able to display and handle the associated regular relay in the correct way.

The wire object is not used in the model, but the wires need to be painted in the presentation layer. To have the wiring painted as on Banedanmark's diagram, every wire has to have its own object that knows how it should be painted.

This leads to the final structure of the presentation. A UML diagram documenting this structure can be found in appendix F.2.

The `Simulator` object runs the application and connects the projects with the available tools. Besides this all of the administrative tools – like creating, saving, loading and closing a project – are located in the object.

The `Project` object keeps as previously mentioned track of the operators panel and all of the diagrams used to display the electrical circuit. Each project knows the model's station object which the project must display. Most of the communication from the presentation layer to the model layer goes through the `Project` and `Station` objects.

All components on the diagrams represents a component in the model. The abstract `Component` object in the presentation layer has thus a reference to the component in the model that it represents.

The circuit in the model is split into a number of `CircuitSegment` objects that each represents and displays a segment of the entire electrical circuit. The abstract `Diagram` object contains all of the functions that are shared by the circuit segments and the operators panel.

4.3.2 Creating and editing components

Every component that is going to be displayed on a diagram must first be created in the model layer.

It does not matter in which order the different components are placed on the diagrams. The only restriction when adding these components is that before any contacts can be displayed; the associated relay must have been created and added to the rack of relays.

When the user wants to add a component to a diagram, he/she first selects the desired tool from either the menu bar or the toolbar. Afterwards the desired location on the diagram is pressed. If the tool requires additional information to be typed a dialog box appears. If ID's must be specified and the application automatically suggests the lowest free ID. If the user accepts this ID or another ID that is free, the model creates a new component with the specified ID. If the user specifies an ID that is used by a previously created component, the model component is reused and displayed.

As soon as a component has been added to a diagram, the add-tool is being deactivated and instead the select tool is activated. This tool can be used to select, move, connect/assemble or disconnect and remove components on the diagrams. Additionally many of the components should be able to be flipped; horizontally and/or vertically to optimize the display of the components and wiring. The following sections describes how these functions must be implemented.

4.3.2.1 Relays

Regardless of the type of relay to create (regular relay or steel core relay) the following must be specified when creating the relay. The position in the rack of relays (field, level and the left/right position in that cell), the size of the relay specified by the total number of contacts and finally how many of these contacts that are upper contacts. Optionally a name and description of the relay can be supplied.

4.3.2.2 Contacts

When creating/displaying a contact it must be specified which relay the contact is on by referring to its position in the rack of relay. Besides this it must be specified which contact on this relay to create. Optionally a description of the contact can be supplied.

4.3.2.3 Buttons

A button is identified through its ID number, so this is the only required information to apply when creating a button. Optionally a name and description of the button can be supplied.

4.3.2.4 Fuses

Each is identified through its ID number, so this is the only required information to apply when creating a fuse. Optionally a name and description of the fuse can be supplied.

4.3.2.5 Resistances

According to the domain a resistance is placed in the rack of relays and thus identified by this position. In this application we have abstracted from this and supplied a resistance with an ID number instead. This number must be provided when creating a resistance and optionally a name and description of the resistance can be supplied.

4.3.2.6 Lamps

Lamps are identified by an ID number so this must be specified by the user when creating the component. Additionally the color of the lamp must be specified. Finally a name and description of the lamp can be supplied.

4.3.2.7 Track sections

As described in the design decisions of the model in section 4.2.3.1 each track section is associated with a relay that in used to reflect whether the track section is free or occupied. This entail that when a track section is created the required information for creating the associated regular relay must be specified by the user. Besides this a name of the track section must be supplied to distinguish between the different track sections of the station.

4.3.2.8 Points

Since a point is somewhat considered a track section, and hence the necessary information for creating a track section must likewise be supplied when creating a point. Additionally two regular relays must be created (as described in section 4.2.3.2). These relays indicate whether the point is in the plus, minus or intermediate position. A name for the point must as well be supplied to distinguish between the points of the station.

To summarize, when creating a point three regular relays must be created and two names must be specified; the name of the track section part of the point and the point name.

4.3.2.9 Select and move

This is not the first application that allows graphically displayed components to be selected. For this reason we have chosen to mimic the actions of these other applications. This makes the use of the tool more intuitive. A single component can be selected by pressing the mouse button on top of it. Multiple components can be selected by pressing the SHIFT key while selecting components. Additionally multiple components can be selected by dragging a square spanning the relevant components.

The selected components are moved by pressing one of the selected components and dragging the mouse.

4.3.2.10 Connect electrical components

One of the key functions in the application is connecting the components on the diagrams with wires. When the user wants to connect two terminals, it must be specified which terminals to connect. The identification of the terminals can be performed in either of two ways: By typing the unique ID's of the terminals or by graphically selecting the terminals. The first solution requires the user to input information about the components and hence the user could easily be frustrated when creating a circuit from scratch. Most of the time would be spent typing in this information. For this reason the latter solution is chosen.

To distinguish this tool from the select and move tool the CTRL key must be pressed while the connection is made. As long as the CTRL key is pressed all terminals on the diagram are painted green or red depending on whether they accept a connection to be made or not. The left mouse button is pressed on top of one of the terminals to connect. This causes the color of some of the other terminals to change from green to red if they do not accept a connection to be made to the selected terminal. Afterwards the mouse is dragged towards the other terminal to connect to and a line is painted from the selected component to the position of the mouse cursor. The line is red until the mouse points to a terminal that accepts the connection. This causes the line to become green. When the mouse button is released the green line is replaced by a wire indicating that the terminals are connected.

4.3.2.11 Assemble track sections

Track sections on the operators panel are assembled very similar to how electrical components are connected on the diagrams. The CTRL key is pressed and the ends of the track sections are painted green or red depending on whether they can be assembled or not. The left mouse button is pressed on top of one of the ends to assemble. This causes the color of some of the other ends to change from green to red if they do not accept an assembly to be made to the selected end. When the mouse afterwards is dragged across the operators panel the pressed end is moved along. If the mouse button is released on top of an end that accepts to be assembled the two ends are being assembled and if the mouse button is released elsewhere no assembly is made.

When assembling track sections it is not allowed to press the mouse button on top of an end of a point. However it is allowed when assembling to release the mouse button on top of an end of a point. One reason for this is that it is not be allowed to connect two points and another reason is that a point cannot be stretched and bend like straight track sections.

4.3.2.12 Delete components and wires

If the user regrets having displayed certain component or having connected one or more wires it is possible to disconnect and remove them from the diagram. This is done by first selecting the desired component or components (including wires) and afterwards pressing either the DELETE key on the keyboard or the "Disconnect and remove selected components" button on the toolbar or in the "Edit" menu on the menu bar. First, if any wires are selected they are being disconnected from the components and removed from the diagram. Afterwards the wires connected to all of the selected components are disconnected and removed and finally the selected components are removed from the diagram.

When components are removed it is only in the presentation layer meaning that they are not removed from or changed in the model layer and the same components can then be displayed elsewhere when desired.

As described in section 3.1.4 this tool has been limited to only applies to the circuit segments and not to the operators panel.

4.3.3 Interacting

When the components are created and displayed on the diagrams the user can interact with them to simulate actions like pushing/releasing a button or occupy a track section to simulate that a foreign object shorts out the track section.

In the following sections the different interactions are explained.

4.3.3.1 Break and repair

As described in the analysis it should be possible to simulate some components to be broken in a certain way and later on repair the specific components. All of these functions are chosen through the menu that pops up when the right mouse button is clicked on top of a component. Only the components mentioned in section 3.1.2 can be broken and repaired.

4.3.3.2 Buttons

Buttons can be pushed and released at any time. This can be done in two ways. One is to right-click on the specific button on a circuit segment and choose the desired function in the pop-up menu. The other way is to choose the specific button in either the "Push button" menu or the "Release button" menu in the menu bar depending on the desired interaction.

4.3.3.3 Lamps

Lamps that are displayed on the circuit segments can at any time have the spare filament added. Any lamp that has got the spare filament added can have it removed. If a wire is connected to the spare filament when this is removed the wire is disconnected prior to the filament being removed. Both of these functions can be accessed from the pop-up menu displayed when the right mouse button is clicked on top of the lamp. The items to choose in the pop-up menu are "Add spare filament" and "Remove spare filament".

4.3.3.4 Track sections

A track section can at any time be occupied or freed by the user. This can be done in two ways. One way is to right-click on the specific track section and choose "Occupy track section" or "Free track section" in the pop-up menu. The other way is to choose the specific track section in either the "Occupy track section" menu or the "Release track section" menu in the menu bar depending on the desired interaction.

4.3.3.5 Points

Points can at any time be switched by the user. This can be done in two ways. One is to right-click on the specific point and choose "Switch point" in the pop-up menu. The other way is to choose the specific point in the "Switch point" menu in the menu bar.

4.3.3.6 Trains

When an entire station is created the application allows the user to simulate a train entering the station. This is done by first selecting the track section on which the train must enter the station. This function can be accessed by right-clicking the desired start track section on the operators panel and then choosing ”Start train from here” in the pop-up menu. Only if the track section is accepted as a start-track section by the model layer the function is enabled.

When a train has entered the station the train can be moved by either moving the front of the train onto the next track section or by moving the rear of the train and thus freeing the rearmost track section occupied by the train. The rear of the train can only be moved if the train occupies more than one track section or the front of the train has left the station. These functions are chosen by selecting ”Move front of train” or ”Move rear of train” either on the toolbar, in the pop-up menu displayed when right-clicking anywhere on a diagram or in the ”Interact” menu in the menu bar. If the user wants to remove the train from the station – for instance if he/she wants to restart the simulation – the train can be cancelled. This is done by choosing ”Cancel train” either in the pop-up menu displayed when right-clicking anywhere on a diagram or in the ”Interact” menu in the menu bar.

4.3.4 Display

When deciding how to display the different component either the symbols used on Banedanmark’s diagrams can be reused or an entirely new set of symbols must be created. Many aspects favor the first option. One is that is less time consuming that designing new symbols and another is that it is more user-friendly to reuse the symbols on Banedanmark’s diagrams, since the users of the application will recognize the notation. For these reasons all of the symbols, that are to be implemented in the application, will be painted as similar to the symbols on Banedanmark’s diagrams.

Is has been stated by out contact at Banedanmark that each component can at most be displayed one place at a time.

In the following sections it is described which further design choices have been made with regards to the display.

4.3.4.1 Names

The names of the different components are displayed as Banedanmark's diagrams meaning that on circuit segments the names are written next to the individual components. On the operators panel the track section names are written below the track section and for points the point name is written above the point near the branching.

4.3.4.2 Flip

Some times the diagrams will look more well-arranged if some components could be flipped either horizontally or vertically. This is by having the lowermost pin/end displayed as the uppermost one and vice versa or having the rightmost pin/end displayed as the leftmost one and vice versa. To to this the user has to right-click on top of a displayed component and from the pop-up menu select the desired function.

4.3.4.3 Wiring

On Banedanmark's diagrams the wires are painted in a way that makes it possible to see exactly which components are connected. This is explained in section 2.3.1.9. To make the presentation paint wires in about the same way as the wires on Banedanmark's diagrams, the wires have to be "intelligent". First of all a wire has to know in which direction the ends must be painted. If one end of a wire is connected to the lower part of a component on the diagram, the wire has to be painted below the component and vice versa when it is connected to the upper part of a component. This means that the wires can be painted in 3 different ways:

- Both ends are connected to the lower parts of two different components.
- Both ends are connected to the upper parts of two different components.
- One end is connected to the lower part of a component and the other end to the upper part of another component.

The first two options are fairly straight forward; the wire has to be painted in the direction away from the first component until the vertical distance to both components reaches a predefined length. When the predefined length is reached the wire bends 90 degrees towards the second component. Next the wire reaches the x-coordinate of the second component, whereafter it should bend 90 degrees towards the second component. The third way to draw a wire is a bit more difficult. The wire can be painted with 0, 2 or 4 bendings as seen on figure 4.13. When it has been determined where the wire should be painted, the bendings

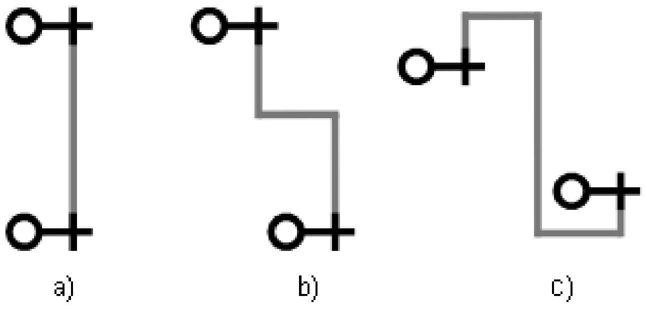


Figure 4.13: *a) a wire with no bendings, b) a wire that bends twice and c) a wire that bends four times.*

are ”cut off”, as seen on figure 4.14. To determine which one of these wire

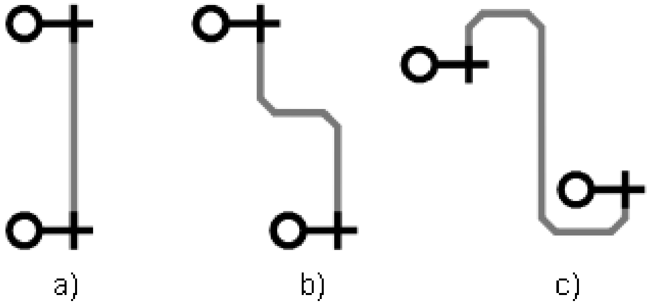


Figure 4.14: *The bendings on the wires are ”cut off” to indicate to which pin a wire is connected.*

”types” to paint, the coordinates of the connected terminals are compared. The wires on figure 4.13 are painted as they are because:

a) The x-coordinates are the same and the y-coordinate of the first terminal is above the y-coordinate of the second terminal.

a) The x-coordinates are different and the y-coordinate of the first terminal is above the y-coordinate of the second terminal.

c) The x-coordinates are different and the y-coordinate of the first terminal is below the y-coordinate of the second terminal.

4.3.4.4 Operators panel

The operators panel is among other things used to display the track sections on a station. Since the real physical track sections are aligned so the only tracks that are placed in a non-horizontal position are parts of the points. This means that it makes sense to have the operators panel displayed as a grid with horizontal lines. The track sections will snap to this grid when placed, moved or assembled.

4.3.4.5 Diagrams in general

Since a project can consist of many diagrams, it must be possible to view more than one diagram at a time. This indicates that every diagram should be shown in a separate window allowing the user to display diagrams next to each other and hide diagrams when they are not needed. The operators panel can similarly be displayed in a separate window.

4.3.4.6 Error messages

When errors of any kind occurs within the application the user should somehow be notified – at least if the error has got consequences for the continuous use of the application. The different types of errors in the application will be presented to the user as different types of messages. The most severe error that either blocks the application or cancels an operation. That is for instance if the user specifies illegal input in an input field or a file could not be loaded. The more minor errors and other information will both be interpreted as information when passed on to the user. These do not affect the use of the application. It is for instance when the user is told that the final state of the circuit has been reached, when a project has been saved or to inform the user which tool is active.

All of the above mentioned types of messages must be sent to a central place in the application. The application itself consists of two different objects: Simulator and Tools. It is most obvious to have the messages sent to the Simulator object and from there dispatch them to the user.

The messages could be passed on to the user by for instance displaying a dialog box with the message or writing the information on a label in the main window of the application. An advantage in the first option is that the user cannot miss to see the message as he/she could do if the message was written on a label somewhere in the application frame. A disadvantage is that the user might get quite frustrated and annoyed by boxes popping up all the time, if he/she has just pressed a few pixels outside a component and the application wants to inform the user that the mouse must be pressed on top of a component.

To avoid the user getting annoyed with the application all messages will be written on a label in the lower part of the main frame. Depending on the severity of the message, the color of the text is changed.

The only messages that are not displayed on the above mentioned label is the information regarding the simulation of the electrical circuit. This information is written on a label on the toolbar next to the buttons used in the simulation. The reason for having this information shown in another way than the other messages is that this information is regarding what the main purpose of the program is about and to have as much of the simulation elements grouped together.

4.3.5 Saving and Loading

The application can be used to create very large projects containing a large amount of components. To reuse projects that have been created it must be possible to save the current project and load it at another time. For the application to be able to recreate the exact same project all displayed components, their attributes, their coordinates and the connected wires must be saved. This can be done in many different ways, e.g. by serialization the objects or by saving the information in a file using character separation or a language like XML.

A very big draw bag with serialization is its compatibility between different versions. To be able to open a serialized object, the object versions must be of the exact same version as when the object was serialized. If this is not the case then the serialized object is completely useless, and for this reason serialization of the objects is not applicable for this application.

The advantage of the character separation in relation to XML is that the saved file will not take up as much disc space since the tags in an XML file are typically longer than the characters used for separation in the first option. In spite of this it has been chosen that the application is going to use XML, which is a language developed for storing data and thus easier to read and understand. This is an advantage with respect to avoiding misreadings.

4.4 Sequence Diagrams

Sequence diagrams are used to document how communication between classes in the application will take place. Sequence diagrams have been made for simulating a step, occupying a track section with a train and for cancelling the train see appendix G. The three sequence diagrams have been made for the application flows evaluated to be the most interesting.

The sequence diagram for a simulation of a step shows how methods are called in the application for the simple circuit as seen in appendix G. The sequence diagram shows the flow from when the user initiates a simulation of the next step, until the result of the simulation is displayed to the user. The sequence diagram is an example of how the model gives a message to the presentation by returning parameters, here a boolean determining that there are no further changes in the circuit.

The next sequence diagram shows the flow of the application when the user wants to move a train a track section forward. In the example which the sequence diagram is based on the track section to be occupied by the train is already occupied by something else. The sequence diagram gives an example of how the model communicates with the presentation using exceptions.

The last sequence diagram depicts a scenario where a train has occupied a single track section on a station. The method returns no parameters or exceptions since there are no special conditions of which the user should be informed.

Chapter 5

Implementation

In the following sections the implementation details will be explained. First the general issues of the application are accounted for and finally the implementation details regarding the model and the presentation will be explained.

5.1 General issues

In the design fase it was decided that an object-oriented programming language was going to be used and the general structure of the application was determined. To be able to implement the chosen design choices a suitable programming language has to be adopted. The most obvious choice would be either Java or C++, as they currently are the most used object-oriented programming languages [2]. During our undergraduate days we have used Java in many projects. Through the knowledge we have gained in these projects we foresee that Java is applicable for solving the requirements for the application. For this reason and that we do not have to spend time learning a new programming language Java is chosen as the programming language for the application.

In the design phase the domain of the project were translated to an outline of how the requirements to the application were to be solved. The implementation phase is now concerned with how the design is to be implemented given the chosen programming language Java. The implementation process has been time-consuming due to the amount of components and the required functionality but translating the design to code has in general been fairly straightforward, and for this reason the implementation section only contains few detailed descriptions of how design is translated to code.

The code of the application is documented with Javadoc which can be seen on the CD handed in with the thesis.

The user's guide of the application is found in appendix K.

5.2 Model

In the design section it was described that the graph structure would be achieved with an adjency list. The adjency list parameter is placed in the `Terminal` class since both poles and pins have an amount of connected terminals. Pins can have three connections at the most (i.e. to the other pin on the contact and to two wires) and poles can have an unlimited amount of connections. Since the adjency list is placed in the `Terminal` class a flexible data structure is chosen to support an indefinite amount of connections to the poles, even though pins only have three connections at the most. The adjency list is thus implemented with an arraylist of terminals `ArrayList<Terminal>`.

5.3 Presentation

The following sections explain how the design choices have been implemented in the presentation layer.

Appendix J depicts how the menubar and tool bar are displayed in the application. Likewise it can be seen how connected electrical components on a diagram and assembled track sections on the operators panel are displayed in the application.

5.3.1 GUI library

When using Java as a programming language 3 different GUI libraries can be used; *AWT*, *Swing* or *SWT*. AWT was the first library to be developed for Java. The performance of AWT and the amount of features provided by AWT are quite poor. The successor GUI library is the Swing library, which has improved performance and functionality. Swing allows the user to control the position of different components in a layout better. The SWT library offers a few more features than Swing and it is said to have a better performance. When weighing this up against the time it would take us to be able to use SWT we decided to use the Swing library. Partly because we know how to use Swing and it seems to be suitable for implementing the requested features of the application.

5.3.2 Singleton pattern

Two classes in the presentation layer (`Simulator` and `Tools`) are used to coordinate the selection and use of the different tools. Because of this it would have many unforeseen consequences if the application somehow was to create more than one instance of both of these classes. To avoid this the *Singleton pattern* is applied on the classes. In both classes the constructor is made private and instead a public static method must be used by external classes to create/get an instance of the class. Each class has got one global field which contains an instance of the class and this same instance is returned in the previous mentioned public static method.

5.3.3 Tools

The `Tools` class is composed of a number of internal classes. Most of these classes are different tools; one class per tool. The reason for this is that most of them only are initiated when the mouse button has been pressed at a certain time and they should not be accessible from other classes. The internal classes work as mouse listeners which allows the individual tools to be added to the diagrams when the tools must be active and removed when they must be inactive.

5.3.4 Frames

The main class of the application, the `Simulator` class, extends `JFrame`, and the abstract class `Diagram`, that is the super class of the `CircuitSegmentView` and `OperatorsPanelView` classes, extends `JInternalFrame`. Every diagram created within the application is added to the desktop pane of the outermost frame. JInternalFrames have been chosen because it – as described in section 4.3.1 – is convenient for the user to be able to look at more than one diagram at the a time. Each of these internal frames can then independently of each other be iconified, minimized, maximized, hidden or restored by the user. When one or more internal frames have been hidden, they can be restored through the "Window" menu. It is not possible through the GUI to dispose any of the internal frames.

