Describing, Modelling and Simulating the Copenhagen Metro

Ida Moltke, Matias Sevel Rasmussen & Tor Justesen

DIKU – 2003

Contents

Preface

This report is a presentation of the bachelor project "Describing, Modelling and Simulating the Copenhagen Metro" formulated by Ida Moltke, Matias Sevel Rasmussen and Tor Justesen at DIKU in connection with the course "Datalogi 2B", held at DIKU, spring 2003. The project is written on the assumption that the reader is at a professional level equivalent to Datalogi 2B and has knowledge of Java. Furthermore, it is expected that the reader has access to the technical documentation used in the work with the project.

Acknowledgements

We would like to thank both Gunni S. Frederiksen, Ørestad Development Corporation, and our supervisor Martin Zachariasen, DIKU, for their support of our work on the project. Especially we would like to thank Gunni S. Frederiksen for providing us with the necessary litterature and technical documentation and for his patience and willingness to answer our questions. We would also like to thank Kjeld Jørgensen, Metro Construction Management, and Nick Pope, Train Dispatcher, Union Switch & Signal, for their supplementation of necessary information as well as elaboration of the technical documentation.

Summary

The goal of this project was to make a simple simulation of the Copenhagen Metro that makes testing of different traffic control strategies possible. This goal seemed reasonable within the limits of a bachelor project, but turned out to be a larger task than first assumed for several reasons.

First of all it turned out to be a rather time-requiring task to read and understand the technical documentation of the Copenhagen Metro system. Ørestad Development Corporation is currently not in possession of a general overall introduction to how the Metro system is technically actually working. The only available overview of the system is [Intro], though this is a very superficial introduction and does not contain sufficient information to provide the basis for a simulation. We therefore had to look through and read a rather huge amount of detailed documentation of the complete system in order to be able to write a description of the system accommodated to our needs.

The available documentation reflects the entire working process of the contruction of the Metro system. Hence, in some cases, the more recent revisions some of the documents were contradictory to older revisions of other documents leading to inconsistency. Furthermore, it turned out that parts of the documentation containing necessary information about e.g. the currently used traffic control subsystem, VR, were not available to us due to business reasons.

In spite of these obstructions we find, however, that we have more or less accomplished our goal. We have accomplished our goal to the extend that we have build a simulation of the operation of the Metro in normal state where it is possible to relatively easy replace the VR function.

We have only implemented a very simple VR function enabling the trains to follow a fixed schedule. And we have only implemented a simplified version of control lines. It is however possible to replace these parts with more realistic implementations. In general, we have implemented the simulation such that it is possible to replace the parts that we have decided only to implement in a simplified version in order to have a running simulation.

The testing, however, revealed an error in the simulation program. Though, the testing still made probable that the program, with a little effort, can corrected to work as intended to.

To justify the correctness of our description of the system, we have presented it to Gunni S. Frederiksen, Project Manager Railway Technique, Ørestad Development Corporation. According to Gunni S. Frederiksen, our description of the Metro system is correct. See [Op] for his opinion on our description.

Chapter 1

Introduction

1.1 Incentive

On October 19 2002 the first phase of the new Copenhagen Metro opened its two lines to the public. The trains are driverless and run by a fully automatic control system. The system involves many aspects of modern computer science, including algorithms to control and coordinate the traffic flow, network communication, network security, etc. The Copenhagen Metro system thus is a good example of how modern computer science and technology is used in a real system.

1.2 Project Description

The goal of the project is to make a realistic, but simple simulation of the Copenhagen Metro.

Our original idea was to analyse how traffic control is handled in the Metro and on this basis develop one or more alternatives to the traffic control strategy currently used.

However, this idea had one disadvantage, namely the fact that we would not have been able to test the 'real life' quality of the developed alternatives. Hence any evaluation of the alternatives would have been purely theoretical, which we did not consider appropriate for a real system like the Copenhagen Metro. For that reason we changed our primary goal into building a simulation of the Metro.

The original idea is still of importance to the project, though, as it gives occasion for a specification of our goal. We wish to focus on traffic control and coordination. Our goal is thus to make a simple simulation of the Copenhagen Metro that makes testing of different traffic control and coordination strategies possible—within the limits of a bachelor project.

1.3 Delimitations

Based on our choice of focus, we will make the following delimitations to the simulation.

We will only simulate parts of the system that have direct connection to the control and coordination of the overall traffic flow. Among the parts we chose to simulate, we will thus only simulate details that are decisive for the traffic flow. We will hence not to go into details with the more engineering aspects of the system.

In order to make our goal reachable within the limits of a bachelor project, we have chosen only to simulate what we will denote 'normal state operation'. Loosely said, this means that we will only simulate the operation of a typical day. More specifically we will use the term to denote

- Operation that does not require human interference.
- Operation that is not interrupted to a degree that human interference is necessary

What this means in practice will be further specified throughout the report.

1.4 Strategy

In order to accomplish our primary goal, we will first give a description of the existing system and, based on this description, design a model that can be used to implement the simulation. Finally, we will evaluate our work in order to determine whether or not it complies with our goals. The project thus consists of the following four parts. We will

1. describe the real system
2. build a model based on the descripion
3. implement a simulation based on the model
4. evaluate our work

The present report describes and discusses each of these parts. The description of the real Metro system is presented in Chapter 2, and the design of the model is discussed in Chapter 3. The simulation is described in Chapter 4 and the evaluation is found in Chapter 5.

Each chapter contains, at the beginning, an outline of its contents and a further specification of the focus for the chapter. In order to ease the reading of the report, we have included a short list of the most frequently used acronyms at the end of the report. The different terms used throughout the report will be explaned at first use.

Chapter 2

Description of the Copenhagen Metro

In this chapter we will give a description of the Copenhagen Metro system and its operation. The focus and level of detail will be accomodated to the purpose and the available material. Hence we will not describe the entire system, but rather give a description of the essentiel parts and their interaction.

As mentioned in Section 1.3, we will not describe the more engineering aspects of the system, but rather the principles of how the system works.

Section 2.1 describes the permanent way, the topography of the permanent way, and the trains. Section 2.3 describes the Automatic Train Control system and Section 2.5 through 2.7 describes its subsystems. Section 2.9 describes how the quality of the operation of the Metro is measured.

2.1 The Permanent Way and the Trains

In the following section we will give a description of the permanent way and its topography followed by a description of the trains.

2.1.1 The permanent way

The Copenhagen Metro alignment constitutes a fork with two branches and a centre part, see Figure 2.1. The first branch covers the stations from the bifurcation at Christianshavn to Vestamager and the other branch the stations from Christianshavn to Lergravsparken. The central part covers the stations from Nørreport to Christianshavn. The Metro provides two servicelines, M1 and M2, running from Nørreport to Vestamager and from Nørreport to Lergravsparken, respectively. Both lines are bidirectional.

Figure 2.1: The main line

As seen on Figure 2.1, the current Copenhagen Metro system is only a part of what is planned to become the complete Metro system. However, we will only describe the current system, since the parts under construction are only prolongations of the branch to Lergravsparken and the centre part and will therefore not change the overall traffic control and coordination radically.

At the end of the line 1 at Vestamager, the Control and Maintenance Center (CMC) is situated. Here the trains are washed, cleaned, maintained and parked, when not in operation, see [Intro] p. 20. The monitoring and control of the operation of the Metro is also located at CMC.

The central part and the two branches are also called the main line. The stations along the main line are called inline stations, and the stations at the ends of the main line are called terminal stations. The terminal stations are the stations at Nørreport, Lergravsparken and Vestamager. We will use the abbreviations in Table 2.1 to denote the different stations.

Direction of travel

The main line consists of two two-rail tracks. Under normal circumstances, the trains are to use the right-hand track relative to their direction of travel. A train must hence a priori arrive from the right when seen from the station as illustrated on Figure 2.2.

If necessary, the trains can travel in both directions on each of the two tracks. This is used, e.g. if some work has to be done on one of the tracks and for this reason only one track can be used for operation, so-called 'single tracking' operation.

Station name	Abbreviations
Nørreport	KN
Kongens Nytorv	KGN
Christianshavn	KHC
Amagerbro	AMB
Lergravsparken	LGP
Islands Brygge	ISB
Universitetet	UNI
Sundby	KHS
Bella Center	BC
Ørestad	ORE
Vestamager	VEA

Table 2.1: Station name abbreviations

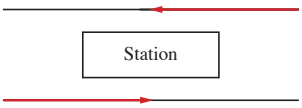

Figure 2.2: Illustration of direction of travel

Crossovers

In order to enable the trains to move from one track to the other, so-called crossovers have been placed at different locations on the main line. A crossover is simply a connection between the two tracks. Crossovers are considered to be part of the main line. There are two types of crossovers, A and B, illustrated on Figure 2.3.

Figure 2.3: Illustration of crossover type A and B, respectively

As it can be seen on Figure 2.3, the difference between the two types of crossovers is that type A requires a shorter track segment than type B. Their functionality is the same. Refer to [CLNDS] to see where they are located.

Pocket tracks

Pocket tracks are additional tracks situated on different locations along the main line, refer to [TS] to see their exact locations. They serve as some sort of storage where trains can be kept without influencing on the rest of the system. Pocket tracks are primarily used in the following situations:

- **Train failure** If a train breaks down or has a failure, it can be placed in one of the pocket tracks, until it can be fetched or checked. This is in some cases faster than driving (or pushing) the train to CMC immediately, and the disturbance of the current traffic is hence minimised.
- **Train frequency adjustments** To facilitate train frequency adjustments the pocket tracks are used to store 'spare' trains during non-peak-hours.

Pocket tracks are not considered to be part of the main line.

Track circuits

The permanent way is divided into numerous track circuits. A track circuit is an arrangement of electrical and electronical equipment, which allows detection and control of trains. Track circuits are seperated by so-called S-bonds that form an electric circuit between the two rails of the track.