5.3.5 Save/load

In the design of the application it was decided to use XML when saving projects. The major advantage using XML is that XML is implemented in Java.

First a schema file is designed by creating en XML-element for each object that needs to be saved. Each of these elements likewise needs a set of attributes corresponding to how much information in the objects that must be saved. This schema is compiled using Java's JAXB compiler (xjc) [3]. The compiler generates a set of java classes – one class for each element in the schema – that can be instantiated and used directly inside the application source code.

The XML schema (see appendix M.3.1) is created to copying the structure of the application. This means that for every object in the application that needs to be saved is created an element in the schema and the relations between these elements are like between the associated objects in the application.

The save function is implemented by for each application object, that needs to be saved, instantiating a corresponding XML-object and copying the necessary information from the application object into the XML-object. When every object is copied Java's own marshal function is used to save this object into a file. The way this function works is to read the XML-schema and then translate the XML-objects into a single XML-file following the rules set in the schema.

XML files usually grow quite large due to the fact that all information in the file is surrounded by start and end tags. Though this also makes it possible to compress the file with a very large ratio – a test XML-file of 206kB is compressed to

5,5kB which gives an approx. 37:1 compression ratio. This compression feature is very well implemented in Java and is suitable for loading and saving XML files through Java. For these reasons the projects saved in the application will be saved as XML files within a compressed zip file.

To make it possible for the user to distinguish the saved files from other files on the computer a the *.ris* extension (Relay Interlocking System) is used. Several examples of a saved simulation object can be found on the CD.

The load function is implemented in the exact opposite sequences; an XML-file is unmarshaled using the schema into a XML-objects and afterwards copying the information in these objects into the objects in the application.

Chapter 6

Test

6.1 Overall test strategy

When an application is developed for simulation purposes, a thorough test becomes especially important. If a simulation is not conducted properly the application becomes completely useless.

Tests have been performed on the model and presentation throughout the project and thus defects have been corrected continuously. The tests in this section is concerned with the test performed on the final version of the application, i.e. tests performed after the implementation phase is completed.

It is a well known rule of thumb that defects in an application is cheaper to correct the sooner they are discovered in a development process. Most often this guideline is used for development projects that stretches over several years and the rule is followed to take out financial costs. The rule can still be applied in this project though since the time spent can be reduced if defects are identified as soon as possible.

The test strategy chosen for the application combines structural and functional testing with emphasis on the tests performed on the model.

In the following sections the specific test strategies for the model and presentation are described.

6.2 Model

Tests should be planned and executed as early as possible in a development project. It is definitely an advantage to automate tests so they can be performed as often as needed. JUnit has been used to automate tests in the model. For each method in every class in the model a structural test has been performed. This means all branches in the code has been tested to assure that the code does what it is expected to do.

The model has also been tested with functional tests. Functional tests test that the model works as expected in relation to the domain. Brief descriptions of the tests are listed in table 6.1 and 6.2, the details can be seen in appendix M.5.

The full version of each structural and functional test cases is on the CD.

In the following sections the general test principles for the model are explained, whereafter test results for the different test areas are examined.

6.2.1 General test principles

For each class a structural test class has been created, which tests each of the methods in a class. In general it should be considered how abstract and non-abstract classes are tested, and how private and protected inherited fields and methods are tested. The test principles used for the model will be discussed in the following sections.

6.2.1.1 Test of Abstract Classes

Abstract classes are not tested directly, since they cannot be instantiated. Therefore the abstract classes will be tested indirectly by testing a child of the abstract class. If the child is also abstract the child of this abstract class will be tested and so forth.

#	Description	Expected
1	The signal control circuit from Stenstrup station is tested in relation to the step simulation. 11 steps have been tested. For each step 36 assertions have been made.	For each step all components should be in the expected state.
2	The locking of train route circuit from Stenstrup station is tested in relation to the step simulation. 11 steps have been tested. For each step 36 assertions have been made.	For each step all components should be in the expected state.
3	The release of train route circuit from Stenstrup station is tested in relation to the step simulation. 15 steps have been tested. For each step 41 assertions have been made.	For each step all components should be in the expected state.
4	The lamp circuit from Stenstrup station is tested in relation to the step simulation. 10 steps have been tested. For each step 59 assertions have been made.	For each step all components should be in the expected state.
5	Current is applied simultaneously to both coilings on a steel core relay.	An exception is thrown.
6	A full simulation has been tested on a cyclic circuit.	The model should detect that the circuit is cyclic.
7	Embedded cycles has been created in a circuit in relation to the step simulation.	The model does not stall.
8	Relays are connected in a series connection in relation to the step simulation.	The relays change state in the same step.

Table 6.1: *Functional test cases*

#	Description	Expected
9	Relays are connected in a parallel connection in relation to the step simulation.	The relays change state in the same step.
10	Various tests concerning a button connected in a circuit: one button, several buttons and buttons blocking current from propagating to relays.	The button works as expected.
11	Various tests concerning a steel core relay connected in a circuit applying current to the coilings in different sequences.	The steel core relay works as expected.
12	Various tests breaking different components.	The components are broken and not affected by the circuit as usual.
13	Various tests concerning a lamp connected in a circuit: adding spare filament, removing spare filament, breaking spare filament, testing maximum number of connections to lamp pins.	The lamp works as expected.
14	Various tests on train simulation: "short" train, "long" train, points not switched properly, freeing all track sections in the middle of the station.	The train simulation works as expected.

Table 6.2: *Functional test cases*

6.2.1.2 Test of Non-abstract Classes

When a class inherits parameters and methods from another class it is only the functionality of a specific class that is tested, not the inherited functionality. If there e.g. are three classes A, B and C with the following relationship:

A
B extends A
C extends B

the test of C, will only test the functionality specific for C. The reason for this is that we assume when A is tested, the functionality of A in B and C is also tested.

The test of A, B and C will be as follows:
Test(A) = $\{A\}$
Test(B) = $\{B \cap^{\neg} A\}$
Test(C) = $\{C \cap^{\neg} B \ \cap^{\neg} A\}$
where $\{X\}$ is test of the functions of the class X.

The inherited methods are tested as an indirect test of the abstract classes. Parameters and methods unique for e.g. the `Button` class are thus testes in StructuralTest/ButtonTest but the class can also be used in StructuralTest/PinGroupTest to test the methods of pin group since `Button` extends the abstract `PinGroup` class.

6.2.1.3 Test of Real Case Scenarios

The functional test tests sub circuits from an interlocking system implemented at Stenstrup station. In this way the domain test covers not only theoretical fictitious examples but also real life scenarios.

6.2.1.4 Test of Private Methods

Since private methods are not accessible a decision has to be made c.f. [4] as to whether the methods should not be tested or the access control mechanism should be subverted so private methods can be accessed. The latter solution is chosen to cover test of private methods an parameters.

A class `Access` has been created to access private methods and parameters similar to [5]. The private methods are accessed by getting a declared list of methods in relation to a class and changing the accessibility with `method.setAccessible(true)`. Additional functionality is added to `Access` so the class of any exceptions that may be thrown is returned, such that validation of an exception type can be performed.

6.2.1.5 Test of protected methods

Protected parameters and methods can be accessed by placing the test classes in the same package as the protected parameters and methods c.f. [4]. Since an `Access` class has been created to access the private parameters and methods, the class is used to access the protected methods as well. Functionality is added to the `Access` class such that inherited methods can be accessed as well.

6.2.2 Results

The automated JUnit tests have been used frequently in the development of the model. Additional functional tests an advantageously be applied since performing the tests has no cost. Throughout the project defects have been identified an corrected, and in the current version of the model no known defects exist.

6.3 Presentation

As previously mentioned emphasis is on test of the model to correct defects as early as possible. This allows less time to test the presentation and for this reason it is only a subset of the functionality that is tested through functional test.

13 test cases have been created to test the main functionality, see appendix H.1. Besides testing the basic functionality a complex circuit from an actual station has been tested as seen in appendix H.1.16. In this test all domain areas of the application is tested i.e. the circuit, the operators panel and the train. 29 simulation steps are performed and for each step the states of 31 relays are validated. The validation of the relays are made in relation to a functional diagram as seen in appendix H.1.16. The functional diagram is a result of an analysis performed on the diagrams of Stenstrup station.

The operations in the application are somewhat simple and quick to execute, and for this reason the documenting phase is the most time-consuming phase. To avoid spending too much time on documentation of trivial test cases exploratory testing is used [6]. In exploratory testing the operations to be tested are decided by the testers impulses. The strength of this test method is that a creative tester might find errors that would not have been found in planned test cases, while the disadvantage is that the quality of the test depends on the individual tester.

The exploratory test is performed in one project. This way of testing also validates that the operations function independently of previous actions performed in the application. 37 test scenarios have been tested through exploratory testing and are documented briefly in appendix H.2.

6.3.1 Results

2 defects were discovered during the exploratory test.

The first defect is related to exploratory test case #15 as seen in appendix H.2. The largest number allowed by a the GUI was entered as an ID of a component. The error message was "The ID must be a positive integer". The error message has been change to "The ID is not valid". If focus were on usability or the application were to be used by personnel at Banedanmark the error messages should be more precise and user-friendly though.

The second defect is related to exploratory test case #19. The lamp component were not repaired when the circuit was reset. The defect has been corrected.

The 13 test cases did not result in any defects and the defects discovered during the exploratory test have been corrected. For this reason no known defects exist in the presentation.

6.4 Conclusion

A thorough structural and functional test on the model combined with a functional test of the application gives in our estimate a very good test coverage. Especially the successful test of Stenstrup station support the functionality of the application.

JUnit has been a very effective tool for testing since the automated tests decreases the time spent executing tests and increases the times the tests are performed at the same time. If the application were to be developed over a longer period of time a capture and replay tool could have been used to automate the test on the GUI as well.

Correcting the two defects detected in the exploratory test results in a final version of the application with no known defects.

Chapter 7

Extensions

Even though only a few entities and functions were delimited in relation to the domain, there are many additional features that can be implemented in the application. One reason for this is that automation of the propagation of current makes is possible to perform different types of analysis and fault location, that would not be possible with the paper diagrams.

In the following 5 extensions that would improve the functionality of the application will be described. Besides the 5 main extensions a list of extensions that would improve the usability of the application are listed in appendix I.

- Internal resistance in the electrical components were delimited in the analysis. In consequence some of Banedanmark's paper diagrams will not function correctly when translated into the application diagrams. Implementing internal resistance in the electrical components will thus allow a more precise translation from paper diagrams to application.
- The procedure for creating a new station using an interlocking system starts at an abstract level i.e. with train route tables and finishes at the concrete circuit level. Validation can be performed on the abstract as well as the concrete level. Being able to frequently validate the abstract train route table in relation to the associated circuit would save time and resources, and prevent errors.

Validating a circuit in relation to a train route table could provide the following validations:

- Do all signals show what they are expected to before the train route is locked?
- Are all track sections in a train route free when the relevant entry signal is green?
- Does the entry signal turn red at the right time?
- Is the start of the release of the train route initiated at the right time?
- Is the release of the train route completed at the right time?
- Is it impossible to lock all conflicting train routes while the current train route is locked?

A useful addition to the current version of the application would thus be a function validating the abstract level.

- Validation on the concrete circuit level would be an advantage as well. As the level of complexity for a circuit grows, so does the number of diagrams. In the current version relay states are validated by flipping through the diagrams of a station for each step in the simulation. This validation could be eased by having the user enter which relays in theory should change state for each step, and then letting the application perform the validation.

  The need for this validation function becomes obvious even for small station as e.g. Stenstrup station. In section 6 a test of Stenstrup station was performed. The test was executed over 29 steps and the circuit contained 31 relays. That leaves 899 validations and the test only covered one out of 8 train routes.

- When a circuit is created the behaviour of the circuit should be deterministic. This means that when certain conditions in a circuit change the final state of the circuit should be predictable. It is possible though to construct circuits that does not always have the same final state.

  If two relays e.g. A and B change state in about the same time in the real world there should be no difference in the state of the circuit if A changes before B or vice versa. An example of an illegal circuit can be seen in figure 7.1. Two functional diagrams are made as seen in table 7.1 and 7.2; a diagram for when A changes before B and a diagram for when B changes before A. The functional diagrams show two different final states of the circuit and thus the circuit is illegal. The step design used in the application assumes current propagates to relays at the exact same time so errors of this type are not detected.

  If relay 1 changes state first, relay 3 will change state. If relay 2 changes

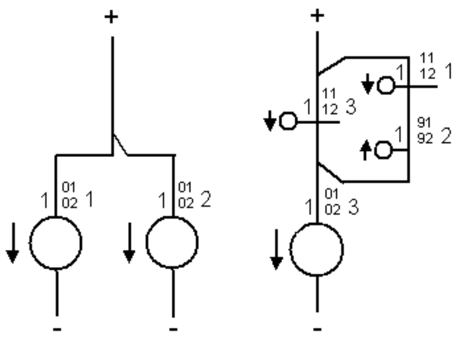


Figure 7.1: *Illegal circuit*

Step	{1,1,l}	{1,2,l}	{1,3,l}
1	↓	↓	↓
2	↑	↓	↓
3	↑	↑	↑

Table 7.1: If relay {1,1,l} changes state first

Step	{1,1,l}	{1,2,l}	{1,3,l}
1	↓	↓	↓
2	↓	↑	↓
3	↑	↑	↓

Table 7.2: If relay {1,2,l} changes state first

state first, relay 3 will *not* change state.

It would be an advantage if a circuit could be investigated for this type of error so a potential error is discovered before the interlocking system is implemented on a station.

- When a station contains several relays it is difficult to maintain an overview of the created relays. Implementing a rack of relays would provide this overview, and furthermore wire connections between relays could be displayed. The advantage of displaying wire connections is to allow the user to optimize the location of the relays with regards to the wiring. A further extension might optimize the location of the relays as well.

Chapter 8

Conclusion

The objective of this thesis was to develop an application for analysis of interlocking systems. The requirements for the application were to simulate propagation of current in a circuit in two different ways: displaying the final static state of a circuit, and displaying intermediate steps in the propagation of current.

The application has been developed and it successfully simulates propagation of current in an electrical circuit in the two required ways. The least amount of electrical component types required for simulating an interlocking system are buttons and the core electrical component, i.e. relays. In the final version of the application lamps, fuses and resistors have additionally been implemented, to achieve a better result when paper diagrams are translated into application diagrams.

Besides simulating propagation of current in a circuit another important part in an interlocking system has additionally been developed i.e. the operators panel. In the design section various solutions were considered in relation to the interpretation of the connection between track sections and interface relays. The chosen design resulted in the creation of an operators panel, where the states of track sections and points are displayed. Through the operators panel track sections can be occupied and freed manually to simulate the track section being shorted out by a foreign object; e.g. by a utility pole. Track sections can also be occupied and freed by a train. The user is able to start the train from a specific track section, move the train forward and cancel the train if necessary.

Another addition made to the application is to simulate broken components. This is useful for evaluating whether additional precautionary measures should be taken for certain components.

The functionality of the application is, on basis of a structural and functional test of the model combined with a functional test of the application, determined to be error-free.

Required and additional functionality has thus been implemented successfully in the application and will ease an otherwise cumbersome analysis of an interlocking system. Personnel from Banedanmark can continuously alter the circuit and instantly see the effects of any changes. A circuit can be "debugged" by simulating the propagation of current step by step, or the full simulation can be used to check that a final state of a circuit is as expected.

Implementing the extensions described in section 7 will provide a powerful tool. If the application were to be used by Banedanmark it would definitely make analysis of interlocking systems easier, faster and less error-prone. Before a station is put into service an isolated test of the implemented interlocking system is performed. The application cannot replace the test in the real world but specific types of errors can be found before the real world test and thus save resources and time.

Bibliography

[1] Cormen et al. *Introduction to algorithms.* The MIT press 2nd edition 2003.

[2] Tiobe. *Tiobe Programming Community Index.*

[3] Inc. Sun Microsystems. *Java Architecture for XML Binding* 2007.

[4] JUnit.org.

[5] Ross Burton. *Subverting Java Access Protection for Unit Testing* 2003.

[6] Poul Staal Vinje. *Softwaretest - Teknik, Struktur, Metode.* Nyt Teknisk Forlag 2nd edition 2005.

Appendix A

Terminology

The terminology describes how the meaning of a word should be concieved in relation to the thesis if the word is ambiguous.

Application Application refers to the program that is developed in the thesis for simulation of current propagation.

Coil pins Coil pins are the pins on a relay that are connected to the coiling in the relay. The pins have the ability to change the state of the relay. The term covers both regular coil pins and steel core pins.

Conflicting train route A train route is conflicting in relation to another train route if the fact that both train routes are used might lead to a collision.

Contact Contact is the common term for upper and lower contacts on a relay.

Closed A contact is said to be closed when the current can pass from one of the pins in the contact to the other.

Open A contact is said to be open when the current cannot pass from one of the pins in the contact to the other.

Diagram A diagram is the form of notation currently used by Banedanmark to document a subset of a circuit.

Live An electrical component is live if it carries current.

Minus direction The term covers two different scenarios. In old terms this is the least straight track through a station, and in new terms it is the left track in relation to the junction on the point. In this thesis the new understanding of the term is used.

Normal state The normal state of a station, is the scenario where current is applied to the system, all points are switched to the plus direction, there are no trains at the station and no train route is locked.

Pin group A pin group is a collection of pins. Whether current can pass from one pin in the pin group to another depends on the state of the pin group.

Plus direction The term covers two different scenarios. In old terms this is the most straight track through a station, and in new terms it is the right track in relation to the junction on the point.

State A state of a component defines whether the component is conductive, live and/or drawn/magnetized and dropped/demagnetized.

User The user is the potential user of the application i.e. personnel from Banedanmark.

Appendix B

Dictionary

English	Danish
Armature	Anker
Aux relay for releasing train routes*	Hjælperelæ for togvejsopløsning
Axle	Hjulaksel
Branch	Forgrening
Circuit	Kredsløb / strømløb
Closed (contact)	Sluttet (kontakt)
Coil	Spole
Coiling	Vikling
Conductive	Elektrisk ledende
Conflicting train route*	Fjendtlig togvej
Current	Strøm
Derailing	Afsporing
Drawn (relay)	Trukket (relæ)
Dropped (relay)	Faldet (relæ)
Entry button	Indkørselsknap
First relay in release of train route*	Indledningsrelæ
Fuse	Sikring
Heat-trated	Varmebehandlet
Interlocking plan*	Sikringsplan
Interlocking system	Sikringsanlæg
Internal resistance	Modstand
Locked train route*	Fastlagt togvej
Locking of train route*	Togvejsfastlægning
Negative pole	Minuspol
Non-conducting	Ikke-ledende
Normal state*	Normalstilling
Open (contact)	Brudt (kontakt)
Operators panel*	Sportavle
Over-current	Overstrøm
Pin	Klemme
Point	Sporskifte
Point detection relay	Sporskiftekontrolrelæ
Point- and signal key relay*	Sporskifte- og signalnøglerelæ
Positive pole	Pluspol
Rack of relays	Relæstativ
Relay	Relæ
Release of train route*	Togvejsopløsning
Replicate relay	Repeterrelæ
Replicate relay for track relay	Repeterrelæ for sporrelæ
Resistor	Modstand

English	Danish
Signal control circuit	Signalstyrestrømløb
Signal control relay	Signalstyrerelæ
Signal lamp relay	Lampekontrolrelæ
State of relay	Relæets tilstand
Steel core relay	Stålkernerelæ
Terminal	Tilslutningsklemme
Track relay	Sporrelæ
Track section	Sporisolation
Train route	Togvej
Train route locking relay	Togvejsspærrerelæ
Train route release relay*	Opløsningsrelæ
Train route table	Togvejstabel

* Translations provided by Kirsten Mark Hansen.

Appendix C

Domain Specifications

The domain specifications are detailed descriptions of the DSB-1954 interlocking system implemented in the Danish railway network by Banedanmark.

The purpose of a domain specification is to give a compact description of a component including details that would cloud the overall domain description.

Each domain specification has two areas that should be described:

Description . The section gives a compact description of the functionality of the component.

Attributes . The section describes each attribute associated with the component. To understand the functionality of the component the purpose of each attribute must be understood.

Exception . The section describes scenarios that are makes a component behave in an unwanted way.

A word is emphasized in the descriptions if the word is described in another domain description.

C.1 Button

C.1.1 Description

Buttons are placed on an operators panel, and are at the same time connected in a circuit. On the operators panel a button can be one of five different types:

entry buttons indicating at which track section a train is going to enter a station,

exit buttons indicating at which track section a train is going to exit a station,

plus buttons indicating that the specific point should be switched to the plus direction,

minus buttons indicating that the specific point should be switched to the minus direction,

track buttons indicating the destination track for the train.

The functionality of the buttons are the same, but the type of the button defines its purpose.

A button can be in one of two states, *pushed* or *released*. A button comprises three pins and a piece of conductive metal. The metal is attached to one of the pins, and the other end switches between the two remaining pins. The switch of the metal piece depends on whether the button is pushed or released. When the metal is in contact with two pins at a time, the metal piece forms a connection from one pin to the other, and current can thus pass from one pin to the other.

A button is uniquely defined by its x- and y-coordinates on the operators panel. When the button is initially created it does not carry current and is thus in the released state.

C.1.2 Attributes

x co-ordinate Specifies the x-coordinate of the button.

y co-ordinate Specifies the y-coordinate of the button.

Type Specifies the type of the button. The types are: entry, exit, plus, minus and track buttons.

C.1.3 Exceptions

None.

C.2 Circuit

C.2.1 Description

A circuit is a collection of electrical components connected with wires. The components connected in the circuit can be in different states and hence the circuit can be in numerous states.

A circuit is depicted on diagrams in its *normal state*. The normal state is when current is applied to the system, all points are switched to the plus direction, there are no trains at the station, and no train route has been locked.

C.2.2 Attributes

Power source Specifies the power source of the circuit.

Relays Specifies the relays of the circuit.

Wires Specifies the wires of the circuit.

Buttons Specifies the buttons of the circuit.

Lamps Specifies the lamps of the circuit.

Fuses Specifies the fuses of the circuit.

Resistors Specifies the resistors of the circuit.

C.2.3 Exceptions

The scenarios that will not provide a valid circuit are:

- The circuit is connected in a way that makes the circuit a cyclic circuit.

- The circuit is not deterministic. If two relays change state at almost the same time and the state of the circuit when relay 1 changes first is not equal to the state of the circuit when relay 2 changes state first.

- The poles are connected.

C.3 Contact

C.3.1 Description

A contact consists of two pins and a piece of conductive metal. A contact can be in one of two states, *open* or *closed*. When the contact is closed current can pass from one pin to the other. When the contact is open current cannot pass from one pin to the other. The state of a contact depends on the state of the *relay* it belongs to. The contact can either be an upper contact or a lower contact on a relay. If it is an upper contact the contact will be closed when the relay is drawn and open when the relay is dropped. If it is an lower contact the contact will be open when the relay is drawn and closed when the relay is dropped.

The contact is uniquely identified by the ID of the relay (level, field and whether it is to the left or right) in addition with the number of the contact.

C.3.2 Attributes

Contact number The contact number combined with the ID of the relay it belongs to uniquely identifies a contact. In figure 2.4 page 7 the contact number is the leftmost number of the two-digit number. Thus the coil pins are contact number 0, the uppermost contact is contact number 1 and so forth.

Field Specifies the field in the rack of relays where the associated relay is placed. The range is $[1 \cdots 1000]$.

Level Specifies the level in the rack of relays where the associated relay is placed. The range is $[0 \cdots 5]$.

Location When the field and level are defined, the associated relay can be placed at a certain location. This location can be to the left or to the right.

Name Optional. Specifies an expressive name of the associated relay.

Pin 1 One of the pins in the contact.

Pin 2 One of the pins in the contact.

C.3.3 Exceptions

None.

C.4 Fuse

C.4.1 Description

A fuse is a metal wire connected to two pins. A fuse is initially intact and hence conductive, but if the metal wire in the fuse is damaged it becomes non-conducting. There are 6 types of fuses in Banedanmark's circuits: central appliance fuse, silized fuse, glass pipe fuse, peak fuse, thermal fuse and circuit breaker.

A fuse is uniquely identified by an integer.

C.4.2 Attributes

ID A positive integer uniquely identifying the fuse.

Ampere The amount of ampere in the fuse.

Fuse type The type of the fuse. The fuse types are: central appliance fuse, silized fuse, glass pipe fuse, peak fuse, thermal fuse, circuit breaker.

C.4.3 Exceptions

None.

C.5 Lamp

C.5.1 Description

A lamp consists of one or two filaments; a regular filament and a spar filament. If the lamp only consists of one filament the filament is connected to a pin in each end. A spare filament can be added to the lamp, one end of the filament connected to an existing pin, and the other end connected to an additional pin. All three pins can beside the connections to the filaments have one additional connection. A lamp is initially conductive, but if the filament burns out the filament is non-conducting.

C.5.2 Attributes

Filament name Optional. Typically specifies the color of the initial filament.

Spare filament name Optional. Typically specifies the color of the regular filament and the letters ”rs”, and abbreviation for ”reserve”, the Danish word for ”spare”.

C.5.3 Exceptions

None.

C.6 Operators panel

C.6.1 Description

For each station there is an operators panel. The panel is the operators tool to communicate with the interlocking system. The operators panel allows interaction through *buttons*, and furthermore the operators panel displays to which direction the points are switched.

C.6.2 Attributes

Buttons A number of buttons.

Track sections A number of track sections.

Points A number of points.

C.6.3 Exceptions

None.

C.7 Pin

C.7.1 Description

A pin is an electrical conductive component. If nothing else is specified at the most two wires can be connected to a pin (the pins used for a lamp can e.g. not have more than one wire connected). A pin is uniquely identified by a positive integer.

C.7.2 Attributes

ID A positive integer uniquely identifying the pin. The pin number combined with the ID of the contact it belongs to, and the ID of the relay it belongs to uniquely identifies a pin. In figure 2.4 page 7 the pin number is the rightmost number of the two-digit number. Thus the coil pins on the right relay in the figure have pin numbers 3 and 4.

C.7.3 Exceptions

None.

C.8 Point

C.8.1 Description

A point is an special type of a *track section*. Besides having the same functionality as a track section, the point has a branching which allows the point to be in one of three states: switched to the plus direction, switched to the minus direction or switched to an intermediate state. A point has a steel core relay associated with the plus and minus direction respectively. The relays change the state of the point, if e.g. the relay associated with the minus direction is magnetized, the point is switched to the minus direction. Initially the point is switched to the plus direction.

A point can uniquely be identified by two ID's: a track section ID or a point ID.

C.8.2 Attributes

Track section ID An annotation uniquely identifying the point.

Point ID An annotation uniquely identifying the point.

Regular relay Specifies the regular relay associated with the track section.

Steel core relay for minus direction Specifies the steel core relay associated with the minus direction.

Steel core relay for plus direction Specifies the steel core relay associated with the plus direction.

C.8.3 Exceptions

None.

C.9 Power source

C.9.1 Description

An interlocking system cannot function without a power source. There is no defined limit on how many wires can be connected to the power source.

C.9.2 Attributes

None.

C.9.3 Exceptions

None.

C.10 Rack of relays

C.10.1 Description

A rack of relays is a frame structure designed to store the relays for a specific station. The rack is divided into rows and columns called levels and fields. Depending on the size of a station, the number of fields in the rack of relays will vary, the number of levels is most often 6. The levels are numbered from 0 to 5.

When a level and a field has been defined there are room for two relays. The placement of a relay is thus defined by specifying a level, a field and whether the relay should be placed to the left or right, as seen on figure C.1.

C.10.2 Attributes

Number of fields A positive integer defining the number of fields in the specific rack of relays. There can be 1000 fields at the most.

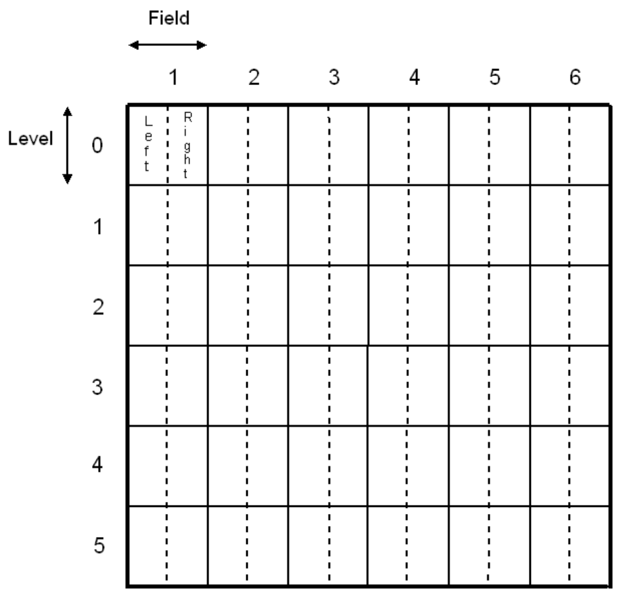


Figure C.1: *The position of a relay is specified by level, field and a position.*

Number of levels A positive integer defining the number of levels in the specific rack of relays. There can be 6 fields at the most with numbers from 0 to 5.

C.10.3 Exceptions

None.

C.11 Regular relay

C.11.1 Description

A regular relay is an electrical component that has the ability to carry current. The relay consists of a number of contacts, and two pins connected to the coil. The pins connected to the coil can change the state of the relay. These pins will be referred to as *coil pins*. The relay can be in two different states, i.e.

the relay can be *drawn* or *dropped*. When the coil pins carries current the relay is drawn, and when the pins connected to the coil does not carry current, the relay is dropped. A certain amount of the contacts on the relay are called *upper contacts*, and the rest of the contacts are called *lower contacts*. When the relay is drawn the upper contacts are able to carry current, and the lower contacts cannot carry current. When the relay is dropped the lower contacts are able to carry current, and the upper contacts cannot carry current.

A relay is uniquely identified by a position in the *rack of relays* defined by a set of attributes, i.e. {field, level, location}. When a relay is initially created it does not carry current, and the relay should thus be dropped.

C.11.2 Attributes

Field Specifies the field in the rack of relays where the relay is placed. The range is $[1 \cdots 1000]$.

Level Specifies the level in the rack of relays where the relay is placed. The range is $[0 \cdots 5]$.

Location When the field and level are defined, the relay can be placed at a certain location. This location can be to the left or to the right.

Number of contacts Specifies the number of contacts on the relay. The range is $\{6, 10, 20\}$.

Number of upper contacts Specifies how many of the contacts should be upper contacts. The range is $[0 \cdots 20]$, although is must apply that the number of upper contacts is less than or equal to the number of contacts.

Name Optional. Specifies an expressive name of the relay.

C.11.3 Exceptions

The scenarios that will not provide a valid relay are:

- The location specified by {field, level, location} is not available in the *rack of relays*.

C.12 Resistor

C.12.1 Description

A resistor is a component connected to two pins. A resistor is initially intact and hence conductive, but if the resistor is damaged it becomes non-conducting. A resistor is uniquely identified by a position in the *rack of relays* defined by a set of attributes, i.e. {field, level, location}.

C.12.2 Attributes

Field Specifies the field in the rack of relays where the resistor is placed. The range is $[1 \cdots 1000]$.

Level Specifies the level in the rack of relays where the resistor is placed. The range is $[0 \cdots 5]$.

Location When the field and level are defined, the resistor can be placed at a certain location. This location can be to the left or to the right.

Internal resistance Specifies the amount of internal resistance in the resistor.

C.12.3 Exceptions

The scenarios that will not provide a valid resistor are:

- The location specified by {field, level, location} is not available in the *rack of relays.*

.

C.13 Signal

C.13.1 Description

A signals is a set of lamps. There are different types of signals placed at the station to signal to the driver of the train.

C.13.2 Attributes

Lamps The number of lamps associated with the signal.

Signal type Specifies the type of the signal. The main signal types are: entry signal, signal in advanced position and dwarf signals.

C.13.3 Exceptions

None.

C.14 Steel core relay

C.14.1 Description

A steel core relay is an electrical component that has the ability to carry current. The relay consists of a number of contacts, and four pins connected to the coil. These four coil pins can change the state of the relay. The relay can be in two different states, i.e. the relay can be *magnetized* or *demagnetized*.

There are two coilings on a steel core relay. A coiling has two ends. Pin 12/14 is connected to one end on coiling 1, and pin 02/04 is connected to the other end on coiling 1. Pin 01/03 is connected to one end on coiling 2, and pin 02/04 is connected to the other end on coiling 2. When coiling 2 carries current the relay is magnetized, and when coiling 1 carries current, the relay is demagnetized. When the relay is magnetized and current is removed from the relay the relay maintains it magnetized state due to the heat-treated steel core.

A certain amount of the contacts on the relay are called *upper contacts*, and the rest of the contacts are called *lower contacts*. When the relay is magnetized the upper contacts are able to carry current, and the lower contacts cannot carry current. When the relay is demagnetized the lower contacts are able to carry current, and the upper contacts cannot carry current.

A relay is uniquely identified by a placement in the *Rack of relays* defined by a set of attributes, i.e. {field, level, position}. When the relay is initially created, it has been heat-treated and current has magnetized the core, such that the relay is initially magnetized.

C.14.2 Attributes

Field Specifies the field in the rack of relays where the relay is placed. The range is $[1 \cdots 1000]$.

Level Specifies the level in the rack of relays where the relay is placed. The range is $[0 \cdots 5]$.

Position When the field and level are defined, the relay can be placed at a certain position. This position can be to the left or to the right.

Number of contacts Specifies the number of contacts on the relay. The range is $\{5, 9, 19\}$, since there are 4 pins connected to the coil.

Number of upper contacts Specifies how many of the contacts should be upper contacts. The range is $[0 \cdots 19]$, although is must apply that the number of upper contacts is less than or equal to the number of contacts.

Name Optional. Specifies an expressive name of the relay.

C.14.3 Exceptions

The scenarios that will not provide a valid steel core relay are:

- The location specified by {field, level, location} is not available in the *rack of relays*.
- Current is applied at both coilings at the same time.

.

C.15 Track section

C.15.1 Description

A track section is an isolated piece of a track. The isolation allows each track section to carry current independent of other track sections. A track section is connected electrically with a regular relay that reflects the state of the track section. When the track section is free the relay is drawn, and when the track section is occupied the track section is dropped. A track section is uniquely identified by an annotation.

C.15.2 Attributes

ID An annotation uniquely identifying the track section.

Regular relay Specifies the regular relay electrically connected with the track section.

C.15.3 Exceptions

None.

C.16 Train

C.16.1 Description

Trains have varying lengths depending on the number of wagons attached to the train. For this reason the number of track sections that a train occupies are varying from train to train.

When a train enters a station, it typically makes a stop at a platform before exiting the station. If the train is not scheduled to make a stop at a station the train drives through the station.

C.16.2 Attributes

Length Specifies how many track sections the train will occupy.

C.16.3 Exceptions

The scenarios that will not provide a valid travel of a train are:

- The track section in front of the train is occupied, and the train moves forward.
- The track section in front of the train is a point, and the point is not switched to the direction where the train comes from.

- The track section in front of the train is a point, and the point is only intermediate switched.

C.17 Wire

C.17.1 Description

A wire connects two electrical components so current can pass from one of the components through the wire to the other component.

C.17.2 Attributes

None.

C.17.3 Exceptions

The scenarios that will not provide a valid connection in a circuit are:

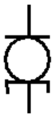
- The wire is only connected in one end.

Appendix D

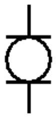
Relay Symbol Definitions

The different types of relays are listed alphabetically below. For each relay a general description is made of how the specific relay type typically is used. When nothing else is mentioned the symbol specifies a regular relay.

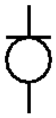
Auxiliary relay for releasing train routes
This symbol is used when a relay is an auxiliary relay in the process of releasing a train route. The relay is used to protect against a false release of a train route in case of voltage failure. The relay is drawn when a train route is locked and a voltage failure has not occured.



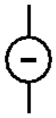
First relay in the release of the train route
This symbol is used when a relay is the first relay in the process of releasing a train route. The relay is drawn when the requirements for starting the release of the train route are fulfilled.



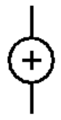
Point and signal key relay
A relay with this symbol is drawn when an associated button has been pushed.



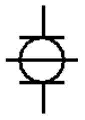
Point detection relay for the minus direction of the point
This symbol is used when a relay reflects the state of the minus direction of a point. The relay is drawn when the point is completely switched to the minus direction.



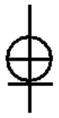
Point detection relay for the plus direction of the point
This symbol is used when a relay reflects the state of the plus direction of a point. The relay is drawn when the point is completely switched to the plus direction.



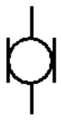
Replicate relay
This symbol is used for a relay that replicate the state of another relay. The relay is drawn when the relay being replicated is drawn.



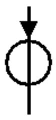
Replicate relay for track relay
This symbol is used for a relay that replicate the state of a *track relay*. The relay is drawn when the associated *track relay* is drawn.



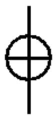
Signal control relay
This symbol is used when a relay is an auxiliary relay for locking a train route. The relay is drawn when the *point and signal key relay* has been drawn, no conflicting train routes are locked, points in the train route are completely switched and the train has not yet entered the station.



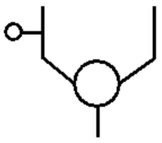
Signal lamp relay
This symbol is used when a relay reflects the state of a filament. The relay is drawn when the connected filament carries current.



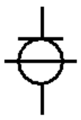
Track relay
This symbol is used when a relay reflects the state of a track section. The relay is drawn when the track section is free.



Train route locking relay
This symbol is used when a steel core relay reflects whether a train route is locked. The relay is demagnetized when the train route is locked, which typically happens when the *signal control relay* has been drawn.



Train route release relay
This symbol is used when a relay is the final relay in the release of a train route. The relay is drawn when the train route has been completely released.



Appendix E

Use Cases

In the following sections use cases concerning the application will be described. The use cases are divided into 4 areas: circuit, operators panel, train and administration.

E.1 Circuit

E.1.1 Create relay

E.1.1.1 Characteristic information

Name		Create relay.
ID		UC-1
Description		When the user wants to add coil pins or a contact on a diagram a relay must be created first.
Preconditions		The application is running, and a project has been created.
Success end condition		A relay has been created.
Basic flow	Step	Action
	1	The user defines at which level the relay should be placed in the rack of relays.
	2	The user defines at which field the relay should be placed in the rack of relays.
	3	The user defines whether the relay should be to the left or right (pursuant to the level and field position) in the rack of relays.
	4	The user defines how many contacts there should be on the relay.
	5	The user defines how many of the contacts should be upper contacts.
	6	The user optionally defines the name of the relay.
	7	The user creates the relay based on the defined parameters.
Extensions	Step	Branching action
	1a	The level number is outside the allowed range $[0 \cdots 5]$.
	2a	The field number is outside the allowed range $[1 \cdots 1000]$.

	4a	The number of contacts is outside the allowed range $\{6, 10, 20\}$.
	5a	The number of upper contacts is outside the allowed range $[0 \cdots 20]$.
	7a	The number of upper contacts is greater than the number of total contacts.
	7b	The position specified in the rack of relays is already occupied.
	7c	The number of contacts on the relay is greater than the number of contacts for a level, and the level is the uppermost level in the rack of relays.
	7d	The number of contacts on the relay is greater than the number of contacts for a level, and the above position in the rack of relays is occupied.

E.1.1.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

E.1.2 Add coil pins

E.1.2.1 Characteristic information

Name		Add coil pins.
ID		UC-2
Description		The user wants to add the pins that changes the state of a relay to a diagram.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created.
Success end condition		The coil pins are added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the coil pins should be added to.
	2	The user chooses the pins to add, by defining the position of the relay in the rack of relays. The position is determined by level, field and whether the relay is placed to the left or right.
	3	The user defines the type of the relay {Auxiliary relay for releasing train routes, First relay in the release of the train route, Point- and signal key relay, Replicate relay, Replicate relay for track relay, Signal control relay, Signal lamp relay, Train route release relay, Steel core relay}.
	4	The user optionally defines a description of the pins.
	5	The user creates the coil pins based on the defined parameters.
Extensions	Step	Branching action
	3a	The relay does not exist.
	5a	The coil pins are not added to any diagram.
	5b	The coil pins are added to the wrong diagram.
	5c	Coil pins with the same ID is already added to a diagram.

E.1.2.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1

E.1.3 Add contact

E.1.3.1 Characteristic information

Name		Add contact.
ID		UC-3
Description		The user wants to add a contact from a relay to a specific diagram.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created.
Success end condition		The contact has been added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the contact should be added to.
	2	The user chooses which relay he/she wants to add a contact from, by defining the relay's position in the rack of relays, that is level, field, left/right.
	3	The user optionally defines a description of the contact.
	4	The user adds the contact to a diagram based on the defined parameters.
Extensions	Step	Branching action
	4a	The contact is not added.
	4b	The contact is added to the wrong diagram.
	4c	A contact with the same ID is already added to a diagram.

E.1.3.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1

E.1.4 Add button

E.1.4.1 Characteristic information

Name		Add button.
ID		UC-4
Description		The user wants to add a button to a diagram.
Preconditions		The application is running, a project and a diagram has been created.
Success end condition		A button has been added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the button should be added to.
	2	The user defines the ID of the button.
	3	The user optionally defines a name and a description for the button.
	4	The user adds the button to a diagram, based on the defined parameters.
Extensions	Step	Branching action
	4a	The button is not added to a diagram.
	4b	The button is added to the wrong diagram.
	4c	A button with the same ID is already added to a diagram.

E.1.4.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

E.1.5 Add fuse

E.1.5.1 Characteristic information

Name		Add fuse.
ID		UC-5
Description		The user wants to add a fuse to a diagram.
Preconditions		The application is running, a project and a diagram has been created.
Success end condition		The fuse has been added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the fuse should be added to.
	2	The user defines the ID of the fuse.
	3	The user optionally defines a name and a description for the fuse.
	4	The user adds the button to a diagram based on the defined parameters.
Extensions	Step	Branching action
	4a	The fuse is not added to a diagram.
	4b	The fuse is added to the wrong diagram.
	4c	A fuse with the same ID is already added to a diagram.

E.1.5.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

E.1.6 Add resistor

E.1.6.1 Characteristic information

Name		Add resistor.
ID		UC-6
Description		The user wants to add a resistor to a diagram.
Preconditions		The application is running, a project and a diagram has been created.
Success end condition		The resistor has been added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the resistor should be added to.
	2	The user defines the ID of the resistor.
	3	The user optionally defines a name and a description for the resistor.
	4	The user adds the fuse to a diagram based on the defined parameters.
Extensions	Step	Branching action
	4a	The resistor is not added to any diagram.
	4b	The resistor is added to the wrong diagram.
	4c	A resistor with the same ID is already added to a diagram.