The length of each track circuit has been designed to correspond to the maximum possible (allowable) speed for a train on that part of the track. The maximum possible speed depends on the location of the track circuit relative to the stations and the curve of the track circuit. For all track circuits, a train arriving with the maximum allowed speed must be able to come to a full stop within the length of the track circuit. As an example, the track circuits surrounding the stations are very short, since a train will always arrive at stations with a low speed. And oppositely, the track circuits between stations are longer since the maximum allowed speed is higher. The main line consists of 182 track circuits with lengths varying from 38 to 291 metres. The specific lengths of each track circuit can be found in [CLNDS].

2.1.2 The trains

There are 34 trains to service the Metro. The length of each train is 39 metres and their maximum speed is 80 kmph, see [Intro], p. 18. The trains are driverless and operated automatically. If necessary, though, the trains can also be operated manually by a train driver.

2.2 Identification

To control the operation of the Metro system, the capability of uniquely identifying trains and their movement, is required. In the following we will describe how this is accomplished in the Metro system.

2.2.1 Direction of travel

To facilitate the description of the system, we will denote the possible directions of travel as eastbound and westbound. Hence a train travelling from Nørreport to either Lergravsparken or Vestamager will have eastbound direction. Further, in order to differentiate between the tracks, we will enumerate the two tracks, the right-hand track in eastbound direction being track number one.

2.2.2 Track circuit identification

The main line is divided into several 'name blocks'. Each name block consists of several track circuits and its assigned name corresponds to the nearest station for that block. Within each name block the track circuits are enumerated from 1 and so forth till the next 'name block' starts, enumerating in eastbound direction.

The names of the track circuits are of the form <NAME BLOCK><TRACK NUMBER>-<TRACK CIRCUIT NUMBER WITHIN NAME BLOCK>T. As an example, track circuit number 5 in name block KHC (near Christianshavn), track 2, has the name KHC2-5T. The names of the track circuits can be found in [CLNDS].

2.2.3 Train identification

Every train has a unique Permanent Vehicle Identification (PVID).

2.2.4 Route identification

In order to systematise the specification of the geographical movement of the trains, the following terms are used:

Route A route is a fixed path along the tracks from an identifiable source track circuit (source) to an identifiable destination track circuit (destination).

Station Stopping Table Along the route from source to destination, the train may pass one or more stations. A Station Stopping Table (SST) specifies for a given route which stations the train must stop at along the way to its specified destination. See section 2.6 for more detail.

Destination ID Each possible route can be combined with several different Station Stopping Tables. A Destination ID (DID) is a unique identification of each such combination.

There are numerous possible route-SST-combinations, however only a limited amount of these are likely ever to be used. A valid DID is a DID identifying a "meaningful" route-SST-combination. Unfortunately we have not been able to get an example of a real DID.

Trip Type The Trip Type designates how a specified route is to be followed. There are four trip types amongst which the following are of our interest

- **Round trip** For round trips, the source-destination pair alternates each time the train reaches a destination. The train hence continually follows the same route, changing direction each time it reaches the destination track circuit.
- **One Way Trip** When the train reaches the destination track circuit, it is held there until it receives a new DID, that is, a new route specification.

From the above it follows that the geographical movement of a train can be uniquely specified from information about DID and trip type. Valid DIDs and their corresponding routing informations are stored in the so-called Route Table located at CMC.

2.3 Automatic Train Control

The operation of the Copenhagen Metro is fully automatic and is handled by the Automatic Train Control (ATC) system. The ATC system consists of three subsystems, see [TSATC] p. 49:

1. The **Automatic Train Protection** (ATP) system providing the primary protection for passengers, personnel and equipment against accidents
2. The **Automatic Train Operation** (ATO) system controlling the operations that would otherwise be performed by a train driver
3. The **Automatic Train Supervision** (ATS) system providing the overall supervision and control of the train traffic

The ATP subsystem is a vital subsystem of ATC, i.e. the ATP subsystem must never fail. Oppositely the ATO and ATS subsystems are non-vital subsystems of ATC and relies on the functioning of the ATP subsystem. In the following sections we will describe each of these subsystems in more detail. We will use the terms ATS (ATO, ATP) and 'the ATS (ATO, ATP) system' interchangeably to denote 'the ATS (ATO,ATP) subsystem'. We will use the term 'Central' to refer to the part of the computer-system located at CMC that provides the data necessary for the operation of the Metro, such as topographical information about the permanent way, DIDs, etc. Central can thus be considered as the database that provides the different subsystems of ATC with the necessary information.

Even though the Metro is fully automatic, the operation of the Metro is constantly supervised by a train dispatcher (TD). The TD is responsible for solving any problem that requires human intervention, and the TD can at any time take over the control of the system, e.g. by putting the system into manual operation mode. The TD is also responsible for choosing between different operational modes and for configuring any configurable parameters in ATC.

2.3.1 Operational modes

The ATC system may operate the Metro in one of four modes:

1. In **AUTO Mode** the trains are operated without the requirements of a driver, i.e. the operation is fully automatic.
2. In **ATO/ATP Mode** the trains are operated manually under the supervision of ATO and ATP. The trains automatically drive from one station to the next, performing both speed regulation and station stopping. A driver has the responsibility of closing the doors and initiating station departure.
3. In **ATP Mode** the trains are operated manually under the supervision of ATP controlling speed restrictions and emergency stops. The ATO and ATS are only used to provide scheduling information. Speed regulation, station stopping, and door handling are the responsibilities of the driver.
4. In **BYPASS Mode** all functions are the responsibility of the driver, i.e. the trains are operated solely by the driver. All ATO, ATS, and ATP functions are disabled or bypassed.

Referring to our focus in Section 1.3, we will only describe how ATC operates when in AUTO Mode. As a consequence we will also only describe a limited part of the TD's role—namely the TD's role in the context of ATS, see Section 2.7.4.

2.4 Interruptions of Service

In the real Metro system many kinds of interruptions can occur. In general, each of these can be classified by one of the following categories:

- **Human interference**, e.g. intrusion on the permant way, door blocking at stations, etc.
- **Equipment failure**, e.g. train breakdown, malfunctioning transmitters and receivers, etc.
- **System failure**, e.g. breakdown of the internal communication between ATC subsystems, etc.

Since we have chosen to focus primarily on the description of how ATC operates in AUTO Mode, the description of how ATC handles interruptions will only concern interruptions that do not impose the need for manual operation or intervention. As an example we will not describe how ATC handles e.g. a train breakdown where the train has to be fetched by another train, since this requires manual intervention. But oppositely we will describe how ATC handles a train breakdown, if the train automatically recovers the operation imposing only a delay of the train.

2.5 Automatic Train Protection

The Automatic Train Protection (ATP) system has two purposes. One purpose is to deliver safety restrictions to the non-vital parts of ATC, the second is to intervene and do, for example, an emergency braking, whenever a safety restriction is violated.

Furthermore, ATP ensures that trains are berthed, i.e. positioned, correctly, and that train doors and platform screen doors do not open or close unless allowed to. In short, ATP

- prevents train collisions
- imposes speed restrictions in track circuits
- intervenes when safety restrictions are violated
- berths trains and controls door opening and closing

The ATP system is divided into two subsystems, ATP Wayside and ATP Vehicle. As the names suggest, ATP Wayside is placed along the tracks, and ATP Vehicle is placed in the trains. ATP Wayside sends information to ATP Vehicle which reacts to this information. This information is called 'control line information', and is sent in every track circuit. (At stations, another type of information is sent, but this case is described in Section 2.5.4). The values in the control line information is determined by so-called 'control lines'.

At first we will describe control lines, the means by which train collisions are prevented and speed restrictions are imposed. Then we will explain how ATP Wayside and ATP Vehicle works and communicates.

2.5.1 Control lines

The cornerstone in the ATP system is the division of the tracks into multiple track circuits. During normal operation, no train is allowed to enter a track circuit occupied by another train. The approaching train must stop at the border of the occupied track circuit. This simple rule is a prevention against train collisions. Only in the very special emergency situation where 'automatic coupling', i.e. a train "rescuing" another train, is performed, a train is allowed to enter an occupied track circuit. However, this situation is outside the scope of this text.

Each track circuit has an associated upper speed limit decided by topographical matters, and ATP prevents trains from violating this upper speed limit.

ATP's prevention of train collisions and ATP's imposition of speed restrictions are implemented using the above mentioned control lines. A control line can be considered as the train's "line of sight".

A control line covers a fixed number (typically between 4–6) of consecutive track circuits. Every track circuit has at least one associated control line.

Basically, control lines are a means to make the train decelerate if necessary. The train must decelerate to respect a speed limit, and the train must also decelerate if another train is situated in a track circuit close ahead of it, recalling that no train is allowed to enter a track circuit occupied by another train.

As the control lines cover more than one track circuit, the train is forewarned of the occupancy some track circuits ahead. Control lines will always ensure that the train comes to a full stop at the border to the occupied track circuit.

The control line information transmitted from ATP Wayside to ATP Vehicle in a given track circuit is based on speed restrictions and possible track occupancy within the control line. This information is described in the following.

Civil speed and other speed limits

Track topography, as for example curves, influences directly on the maximum speed in a track circuit. The main line is divided into 'civil speed zones', which sets a maximum for the allowed speed. Often the civil speed is 80 kmph.

Furthermore, a 'speed restriction zone' can be applied to all track circuits between two stations.

Control line information

For every possible way through the track circuits on the Copenhagen Metro main line, the control line length and the control line information (refered to as 'control line info' in the remains of the report) are predefined in the diagrams [CLNDS], [CLNDX], [CLRDS], and [CLRDX].

The most relevant parts of the control line info are shown in Table 2.2. See [WASS] p. 57 and [VASS] pp. 49–50 for a full listing. In the following the data fields shown in Table 2.2 are further described.