E.1.6.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

E.1.7 Add lamp

E.1.7.1 Characteristic information

Name		Add lamp.
ID		UC-7
Description		The user wants to add a lamp to a diagram.
Preconditions		The application is running, a project and a diagram has been created.
Success end condition		The lamp has been added to a diagram.
Basic flow	Step	Action
	1	The user chooses which diagram the lamp should be added to.
	2	The user defines the ID of the lamp.
	3	The user defines the color of the lamp.
	4	The user optionally defines a name and a description for the lamp.
	5	The user adds the lamp to a diagram based on the defined parameters.
Extensions	Step	Branching action
	5a	The lamp is not added to a diagram.
	5b	The lamp is added to the wrong diagram.
	5c	A lamp with the same ID is already added to a diagram.

E.1.7.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

E.1.8 Add spare filament

E.1.8.1 Characteristic information

Name		Add spare filament.
ID		UC-8
Description		The user wants to add a spare filament to a lamp.
Preconditions		The application is running, a project and a diagram has been created. A lamp has been added to a diagram.
Success end condition		The spare filament has been added to a lamp.
Basic flow	Step	Action
	1	The user chooses which lamp the spare filament should be added to.
	2	The user adds the spare filament based on the defined parameters.
Extensions	Step	Branching action
	2a	The spare filament is not added.
	2b	The spare filament is added to the wrong lamp.
	2c	The spare filament is added to the diagram without being associated with a lamp.

E.1.8.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-7

E.1.9 Add power supply

E.1.9.1 Characteristic information

Name		Add power supply.
ID		UC-9
Description		The user wants to add power supply to a diagram.
Preconditions		The application is running, a project and a diagram has been created.
Success end condition		Power supply is added to a diagram.
Basic flow	Step 1	Action The user chooses which diagram the power supply should be added to.
	2	The user adds the power supply to a diagram based on the defined parameters.
Extensions	Step	Branching action
	2a	Power supply is not added.
	2b	Power supply is added to the wrong diagram.

E.1.9.2 Related information

Priority		High
Subordinates		UC-46, UC-47

E.1.10 Remove component

E.1.10.1 Characteristic information

Name		Remove component.
ID		UC-10
Description		The user wants to remove the coil pins, a contact, a button, a fuse, a resistor, a lamp or the power supply from a diagram.
Preconditions		The application is running, a project and a diagram has been created. The component to be removed has been added.
Success end condition		The component has been removed from a diagram.
Basic flow	Step 1	Action The user chooses which component should be removed.
	2	The user removes the chosen component.
Extensions	Step	Branching action
	2a	The component is not removed.
	2b	The wrong component is removed.

E.1.10.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-2/UC-3/UC-4/UC-5/UC-6/UC-7/UC-9

E.1.11 Remove spare filament

E.1.11.1 Characteristic information

Name		Remove spare filament.
ID		UC-11
Description		The user wants to remove a spare filament from a lamp.
Preconditions		The application is running, a project and a diagram has been created. A lamp with a spare filament has been added.
Success end condition		The spare filament has been removed from a lamp.
Basic flow	Step	Action
	1	The user chooses which spare filament should be removed.
	2	The user removes the chosen spare filament.
Extensions	Step	Branching action
	2a	The spare filament is not removed.
	2b	The wrong spare filament is removed.

E.1.11.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-7, UC-7

E.1.12 Connect terminals

E.1.12.1 Characteristic information

Name		Connect terminals.
ID		UC-12
Description		The user wants to connect two terminals on a diagram.
Preconditions		The application is running, a project and a diagram has been created. Two components (coil pins, a contact, a button, a fuse, a lamp, power supply) has been added to a diagram.
Success end condition		Two terminals has been connected.
Basic flow	Step	Action
	1	The user chooses the two terminals that should be connected.
	2	The user connects the chosen terminals.
Extensions	Step	Branching action
	1a	One or both of the terminals are already connected to the maximum allowed number of connections, and can thus not be chosen.
	1b	The chosen terminals belongs to the same component, and can thus not be chosen.
	1c	The chosen terminals are both poles and can thus not be chosen.

E.1.12.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-2/UC-3/UC-4/UC-5/UC-6/UC-7/UC-8/UC-9

E.1.13 Disconnect terminals

E.1.13.1 Characteristic information

Name		Disconnect terminals.
ID		UC-13
Description		The user wants to disconnect two terminals on a diagram.
Preconditions		The application is running, a project and a diagram has been created. Two terminals are connected.
Success end condition		Two terminals has been disconnected.
Basic flow	Step	Action
	1	The user chooses which terminals should be disconnected.
	2	The user disconnects the chosen terminals.
Extensions	Step	Branching action
	2a	The terminals are not disconnected.
	2b	The wrong terminals are disconnected.

E.1.13.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-2/UC-3/UC-4/UC-5/UC-6/UC-7/UC-8/UC-9, UC-12

E.1.14 Push button

E.1.14.1 Characteristic information

Name		Push button.
ID		UC-14
Description		The user wants to push a button on a diagram.
Preconditions		The application is running, a project and a diagram has been created. A button has been added to a diagram.
Success end condition		The button on a diagram has been pushed.
Basic flow	Step	Action
	1	The user chooses the button to be pushed.
	2	The user pushes the chosen button.
Extensions	Step	Branching action
	2a	The button is not pushed.
	2b	The wrong button is pushed.

E.1.14.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-4

E.1.15 Release button

E.1.15.1 Characteristic information

Name		Release button.
ID		UC-15
Description		The user wants to release a pushed button on a diagram.
Preconditions		The application is running, a project and a diagram has been created. A button has been added to a diagram and pushed.
Success end condition		The button is released.
Basic flow	Step	Action
	1	The user chooses the button to be released.
	2	The user releases the chosen button.
Extensions	Step	Branching action
	2a	The button is not released.
	2b	The wrong button is released.

E.1.15.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-4, UC-14

E.1.16 Draw relay permanently

E.1.16.1 Characteristic information

Name		Draw relay permanently.
ID		UC-16
Description		The user wants to simulate a broken relay. The relay is simulated to be permanently drawn.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created and the coil pins has been added to a diagram.
Success end condition		The relay is been permanently drawn.
Basic flow	Step	Action
	1	The user chooses a relay.
	2	The chosen relay is simulated as being permanently drawn.
Extensions	Step	Branching action
	2a	The relay does not remain drawn.

E.1.16.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-2

E.1.17 Drop relay permanently

E.1.17.1 Characteristic information

Name		Drop relay permanently.
ID		UC-17
Description		The user wants to simulate a broken relay. The relay is simulated to be permanently dropped.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created, and the coil pins have been added to a diagram.
Success end condition		The relay is permanently dropped.
Basic flow	Step	Action
	1	The user chooses a relay.
	2	The chosen relay is simulated as being permanently dropped.
Extensions	Step	Branching action
	2a	The relay is initially dropped, but does not remain dropped.

E.1.17.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-2

E.1.18 Set relay non-conducting

E.1.18.1 Characteristic information

Name		Set relay non-conducting.
ID		UC-18
Description		The user wants to simulate a broken relay. The relay is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created, and the coil pins have been added to a diagram.
Success end condition		The relay is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a relay.
	2	The chosen relay is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The relay is initially non-conducting, but does not remain non-conducting.

E.1.18.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-2

E.1.19 Set contact non-conducting

E.1.19.1 Characteristic information

Name		Set contact non-conducting.
ID		UC-19
Description		The user wants to simulate a broken contact. The contact is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created, and a contact has been added to a diagram.
Success end condition		The contact is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a contact.
	2	The chosen contact is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The contact is initially non-conducting, but does not remain non-conducting.

E.1.19.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-3

E.1.20 Set contact conductive

E.1.20.1 Characteristic information

Name		Set contact conductive.
ID		UC-20
Description		The user wants to simulate a broken contact. The contact is simulated to be conductive.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created, and a contact has been added to a diagram.
Success end condition		The contact is permanently conductive.
Basic flow	Step	Action
	1	The user chooses a contact.
	2	The chosen contact is simulated as being permanently conductive.
Extensions	Step	Branching action
	2a	The contact is initially conductive, but does not remain conductive.

E.1.20.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-3

E.1.21 Set fuse non-conducting

E.1.21.1 Characteristic information

Name		Set fuse non-conducting.
ID		UC-21
Description		The user wants to simulate a broken fuse. The fuse is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A fuse has been added to a diagram.
Success end condition		The fuse is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a fuse.
	2	The chosen fuse is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The fuse is initially non-conducting, but does not remain non-conducting.

E.1.21.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-5

E.1.22 Set resistor non-conducting

E.1.22.1 Characteristic information

Name		Set resistor non-conducting.
ID		UC-22
Description		The user wants to simulate a broken resistor. The resistor is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A resistor has been added to a diagram.
Success end condition		The resistor is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a resistor.
	2	The chosen resistor is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The resistor is initially non-conducting, but does not remain non-conducting.

E.1.22.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-6

E.1.23 Set lamp non-conducting

E.1.23.1 Characteristic information

Name		Set lamp non-conducting.
ID		UC-23
Description		The user wants to simulate a broken lamp. The lamp is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A lamp has been added to a diagram.
Success end condition		The lamp is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a lamp.
	2	The chosen lamp is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The lamp is initially non-conducting, but does not remain non-conducting.

E.1.23.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-7

E.1.24 Set spare filament non-conducting

E.1.24.1 Characteristic information

Name		Set spare filament non-conducting.
ID		UC-24
Description		The user wants to simulate a broken spare filament. The spare filament is simulated to be non-conducting.
Preconditions		The application is running, a project and a diagram has been created. A spare filament has been added to a diagram.
Success end condition		The spare filament is permanently non-conducting.
Basic flow	Step	Action
	1	The user chooses a spare filament.
	2	The chosen spare filament is simulated as being permanently non-conducting.
Extensions	Step	Branching action
	2a	The spare filament is initially non-conducting, but does not remain non-conducting.

E.1.24.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-7, UC-8

E.1.25 Repair relay

E.1.25.1 Characteristic information

Name		Repair relay.
ID		UC-25
Description		The user wants to repair a broken relay.
Preconditions		The application is running, a project and a diagram has been created. A relay has been created, and the coil pins have been added to a diagram. The relay is broken, that is either permanently drawn, dropped and/or non-conducting.
Success end condition		The relay is repaired.
Basic flow	Step	Action
	1	The user chooses a relay.
	2	The chosen relay is repaired.
Extensions	Step	Branching action
	2a	The relay is not repaired.

E.1.25.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-2, UC-16/UC-17/UC-18

E.1.26 Repair contact

E.1.26.1 Characteristic information

Name		Repair contact.
ID		UC-26
Description		The user wants to repair a broken contact.
Preconditions		The application is running, a project and a diagram has been created. A contact has been added to a diagram. The contact is broken.
Success end condition		The contact is repaired.
Basic flow	Step	Action
	1	The user chooses a contact.
	2	The chosen contact is repaired.
Extensions	Step	Branching action
	2a	The contact is not repaired.

E.1.26.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-1, UC-3, UC-19/UC-20

E.1.27 Repair fuse

E.1.27.1 Characteristic information

Name		Repair fuse.
ID		UC-27
Description		The user wants to repair a broken fuse.
Preconditions		The application is running, a project and a diagram has been created. A fuse has been added to a diagram. The fuse is broken.
Success end condition		The fuse is repaired.
Basic flow	Step	Action
	1	The user chooses a fuse.
	2	The chosen fuse is repaired.
Extensions	Step	Branching action
	2a	The fuse is not repaired.

E.1.27.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-5, UC-21

E.1.28 Repair resistor

E.1.28.1 Characteristic information

Name		Repair resistor.
ID		UC-28
Description		The user wants to repair a broken resistor.
Preconditions		The application is running, a project and a diagram has been created. A resistor has been added to a diagram. The resistor is broken.
Success end condition		The resistor is repaired.
Basic flow	Step	Action
	1	The user chooses a resistor.
	2	The chosen resistor is repaired.
Extensions	Step	Branching action
	2a	The resistor is not repaired.

E.1.28.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-6, UC-22

E.1.29 Repair lamp

E.1.29.1 Characteristic information

Name		Repair lamp.
ID		UC-29
Description		The user wants to repair a broken lamp.
Preconditions		The application is running, a project and a diagram has been created. A lamp has been added to a diagram. The lamp is broken.
Success end condition		The lamp is repaired.
Basic flow	Step	Action
	1	The user chooses a lamp.
	2	The chosen lamp is repaired.
Extensions	Step	Branching action
	2a	The lamp is not repaired.

E.1.29.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-4, UC-7, UC-23

E.1.30 Repair spare filament

E.1.30.1 Characteristic information

Name		Repair spare filament.
ID		UC-30
Description		The user wants to repair a broken spare filament.
Preconditions		The application is running, a project and a diagram has been created. A spare filament has been added to a diagram. The spare filament is broken.
Success end condition		The spare filament is repaired.
Basic flow	Step	Action
	1	The user chooses a spare filament.
	2	The chosen spare filament is repaired.
Extensions	Step	Branching action
	2a	The spare filament is not repaired.

E.1.30.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47, UC-7, UC-8, UC-24

E.1.31 View normal state

E.1.31.1 Characteristic information

Name		View normal state.
ID		UC-31
Description		The user wants to see the normal state of a station.
Preconditions		The application is running.
Success end condition		The normal state of the station is displayed.
Basic flow	Step	Action
	1	The user chooses to display the normal state of the station.
Extensions	Step	Branching action
	1a	The normal state of the station is not displayed correctly.

E.1.31.2 Related information

Priority		Medium
Subordinates		UC-46

E.1.32 Step simulation

E.1.32.1 Characteristic information

Name		Step simulation.
ID		UC-32
Description		The user wants to view the next state of a circuit.
Preconditions		The application is running.
Success end condition		The next state of the circuit is displayed.
Basic flow	Step	Action
	1	The user chooses to progress the state of a circuit one step.
Extensions	Step	Branching action
	1a	The propagation of the current is not displayed correctly.

E.1.32.2 Related information

Priority		Medium
Subordinates		UC-46

E.1.33 Full simulation

E.1.33.1 Characteristic information

Name		Full simulation.
ID		UC-33
Description		The user wants to simulate how the current propagates in the circuit until the circuit has no further changes.
Preconditions		The application is running.
Success end condition		The final state of the circuit is displayed.
Basic flow	Step	Action
	1	The user chooses to simulate the final state of the circuit .
Extensions	Step	Branching action
	1a	The displayed state of the circuit is not the final state.
	2a	The circuit is cyclic and the application stalls.

E.1.33.2 Related information

Priority		Medium
Subordinates		UC-46

E.2 Operators Panel

E.2.1 Add track section

E.2.1.1 Characteristic information

Name		Add track section.
ID		UC-34
Description		The user wants to add a track section to the operators panel.
Preconditions		The application is running, a project and the operators panel has been created.
Success end condition		The track section is added to the operators panel.
Basic flow	Step	Action
	1	The user defines the name of the track section.
	2	The user defines at which level the interface relay should be placed in the rack of relays.
	3	The user defines at which field the interface relay should be placed in the rack of relays.
	4	The user defines whether the interface relay should be to the left or right (pursuant to the level and field position) in the rack of relays.
	5	The user defines how many contacts there should be on the interface relay.
	6	The user defines how many of the contacts should be upper contacts.
	7	The user optionally defines the description of the interface relay.
	8	The user adds a track section to the operators panel based on the defined parameters.

E.2.1.2 Related information

Extensions	Step	Branching action
	1a	The name is left blank.
	2a	The level number is outside the allowed range $[0 \cdots 5]$.
	3a	The field number is outside the allowed range $[1 \cdots 1000]$.
	5a	The number of contacts is outside the allowed range $\{6, 10, 20\}$.
	6a	The number of upper contacts is greater than the number of total contacts or negative.
	8a	The track section is not added to the operators panel.
	8b	An interface relay is created, even though the track section is not added to the operators panel.

E.2.1.3 Related information

Priority		Medium
Subordinates		UC-46, UC-48

E.2.2 Add point

E.2.2.1 Characteristic information

Name		Add point.
ID		UC-35
Description		The user wants to add a point to the operators panel.
Preconditions		The application is running, a project and the operators panel has been created.
Success end condition		The point has been added to the operators panel.
Basic flow	Step	Action
	1	The user defines the name of the part of the point that functions as a track section.
	2	The user defines the name of the point.
	3	The user defines at which level the track section interface relay should be placed in the rack of relays.
	4	The user defines at which field the track section interface relay should be placed in the rack of relays.
	5	The user defines whether the track section interface relay should be to the left or right (pursuant to the level and field position) in the rack of relays.
	6	The user defines how many contacts there should be on the interface relay.
	7	The user defines how many of the contacts should be upper contacts.
	8	The user optionally defines the description of the track section interface relay.
	9	The user defines at which level the plus interface relay should be placed in the rack of relays.

E.2.2.2 Related information

	10	The user defines at which field the plus interface relay should be placed in the rack of relays.
	11	The user defines whether the plus interface relay should be to the left or right (pursuant to the level and field position) in the rack of relays.
	12	The user defines how many contacts there should be on the plus interface relay.
	13	The user defines how many of the contacts should be upper contacts.
	14	The user optionally defines the description of the plus interface relay.
	15	The user defines at which level the minus interface relay should be placed in the rack of relays.
	16	The user defines at which field the minus interface relay should be placed in the rack of relays.
	17	The user defines whether the minus interface relay should be to the left or right (pursuant to the level and field position) in the rack of relays.
	18	The user defines how many contacts there should be on the minus interface relay.
	19	The user defines how many of the contacts should be upper contacts.
	20	The user optionally defines the description of the minus interface relay.

E.2.2.3 Related information

	21	The user adds the point based on the defined parameters.
Extensions	Step	Branching action
	1a, 2a	The name is left blank.
	3a, 9a, 15a	The level number is outside the allowed range $[0 \cdots 5]$.
	4a, 10a, 16a	The field number is outside the allowed range $[1 \cdots 1000]$.
	6a, 12a, 18a	The number of contacts is outside the allowed range $\{6, 10, 20\}$.
	7a, 13a, 19a 21a	The number of upper contacts is greater than the number of total contacts or negative. The point is not added to the operators panel.
	21b	The point is not added to the operators panel, but one or more interface relays are still created.
	21c	One or more of the interface relays already exists.
	21d	Two or more of the interface relays has the same position in the rack of relays.

E.2.2.4 Related information

Priority		Medium
Subordinates		UC-46, UC-48

E.2.3 Assemble track sections/points

E.2.3.1 Characteristic information

Name		Assemble track sections/points.
ID		UC-36
Description		The user wants to assemble one of the ends on a track section with an end on another track section or on a point.
Preconditions		The application is running, a project and the operators panel has been created. Two track sections, or a track section and a point, has been added to the operators panel.
Success end condition		Two ends has been assembled.
Basic flow	Step	Action
	1	The user chooses which ends should be assembled.
	2	The user assembles the chosen ends.
Extensions	Step	Branching action
	1a	One or both of the ends are already assembled to another end.
	2a	The chosen ends belongs to the same track sections.
	2b	The ends both belongs to a point.
	2c	Two ends that were not chosen are assembled.
	2d	One of the ends belong to a track section/-point which the other end also is assembled to.

E.2.3.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34/UC-35

E.2.4 Occupy track section

E.2.4.1 Characteristic information

Name		Occupy track section.
ID		UC-37
Description		The user wants to occupy a track section on the operators panel.
Preconditions		The application is running, a project and the operators panel has been created. A track section has been added to the operators panel.
Success end condition		The track section has been occupied.
Basic flow	Step	Action
	1	The user chooses the track section to occupy.
	2	The user occupies the chosen track section.
Extensions	Step	Branching action
	2a	The track section is not occupied.
	2b	A wrong track section is occupied.
	2c	The track section is not occupied since it is already occupied by a train.

E.2.4.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34

E.2.5 Free track section

E.2.5.1 Characteristic information

Name		Free track section.
ID		UC-38
Description		The user wants to free an occupied track section on the operators panel.
Preconditions		The application is running, a project and the operators panel has been created. A track section has been added to the operators panel. The track section is occupied.
Success end condition		The track section has been freed.
Basic flow	Step	Action
	1	The user chooses the track section to free.
	2	The user frees the chosen track section.
Extensions	Step	Branching action
	2a	The track section on the operators panel is not freed.
	2b	A wrong track section is freed.
	2c	The track section on the operators panel is not freed since it is occupied by a train.

E.2.5.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34, UC-37

E.2.6 Occupy point

E.2.6.1 Characteristic information

Name		Occupy point.
ID		UC-39
Description		The user wants to occupy a point on the operators panel.
Preconditions		The application is running, a project and the operators panel has been created. A point has been added to the operators panel.
Success end condition		The point has been occupied.
Basic flow	Step	Action
	1	The user chooses the point to occupy.
	2	The user occupies the chosen point.
Extensions	Step	Branching action
	2a	The point on the operators panel is not occupied.
	2b	A wrong point is occupied.
	2c	The point on the operators panel is not occupied, since it is already occupied by a train.

E.2.6.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-35

E.2.7 Free point

E.2.7.1 Characteristic information

Name		Free point.
ID		UC-40
Description		The user wants to free an occupied point on the operators panel.
Preconditions		The application is running, a project and the operators panel has been created. A point has been added to the operators panel. The point is occupied.
Success end condition		The point has been freed.
Basic flow	Step	Action
	1	The user chooses the point to free.
	2	The user frees the chosen point.
Extensions	Step	Branching action
	2	The point on the operators panel is not freed.
	2	A wrong point is freed.
	2	The point on the operators panel is not freed, since it is occupied by a train.

E.2.7.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-35, UC-39

E.2.8 Switch point

E.2.8.1 Characteristic information

Name		Switch point.
ID		UC-41
Description		The user wants to switch a point on the operators panel. The switch operation has to be used twice before to switch a point from one direction to another. When the switch operation is used once the point is in the intermediate position i.e. neither switched to the plus nor the minus direction.
Preconditions		The application is running, a project and the operators panel has been created. A point has been added to the operators panel.
Success end condition		The point has been partially switched.
Basic flow	Step	Action
	1	The user chooses the point to switch.
	2	The user switches the chosen point.
Extensions	Step	Branching action
	2a	The point on the operators panel is not switched.
	2b	The wrong point is switched.
	2c	The point on the operators panel is not switched, because the point is occupied.
	2d	The point is not switched correctly.

E.2.8.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-35

E.3 Train

E.3.1 Start train

E.3.1.1 Characteristic information

Name		Start train.
ID		UC-42
Description		The user wants to simulate a train entering the station.
Preconditions		The application is running, and the operators panel is created. A track section or a point with exactly one available end is added to the operators panel.
Success end condition		The train has entered a station by occupying the first track section/point on the station.
Basic flow	Step	Action
	1	The user chooses a track section where the train should enter the station.
	2	The user start the simulation of the train from the chosen track section/point.
Extensions	Step	Branching action
	1a	The chosen track section is not a valid start track section since it has more or less than one free end.
	2a	The chosen track section is not occupied.

E.3.1.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34/UC-35, UC-12

E.3.2 Occupy next track section

E.3.2.1 Characteristic information

Name		Occupy next track section.
ID		UC-43
Description		The user wants to occupy the next tract section in front of the train.
Preconditions		The application is running, and the operators panel is created. A track section with exactly one available end is added to the operators panel.
Success end condition		The next free track section in front of the train has been occupied.
Basic flow	Step	Action
	1	The user makes the train occupy the next free track section in front of the train.
Extensions	Step	Branching action
	1a	No track section is occupied.
	1b	There is no track section in front of the train, since the train is on the last track section of the station.
	1c	The train crashes since the point in front of the train was not switched to the correct track.
	1d	The train crashes since the track section in front of the train is already occupied (by e.g. a utility pole).

E.3.2.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34/UC-35, UC-12, UC-42

E.3.3 Free rear track section

E.3.3.1 Characteristic information

Name		Free rear track section.
ID		UC-44
Description		The user wants to free the rearest track section occupied by the train.
Preconditions		The application is running, and the operators panel is created. A track section with exactly one available end is added to the operators panel.
Success end condition		The rearest track section occupied by the train is freed.
Basic flow	Step	Action
	1	The user makes the train free the rearest track section occupied by the train.
Extensions	Step	Branching action
	1a	The train has only occupied a single track section, and therefore the rear cannot be freed.

E.3.3.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34/UC-35, UC-12, UC-43

E.3.4 Cancel train

E.3.4.1 Characteristic information

Name		Cancel train.
ID		UC-45
Description		The user wants to cancel a train currently occupying one or more track sections on the station.
Preconditions		The application is running, and the operators panel is created. A train is occupying track sections at the station.
Success end condition		The train is cancelled, and all track sections occupied by the train are freed.
Basic flow	Step	Action
	1	The user cancels the train on the station.
Extensions	Step	Branching action
	1a	Not all track sections occupied by the train are freed.

E.3.4.2 Related information

Priority		Medium
Subordinates		UC-46, UC-48, UC-34/UC-35, UC-12, UC-42, UC-43/UC-44

E.4 Administration

E.4.1 Create project

E.4.1.1 Characteristic information

Name		Create project.
ID		UC-46
Description		The user wants to create a new station and thus a new project.
Preconditions		The application is running.
Success end condition		The project has been created.
Basic flow	Step	Action
	1	The user creates a project.
Extensions	Step	Branching action

E.4.1.2 Related information

Priority		High
Subordinates		

E.4.2 Create diagram

E.4.2.1 Characteristic information

Name		Create new diagram.
ID		UC-47
Description		The user wants to create a diagram for a part of the circuit.
Preconditions		The application is running, and a project has been created.
Success end condition		The diagram has been created.
Basic flow	Step	Action
	1	The user creates a new diagram.
Extensions	Step	Branching action
	1a	The diagram is not created.

E.4.2.2 Related information

Priority		High
Subordinates		UC-46

E.4.3 Create operators panel

E.4.3.1 Characteristic information

Name		Create operators panel.
ID		UC-48
Description		The user wants to create the operators panel, such that track sections can be added.
Preconditions		The application is running, and a project has been created.
Success end condition		The operators panel has been created.
Basic flow	Step	Action
	1	The user creates the operators panel.
Extensions	Step	Branching action

E.4.3.2 Related information

Priority		Medium
Subordinates		UC-46

E.4.4 Save project

E.4.4.1 Characteristic information

Name		Save project.
ID		UC-49
Description		The user wants to save the currently opened project.
Preconditions		The application is running.
Success end condition		The project is saved.
Basic flow	Step	Action
	1	The user saves the project.
Extensions	Step	Branching action
	1a	Some or all of the components are not saved.

E.4.4.2 Related information

Priority		Medium
Subordinates		UC-46

E.4.5 Load project

E.4.5.1 Characteristic information

Name		Load project.
ID		UC-50
Description		The user wants to load a previously saved project.
Preconditions		The application is running.
Success end condition		The project is opened.
Basic flow	Step	Action
	1	The user opens a project.
Extensions	Step	Branching action
	1a	Some or all of the saved components are not displayed.
	1b	Some or all of the connections between the components are not displayed.
	1c	The file is not a *.ris file.

E.4.5.2 Related information

Priority		Medium
Subordinates		UC-46, UC-49

E.4.6 Reset station

E.4.6.1 Characteristic information

Name		Reset station.
ID		UC-51
Description		The user wants to reset all components to their initial state.
Preconditions		The application is running.
Success end condition		The station is reset.
Basic flow	Step	Action
	1	The user resets the station.
Extensions	Step	Branching action
	1a	Some or all of the components has not the same conductive state as when they were created.
	1b	Some or all of the components are still live.

E.4.6.2 Related information

Priority		Medium
Subordinates		UC-46

E.4.7 Rename diagram

E.4.7.1 Characteristic information

Name		Rename diagram.
ID		UC-52
Description		The user wants to rename a diagram.
Preconditions		The application is running and a diagram is created.
Success end condition		The diagram is renamed.
Basic flow	Step	Action
	1	The user chooses a diagram to rename.
	2	The user renames the chosen diagram.
Extensions	Step	Branching action
	2a	The diagram is not renamed.
	2b	The wrong diagram is renamed.

E.4.7.2 Related information

Priority		Medium
Subordinates		UC-46, UC-47

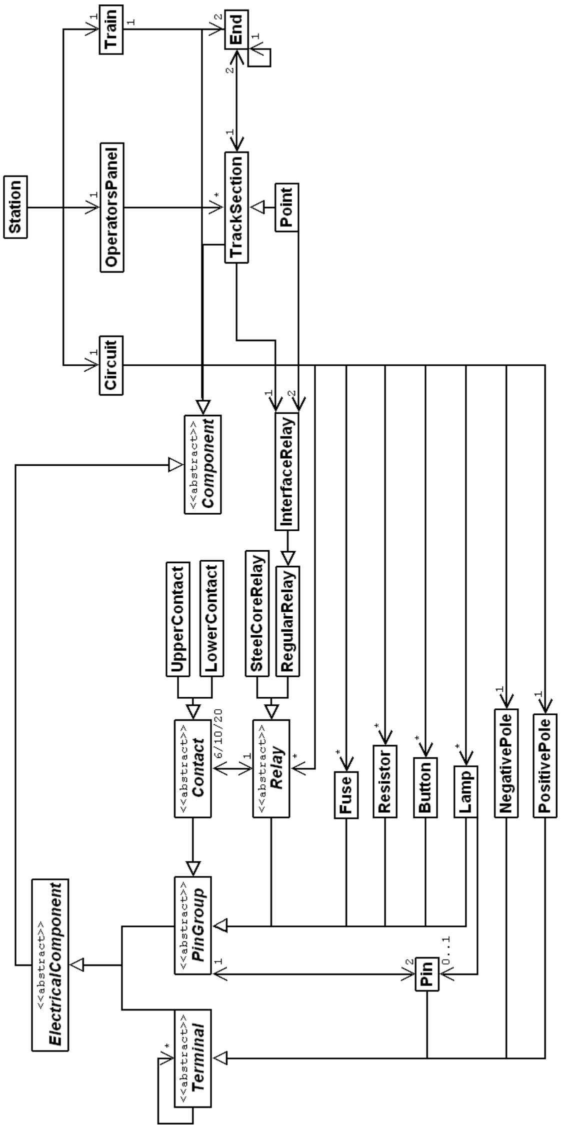
Appendix F

UML diagrams

The structure of the model and presentation layers of the application are documented using UML diagrams. The UML diagrams reflects the relation and inheritance between the different classes of the application.

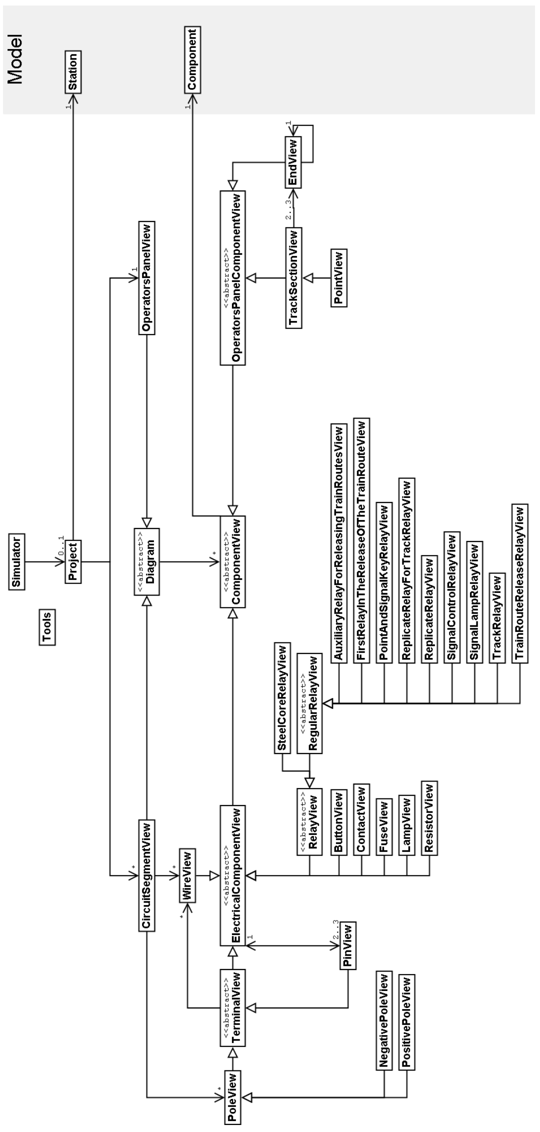
F.1 Model

The image below documents the relation and inheritance between the classes in the model layer of the application.



F.2 Presentation

This image documents the relation and inheritance in the presentation layer and the link between the model and presentation layers. The objects within the gray area are part of the model layer.



Appendix G

Sequence Diagrams

Sequence diagrams are used to document how the communication between classes in the application will take place.

The diagram for simulating a step is a sequence diagram for a circuit as on figure G.1.

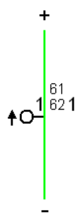
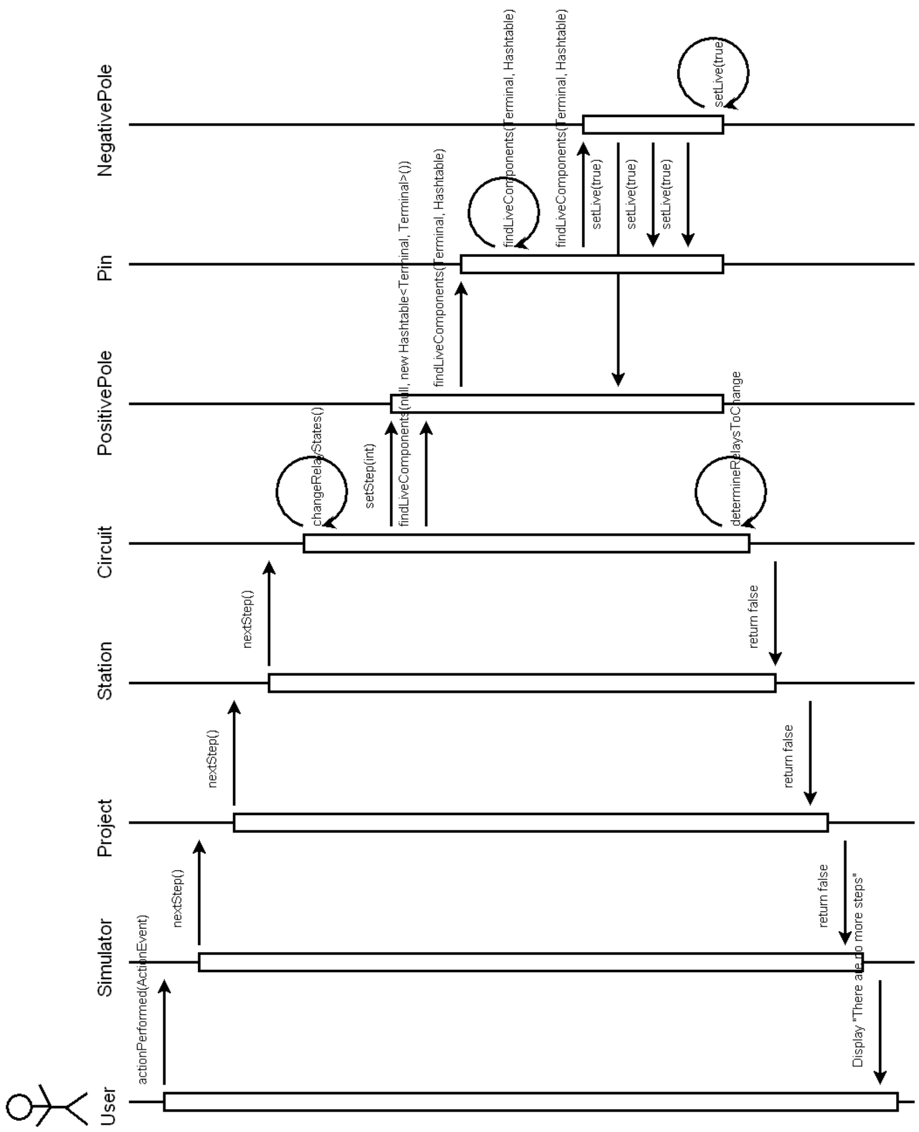


Figure G.1: *The circuit for the sequence diagram depicting a simulation of a step.*

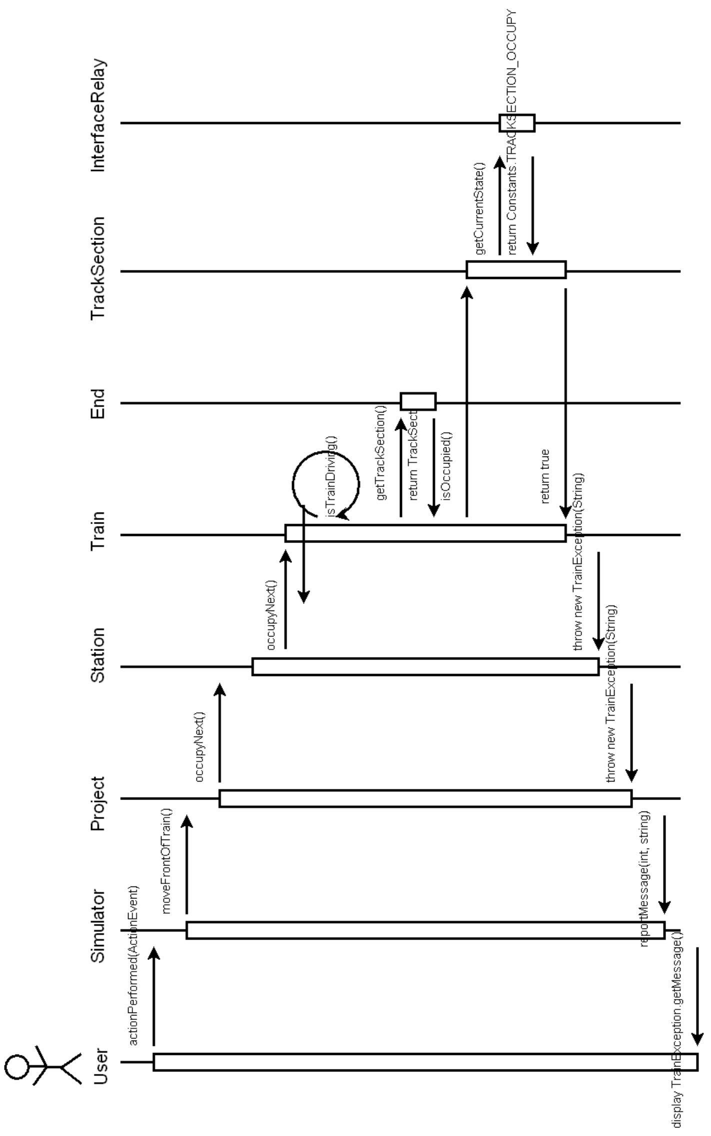
The second diagram depicts the scenario where the track section in front of the train is already occupied by a another conductive object.

The last diagram shows the application flow when a train is cancelled.

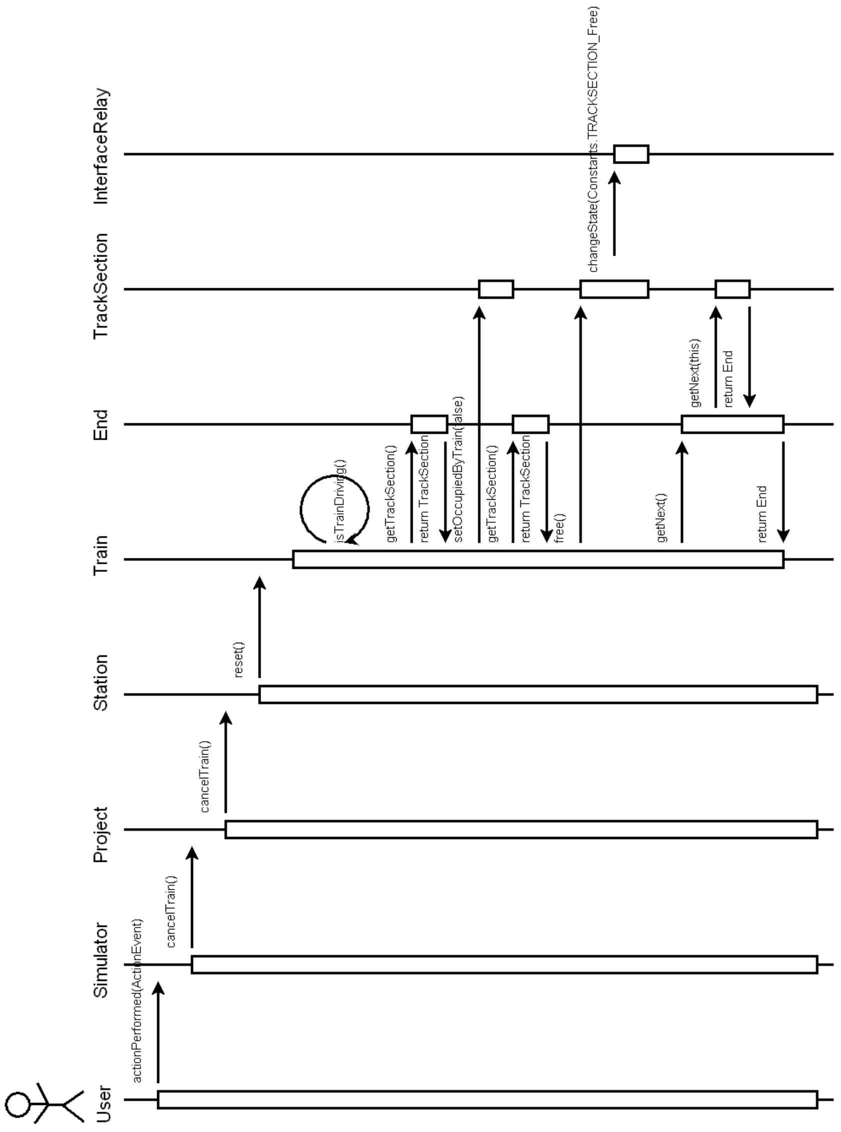
G.1 Simulate a Step


User
Simulator
Project
Station
Circuit
PositivePole
Pin
NegativePole
actionPerformed(ActionEvent)
nextStep()
nextStep()
nextStep()
changeRelayStates()
setStep(int)
findLiveComponents(null, new Hashtable<Terminal, Terminal>())
findLiveComponents(Terminal, Hashtable)
findLiveComponents(Terminal, Hashtable)
findLiveComponents(Terminal, Hashtable)
setLive(true)
setLive(true)
setLive(true)
setLive(true)
determineRelaysToChange
return false
return false
return false
Display "There are no more steps"

G.2 Occupy Next Track Section



G.3 Cancel train



Appendix H

Test Cases

H.1 Functional tests

The functional test cases of the presentation are documented in this appendix.

H.1.1 Diagram

H.1.2 TC-1: Create a regular relay

H.1.2.1 Description

This test case tests UC-1 which is used when a relay is placed in the rack of relay. The relay must be placed in the rack of relays before the relay can be connected to other components. In this test a regular relay with a total of 6 contacts, of which 4 are upper contacts, will be created and placed as the right relay in field 10 and level 4.