Field	**Explanation**
Track Circuit ID	unique id for the current track circuit
Line Speed	default speed in this track circuit
Target Speed	desired speed
Distance-to-Go	distance to travel before the target speed should be reached
Direction Control	eastbound or westbound, the allowed direction
Next Carrier Frequency	the frequency in the next track circuit

Table 2.2: Control line info

Track Circuit ID The train has a table of sequences of track circuits. If there is a difference between the received Track Circuit ID and the track circuit, which the train should be in according to the table, ATP Vehicle will stop the train immediately. The Track Circuit ID value refers to the track circuit where the transmitter is situated.

Line Speed The maximum speed allowed in the current track circuit, set by civil speed or speed restrictions. The possible values in this field are 0, 3, 5, 10, 15, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, and 80 kmph.

Target Speed This is the maximum allowed speed for the train, when Distance-to-Go metres are travelled. The possible values in this field are the same as for Line Speed.

Distance-to-Go When the train has travelled Distance-to-Go metres, the train speed is allowed to be at most Target Speed. There are 128 possible values from 0 to 2900 metres for this field. The step is ranging from 6 to 500 metres. An actual physical distance is rounded to the nearest value lower than the actual distance which is the most restrictive.

In a situation where no obstructions occur in the control line, the value in the Distance-to-Go field is often 2900. This value is far more than the distance between any two consecutive stations, and thereby it can be looked upon as a "clear road"-signal.

Direction Control A train has an A-end and a B-end. At ATC system start-up or at train start-up the orientation of the ends is calibrated, see [VASS] pp. 67–69, allowing the train to, at any time, determine the direction of travel. Upon every reception of control line info, ATP Vehicle checks, whether its actual direction of travel matches the "commanded" direction, given in the Direction Control field. If there is a mismatch, ATP Vehicle will stop the train immediately.

The values in this field can be eastbound or westbound.

Next Carrier Frequency This frequency will be used in the next track circuit for sending control line info.

Track Circuit ID, Direction Control, and Next Carrier Frequency are fixed per transmitter, see Section 2.5.2. The fields which contain variable values per transmitter are Line Speed, Target Speed, and Distance-to-Go. The values of these fields form a triple that is defined in [CLNDS], [CLNDX], [CLRDS], and [CLRDX].

Principles for reading control lines

The control lines in the documents [CLNDS], [CLNDX], [CLRDS], and [CLRDX] are all read in the same way, and this method should also be use to read the control lines in the below scenario's Figure 2.4.

The control line of track circuit X starts with the dashed line and ends with an arrow. The principles, for reading which control line info that will be received by a train in a given track circuit, are:

- Check if there are any track occupancies in the track circuits covered by the control line. The track occupancy closest to track circuit X determines the control line info.
- The control line info is the triple printed in the track circuit left of the occupied track circuit.
- If there are no track occupancies, the control line info is the triple in the rightmost track circuit of the control line.

Scenario

In this section we will describe a scenario, that shows how control lines work. Figure 2.4 shows the actual (Line Speed, Target Speed, Distance-to-Go)-triples in the control lines at Nørreport station, as they are described in [CLNDS], sheet CL007N and CL008N. The control lines in Figure 2.4 are read as described above.

The scenario starts with a train, T1, occupying track circuit KN1-2T. Hence KN1-2T (which holds the Nørreport station) is marked as occupied in the ATP system. The following describes the succesive steps taking place, when another train, T2, travels toward Nørreport station.

1. T2 enters FOR1-7T, and receives (80,0,2900) as nothing is in the control line.
2. T2 enters FOR1-8T, and receives (80,0,2900) as nothing is in the control line.

3. T2 enters FOR1-9T, and receives (80,0,581), because the control line signals that there is an occupancy in track circuit KN1-2T, why T2 must stop (i.e. run at 0 kmph) in 581 metres.
4. T2 enters FOR1-10T, and receives (65,0,359), because the control line signals that there is an occupancy in track circuit KN1-2T, why T2 must stop in 359 metres.
5. T2 enters FOR1-11T, and receives (65,0,159), because the control line signals that there is an occupancy in track circuit KN1-2T, why T2 must stop in 159 metres.
6. T2 enters KN1-1T, and receives (65,0,50), because the control line signals that there is an occupancy in track circuit KN1-2T, why T2 must stop in 50 metres.
7. T2 can now travel to the end of the 50 metres track circuit KN1-1T and then stop. T2 must wait for T1 leaving KN1-2T completely before it can proceed into that track circuit.
8. T1 leaves KN1-2T. T2, still in KN1-1T, now receives (65,0,88), as T1 is no longer occupying KN1-2T.
9. T2 enters KN1-2T, and receives (40,0,38), because the control line signals that there is an occupancy in track circuit KN1-3T, why T2 must stop in 38 metres.

From Figure 2.4 it can also be read, that the civil speed zone containing KN1-2T (Nørreport station) has a 40 kmph speed limit. In fact, civil speed zones containing stations, always have this limit.

Interlocking

When two trains are running the same way through a sequence of consecutive track circuits, it follows that control lines will prevent the two trains from colliding. But a problem arises whenever trains need access to the same resource such as a crossover or a bifurcation switch at the same time. Situations like the ones shown on Figure 2.5 and Figure 2.6 are very undesirable as they may lead to train collisions.

Both situations are prevented by so-called 'interlocking'. The ATP Wayside interlocking computer maintains a queue of requested positions of a bifurcation switch or a crossover. Requests are sent to ATP by interlocking reservations from ATS. If, for example, T1's needed switch position was first in the queue, the situation on Figure 2.5 would be avoided by setting a pseudo occupancy in track circuit TC1 until the T1 has passed through the bifurcation switch. Then the pseudo occupancy is moved from TC1 to TC4 (if the needed switch postion of T2 is next in the queue). The pseudo occupancy will cause control lines "reaching" the pseudo occupied track circuit to decelerate an approaching train.

The situation on Figure 2.6 is handled in a similar way with pseudo occupancies. In a situation where trains must run towards each other at a single track, collision prevention is also controlled by the interlocking computers. Thus interlocking extends the control line concept to cover switch and crossover problems.

ATP Wayside interlocking computers are located in Service Equipment Room (SER). In each SER an ATP Wayside interlocking computer controls all interlocking within its belonging track circuits. Typically a SER is situated at every station, and the interlocking computer at the SER will control the interlocking of nearby crossovers. The interlocking computer at Christianshavn station will also control the bifurcation switches. The interlocking computers at each SER are connected.

2.5.2 ATP Wayside

The ATP Wayside equipment is physically placed along the tracks and connected to ATP Wayside computers situated in SERs. These ATP Wayside computers are again connected to computers at CMC.

The wayside equipment consists of various kinds of transmitter/receivers, placed at the beginning and at the end of each track circuit, and of a collection of computers that perform calculations and communication with ATS.

Figure 2.4: Control lines at Nørreport station

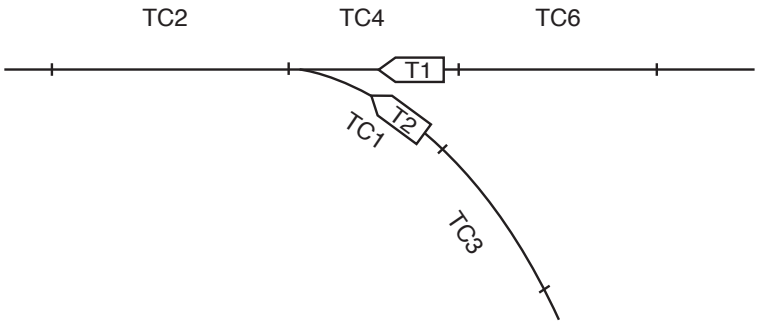

Figure 2.5: Undesirable situation at a bifurcation

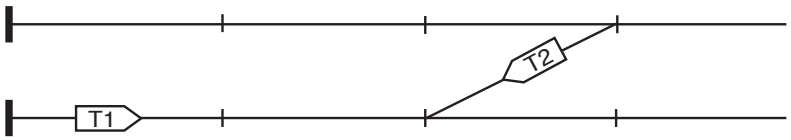

Figure 2.6: Undesirable situation at a crossover

A transmitter sends out control line info at a given frequency using the rails of the tracks as transmission medium. Two consecutive track circuits do not have the same frequency. This allows a train to distinguish the data signals. At every track circuit boundary, the train tunes in for the new correct frequency.

Every train travelling against the direction of transmission will pick up the data signal via an antenna. This is how ATP Vehicle receives control line info, and it is also how ATP Wayside monitors whether a given track circuit is occupied by a train or not; if the transmitted signal does not reach the receiver (because the train shortcuts the signal), the track circuit is presumed occupied. It follows, that a train occupies two track circuits when it is crossing a track circuit boundary.

Because the trains are able to run in both directions on the tracks, the transmitters must be capable of sending data both ways.

2.5.3 ATP Vehicle

The ATP Vehicle system constantly receives control line info from ATP Wayside via the rails of the tracks. Upon every reception of this information ATP Vehicle calculates a so-called Safe Braking Distance (SBD) profile for the train.

The calculation of the SBD profile is based on the values contained in the Line Speed, Target Speed and Distance-to-Go fields of the control line info. The SBD profile gives an upper bound for the speed of the train as a function of the current distance to the point on the tracks specified by the Distance-to-Go field, such that the train is runnning with the specified Target Speed at that point. That is, the train must obey the SBD profile in order to obtain the specified Target Speed after Distance-to-Go metres measured from the beginning of the current track circuit. Since the SBD profile is a safe braking profile the train, i.e. the ATO unit in the train, must constantly run with a speed that is below the one prescribed by the SBD profile. Otherwise ATP Vehicle will intervene and force the braking.