H.1.2.2 Precondition

Before a relay can be added, a new project has to be created(UC-46). No relay has been created and added as the right relay in field 10, level 4 in the rack of relays.

H.1.2.3 Postcondition

The relay is created and placed at the specified location in the rack of relays. It is possible to show contacts from this relay and connect them to other components.

H.1.2.4 Test procedure

#	Procedure	Expected result
1	Click the "Create new relay" icon on the toolbar or the "Relay".	A dialog box with the title "Create new relay" is displayed.
2	Type in "A" as name for the relay. (Optional)	
3	Choose "Regular relay" in the drop-down box.	
4	Type "10" in the field field.	
5	Type "4" in the level field.	
6	The button "Right" is selected	
7	The number of total contacts are set to 6.	
8	The number of upper contacts are set to 4.	
9	The "Ok" button is pushed.	A message is displayed to confirm that the relay has been created.

H.1.3 TC-2: Add coil pins

H.1.3.1 Description

This test case tests UC-2 which is used when the user wants to add the pins that changes the state of a relay to a diagram.

H.1.3.2 Precondition

This test is expected to follow directly after TC-1, which means that a project has to created and a relay has to be created and added to the rack of relays.

H.1.3.3 Postcondition

The coil pins are displayed on the specific diagram.

H.1.3.4 Test procedure

#	Procedure	Expected result
1	Click the "Display regular coil pins" icon on the toolbar.	
2	Click on the diagram where the icon should be displayed.	A dialog box with the title "Display regular coil pins" is displayed.
3	Select the type of relay the icon should be displayed as.	
4	Choose "10" in the field drop-down box.	The level drop-down box is updated and contains only the numbers on the levels at field 10 that are not empty.
5	Type "4" in the level drop-down box.	The "Left" and "Right" buttons are enabled/disabled depending on whether the left and right relay positions in field 10 and level 4 are occupied by a relay.
6	Select the "Right" button.	
7	Leave the description field blank.	
8	Press the "Ok" button.	The dialog disappears and the icon is displayed on the diagram.

H.1.4 TC-3: Add contact

H.1.4.1 Description

This test case testes UC-3 which is used when the user wants to display a contact on a diagram.

H.1.4.2 Precondition

This test is expected to follow directly after TC-1, which means that a project and a relay has been created.

H.1.4.3 Postcondition

The specified contact is displayed on the specific diagram.

H.1.4.4 Test procedure

#	Procedure	Expected result
1	Click the "Display contact" icon on the toolbar or in the "Components" menu.	
2	Click on the diagram where the contact should be displayed.	A dialog box with the title "Display contact" is displayed.
3	Choose "10" in the field drop-down box.	The level drop-down box updates to only contain the numbers on the levels at field 10 that are not empty.
4	Type "4" in the level drop-down box.	The "Left" and "Right" buttons are enabled/disables depending on whether the left and right relay positions in field 10 and level 4 are occupied by a relay/empty.
5	Select the "Right" button.	
6	Leave the description field blank.	
7	Press the "Ok" button.	The dialog disappears and the contact is displayed on the diagram.

H.1.5 TC-4: Add lamp

H.1.5.1 Description

This test case testes UC-7 which is used when the user wants to add and display a lamp on a diagram. In this test a red lamp is added and displayed on a diagram.

H.1.5.2 Precondition

Before a lamp can be added, a project and a diagram has to be created.

H.1.5.3 Postcondition

A red lamp is added and displayed on the diagram.

H.1.5.4 Test procedure

#	Procedure	Expected result
1	Click the "Add lamp" on the toolbar or in the "Components" menu.	
2	Click on the diagram where the lamp should be displayed.	A dialog box with the title "Display new lamp" is displayed.
3	Type "A" in the name field.	
4	Type "1" in the ID field.	Select "Red" in the color drop-down box.
5	Leave the description field blank.	
6	Press the "Ok" button.	The dialog disappears and a red lamp is displayed on the diagram.

H.1.6 TC-5: Add spare filament to lamp

H.1.6.1 Description

This test case tests UC-8 which is used when the user wants to add a spare filament to a displayed lamp on a diagram.

H.1.6.2 Precondition

This test is expected to follow directly after TC-4, which means that a project and a diagram has to be created, and a lamp (with no added spare filament) has to be displayed on a diagram.

H.1.6.3 Postcondition

The spare filament is added to the existing red lamp and displayed on the diagram.

H.1.6.4 Test procedure

#	Procedure	Expected result
1	Right-click the lamp displayed on the diagram.	A short-cut menu pops up.
2	Click "Add spare filament" in the short-cut menu.	The previously displayed lamp icon is replaced with a new containing two filaments. Both of the filaments are red.

H.1.7 TC-6: Connect terminals

H.1.7.1 Description

This test case tests UC-12 which is used when the user wants to connect two terminals displayed on the same diagram.

H.1.7.2 Precondition

A project and a diagram have been created. On the diagram two terminals are displayed (a lamp and a button).

H.1.7.3 Postcondition

A wire is drawn between the selected pins on the lamp and the button.

H.1.7.4 Test procedure

#	Procedure	Expected result
1	Press and hold the CTRL key on the keyboard.	All terminals on the diagram are shown with green circles, since they all allow more connections to be made to them.
2	Move the cursor on top of the lowest green circle on the lamp displayed on the diagram, and press and hold the left mouse button.	The pins on the lamp changes to red, since a component cannot be connected to itself.
3	Move the cursor towards (but not over) the lower green circle on the button displayed on the same diagram as the lamp.	A red line is drawn from the lowest circle on the lamp to where the cursor is pointing.
4	Move the cursor on top of the lowest green circle on the button displayed on the diagram.	The line color is changed from red to green to inform that a connection to this pin is allowed.
5	Release the left mouse button.	The green line is replaced by a wire connecting the pins selected by the user.

H.1.8 TC-7: Draw relay permanently

H.1.8.1 Description

This test case tests UC-16 which is used when the user wants to simulate a broken relay. The relay is permanently in the drawn state even when current is not applied to the coil pins.

H.1.8.2 Precondition

This test is expected to follow directly after TC-2, which means that a project and a diagram has to be created, a relay is created and the coil pins are displayed on a diagram.

H.1.8.3 Postcondition

The arrow next to the icon indicates that the relay is drawn.

H.1.8.4 Test procedure

#	Procedure	Expected result
1	Right-click the relay icon displayed on the diagram.	A short-cut menu pops up.
2	Click ”Set drawn” in the short-cut menu.	The arrow on the left side of the icon points upwards and the color is changed to red.

H.1.9 TC-8: Repair relay

H.1.9.1 Description

This test case tests UC-25 which is used when the user wants to repair a relay. The relay might previously have been set to either parmanently drawn, permanently dropped or permanently non-conducting .

H.1.9.2 Precondition

A project, a diagram and a relay is created. The coil pins are displayed on the diagram and the associated relay is set to permanently drawn.

H.1.9.3 Postcondition

The state of the relay corresponds to whether current is applied to or removed from the relay.

H.1.9.4 Test procedure

#	Procedure	Expected result
1	Right-click the icon displayed on the diagram.	A short-cut menu pops up.
2	Click ”Repair” in the short-cut menu.	The icon does not change, but when current is applied to or removed from the relay the state of the relay can be changed.

H.1.10 TC-9: Simulate propagation of current in steps

H.1.10.1 Description

This test case tests UC-32 which is used when the user wants to simulate the propagation of current step by step.

H.1.10.2 Precondition

This test is expected to follow directly after TC-2, which means that a project and a diagram has to be created, a relay is created and the coil pins are displayed on a diagram. A positive pole and a negative pole has been added to the diagram, the positive pole is connected to pin 01 on the relay, and the negative pole is connected to pin 02 on the relay.

H.1.10.3 Postcondition

The relay, the positive and negative poles are live. The current is illustrated with green, thus all wires are green. After step 2 the relay has changed state.

H.1.10.4 Test procedure

#	Procedure	Expected result
1	Click on the "Simulation - next step " icon on the toolbar.	Current has been applied to the system and the wires are thus green.
2	Click on the "Simulation - next step " icon on the toolbar.	The arrow next to the relay is flipped so it point upwards.

H.1.11 Operators panel

H.1.12 TC-10: Create track section

H.1.12.1 Description

This test case tests UC-34 which is used when a track section is added to an operators panel.

H.1.12.2 Precondition

A project has been created(UC-46).

H.1.12.3 Postcondition

A track section has been created and added to the operators panel. The track section is initially free.

H.1.12.4 Test procedure

#	Procedure	Expected result
1	Access the operators panel using CTRL+0.	The operators panel is displayed.
2	Choose the "Add track section" icon in the toolbar and choose where the track section should be placed by clicking on the operators panel.	The dialog box "New Track Section" becomes visible.
3	Enter "A12" in the "Name" field.	
4	Enter "103" in the "Field" field.	
5	Type "4" in the "Level" field.	
6	Select the radiobutton "Right".	
7	Set the number of total contacts to "20".	
8	Set the number of upper contacts to "11".	
9	Enter the description "Odense".	
10	Click "Ok".	A track section named "A12" is placed on the operators panel.

H.1.13 TC-11: Create point

H.1.13.1 Description

This test case tests UC-35 which is used when a point is added to an operators panel.

H.1.13.2 Precondition

A project has been created(UC-46).

H.1.13.3 Postcondition

A point has been created and added to the operators panel. The point is initially free.

H.1.13.4 Test procedure

#	Procedure	Expected result
1	Access the operators panel using CTRL+0.	The operators panel is displayed.
2	Choose the "Add point" icon in the toolbar and choose where the point should be placed by clicking on the operators panel.	The dialog box "New Point" becomes visible.
3	Enter "B12" in the "Name" field for the relay associated with the track section.	
4	Enter "103" in the "Field" field for the relay associated with the track section.	
5	Type "4" in the "Level" field for the relay associated with the track section.	
6	Select the radiobutton "Right" for the relay associated with the track section.	
7	Set the number of total contacts to "20" for the relay associated with the track section.	
8	Set the number of upper contacts to "7" for the relay associated with the track section.	
9	Enter "P1" in the "Name" field for the relay associated with the plus direction of the point.	
10	Enter "104" in the "Field" field for the relay associated with the plus direction of the point.	
11	Type "4" in the "Level" field for the relay associated with the plus direction of the point.	
12	Select the radiobutton "Left" for the relay associated with the plus direction of the point.	

#	Procedure	Expected result
13	Set the number of total contacts to ”20” for the relay associated with the plus direction of the point.	
14	Set the number of upper contacts to ”7” for the relay associated with the plus direction of the point.	
15	Enter ”104” in the ”Field” field for the relay associated with the minus direction of the point.	
16	Type ”4” in the ”Level” field for the relay associated with the minus direction of the point.	
17	Select the radiobutton ”Right” for the relay associated with the minus direction of the point.	
18	Set the number of total contacts to ”20” for the relay associated with the minus direction of the point.	
19	Set the number of upper contacts to ”10” for the relay associated with the minus direction of the point.	
20	Click ”Ok”.	A point named ”B12” is placed on the operators panel.

H.1.14 TC-12: Assemble track sections and points

H.1.14.1 Description

This test case tests UC-36 which is used when two track sections or a track section and a point is connected.

H.1.14.2 Precondition

A project has been created(UC-46), and two track sections have been created.

H.1.14.3 Postcondition

Two track sections have been assembled.

H.1.14.4 Test procedure

#	Procedure	Expected result
1	Access the operators panel using CTRL+0.	The operators panel is displayed.
1	Press and hold the "Ctrl" key on the keyboard.	All ends on the diagram are shown with green circles, since they all allow more connections to be made to them.
2	Move the cursor on top of the leftmost green circle on the track section displayed on the operators panel, and press and hold the left mouse button.	The ends on the track section changes to red, since a component cannot be assembled to itself.
3	Move the cursor on top of the the green circle on the plus direction of the point displayed on the diagram.	
5	Release the left mouse button.	The track section is lengthened to reach the point.

H.1.15 TC-13: Simulate train

H.1.15.1 Description

This test case tests UC-42, UC-43, UC-44 which is used when a train is simulated. The train is simulated as having a length of appoximately 2 track sections.

H.1.15.2 Precondition

A project has been created. 4 track sections and two points have been created. The track sections are connected as in stenstrup.ris which can be seen on the CD.

Point 01 is switched to the minus direction and point 02 is switched to the plus direction. The train is started at track section ”B12”.

H.1.15.3 Postcondition

The train has left the station and all track sections and points are thus free.

H.1.15.4 Test procedure

#	Procedure	Expected result
1	Right-hand click on track section ”B12” and choose ”Start train from here”.	Track section ”B12” is colored black indicating that the track section has been occupied.
2	Click on the ”Move front of train” icon in the toolbar.	Point ”02” is colored black indicating that the track section has been occupied.
3	Click on the ”Move front of train” icon in the toolbar.	Track section ”04” is colored black indicating that the track section has been occupied.
4	Click on the ”Move rear of train” icon in the toolbar.	Track section ”B12” is colored blue indicating that the track section has been freed.
5	Click on the ”Move front of train” icon in the toolbar.	Point ”01” is colored black indicating that the track section has been occupied.
6	Click on the ”Move rear of train” icon in the toolbar.	Point ”03” is colored blue indicating that the track section has been freed.
7	Click on the ”Move front of train” icon in the toolbar.	Track section ”A12” is colored black indicating that the track section has been occupied.
8	Click on the ”Move rear of train” icon in the toolbar.	Track section ”04” is colored blue indicating that the track section has been freed.
9	Click on the ”Move rear of train” icon in the toolbar.	Point ”01” is colored blue indicating that the track section has been freed.
10	Click on the ”Move rear of train” icon in the toolbar.	Track section ”A12” is colored blue indicating that the track section has been freed.

H.1.16 Stenstrup

H.1.17 Description

In this section a Danish railway station called Stenstrup station will be tested. The specific test case tests train route 2 on Stenstrup station. Train route 2 is the procedure when a train enters track section "A12" and stops at track section "02". The details can be seen by opening stenstrup.ris in the application both placed on the CD.

Stenstrup station is one of the least complicated stations since it only has two tracks. Diagrams for Stenstrup station made by Banedanmark is used to create Stenstrup station in the application. The diagrams only display the circuit for when a train arrives from Odense. For this reason contacts used when a train arrives from Svendborg are omitted, since they will never change state because the associated relays are on the diagrams for when a train approaches the station from Svendborg. Components not implemented in the application, are also omitted e.g. relays for protection against false voltage.

Stenstrup station uses an old notation concerning plus and minus direction for the points. For this reason the circuit has been altered to translate the old notation to the new notation used in the application. Stenstrup station furthermore uses two types of ID's to uniquely identify relays. An old notation using one number to identify a relay, and a new notation using three parameters to identify a relay. The application only uses the new notation so relays with the old notation on the Stenstrup diagrams have been delegated the three parameters *level*, *field* and *position* described in the domain description. These parameters are determined by looking on the diagrams for the rack of relays. Internal resistance is not implemented in the application and thus two circuit constructions on the diagram for the release of the train route have been modified.

H.1.18 Precondition

stenstrup.ris has been loaded in the application. The stenstrup.ris file can be found on the CD.

H.1.19 Post condition

The expected state of the circuit is described using Banedanmark's functional diagrams. The ID of the relay, i.e. the location of the relay in the rack of relays, is compactly described in curly brackets {level, field, position}. The type of the relay will be abbreviated so RR means regular relay, SCR means steel core relay and IR means interface relay. SCR{3, 5, r} denotes thus a steel core relay located at level 3 field 5 in the right position.

One row in table H.1 to H.8 describes a state in the circuit, i.e. the test case comprise 29 states in the circuit. State -1 is the state where no current is applied.

State	Button	RR{3,3,l}	RR{3,7,r}	SCR{2,3,l}
-1	Open	↓	↓	↑
0	Open	↓	↓	↑
1	Open	↓	↓	↑
2	Open	↓	↓	↑
3	Open	↓	↓	↑
4	Open	↓	↓	↑
5	Open	↓	↓	↑
6	Open	↓	↓	↑
7	Closed	↓	↓	↑
8	Closed	↓	↑	↑
9	Closed	↑	↑	↑
10	Open	↑	↑	↑
11	Open	↑	↓	↓
12	Open	↑	↓	↓
13	Open	↑	↓	↓
14	Open	↑	↓	↓
15	Open	↓	↓	↓
16	Open	↓	↓	↓
17	Open	↓	↓	↓
18	Open	↓	↓	↓
19	Open	↓	↓	↓
20	Open	↓	↓	↓
21	Open	↓	↓	↓
22	Open	↓	↓	↓
23	Open	↓	↓	↓
24	Open	↓	↓	↑
25	Open	↓	↓	↑
26	Open	↓	↓	↑
27	Open	↓	↓	↑

Table H.1: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	SCR{4,7,r}	RR{2,16,l}	RR{1,15,r}	RR{2,15,l}
-1	↑	↓	↓	↓
0	↑	↓	↓	↓
1	↑	↑	↓	↑
2	↑	↑	↓	↑
3	↑	↑	↓	↑
4	↑	↑	↓	↑
5	↑	↑	↓	↑
6	↑	↑	↓	↑
7	↑	↑	↓	↑
8	↑	↑	↓	↑
9	↑	↑	↓	↑
10	↓	↑	↓	↑
11	↓	↑	↓	↑
12	↓	↓	↑	↓
13	↓	↓	↑	↓
14	↓	↓	↑	↓
15	↓	↓	↓	↓
16	↓	↑	↓	↑
17	↓	↑	↓	↑
18	↓	↑	↓	↑
19	↓	↑	↓	↑
20	↓	↑	↓	↑
21	↓	↑	↓	↑
22	↓	↑	↓	↑
23	↑	↑	↓	↑
24	↑	↑	↓	↑
25	↑	↑	↓	↑
26	↑	↑	↓	↑
27	↑	↑	↓	↑

Table H.2: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	RR{1,15,l}	RR{3,9,l}	RR{4,9,l}	RR{5,9,l}
-1	↓	↓	↓	↓
0	↓	↓	↓	↓
1	↓	↓	↓	↓
2	↓	↓	↓	↓
3	↓	↓	↓	↓
4	↓	↓	↓	↓
5	↓	↓	↓	↓
6	↓	↓	↓	↓
7	↓	↓	↓	↓
8	↓	↓	↓	↓
9	↓	↓	↓	↓
10	↓	↓	↓	↓
11	↑	↓	↓	↓
12	↑	↓	↓	↑
13	↑	↓	↓	↑
14	↑	↓	↓	↑
15	↓	↓	↓	↑
16	↓	↓	↓	↑
17	↓	↑	↓	↑
18	↓	↑	↓	↑
19	↓	↑	↓	↑
20	↓	↑	↓	↑
21	↓	↑	↓	↑
22	↓	↑	↑	↑
23	↓	↑	↑	↓
24	↓	↑	↑	↓
25	↓	↓	↑	↓
26	↓	↓	↓	↓
27	↓	↓	↓	↓

Table H.3: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	RR{2,9,l}	RR{2,9,r}	RR{2,10,l}	RR{2,10,r}
-1	↓	↓	↓	↓
0	↓	↓	↓	↓
1	↑	↓	↓	↑
2	↑	↓	↓	↑
3	↑	↓	↓	↑
4	↑	↓	↓	↓
5	↑	↓	↑	↓
6	↑	↓	↑	↓
7	↑	↓	↑	↓
8	↑	↓	↑	↓
9	↑	↓	↑	↓
10	↑	↓	↑	↓
11	↑	↓	↑	↓
12	↑	↓	↑	↓
13	↑	↓	↑	↓
14	↑	↓	↑	↓
15	↑	↓	↑	↓
16	↑	↓	↑	↓
17	↑	↓	↑	↓
18	↑	↓	↑	↓
19	↑	↓	↑	↓
20	↑	↓	↑	↓
21	↑	↓	↑	↓
22	↑	↓	↑	↓
23	↑	↓	↑	↓
24	↑	↓	↑	↓
25	↑	↓	↑	↓
26	↑	↓	↑	↓
27	↑	↓	↑	↓

Table H.4: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	RR{4,3,l}	RR{4,3,r}	RR{4,4,l}	RR{4,4,r}
-1	↓	↓	↓	↓
0	↓	↓	↓	↓
1	↑	↑	↑	↑
2	↑	↑	↑	↑
3	↑	↑	↑	↑
4	↑	↑	↑	↑
5	↑	↑	↑	↑
6	↑	↑	↑	↑
7	↑	↑	↑	↑
8	↑	↑	↑	↑
9	↑	↑	↑	↑
10	↑	↑	↑	↑
11	↑	↑	↑	↑
12	↑	↑	↑	↑
13	↑	↑	↑	↑
14	↑	↑	↑	↑
15	↑	↑	↑	↑
16	↓	↑	↑	↑
17	↓	↑	↑	↑
18	↓	↑	↑	↑
19	↓	↑	↑	↑
20	↓	↑	↑	↑
21	↑	↓	↑	↑
22	↑	↓	↑	↑
23	↑	↓	↑	↑
24	↑	↓	↑	↑
25	↑	↓	↑	↑
26	↑	↓	↑	↑
27	↑	↓	↑	↑