When calculating the SBD profile, it is assumed that the emergency braking rate (0.7 m/s^2) is used. The value contained in the Distance-to-Go field is measured from the beginning of the current track circuit. To obtain the actual Distance-to-Go, the train must hence know its current position within the track circuit. And in order to verify that the train is currently running with an allowed speed, ATP Vehicle must also know the current speed of the train.

Speed and position determination

The results of the speed and position determination are used by both ATP Vehicle and ATO. Because the results are used in the vital ATP Vehicle system, the determinination itself becomes vital.

The train uses two different sensors to determine train motion parameters, such as position, speed, and zero-speed. If there is a difference in the readings of the two sensors (except for an error tolerance), the train will assume being at the end of the track circuit. This is the most restrictive.

An exact description of how the sensors determines a train's motion parameters can be found in [VASS] pp. 93–94.

Zero speed (when the train has stopped) is detected when the measured speed over a single fixed time cycle is less than a certain limit. This limit is currently set to 0.4 kmph.

2.5.4 Vital station tasks

The vital tasks ATP is performing at stations are:

- train berthing
- door opening and closing control
- intrusion detection

Via equipment situated at each station, ATP Wayside checks, whether the train is correctly berthed, i.e. positioned, at stations. Only when the train is correctly positioned the train receives the special 'berthed bit', which allows ATO to open the train doors. This is synchronised with the opening of the platform screen doors (at underground stations). ATP also ensures that train communication equipment is functioning before train doors may close. See [VASS] p. 104 for details.

If intrusion on the tracks is detected by ATP, ATP prevents any train from entering the specific track circuit(s).

2.6 Automatic Train Operation

An ATO unit is placed in each train. As mentioned in Section 2.3 it controls the operations that would otherwise be performed by a train driver. The most important ones are

- Speed regulation
- Programmed stopping
- Door handling
- Station departure

In the following sections we will describe how ATO handles each of these operations.

2.6.1 Communication

ATO communicates with ATP Vehicle, ATP Wayside, ATS and Central. The communication with ATP Wayside, ATS and Central may only take place at the stations, whereas ATO may constantly internally communicate with ATP Vehicle.

2.6.2 Operational requirements

In order to execute the mentioned operations, ATO needs to have information about the train's current speed as well as the exact position within the current track circuit. As mentioned in Section 2.5.3, these informations are constantly calculated by the train and thus directly available to ATO.

2.6.3 Speed regulation

ATO regulates the train speed in accordance with the speed restrictions provided by ATP Vehicle and the performance level currently assigned to the train. A performance level is an indication of a desired level of performance for the train. What performance level is assigned to the train is dependent on the current operation strategy. Performance level data include speed limits and recommended acceleration and deceleration rates. There are nine different performance levels, each identified by a one digit index value, as shown in Table 2.3.

Level	**Speed limit**	**Commanded Acceleration**	**Programmed Deceleration**
1	Maximum	Maximum (1.2 m/s^2)	0.9 m/s^2
2 (normal)	70 kmph	1.0 m/s^2	0.9 m/s^2
3	60 kmph	0.8 m/s^2	0.8 m/s^2
4	55 kmph	0.8 m/s^2	0.8 m/s^2
5	50 kmph	0.6 m/s^2	0.7 m/s^2
6	60 kmph	1.0 m/s^2	0.5 m/s^2
7	30 kmph	0.8 m/s^2	0.5 m/s^2
8	15 kmph	0.8 m/s^2	0.5 m/s^2
9	5 kmph	0.8 m/s^2	0.5 m/s^2

Table 2.3: Performance levels

The operation strategy is decided by ATS and a train's performance level can be changed at stations. Refer to Section 2.7 for more detail. ATO will strive to operate the train in compliance with the current performance level, though the speed and acceleration/deceleration limits in the performance level data are not safety-related. The actual speed and acceleration/deceleration of the train are hence subjects to restrictions imposed by ATP Vehicle. More specifically the speed of the train must always be below the minimum of the performance level speed limit and the speed limit imposed by ATP Vehicle.

2.6.4 Speed regulation according to control line info

As mentioned in Section 2.5.3, ATP Vehicle constantly receives control line info from ATP Wayside and calculates a SBD profile for the train. ATO regulates the train speed so it constantly stays at least 5 kmph below the SBD profile. Notice, that if the speed limit of the performance level is lower than the one imposed by the SBD profile, ATO will regulate the speed of the train in accordance with this.

If the train is approaching an occupied track circuit, the SBD profile will indicate to ATO that it must start decelerating the train from a certain point, so it comes to a full stop at the end of the track circuit immediately before the occupied track circuit. This situation is illustrated in the Figure 2.7, where TC1–TC5 denotes track circuits and T1 and T2 denotes two different trains.

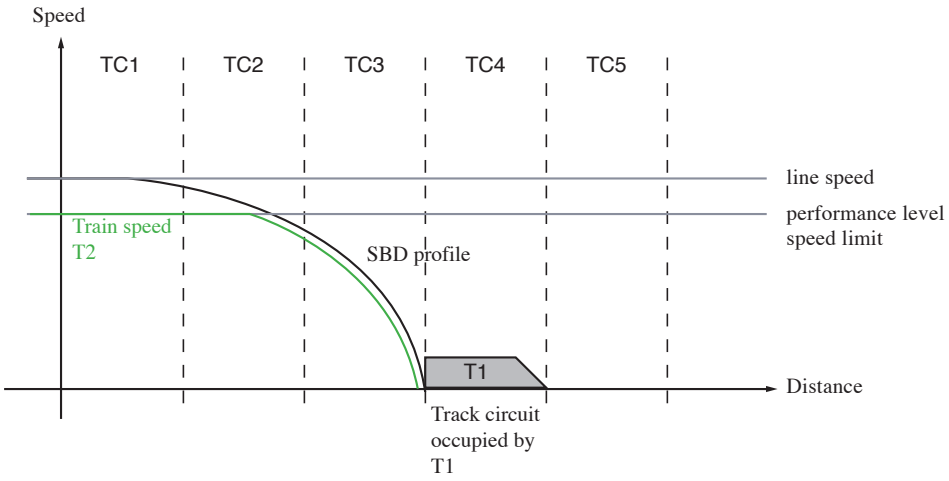

Figure 2.7: A train approching an occupied track circuit

The train occupying TC4 may however leave TC4 and continue to TC5 after T2 has started its deceleration. In this case new control line info will be sent to T2 and ATP Vehicle in T2 will calculate a new SBD profile. Since T1 moved to TC5 and since T2 had already started decelerating, the current speed of T2 may hence be lower than the actual desired speed according to either the new SBD profile or the speed limit provided by the current performance level assigned to T2. This situation is illustrated in Figure 2.8.

In this case, ATO must accelerate the train so it regains the desired speed, that is, the minimum of the performance level speed limit and the SBD profile speed limit.

The desired speed at the time the new control line is received may not be the same as the desired speed when the train has accelerated. As an example, assume that the desired speed of the train is the speed limit indicated by the performance level. The train speed may then be bounded by the new SBD profile before the train has actually accelerated to the desired speed limit. This situation is illustrated in Figure 2.9.

In this case, ATO must stop the acceleration of the train and obey the SBD profile.

If on the opposite, the speed limit prescribed by the performance level is reached first, ATO must also stop the acceleration and obey this speed limit. This situation is illustrated in Figure 2.10.

Notice that in the figures above we have assumed that the performance level speed limit is always lower than the speed limit imposed by the SBD profile. However, this might not always be the case,

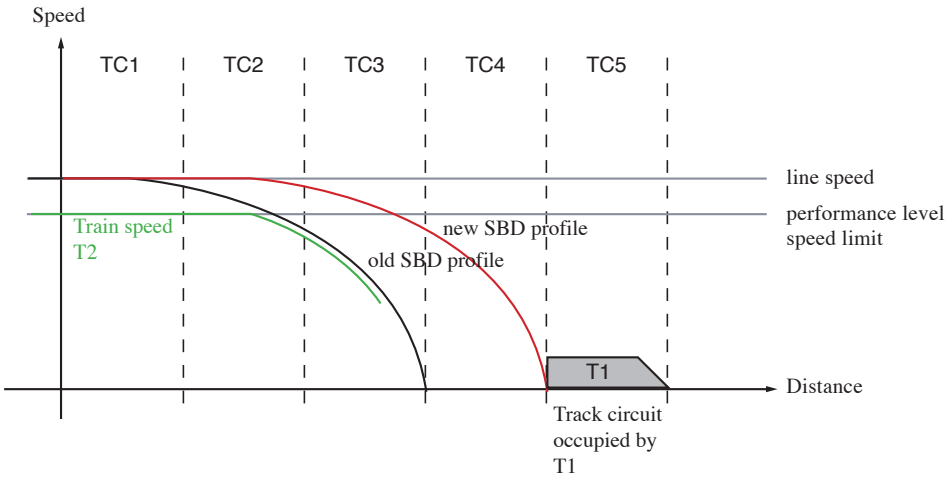

Figure 2.8: A train approching an occupied track circuit. T1 moves from TC4 to TC5 before T2 has fully stopped.