Table H.5: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	RR{1,8,l}	RR{1,8,r}	IR{2,1,l}	IR{2,1,r}
-1	↓	↓	↑	↓
0	↓	↓	↑	↓
1	↑	↑	↑	↓
2	↑	↑	↑	↓
3	↑	↑	↑	↓
4	↑	↑	↑	↓
5	↑	↑	↑	↓
6	↑	↑	↑	↓
7	↑	↑	↑	↓
8	↑	↑	↑	↓
9	↑	↑	↑	↓
10	↑	↑	↑	↓
11	↑	↑	↑	↓
12	↑	↑	↑	↓
13	↑	↑	↑	↓
14	↓	↑	↑	↓
15	↓	↑	↑	↓
16	↑	↑	↑	↓
17	↑	↑	↑	↓
18	↑	↑	↑	↓
19	↑	↑	↑	↓
20	↑	↑	↑	↓
21	↑	↑	↑	↓
22	↑	↑	↑	↓
23	↑	↑	↑	↓
24	↑	↑	↑	↓
25	↑	↑	↑	↓
26	↑	↑	↑	↓
27	↑	↑	↑	↓

Table H.6: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	IR2,2,left	IR2,2,right	IR5,1,right	IR5,2,right
-1	↓	↑	↑	↑
0	↓	↑	↑	↑
1	↓	↑	↑	↑
2	↓	↑	↑	↑
3	↑	↓	↑	↑
4	↑	↓	↑	↑
5	↑	↓	↑	↑
6	↑	↓	↑	↑
7	↑	↓	↑	↑
8	↑	↓	↑	↑
9	↑	↓	↑	↑
10	↑	↓	↑	↑
11	↑	↓	↑	↑
12	↑	↓	↑	↑
13	↑	↓	↑	↑
14	↑	↓	↑	↑
15	↑	↓	↓	↑
16	↑	↓	↓	↑
17	↑	↓	↓	↑
18	↑	↓	↓	↑
19	↑	↓	↓	↑
20	↑	↓	↑	↓
21	↑	↓	↑	↓
22	↑	↓	↑	↓
23	↑	↓	↑	↓
24	↑	↓	↑	↓
25	↑	↓	↑	↓
26	↑	↓	↑	↓
27	↑	↓	↑	↓

Table H.7: *Expected states of the circuit for Stenstrup station c.f. test case "Stenstrup".*

State	IR5,3,right	IR5,4,right	IR1,6,right	IR1,7,right
-1	↑	↑	↑	↑
0	↑	↑	↑	↑
1	↑	↑	↑	↑
2	↑	↑	↑	↑
3	↑	↑	↑	↑
4	↑	↑	↑	↑
5	↑	↑	↑	↑
6	↑	↑	↑	↑
7	↑	↑	↑	↑
8	↑	↑	↑	↑
9	↑	↑	↑	↑
10	↑	↑	↑	↑
11	↑	↑	↑	↑
12	↑	↑	↑	↑
13	↑	↑	↓	↑
14	↑	↑	↓	↑
15	↑	↑	↑	↑
16	↑	↑	↑	↑
17	↑	↑	↑	↑
18	↑	↑	↑	↑
19	↑	↑	↑	↑
20	↑	↑	↑	↑
21	↑	↑	↑	↑
22	↑	↑	↑	↑
23	↑	↑	↑	↑
24	↑	↑	↑	↑
25	↑	↑	↑	↑
26	↑	↑	↑	↑
27	↑	↑	↑	↑

Table H.8: *Expected states of the circuit for Stenstrup station c.f. test case ”Stenstrup”.*

H.1.20 Test procedure

#	Procedure	Expected result
1	Verify the states of the relays according to state (-1).	The state of the circuit matches the expected state -1 in table H.1 to H.8.
2	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 0 in table H.1 to H.8.
3	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 1 in table H.1 to H.8.
4	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 2 in table H.1 to H.8.
5	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 2 in table H.1 to H.8.
6	Switch point 02 to the minus direction.	Point 02 is completely switched to the minus direction.
7	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 3 in table H.1 to H.8.
8	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 4 in table H.1 to H.8.
9	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 5 in table H.1 to H.8.
10	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 6 in table H.1 to H.8.
11	Push entry button 1 on the signal control circuit.	The entry button icon changes.
12	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 7 in table H.1 to H.8.

#	Procedure	Expected result
13	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 8 in table H.1 to H.8.
14	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 9 in table H.1 to H.8.
15	Release entry button 1 on the signal control circuit.	The button icon changes.
16	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 10 in table H.1 to H.8.
17	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 11 in table H.1 to H.8.
18	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 12 in table H.1 to H.8.
19	Start the train from track section A12.	Track section A12 is colored black indicating that the train has occupied track section A12.
20	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 13 in table H.1 to H.8.
21	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 14 in table H.1 to H.8.
22	Occupy point 01 and free track section A12 with the train.	Point 01 is colored black and track section A12 is colored blue.
23	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 15 in table H.1 to H.8.
24	Click the "Next step" icon on the toolbar.	The state of the circuit matches the expected state 16 in table H.1 to H.8.

#	Procedure	Expected result
25	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 17 in table H.1 to H.8.
26	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 18 in table H.1 to H.8.
27	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 19 in table H.1 to H.8.
28	Occupy track section 02 and free point 01 with the train.	Point 01 is colored blue and track section 02 is colored black.
29	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 20 in table H.1 to H.8.
30	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 21 in table H.1 to H.8.
31	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 22 in table H.1 to H.8.
32	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 23 in table H.1 to H.8.
33	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 24 in table H.1 to H.8.
34	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 25 in table H.1 to H.8.
35	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 26 in table H.1 to H.8.
36	Click the ”Next step” icon on the toolbar.	The state of the circuit matches the expected state 27 in table H.1 to H.8.

H.2 Exploratory test

Exploratory testing allows a tester to ”test on impulse” which sometimes leads to detection of defects that would not have been found with planned test cases.

Usually exploratory tests are not documented in much detail. In this case though each test is documented briefly. The documentation is used to analyze the test coverage.

#	Procedure	Expected result
1	Create relay, name more than 10 characters.	Relay is not created.
2	Create relay, name = 10 characters.	Relay is created.
3	Create relay, level < 0.	Relay is not created.
4	Create relay, level = 0.	Relay is created.
5	Create relay, level > 1000.	Relay is not created.
6	Create relay, level is a string.	Relay is not created.
7	Create relay, field < 1.	Relay is not created.
8	Create relay, field = 1.	Relay is created.
9	Create relay, field > 5.	Relay is not created.
10	Create relay, field is a string.	Relay is not created.
11	Display coil pins.	Coil pins are displayed.
12	Create button, ID is a string.	Fuse is not created.
13	Create resistor, ID is a string.	Relay is not created.
14	Create fuse, ID is a string.	Fuse is not created.
15	Create fuse, enter greatest integer possible.	An appropriate message is displayed. A fuse is not created.
16	Connect button in circuit, push and release.	The button does not carry current when it is released. The button carries current when it is pushed.
17	Connect fuse in circuit, break and repair.	The fuse does not carry current when it is broken. The fuse carries current when it is repaired.
18	Connect lamp in circuit, break and repair.	The lamp does not carry current when it is broken. The lamp carries current when it is repaired.
19	Connect button, fuse, lamp and resistor in circuit, break and reset circuit.	When the circuit is reset the components are repaired.

#	Procedure	Expected result
20	Create fuse with an ID of an already existing fuse.	An appropriate error message is displayed.
21	Add two negative poles and connect them.	The negative poles cannot be connected.
22	Add component and save project. Load project to validate that the component has been saved. Repeat process.	The added components are saved.
23	Assemble two points.	An error message is displayed.
24	Add two negative poles and connect them.	The negative poles cannot be connected.
25	Drive train into a point in an intermediate switch.	The train crashes.
26	Drive train into a track section already occupied.	The train crashes.
27	Start train from track sections where no ends are assembled to other track sections/points.	The function is disabled and cannot be selected.
28	Apply current on both sides of steel core relay.	An error message is displayed.
29	Rename diagram, close project and open project again.	The diagram has the exptected name.
30	Display interface coil pins in a diagram.	An error message is displayed.
31	Switch point while it is occupied.	An error message is displayed.
32	Switch point while it is occupied by a train.	An error message is displayed.
33	Give relays associated with a point the same location in the rack of relays.	An error message is displayed.
34	Set description of component in the properties frame. Validate that the description is saved.	The description of the component is saved.
35	Start train from point with one available end.	The train occupies the point.
36	Start train from minus direction of plus switched point.	An error message is displayed.
37	Open a file that is not *.ris.	An error message is displayed.

Appendix I

Additional Extensions

The current version of the application supports the opportunity to create and display diagrams and an operators panel as they are depicted on the paper diagrams. Furthermore the application supports features which allows the user to interact with it and simulate current propagate through the electrical circuit in different ways.

Suggestions for additional features in the application has been divided into 4 different areas: circuit, operators panel, train and GUI.

I.1 Circuit

- In the current version of the application the display symbol (aux. relay, replicate relay, etc.) of a relay is determined when the coil pins are added to a diagram. This display symbol should be determined when a relay is created.

- Not all types of electrical components are implemented in the current version of the application, e.g. the relay for protection against false voltage. For the application to be able to fully translate a paper diagram to an application diagram all types of electrical components must be implemented.

- In the current version a relay cannot be deleted from the model level. If a relay becomes redundant it is useful to be able to delete it.
- The application would become more usable if it was possible to change position of a relay in the rack of relays.
- For the same reason as the above item, it should be possible to change the name of a component after it has been created.
- In the domain it is possible to create a button that is initially pushed, and thus it should be possible in the application.
- Having a "history"-list, displaying which relays change state in each step would provide more overview, when several relays change state in the same step
- It would improve the usability of the application if the properties pop-up window displayed more information. The properties for a relay could e.g. show the number of connected contacts and so forth.
- Besides regular track sections and points, cross-track sections (where two track sections intersect) can be used on a station. Implementing a cross track section would thus be an advantage.

I.2 Operators panel functionality

The implemented operators panel does not support the functionality of the operators panel in the domain. Below are listed some of the extensions that can be added to the existing functionality of the operators panel.

- Disassemble track sections/points.
- Remove track sections/points.
- Move track sections that are assembled to other track sections.
- Display signals (with multiple lamps attached).
- Display buttons an only allow button interaction through the operators panel.

I.3 Train

- The current version of the simulation only allows one train on a station at a time. Several train routes can be locked at the same time, given that they are not conflicting, and thus several trains can be on a station at a time. Simulating more than one train on a station at the same time would thus be an improvement of the application.

I.4 GUI

Almost all of the components displayed on the paper diagrams have been implemented in the application. Even though this is the case there are many smaller extensions to the diagram functionality, which makes the application more useable and the output more in accordance with the domain. Some of these extensions are as follows:

- In accordance with Banedanmark's diagrams it should be possible to display lamps "horizontally" instead of vertically.
- In accordance with Banedanmark's diagrams it should be possible to connect terminals displayed on different diagrams.
- Wires should not be painted on top of components.
- The application would become more usable if it is displayed graphically when a component is broken.
- It should only be allowed for the user to choose to display a contact that is not already displayed on a diagram.
- The size of a diagram is currently limited by the resolution of the screen. Adding scrollbars to allow larger diagrams would thus be an advantage. Alternatively a zoom-function could be implemented.

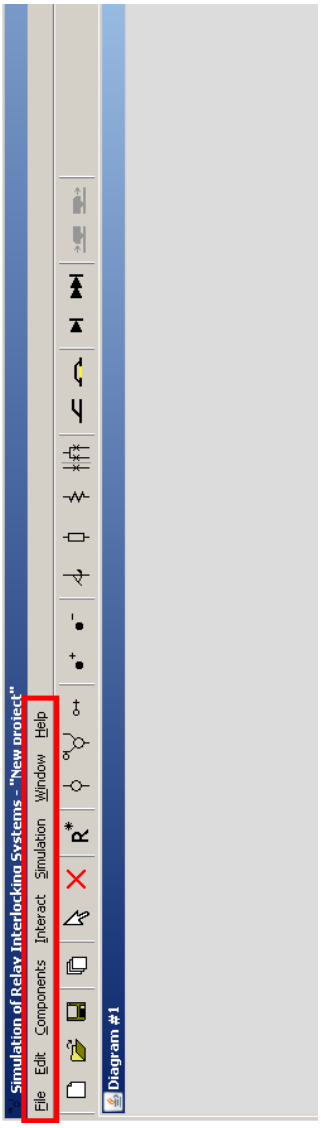
Appendix J

Screen shots

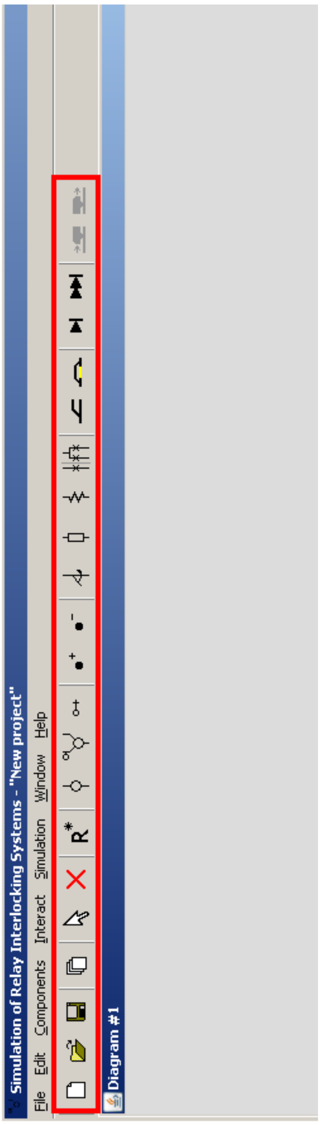
This appendix contains four screenshots of the application:

1. The menu bar that provides almost all tools and features available in the application.
2. The toolbar that provides the most important and used tools of the application.
3. An example of how a circuit can be displayed and connected on a diagram in the application. This diagram contains a positive pole, two negative poles, 4 set of coil pins, 21 contacts, 4 fuses, 4 resistors and 4 lamps. The current is turned on (the green lines) and the red and yellow lamps are turned on.
4. An example of how the operators panel of a project can be displayed in the application. This operators panel contains 14 track sections (A-N) of which 5 are points (01-05). All points are set in the plus direction.

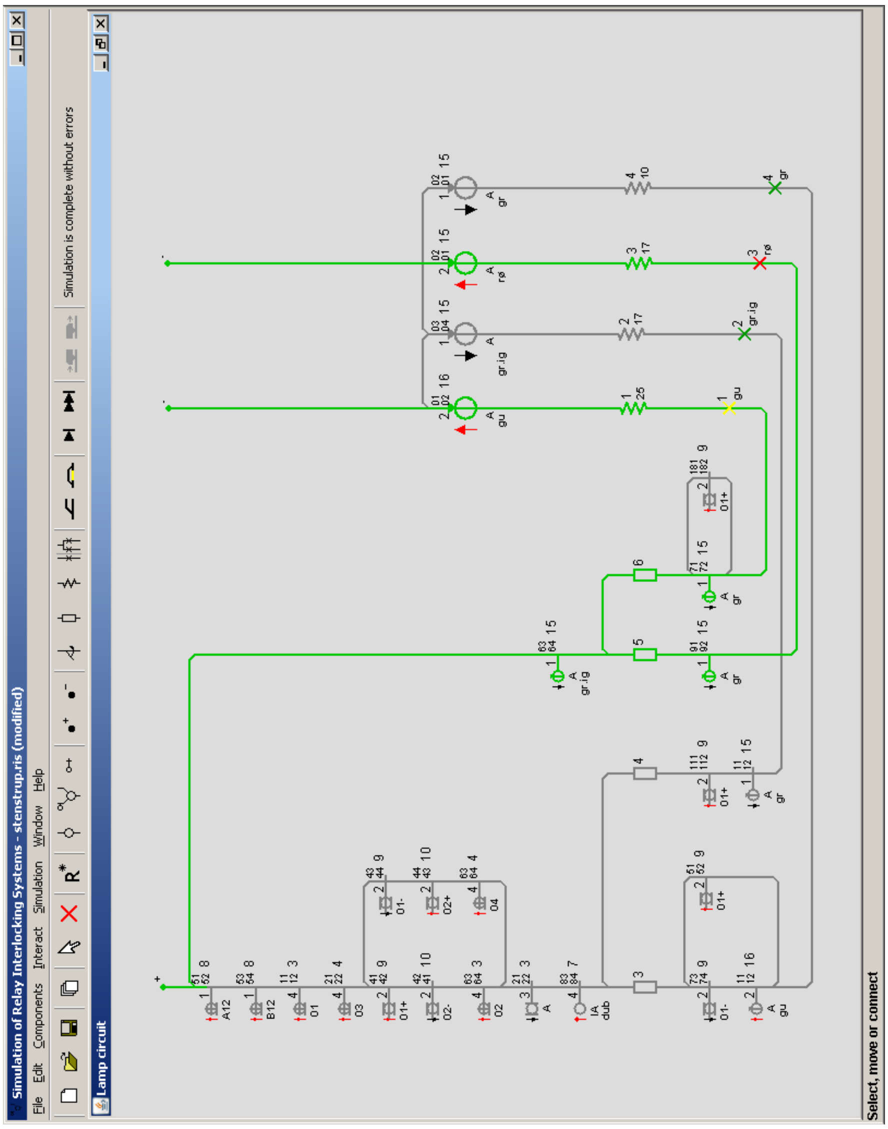
J.1 Menu bar


Simulation of Relay Interlocking Systems - "New project"
File Edit Components Interact Simulation Window Help
Diagram #1

J.2 Toolbar


Simulation of Relay Interlocking Systems - "New project"
File Edit Components Interact Simulation Window Help
Diagram #1

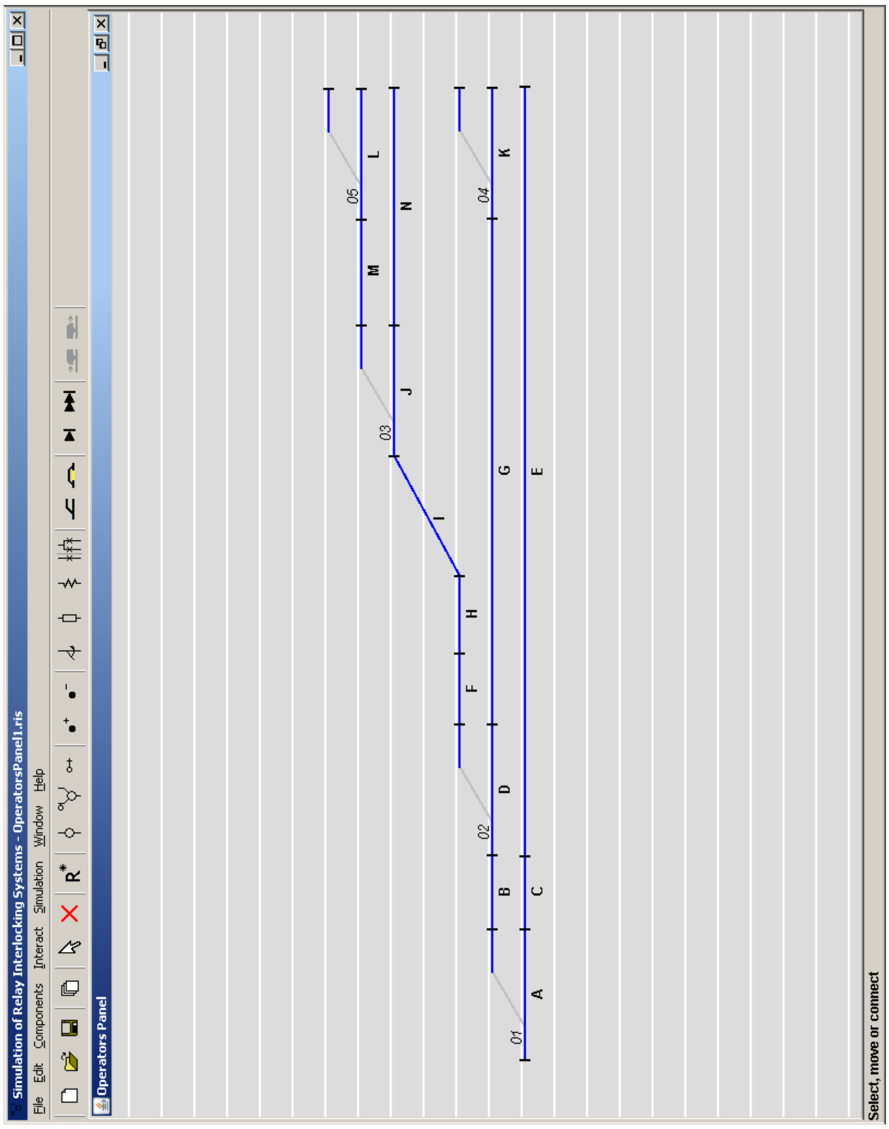
J.3 Diagram


Simulation of Relay Interlocking Systems - stenstrup.ris (modified)
File
Edit
Components
Interact
Simulation
Window
Help
Lamp circuit

Simulation is complete without errors

Select, move or connect

J.4 Operators panel


Simulation of Relay Interlocking Systems - OperatorsPanel1.ris
File Edit Components Interact Simulation Window Help
Operators Panel
01
02
03
04
05
A
B
C
D
E
F
G
H
I
J
K
L
M
N
Select, move or connect

Appendix K

User's guide

K.1 Introduction

This application provides a tool that makes it easier to create and simulate a virtual relay interlocking system.

This user's guide describes how to add, remove, interact with and edit the relay interlocking system together. It is also described how to simulate the propagation of current through the electrical circuit and the movement of the train on the station.

K.2 Requirements

To be able to run the application Java's JRE 1.6 must be installed. It can be downloaded from the following site on the internet:
"http://java.sun.com/javase/downloads/index.jsp". The CD provides the JRE 1.6 for Windows and Sun platforms.