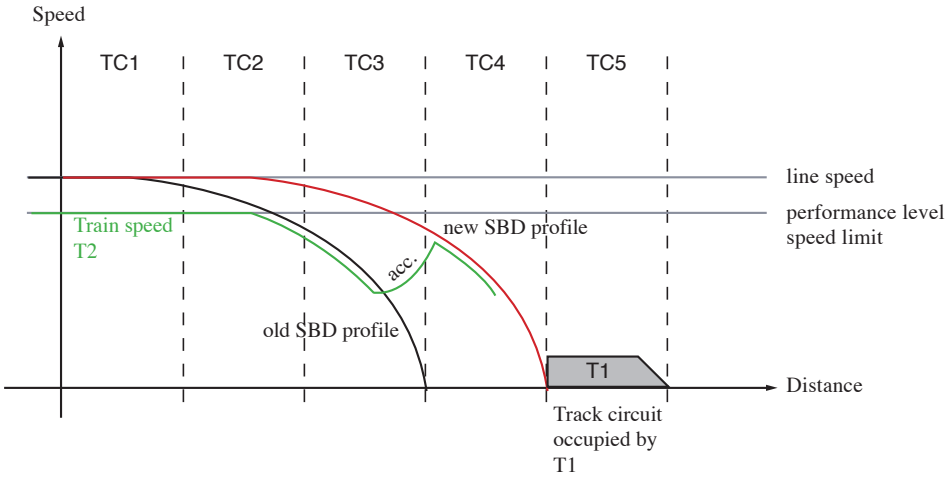

Figure 2.9: A train approching an occupied track circuit. T1 moves from TC4 to TC5 before T2 has fully stopped, why T2 must commence acceleration to regain the desired speed. The desired speed is bounded by the SBD profile.

but if the SBD profile speed limit is below the performance level speed limit, ATO will just regulate the in accordance with this speed limit.

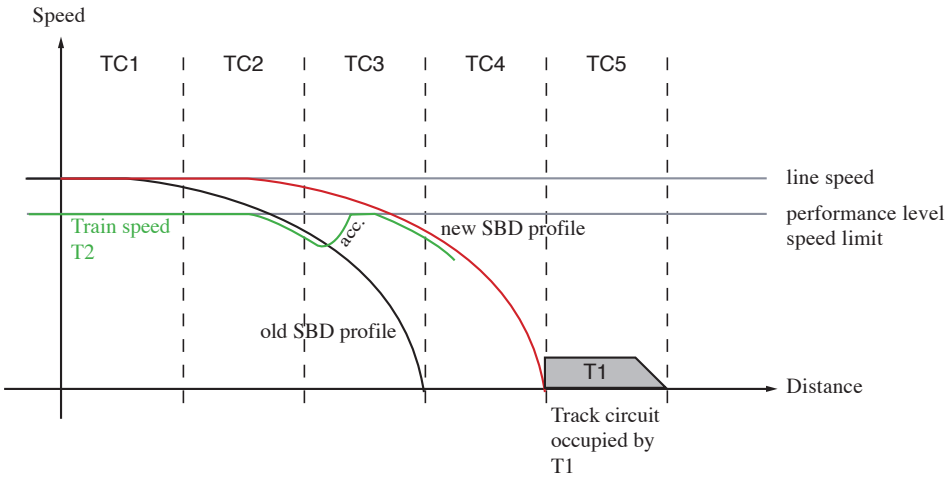

Figure 2.10: A train approching an occupied track circuit. T1 moves from TC4 to TC5 before T2 has fully stopped, why T2 must commence acceleration to regain the desired speed. The desired speed is bounded by the performance level speed limit.

2.6.5 Programmed stopping

When a train is assigned a DID, the DID's Station Stopping Table (SST) is downloaded to the train from Central and can hence be directly accessed by ATO during operation. Beside, information about whether the train must stop at a given station or not, the SST also contains information about the station's corresponding so-called "beacon track circuits'. Physically a beacon is a transmitter constantly emitting a signal. This signal is used to indicate to the train that it must initiate a programmed stop in order to stop at the approaching station. The term 'beacon track circuit' is a denotation for a specific track circuit, located a certain distance from the station, in which a beacon is placed. For each station there is an eastside and a westside beacon track circuit.

Every 350 ms ATO receives a data message from ATP Vehicle containing the current track circuit ID and train direction. For each message, ATO looks in the SST for a beacon track circuit ID/direction match. If a match is found and the train is designated to stop at the approaching station, a programmed stop is initiated. A programmed station stop is performed in three stages, see [VASS] pp. 99–101:

- **Constant speed** The vehicle continues with its current speed until reaching a distance D from the station. The distance D is calculated by ATO with the equation

$$D = \frac{v^2}{2a}$$

where v is the current speed of the train and a is the deceleration rate. According to [VASS] p. 99 the station stopping braking rate a is a constant for the current track segment between the previous and the approaching station, and was provided to ATO from Central at the previous station. [1] The normal station stopping braking rate is -0.9 m/s^2 (which corresponds exactly

[1] We have not been able to find any documentation about deceleration constants for track segments between stations. We therefore assume that what is meant, is that a denotes the programmed deceleration rate provided by the current performance level. This makes sense since this deceleration rate complies with its corresponding performance level speed limit and since a performance level is assigned to a train from station to station.

to the programmed deceleration rate for the *normal* performance level).

- **Constant deceleration** When the vehicle reaches the distance D from the current destination station, ATO initiates deceleration of the train at a continually adjusted braking rate a given by the following equation:

$$a = \frac{v^2}{2d}$$

  where v is the train's current speed and d is the current distance to the station stopping point. The distance d is calculated from the position of the train within the current track circuit and knowledge of the actual distance from track circuit boundaries to a specified station platform stopping point. Information about the platform station stopping point is provided to ATO from Central upon initialisation of the train.

- **Crawl and final braking** Once the train speed v is below a certain threshold and the vehicle pre-stopping distance is zero, ATO puts the train into 'crawl mode' and initiates final braking for normal precision stopping at the station stopping point. Once the train is fully stopped, ATO evaluates the accuracy of the stop with reference to the last calculated distance to the station stopping point. If the distance is within a margin of ± 1.1 metres from the desired station stopping point and no abnormalities have occured during the stopping of the train, ATO declares the stop to be successful.

For all trains, a 'Skip Stop' command transmitted to the train from ATS, will instruct the train to skip the next scheduled stop, that is, to bypass the next station. A 'Skip Stop' will be send to the train if for instance the platform screen doors at the next station are out of service. The train may also receive a "Skip Stop" command during an already initiated programmed stop. In this case the programmed stop is abandoned and the train proceeds to the next station.

Overshoot and undershoot

The terms 'overshoot' and 'undershoot' are used to denote the distance between the desired station stopping point and the actual train position when the train has stopped. An overshoot denotes a positive distance (the train has stopped too late), and undershoot denotes a negative distance.

If the undershoot/overshoot is within a range of ± 1.1 metres, and no abnormalities have occured during the stopping of the train, ATO declares the stop to be successful. In this case, ATO will open the train doors as normal. If the undershot is more than 1.1 metres, the train will go back to crawl mode and move the train forward to the desired stopping point. If the overshoot is between 1.1 and 2 metres, and no other errors have occurred, ATO will execute a reverse crawl to back the train to the desired stopping point. If the overshoot is greater than 2 metres, ATP Wayside will not allow the train to open its doors, and the train will proceed to the next station.

2.6.6 Door handling

If a train has completed a successful programmed stop at a station, ATO will send a request to ATP Wayside, to open the doors. If the request is accepted, ATP Wayside will open the platform screen doors and send a message to ATO to open the train doors.

Along with the request, ATO also indicates to ATP Wayside, if any of the train doors shouldn't or can't be opened e.g. due to a failure. If a certain train door is inhibited from opening, the corresponding platform screen door will not be opened. And reversely, if a given platform screen door cannot be opened, ATP Wayside will tell ATO to inhibit the corresponding train door from opening. When the train and platform screen doors have been opened, the control of the train is passed to ATS and a default dwell timer for the doors is started.

ATO will send a request to ATP Wayside to close the doors if one of the following two conditions is satisfied:

- ATO has received a message from ATS to leave the station

- The default dwell time has expired

If the request is accepted, ATP Wayside will tell ATO to close the train doors.

Note that ATO is neither responsible for managing dwell times at stations nor making sure the route to the next stop specified by the train's DID is prepared for the train. These task are responsibilities of ATS, refer to Section 2.7.

Obstructions

If an obstruction prevents any door from closing after ATP has initiated a door closure request, the door closing procedure will be repeated. If any of the doors still fail to close, ATO will not attempt to re-close the doors until commanded to do so by ATS.

2.6.7 Station departure

If both the train doors and the platform screen doors have been successfully closed, ATO will prepare the train for departure. Before a train can leave a station several conditions must all be satisfied among which the following are of interest:

- The train is assigned a valid DID
- The train has been assigned a valid performance level
- The train has been told to leave by ATS or the default dwell time has expired

If all conditions are satisfied, the train can depart and proceed to the next scheduled stop.

2.7 Automatic Train Supervision

As mentioned in Section 2.3 ATS provides the overall supervision and control of the traffic movement. In a little more detail, ATS continously gathers information about the status of the entire system, and based on this information:

1. controls and coordinates the overall traffic movement
2. provides the TD with a schematic overview of the entire main line
3. provides data to the stations about each individual train. This data is the basis for the information given to passengers waiting at the stations.

Since the focus of this project is the automatic operation and coordination of trains, see Section 1.2, we will only give attention to the control and coordination of the overall traffic movement.

First we will enlarge on the concept of traffic control. Subsequently we will describe the subsystem of ATS that handles traffic control. And finally, we will give an overview of the means that ATS provides to this subsystem both informatio-nwise and communication-wise. The latter serves the purpose of giving a description of what means any replacement of the current traffic control subsystem would have at its disposal.

2.7.1 Controlling the traffic

The traffic control in the Metro consists in outline of:

- putting trains into operation from either CMC or a pocket track (put-in)
- assigning DIDs to the trains, i.e. choose the routes they are to follow.
- controlling the dwell times of the individual trains in operation
- controlling the performance levels of the individual trains in operation

- reserving interlockings along the route of any train in operation.
- taking trains out of operation to either CMC or a pocket track (lay up)

An important goal of the traffic control is to make the traffic flow as promised to the passengers[2]. The Metro is not officially running according to a fixed schedule. Instead the passengers are promised fixed train arrival frequencies and estimated travel times. The current train arrival frequencies throughout the day can be seen in Table 2.4 and the current estimated travel times can be seen in Table 2.5 and Table 2.6.

In the following we will describe how the currently used traffic control subsystem of ATS complies with these requirements.

2.7.2 The Vehicle Regulation subsystem

The traffic control subsystem of ATS is called the Vehicle Regulation (VR) system. It supports two different automatic modes of operation, see [CNAOM] p. 61:

- **Schedule Regulation:** In this mode, VR controls train operation according to a pre-programmed schedule and VR seeks to minimise deviations from this schedule.
- **Headway Regulation:** In this mode, VR controls train operation to balance time intervals (headway) between trains. Each of the branches on the main line can have different headways, but the central section always has one common headway.

Schedule Regulation is the normal mode of operation, refer [CNAOM] p. 61.

Headway Regulation is meant for situations where deviations from the schedule have become too large[3].

Unfortunately the official documentation on either of these modes is somewhat sparse—probably because the developing company, Union Switch & Signal, wants to keep the details secret for business reasons. Especially the official documentation on VR contains very little information on Headway Regulation. Due to this and to the fact that Headway Regulation is not meant to be the normal mode of operation, we will not give a further description of that mode, cf. Section 1.3. We will, though, take the mode into account when describing the means provided to VR by ATS.

[2]There are several other goals. We will address this briefly in Section 3.7.

[3]We use the rather vague term 'too large' due to the fact that it is entirely up to the TD to decide, when deviations have become too large. In Section 2.7.4 we will give an overview of the TD's role with respect to VR and the traffic control.

Time of day	**M1**	**M2**	**On the centre part**
05.00–06.00	10	10	5
06.00–09.00(1)	6	6	3
09.00–15.00	8	8	4
15.00–18.00(1)	6	6	3
18.00–24.00	8	8	4
24.00–01.00	10	10	5
01.00–05.00(2)	15	15	7.5
(1) Saturday, Sunday and Holidays (2) Only the night after Friday and Saturday			

Table 2.4: Number of minutes between consecutive trains

2.7.3 Schedule Regulation

First we will give a short description of the schedule that VR uses as a basis for the train operation. Then we will describe how VR uses this in practise.

The fixed schedule

For every day, a fixed time schedule is generated. This is done by using a scheduling application, called HASTUS, that is not a part of ATC. We will in this report call any such schedule a 'hastus plan'.

A hastus plan specifies DID, trip, performance levels and arrival/departure times for each train. Also, the hastus plan indicates when a train is to be put-in and laid up. The indication simply consists of the first and the last entry in the hastus plan for each train. The station and arrival time in the first entry indicates when and where the train should be put-in, and in the last entry the station and departure time indicates when and where a train should be laid up. See Appendix [Hastus] for an example of a real hastus plan. Also the hastus plan indicates when a train is to be

HASTUS bases its planning on the described promises to the passengers about train frequencies and travel times. And it bases its planning on the routing requirements defined by Metro Service, the company in charge of the operation of the Metro. During the first couple of months of operation the trains were required to drive in two seperate loops, the M1 loop: VEA–KN–VEA and the M2 loop: LGP–KN–LGP, see Figure 2.11.

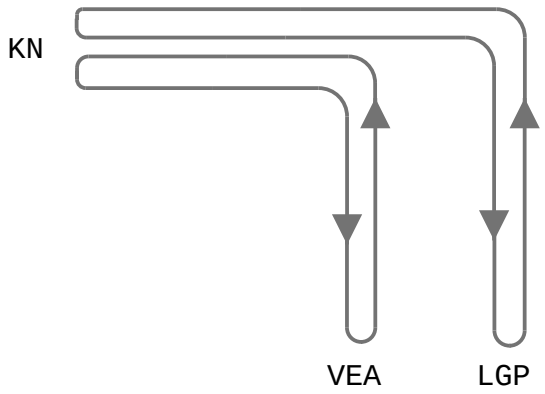

Figure 2.11: Illustration of two loops

From/To	KN	KGN	KHC	ISB	UNI	KHS	BC	ORE	VEA
KN	-	2	3	5	7	9	10	12	14
KGN	2	-	1	3	5	7	8	10	12
KHC	3	1	-	2	4	6	7	9	11
ISB	5	3	2	-	2	4	5	7	9
UNI	7	5	4	2	-	2	3	5	7
KHS	9	7	6	4	2	-	1	3	5
BC	10	8	7	5	3	1	-	2	4
ORE	12	10	9	7	5	3	2	-	2
VEA	14	12	11	9	7	5	4	2	-

Table 2.5: Number of minutes between any two stations on M1

Currently all trains are all required to drive in a so-called 'single butterfly loop' VEA-KN-LGP-KN-VEA, see Figure 2.12.

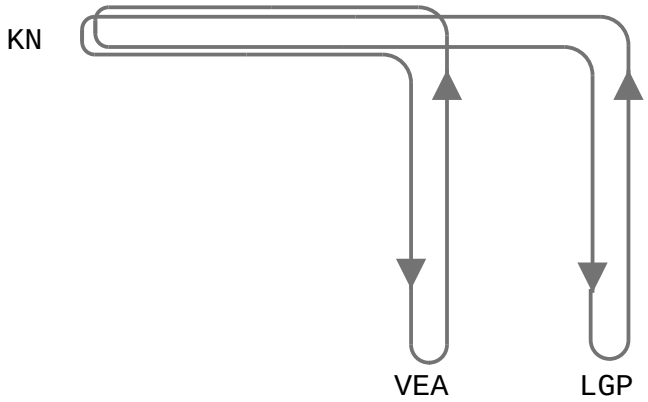

Figure 2.12: Illustration of butterfly loop

If informed of any day to day limitations to equipment, such as trains out of order and stations closed for maintenance, HASTUS will also take these into account when generating a plan.

Usage of the fixed schedule

The hastus plan is first of all used by VR for train put-in and train lay up. VR follows the hastus plan's information about this as closely as possible. Second of all VR uses the plan for controlling the trains that are in operation. Whether all trains are on time or not, VR automates the operation by continually generating an operation plan that is more detailed than the hastus plan. We will denote any such plan a 'VR plan'.

A VR plan is based not only on the hastus plan, but also on the topography of the permanent way and operating conditions such as actual arrival times. Hence the hastus plan is not used directly as an operation plan, but rather as an ideal for arrival and departure times and as a basis for the assignment of DIDs to trains.

A VR plan consists of two subplans for each train in operation: a route plan and a station plan, see [CASS] p. 126. The station plan in many ways resembles a hastus plan in that it contains information about the exact same things. The only difference is that the station plan is continually adjusted to the actual operation conditions, whereas the hastus plan is static and only theoretical, see [CASS] p. 125.

The route plan on the other hand has no resemblance to a hastus plan. It contains information about when the train is planned to traverse the interlockings on its route.

From/To	KN	KGN	KHC	AMB	LGP
KN	-	2	3	5	7
KGN	2	-	1	3	5
KHC	3	1	-	2	4
AMB	5	3	2	-	2
LGP	7	5	4	2	-

Table 2.6: Number of minutes between any two stations on M2

Generating plans

When planning, VR seeks to reach the closest match between the actual arrival and departure times and the arrival and departure times from the hastus plan, see [CNAOM] p. 64.

This is done by means of performance level adjustments and/or dwell time adjustments, see [CNAOM] p. 62.

Notice that even though VR can change the DID of a train, this is currently not used as a means to reaching the wanted match.

How often a new plan is generated

For each train a plan, including at least two complete station stops ahead, is made and for the system as a whole, VR will typically plan two hours ahead, refer [CASS] p. 125 and [GSF].

A new plan is made if a plan is about to expire. Moreover a new plan is generated whenever a train arrives at a station more than three seconds early or late compared to the current VR plan, see [CNAOM] p. 63.

If a train arrives at a station 1–3 seconds earlier or later than planned by VR, this is mended by adjusting the dwell time allowing the train to leave at the time VR had planned. Hence the deviation does not cause VR to replan, see [CNAOM] p. 63. If VR for some reason is not allowed to adjust the dwell time then even a 1–3 second deviation will cause VR to replan, though. See Section 2.7.4 for information about VR's limitations.

The time a plan is generated for

According to [CNAOM] p. 63:

> "It can not be assumed that the system plan can be built and a vehicle can accept a performance level change at the current in-line station with a minimum of 15 second dwell. Therefore, for most minor delays adjustments, VR will plan to make performance level adjustments and dwell adjustments at the next platform."

Thus if a train, T, arrives at a station S off schedule and thereby causes VR to replan, VR is not always capable of generating a new plan for T within the time available before T is supposed to leave S. We have, however, not found any documentation saying what the criterias are for VR to be capable of doing so. Moreover we have not found any documentation saying whether VR in the described situation will always try to make a new plan for T that is intended to be used at S nor have we found documentation saying how VR will react in case it fails time-wise to make a plan.

The way plans are generated

According to [CASS], p. 125–127, the process of generating a plan is divided into three stages:

Base planning For each train a base plan is made. A train's base plan is made as if that train was the only train situated on main line. It is made using the following guidelines:

If a train arrives a little late or a little early, VR will try to find a performance level/dwell time combination that allows the train to arrive at the next station on schedule[4]. If there are more than one such combination, VR will choose the one in which the dwell times are the closest to the scheduled ones, see [CNAOM] p. 62.

If a train arrives very late[5], the plan will be based on the use of the fastest possible performance level and the minimum dwell until the train is only a little late. Likewise, if a train is very early the

[4]The use of the vague expression 'little late/early' is against our intensions, but it is as precise as the documentation allows us to be. Based on the fact that a new plan is not generated if a deviation is less than three seconds, our guess is that 'a little' is normally more than three seconds.

[5]Again we use a vague expression.

plan will be based on the use of the slowest possible performance level and the maximum dwell, see [CNAOM] p. 62.

However, if a huge delay occurs to a scheduled train, such that the train is now closer to the next scheduled trip instead of the current trip, VR will consider the train early for the next trip rather than late on the current trip, see [CNAOM] p. 64. Whether this affects the following trains or not is not clear from the documentation.