It is recommended to use a monitor resolution on at least $1024 * 768$ pixels to be able to display more than one diagram at a time.

K.3 Installation and execution

The application can be run without having Java's JRE 1.6 installed on the computer (only for Windows and Sun platforms):

K.3.1 Windows users

- To run the application directly from the CD, execute SimulatorWin.bat from the root of the CD.
- To run the application from the computer, copy the Windows folder and the Simulator.jar and SimulatorWin.bat files to a folder on the computer. Afterwards the application is run by executing the SimulatorWin.bat file directly from its new location.

K.3.2 Sun users

- To run the application directly from the CD, execute SimulatorSun.sh from the root of the CD.
- To run the application from the computer, copy the Sun folder and the Simulator.jar and SimulatorSun.sh files to a folder on the computer. Afterwards the application is run by executing the SimulatorSun.sh file directly from its new location.

K.4 How to use the application

The first part of creating a relay interlocking system is to create a project. Each project contains all information about a single station, e.g. the track sections, points, relays, buttons, fuses, etc. All of these components are displayed on either a diagram or on the operators panel.

Almost all features in the application are available through the menu bar in the top of the application frame. Below this menu bar is a toolbar that provides the most important and most frequently used features. The features for an individual component are available through a short cut menu that can be accessed by right-clicking the component. A final access to the features is to use the shortcut keys written in the menu bars.

In the buttom of the main window there is a message bar. It is used to inform about errors in the input or use of the application, to inform about what has just happened or which tool is currently active.

The following sections describe how the components are created and used, and the more specific use of the application.

K.4.1 New project

When the application is started a project is created. Afterwards, a new project can be created by either:

- selecting "Project" in the "File" → "New" menu,
- clicking the "New project" on the toolbar or
- using the shortcut key CTRL + P.

When a project is created a diagram and an operators panel are created automatically .

K.4.2 Diagrams

The diagrams of a project display a segment of the entire electrical circuit by displaying any number of components (coil pins, contacts, buttons, fuses, resistors, lamps, negative poles and positive poles) and how they are individually connected. A project can consist of any number of diagrams.

The "Window" menu in the menu bar contains a list of the names of all created diagrams. When pressing either of the diagram items in this menu the associated diagram is displayed. Created diagrams can be renamed using the "Rename diagram" item in the "Window" menu.

K.4.2.1 New diagram

A new diagram can be created by either:

- selecting "Diagram" in the "File" → "New" menu,
- clicking the "New diagram" button on the toolbar or
- using the shortcut key CTRL + P.

The following sections describe all components that can be displayed on a diagram, how the components are used and how the user can interact with an electrical circuit through the diagrams.

K.4.2.2 Positive pole

At least one positive pole is required on a diagram to create a functional electrical circuit segment. Without a positive pole it is not possible to add current to the other components on the diagram. A positive pole is displayed by either:

- selecting "Display positive pole" in the "Components" menu,
- clicking the "Display positive pole" button on the toolbar or
- using the shortcut key CTRL + SHIFT + P.

A positive pole can have any number of wires connected.

K.4.2.3 Negative pole

At least one negative pole is required on a diagram to create a functional electrical circuit segment. Without a negative pole it is not possible to add current to the other components on the diagram. A positive pole is displayed by either:

- selecting "Display negative pole" in the "Components" menu,
- clicking the "Display negative pole" button on the toolbar or
- using the shortcut key CTRL + SHIFT + N.

A negative pole can have any number of wires connected.

K.4.2.4 Create a relay

Before any part of a relay (coil pins or contacts) can be displayed on a diagram the relay has to be created and placed in the rack of relays. This is done by first:

- selecting "Relay" in the "Components" → "Create" menu,
- clicking the "Create new relay" button on the toolbar or
- using the shortcut key CTRL + SHIFT + Y.

In the dialog box that pops up, type in the information used to create a new relay. The name can as the only field be omitted. All other fields must be filled out:

Name The name of the relay (optional).

Type Choose either of the items: Regular relay or steel core relay.

Field The field to use in the rack of relays ($[1, \ldots, 5]$).

Level The level to use in the rack of relays ($[1, \ldots, 1000]$).

Left/right The position in the rack of relays.

Total contacts The size of the relay, specified by the total amount of contacts.

Upper contacts How many of the contacts that must be upper contacts.

By default the first free location in the rack of relay is suggested. When all required fields have been filled out the OK button is pressed and the relay is created and placed in the rack of relays. For confirmation purposes a message is printed in the message bar. The relay can only be created if the specified location in the rack of relay is available.

K.4.2.5 Coil pins

To be able to display the coil pins on a diagram the relay on which the pins are located, must be created. Depending on the type of relay, different tools must be activated. To display the coil pins on a regular relay activate the tool by either:

- selecting "Display regular coil pins" in the "Components" menu,
- clicking the "Display regular coil pins" button on the toolbar or
- using the shortcut key CTRL + SHIFT + R.

To display the coil pins on a steel core relay activate the tool by either:

- selecting "Display steel core coil pins" in the "Components" menu,
- clicking the "Display steel coil pins" button on the toolbar or
- using the shortcut key CTRL + SHIFT + O.

With the mouse the desired spot for the relay on a diagram is clicked. A dialog box with the following fields appears:

Type The use of the relay (only for regular coil pins).

Field The field of the relay in the rack of relay.

Level The level of the relay in the rack of relays.

Left/right The position of the relay in the rack of relays.

Description A description of the coil pins (optional).

When all required fields have been filled out, the OK button is pressed and the coil pins are displayed on the diagram. A group of coil pins can only be displayed at one diagram at a time.

Each of the coil pins can have a maximum of two wires connected simultaneously.

When the coil pins are displayed there are different ways of interacting with them. All of the following features are accessible by right-clicking on the coil pins:

- Set state of the associated relay to be drawn – even when current is not applied.
- Set state of the associated relay to be dropped – even when current is applied.
- Set coil pins to be non-conducting.

- Repair the coil pins if any of the above features have been applied.
- Flip the displayed coil pins upside down (vertical).

K.4.2.6 Contacts

To be able to display a contact on a diagram the relay on which the contact is located must be created. Activate the tool for displaying a contact by either

- selecting "Display contact" in the "Components" menu,
- clicking the "Display contact" button on the toolbar or
- using the shortcut key CTRL + SHIFT + C.

Specify where on the diagram the contact must be placed by pressing the left mouse button at the desired point. A dialog box with the following fields appears:

Field The field of the associated relay in the rack of relay.

Level The level of the associated relay in the rack of relays.

Left/right The position of the associated relay in the rack of relays.

Upper contact If an upper contact is to be placed on the diagram, select the "Upper contact" radio button and choose the desired contact number in the drop-down box.

Lower contact If a lower contact is to be placed on the diagram, select the "Lower contact" radio button and choose the desired contact number in the drop-down box.

Description A description of the contact (optional).

When all required fields have been filled out, the OK button is pressed and the contact is displayed on the diagram. Each contact can only be displayed on one diagram at a time.

Each pin on a contact can have a maximum of two wires connected simultaneously.

When the contact is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the contact:

- Set the contact to be non-conducting even when it should not be (simulating a defective contact).
- Set the contact to be conductive even when it should not be (simulating a defective contact).
- Repair the contact if any of the above features have been applied.
- Flip the displayed contact upside down (vertical).

K.4.2.7 Buttons

Activate the tool for displaying a button by either:

- selecting "Add button" in the "Components" menu,
- clicking the "Add button" button on the toolbar or
- using the shortcut key CTRL + SHIFT + B.

Specify where on the diagram the button must be placed by clicking the left mouse button at the desired point. A dialog box with the following fields appears:

Name The name of the button (optional).

ID The ID of the button.

Description A description of the button (optional).

Only the ID field has to be specified, the other fields can be omitted. Click the OK button when the required fields have been filled out. If a button has previously been created with the specified ID the same button is reused and displayed, otherwise a new button is created and displayed. A button can only be displayed on one diagram at a time.

Each pin on a button can have a maximum of two wires connected simultaneously.

When the button is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the button:

- Push the button.
- Release the button.
- Flip the displayed contact upside down (vertical).

Additionally the button can be pushed/released by selecting the item with the button's ID in the "Interact" → "Push button" and "Release button" menus.

K.4.2.8 Fuses

Activate the tool for displaying a fuse by either

- selecting "Add fuse" in the "Components" menu,
- clicking the "Add fuse" button on the toolbar or
- using the shortcut key CTRL + SHIFT + F.

Specify where on the diagram the fuse must be placed by pressing the left mouse button at the desired point. A dialog box with the following fields appears:

Name The name of the fuse (optional).

ID The ID of the fuse.

Description A description of the fuse (optional).

Only the ID field has to be specified, the other fields can be omitted. Click the OK button when the required fields have been filled out. If a fuse has previously been created with the specified ID the same fuse is reused and displayed, otherwise a new fuse is created and displayed. Each fuse can only be displayed on one diagram at a time.

Each pin on a fuse can have a maximum of two wires connected simultaneously.

When the fuse is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the fuse:

- Set the fuse to be non-conducting to simulate the metal wire inside the fuse being melted.
- Repair the fuse by making it conductive.

K.4.2.9 Resistors

Activate the tool for displaying a resistor by either:

- selecting "Add resistor" in the "Components" menu,
- clicking the "Add resistor" button on the toolbar or
- using the shortcut key CTRL + SHIFT + E.

Specify where on the diagram the resistor must be placed by pressing the left mouse button at the desired point. A dialog box with the following fields appears:

Name The name of the resistor (optional).

ID The ID of the resistor.

Description A description of the resistor (optional).

Only the ID field has to be specified, the other fields can be omitted. Click the OK button when the required fields have been filled out. If a resistor has previously been created with the specified ID the same resistor is reused and displayed, otherwise a new resistor is created and displayed. Each resistor can only be displayed on one diagram at a time.

Each pin on a resistor can have a maximum of two wires connected simultaneously.

When the resistor is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the resistor:

- Set the resistor to be non-conducting to simulate over-current.
- Repair the resistor by making it conductive.

K.4.2.10 Lamps

Activate the tool for displaying a lamp by either:

- selecting "Add lamp" in the "Components" menu,
- clicking the "Add lamp" button on the toolbar or
- using the shortcut key CTRL + SHIFT + L.

Specify where on the diagram the lamp must be placed by pressing the left mouse button at the desired point. A dialog box with the following fields appears:

Name The name of the lamp (optional).

ID The ID of the lamp.

Color The color of the lamp (green, yellow or red).

Description A description of the lamp (optional).

Only the ID and color fields have to be specified, the other fields can be omitted. Click the OK button when the required fields have been filled out. If a lamp has previously been created with the specified ID the same lamp is reused and displayed, otherwise a new lamp is created and displayed. Each lamp can only be displayed on one diagram at a time.

Each pin on a lamp can have a maximum of one wire connected simultaneously.

When the lamp is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the lamp:

- Add a spare filament to the lamp.
- Remove the spare filament from the lamp.
- Set the main filament on the lamp to be non-conducting to simulate a burn-out.
- Set the spare filament on the lamp to be non-conducting to simulate a burn-out.
- Repair the main filament on the lamp by making it conductive.
- Repair the spare filament on the lamp by making it conductive.
- Flip the displayed lamp upside down (vertical).
- Flip the displayed lamp right to left (horizontal).

K.4.2.11 Connect

If the select/move/connect/assemble tool is not activated it is done by either:

- selecting "Select/move/connect/assemble" in the "Edit" menu,
- clicking the "Select/move/connect/assemble" button on the toolbar or
- using the shortcut key ALT + SHIFT + S.

Afterwards the CTRL key should be pressed. The pins and poles on the active diagram are painted in either a green or red color indicating whether they accept another connection to be made or not. The left mouse button is pressed on top of one of the pins or poles to connect. Now the color of some of the pins on the diagram change from green to red to indicate that they do not accept a connection to be made. While the mouse is moved across the diagram a red line is depicted from the pin on which the mouse was pressed to where the mouse pointer is. Again the red line indicates that it is not possible to make a connection. When the mouse is pointing at a green pin or pole, the line color is switched to green to indicate that this connection is acceptable. To finish the connection release the mouse button. The green line is replaced by a wire. Be aware that the CTRL key must be kept pressed from when the mouse button is pressed until the mouse button is released.

K.4.2.12 Edit components

If the components on the diagrams are not exactly where they are supposed to be, they can be selected and moved. Likewise components can be selected, disconnected and removed if they should no longer be displayed on the current diagram. The select/move/connect/assemble tool is activated by either:

- selecting "Select/move/connect/assemble" in the "Edit" menu,
- clicking the "Select/move/connect/assemble" button on the toolbar or
- using the shortcut key ALT + SHIFT + S.

A component has to be selected before it can be moved or removed. This is done by clicking them individually or dragging a rectangle that spans across all components that must be selected. To select additional components than the ones already selected the SHIFT key must be pressed and held down while selecting as just described. When the selecting is done the SHIFT key should be released.

To move the selected components simply press the mouse button on top of one of the selected components and drag them to where they should be placed and then release the mouse button.

To disconnect and remove the selected components either:

- select "Disconnect and remove selected" from the "Edit" menu,
- click the "Disconnect and remove selected" button on the toolbar or
- use the shortcut key DELETE.

First all selected components are disconnected (including wires) and afterwards the components are removed from the diagram.

K.4.2.13 Simulation

The propagation of current through an electrical circuit can be simulated in two different ways. Either a step by step simulation or a full simulation. The full simulation can be run to get to the final state even after any number of steps has been simulated using the step by step simulation.

The next step in the step by step simulation can be simulated either by:

- selecting "Next step" in the "Simulation" menu,
- clicking the "Simulation - next step" button on the toolbar,
- using the shortcut key CTRL + ALT + N or
- right-clicking anywhere on a diagram and choosing "Next step" from the pop-up menu.

The full simulation can be started by either:

- selecting "Full" in the "Simulation" menu,

- clicking the "Simulation - full" button on the toolbar,
- using the shortcut key CTRL + ALT + S or
- right-clicking anywhere on a diagram and choosing "Full simulation" from the pop-up menu.

The normal state of the station can easily be attained by either:

- selecting "Go to normal state" in the "Simulation" menu or
- right-clicking anywhere on a diagram and choosing "Go to normal state" from the pop-up menu.

The components can be set to the state they initially were in when they were created by selecting "Reset all components" from the "Simulation" menu.

K.4.3 Operators panel

The operators panel of a project displays how track sections and points are assembled and which states they are in; which are occupied or free and which directions are the points switched to. A project can contain only one operators panel. The operators panel is created as soon as a project is created.

The "Window" menu in the menu bar contains a list of the names of all created diagrams. When pressing the "Operators panel" button in the "Windows" menu, the operators panel is displayed on top of all other diagrams.

K.4.3.1 Track sections

Activate the tool for displaying a track section by either:

- selecting "Add track section" in the "Components" menu,
- clicking the "Add track section" button on the toolbar or
- using the shortcut key CTRL + SHIFT + T.

In the dialog box that pops up, type in the information for the relay associated with the track section. The description can as the only field be omitted. All other fields must be filled out:

Name The name of the track section.

Field The field to use for the relay in the rack of relays ($[1, \ldots, 1000]$).

Level The level to use for the relay in the rack of relays ($[0, \ldots, 5]$).

Left/right The position for the relay in the rack of relays.

Total contacts The size of the relay, specified by the total amount of contacts.

Upper contacts How many of the contacts that must be upper contacts.

Description A description of the track section (optional).

By default the first free location in the rack of relay is suggested. When all required fields have been filled out, the OK button is pressed and the relay and track section are created and the relay is placed in the rack of relays. The relay can only be created if the specified location in the rack of relay is available.

Each electrically isolated end on a track section can be assembled to only one other end.

When a track section is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the track:

- Occupy the track section.
- Free the track section.

Additionally a track section can be occupied/freed by selecting the item with the track section's name in the "Interact" → "Occupy track section" and "Free track section" menus.

K.4.3.2 Points

Activate the tool for displaying a point by either:

- selecting "Add point" in the "Components" menu,

- clicking the "Add point" button on the toolbar or
- using the shortcut key CTRL + SHIFT + I.

In the dialog box that pops up, type in the information used to create three new relays. One relay to indicate whether the point is occupied or free, one to indicate whether the point is switched to the plus direction and finally one to indicate whether the point is swithced to the minus direction. The descriptions can be omitted. All other fields must be filled out for each relay:

Name The name of the track section relay and point relays.

Field The field to use for the relays in the rack of relays ($[1, \ldots, 1000]$).

Level The level to use for the relays in the rack of relays ($[0, \ldots, 5]$).

Left/right The position for the relays in the rack of relays.

Total contacts The size of the relays, specified by the total amount of contacts.

Upper contacts How many of the contacts that must be upper contacts.

Description A description of the track section and point relays (optional).

By default the first three free locations in the rack of relay are suggested. When all required fields have been filled out, the OK button is pressed and the relays and point are created and the relays are placed in the rack of relays. The relays can only be created if the specified places in the rack of relay are available.

Each electrically isolated end on a point can be assembled to only one other end.

When a point is displayed there are different ways of interacting with it. All of the following features are accessible by right-clicking on the point:

- Switch the point to the intermediate state or finalize the switch of the point.
- Occupy the point.
- Free the point.
- Flip the displayed point upside down (vertical).
- Flip the displayed point right to left (horizontal).

Additionally a point can be switched by selecting the item with the point's name in the "Interact" → "Switch point" menu. Likewise the point can be occupied/freed by selecting the item with the point's track section name in the "Interact" → "Occupy track section" and "Free track section" menus.

K.4.3.3 Assembly

To assemble a track section with either another track section or a point on the operators panel the select/move/connect/assemble tool must be activated. If it is not active it is done by either:

- selecting "Select/move/connect/assemble" in the "Edit" menu,
- clicking the "Select/move/connect/assemble" button on the toolbar or
- using the shortcut key ALT + SHIFT + S.

Afterwards the CTRL key must be pressed. The ends of the track sections and points are painted by either a green or red color indicating whether they accept an assembly to be made or not. The left mouse button is pressed on top of one of the track section ends to assemble. Now the color of some of the ends on the operators panel change from green to red to indicate that they do not accept an assembly to be made. While the mouse is moved across the operators panel the end on which the mouse was pressed is moved along. When the mouse button is released on top of a green painted end these two ends are assembled. Be aware that the CTRL key must be kept pressed from when the mouse button is pressed until the mouse button is released.

K.4.3.4 Edit components

If the components on the diagrams are not exactly where they are supposed to be, they can be moved. If the select/move/connect/assemble tool is not activated it is done by either

- selecting "Select/move/connect/assemble" in the "Edit" menu,
- clicking the "Select/move/connect/assemble" button on the toolbar or
- using the shortcut key ALT + SHIFT + S.

To move any component simply press the mouse button on top of it and drag it to where it should be placed and then release the mouse button.

K.4.3.5 Train

The movement of a train on a station can be simulated. First it needs to be specified where the train must enter the station. This is done by right-clicking on a track section/point via which the train must enter and selecting "Start train from here" in the pop-up menu. Be aware that the chosen track section/point must have exactly one end that is not assembled with any other end. The end that is not assembled with another end indicates the assembly to the open track. The movement of the train is controlled by either letting the front end of the train move forward to occupy the track section in front of the train or by moving the rear end of the train forward to free the rearmost track section/point occupied by the train. The front of the train is moved by either:

- selecting "Move front of train" in the "Interact" menu,
- clicking the "Move front of train" button on the toolbar,
- using the shortcut key CTRL + ALT + F or
- right-clicking anywhere on a diagram and choosing "Move front of train" from the pop-up menu.

The rear of the train is moved by either:

- selecting "Move rear of train" in the "Interact" menu,
- clicking the "Move rear of train" button on the toolbar,
- using the shortcut key CTRL + ALT + R or
- right-clicking anywhere on a diagram and choosing "Move front of train" from the pop-up menu.

The train is cancelled and removed from the station by either:

- selecting "Cancel train" in the "Interact" menu,
- using the shortcut key CTRL + ALT + C or
- right-clicking anywhere on a diagram and choosing "Cancel train" from the pop-up menu.

Appendix L

Files on CD

The following files have been handed in on a CD together with the thesis:

README.txt A detailed description of the files on the CD.

bachelorthesis.pdf The thesis including source code.

Simulator.jar The application as an executable jar-file.

Javadoc Folder containing Javadoc documenting the code of the application. To access the Javadoc use the index file in the Javadoc folder.

Unix Folder containing a standalone version of the Java 1.6 JRE for Unix platforms.

Windows Folder containing a standalone version of the Java 1.6 JRE for Windows platforms.

SimulatorUnix.sh Script to execute the application on a Unix platform using the provided JRE on the CD. Thus the application can be executed without having Java 1.6 JRE installed locally.

SimulatorWin.bat Script to execute the application on a Windows platform using the provided JRE on the CD. Thus the application can be executed without having Java 1.6 JRE installed locally.

stenstrup.ris An application project for Stenstrup station.

cycle.ris An application project for the cyclic circuit in figure 3.4 at page 32.

cycle2.ris An application project for a cyclic circuit.

notCycle.ris An application project for a circuit that goes through the max amount of states without being a cycle.

example.ris An application project for the step simulation in figures 4.3 to 4.7 starting on page 44.

parallel.ris An application project for the parallel example on figure 4.8 on page 46.

serial.ris An application project for the series connection on figure 4.9 on page 47.

AandB.ris An application project for the "A and B" example on figure 3.1 on page 30.

AorB.ris An application project for the "A or B" example on figure 3.2 on page 30.

wires.ris An application project for the wires example on figure 4.14 on page 73.

Appendix M

Source code

An electronic version of the full thesis including the source code is found on the CD.