At terminal stations, if possible, deviations are handled only by means of dwell time adjustments.

Determining dependencies All dependencies are determined. A dependency is a situation, where the base plans of two different trains specify the need for the same resource, such as the same station or the same interlocking. Each dependency is either a *follow* or an *overlap*.

It is a follow if an order of the two trains involved is already established, hence there is no conflict. It is an overlap if the order is not yet established and therefore up to VR.

Resolving dependencies All overlaps are resolved. For each combination of orderings between any two overlapping trains, VR builts a plan. And for each plan, VR calculates start and end times for the use of resources. Using a predefined metric—based on deviations from schedule and some so-called train priorities—the cost of each plan is calculated. For more details, see [CASS] p. 125–128, though notice that the metric is only outlined not actually defined in detail.

After having finished all the calculations, VR selects the plan with the lowest cost.

Notice that under normal circumstances the base plan will be valid, refer to [CASS] p. 126. In this case the third stage is not relevant.

Plan execution

The VR plan is the basis for the compliance of the requirements from Section 2.7.1. In this subsection we will specify how and when VR executes a VR plan.

How a plan is used

VR uses the station plan for dispatching the individual trains from stations. The route plan is used for reserving interlocks for a train before the train is planned to traverse them.

Station Dispatching Each time a train enters a station VR sets a timer to terminate the dwell according to the planned departure time. In case a new plan is available with a different dwell time before the timer expires, this new dwell time is used as a new basis for the timer. When the timer expires, VR checks for abnormal conditions. An abnormal condition is in short a condition that is likely to prevent the train from reaching the next station. See [CNAOM] pp. 63–64 for more details on these conditions. If no abnormal conditions exist, VR tells ATO which performance level to use and signals that it can now depart, see [CASS] p. 128–129. If an abnormal condition exists, VR will extend the dwell indefinitely, see [CNAOM] p. 64. Unfortunately we have not been able to find out how this affects the VR planning.

Route Control According to the route specified by the DIDs of the trains in operation, VR will try to make sure that interlockings are reserved ahead of the trains so that no train has to slow down or entirely stop before traversing an interlocking.

In practice, VR will—based on the route plan—try to reserve as many interlockings ahead of every train as possible, though within some limits. The most important limits are:

- VR should not reserve an interlocking for a train, if it has not yet planned the train's traversal of it.

- VR should not reserve an interlocking for a train more than X seconds before the train is planned to traverse it. The value of X is configurable.
- VR should not, for a train that is planned to change turn back, e.g change direction of travel, reserve an interlocking on the turn back route until the train is in the turn back area[6].

For more details see [CASS] p. 130.

2.7.4 The Train Dispatcher's role

The extend to which VR controls traffic is decided by the TD. First of all a TD can set the system into *No Regulation Mode* which means that all VR functionalities are disabled. In this case the TD is responsible for controlling the entire operation, see [CNAOM] p. 61. Assuming the system is not in No Regulation Mode, the TD can control the use of VR in the following ways:

1. The TD is responsible for deciding which operation mode, Schedule Regulation or Headway Regulation, VR should be in.
2. The TD can set one or more devices into different modes at any time:
    - **Train Device Modes**:
      Each train can individually be set in Central Automatic Mode or in Central Manual Mode. If a train is in Central Automatic Mode it is controlled by VR. If it is in Central Manual Mode it is controlled by either the TD or a driver.
    - **Location Device Modes** Each station in the system can be individually set into several different modes. The most important ones are Central Automatic Mode and Central Manual Mode. If a station is in Central Automatic Mode, VR controls the operation. If it is in Central Manual Mode, a default dwell and a default performance level are used for all trains arriving at the station.
3. The TD can configure a series of VR-parameters. Among the most important ones are:
    - the set of performance levels that VR is allowed to use at each individual station
    - the maximum and the minimum dwell time that VR is allowed to use at each indvidual station

For a complete list of VR-parameters see [CASS] p. 124–125.

2.7.5 The means provided to the Vehicle Regulation subsystem

The information available to VR in the real system is in outline:

- A HASTUS plan
- The actual arrival and departure times of all trains in operation
- The topography of the permanent way, cf. [CASS] p. 125. E.g the position of the positions of the stations relative to each other and the positions of crossovers and the bifurcation.
- Speed restrictions that may affect the travel time of a train.
- The DIDs and performance levels supported by the system.
- Configurations made by the TD.

All this information, except for the actual arrival and departure times and the TD's configurations, are available at Central. The arrival and departure times are measured by ATS at each station. The TD's configurations are given directly to ATS by the TD at CMC.

On the other hand VR is for any given train capable of

[6] We have unfortunately not been able to find the exact definition of a turn back area.

- putting in the train from either CMC or a pocket track (of course depending on where the train is situated)
- assigning DID, trip and performance level to a train's ATO, when the train is situated at a station.
- commanding a train situated at a station to leave that station.
- reserve a specific interlocking through ATP Wayside.
- laying up a train either to CMC or to a pocket track.

2.8 Communication between ATC Subsystems

The communication between the different subsystems of ATC is illustrated on Figure 2.13.

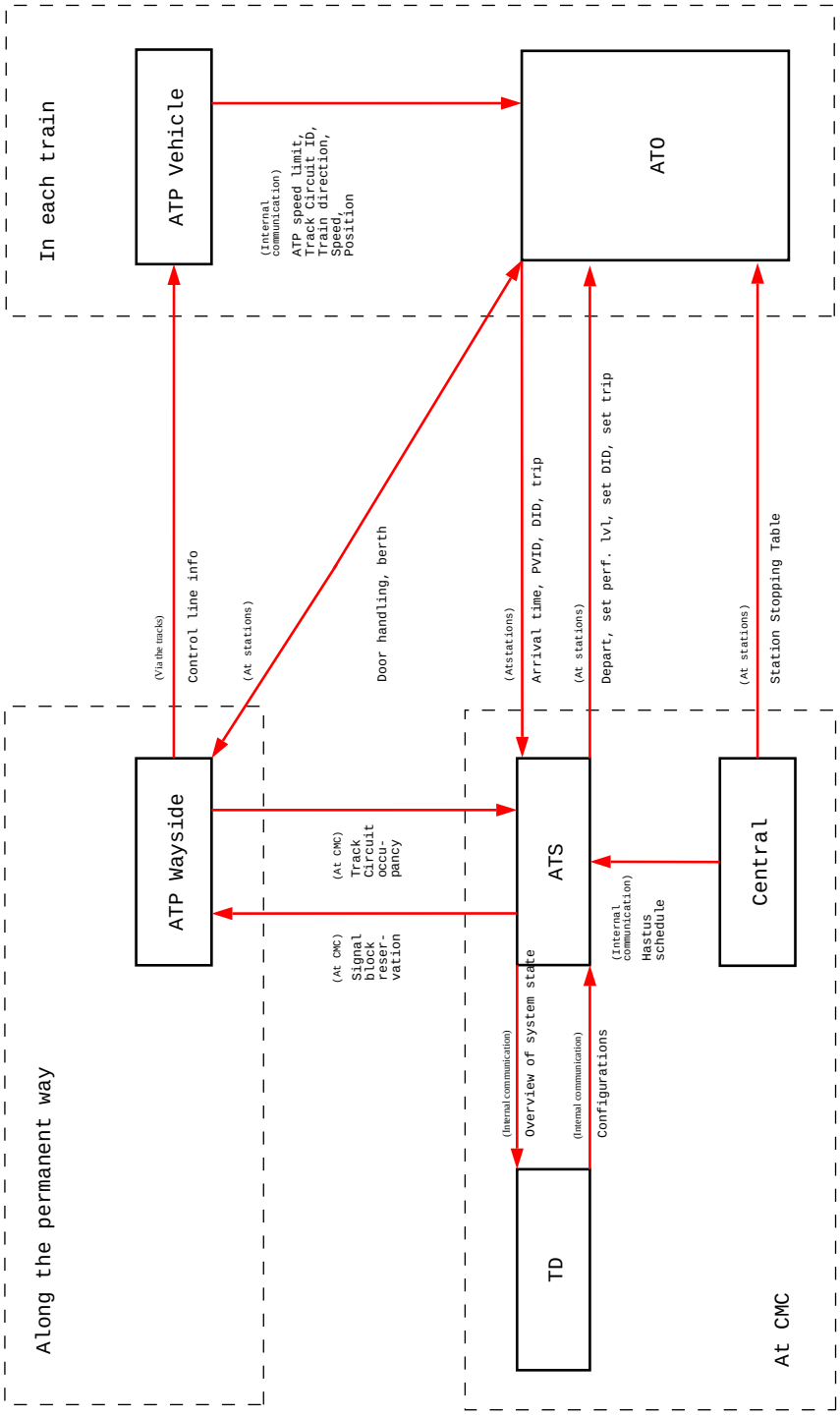

Figure 2.13: Communication between ATC subsystems

2.9 Evaluation of the Traffic Control Strategy

The Ørestad Development Corporation, the owners of the Metro, does not isolatedly evaluate the traffic control strategy implemented by VR. They evaluate the operation of the Metro as a whole. This is done using a metric called Service Availability, that uses Sevice Availability Points (SAPs) as measure. In outline SAP is a measure of how well the promise of a given train arrival frequency is kept.

The principle behind the evaluation is simple. The operation time is divided into time intervals of a specified length. And the promised train arrival frequency is translated into an amount of promised train arrivals for the individual stations within each time interval.

For each station, for each time interval the actual amount of train arrivals is measured. If the actual amount of arrivals is less than the promised amount, the difference between the two amounts is given as punishment—minus one SAP per train. This punishment is kept unless the amount of train arrivals in the immediate next time interval is greater than the promised amount. If this is the case some or all of the punishment is deleted.

For instance KGN could have a promised arrival amount of three trains per time interval. And in case only one train arrives to KGN in the period of an interval, KGN is given minus two SAPs. If within the immediate following interval four trains arrive at KGN one minus point is deleted. If five or more trains arrive both minus points are deleted. Otherwise the minus points are kept.

Notice that this metric is well defined no matter what operation mode the Metro is in.

Chapter 3

Model of the Copenhagen Metro

In this chapter we describe how we will model the Copenhagen Metro system. The modelling will be based on the description of the system presented in Chapter 2.

Our goal is to make a simple simulation of the Copenhagen Metro that makes testing of different traffic control and coordination strategies possible. Hence we will only model the parts of the system that are relevant to the overall traffic control and coordination.

Ideally the model should give occasion to a simulation that is as close to the real system as possible in its effect traffic-flow-wise. Though, since this is a time-requiring task and our goal is a *simple* simulation, we will only strive for realism to the extend that time allows it. In cases where we chose to simplify parts of the model in ways that might have decisive influence on the effect of the simulation, we will seek to make it possible to replace the parts with more realistic versions. Also, we will seek to make VR replaceble.

In Section 3.1 we will discuss some general modelling considerations. In Section 3.2 we will discuss the modelling of the interruptins of service presented in Section 2.4. In Section 3.3 we will discuss the modelling of the ATC system and its subsystems. Finally, we will give a description of how we will model Central, the permanent way and the trains in accordance with the requirements imposed by the modelling of ATC, see Sections 3.4 and 3.5, respectively. In Section 3.7 we will discuss how the evaluation of different traffic control strategies can be modelled.

Notice that we will address the modelling of concepts like DIDs when discussing how to model Central. Until then we will use the terms in an abstract sense.

3.1 General Modelling Considerations

In the following we will discuss some general modelling considerations that are decisive for the modelling of the entire system.

3.1.1 Direction of travel

To simplify ATP to an extend where an implementation of a simulation seems possible, we will only model right-hand side driving, i.e. the trains will run on track 1 on the stretches from KN to LGP and from KN to VEA, and on track 2 on the stretches from LGP to KN and from VEA to KN. Under normal conditions this is actually also the way the Metro trains run.

Furthermore, we choose not to model crossovers. The sole purpose of a crossover is the change of track for a train. In a normal state operation, the trains will only change track at the terminal stations. To subsitute this need, we will allow trains to be moved in an undefined manner from one track to the other at the terminal stations.

We are aware that the above simplifications are limiting VR maneuvers, but we find them reasonable within the time frame of the project and the delimitations discussed in Section 1.3.

3.1.2 Speed adjustment

As described in Section 2.6.3, ATO constantly regulates the train speed in accordance with the speed restrictions provided by ATP Vehicle and the current performance level. The actual train speed must at most be the minimum of the speed limit provided by the performance level and the speed limit prescribed by the SBD profile minus 5 kmph. ATO adjusts the speed by either accelerating or decelerating the train. In the following we will discuss how this can be modelled. There are basically two ways of modelling train speed adjustments:

1. 'Direct speed adjustment', where it is assumed that the train can accelerate/decelerate from one speed to another in zero seconds
2. 'Real speed adjustment', where the train performs speed adjustments by actual acceleration/deceleration.

In accordance with Section 3 we will choose the modelling method whose effect is the closest to the effect of the real system. If the two methods have more or less the same effect, we will chose the simplest of the two. Basically, the choice is between direct and real speed adjustment. Real speed adjustment is the most realistic method, but it adds some complexity to the model. Oppositely, direct speed adjustment is simple, but less realistic.

In order to choose between direct and real speed adjustment we will now consider the time of travelling for a train travelling on the longest stretch between any two consecutive stations. For each performance level we will calculate the time of travelling for a train using direct speed adjustment respectively normal speed adjustment. In order to do this we will need the following formulae, cf [Mek] p. 2.6 :
The total time t_s spent to travel a given distance s with constant speed v is given by the formula

$$t_s = \frac{s}{v} \quad .$$

The total time t_a spent to accelerate/decelerate the train from an initial speed v_0 to a specified speed v is given by the formula

$$t_a = \frac{v - v_0}{a} \quad .$$

where a is the acceleration/deceleration rate. We assume a to be constant.

The total distance s_a travelled during constant acceleration/deceleration for a specified amount of time t, is given by the formula

$$s_a = v_0 t + \frac{1}{2} a t^2 \quad .$$

where a is the acceleration/deceleration rate. We assume a to be constant.

From [BCLDC] we have that the longest distance between any two consecutive stations is 1293 metres—measured from the ends of the station track circuits—between the stations at KHC and AMB in eastbound direction. We will now consider the time a train spends to get from KHC to AMB using direct speed adjustment respectively normal speed adjustment. We will make the following assumptions, in order to provide a worst-case scenario:

- The train is allowed to run with a maximum speed of 80 kmph on the whole stretch between KHC and AMB.
- There are no obstructions causing delays on the way from KHC to AMB.
- Once the train has obtained its maximum allowable speed it keeps this speed at all times.

We will not show all of the calculations here, but just give a single example. With performance level 1, the speed limit is 80 kmph $= \frac{80000}{3600}\frac{\text{m}}{\text{s}} = \frac{200}{9}\frac{\text{m}}{\text{s}}$ and the acceleration and deceleration rates are 1.2 m/s^2 and 0.9 m/s^2 respectively. By using direct speed adjustment it will take the train

$$t_{dir} = \frac{s}{v} = \frac{1293 \text{ m}}{\frac{200}{9}\frac{\text{m}}{\text{s}}} = 58.2 \text{ s}$$

to get from KHC to AMB.

The time t_1 spent to accelerate the train from zero speed $v_0 = 0$ m/s^2 to $v = 80$ kmph $= \frac{200}{9}\frac{\text{m}}{\text{s}}$ with acceleration rate $a = 1.2$ m/s^2 is

$$t_1 = \frac{v - v_0}{a} = \frac{\frac{200}{9}\frac{\text{m}}{\text{s}} - 0\frac{\text{m}}{\text{s}}}{1.2\frac{\text{m}}{\text{s}^2}} = 18.5 \text{ s}$$

and the time t_2 spent to decelerate the train from speed $v_0 = 80$ kmph $= \frac{200}{9}\frac{\text{m}}{\text{s}}$ with deceleration rate $d = 0.9$ m/s^2 is

$$t_2 = \frac{v - v_0}{-d} = \frac{0\frac{\text{m}}{\text{s}} - \frac{200}{9}\frac{\text{m}}{\text{s}}}{-0.9\frac{\text{m}}{\text{s}^2}} = 24.7 \text{ s}$$

The distance s_1 travelled during acceleration is

$$s_1 = v_0 t_1 + \frac{1}{2}at_1^2 = \frac{1}{2}at_1^2 = 205.8 \text{ m}$$

and the distance s_2 travelled during deceleration is

$$s_2 = v_0 t_2 - \frac{1}{2}dt_2^2 = \frac{200}{9}\frac{\text{m}}{\text{s}}t_2 - \frac{1}{2}dt_2^2 = 274.3 \text{ m}$$

The distance travelled with constant speed is thus

$$s_3 = 1293 \text{ m} - s_1 - s_2 = 812.9 \text{ m}$$

which takes

$$t_3 = \frac{s_3}{\frac{200}{9}\frac{\text{m}}{\text{s}}} = 36.6 \text{ s}$$

It will thus take

$$t_{norm} = t_1 + t_2 + t_3 = 79.8 \text{ s}$$

to get from KHC to AMB when using real speed adjusment.

It can be seen that it requires the most time to get from KHC to AMB when using normal speed adjustment rather than direct speed adjustment, and the relative difference is

$$\frac{t_{real} - t_{dir}}{t_{real}}100\% = 27.1\%$$

PL	Real speed adjustment	Direct speed adjustent	Relative difference
1	79.8 s	58.2 s	27.1 %
2	87.0 s	66.5 s	23.6 %
3	98.4 s	77.6 s	21.2 %
4	103.7 s	84.6 s	18.4 %
5	114.6 s	93.1 s	18.8 %
6	102.6 s	77.6 s	24.4 %
7	168.7 s	155.2 s	8.0 %
8	317.1 s	310.3 s	2.1 %
9	933.2 s	931.0 s	0.2 %

Table 3.1: Times of travelling between KHC and AMB with direct and real speed adjustment

The results of the calculations for the rest of the performance levels are presented in Table 3.1.

From the results of the calculations it can be seen that it is of almost no importance whether we model speed regulation by either direct or real speed adjustment when considering performance levels 7, 8 and 9. However, when considering the performance levels 1–6 there is a relatively large difference between the times of travelling with real speed adjustment implying the longest travelling time. For instance it can be seen that a train assigned the normal performance level (2) will arrive 23.6% faster at AMB when using direct speed adjustment than when using real speed adjustment.

Since we have now showed the importance of choosing real speed adjustment rather than direct adjustment in the worst-case, this will also be the case in general. Modelling speed adjustment by real speed adjustment will thus give an effect that is closer to the effect of the real system's train speed adjustment. We will hence model train speed adjustments by real speed adjustment.

The extend to which real speed adjustment is modelled

Section 2.5 describes that the main line is divided into civil speed zones. The train is notified about these speed zones via the control lines, i.e. via the SBD profile calculated by ATP Vehicle, and will hence automatically obey the speed limits prescribed by the different civil speed zones. According to [KJG] the time spent to regulate the speed of the train due to the different civil speed zones is in general negligible when compared to the time spent to accelerate/decelerate the train when departing/arriving at a station.

One possibility is thus to assume that speed adjustments are only performed when the train is departing/arriving at a station, i.e. speed adjustments due to e.g. civil speed zones, are not modelled. However from Section 2.7 we have that the VR function is very sensitive to minor delays, that is, even very small deviations from schedule (1–3 seconds) will have influence on the way VR performs its planning. We thus choose to model real speed adjustment by having ATO regulate the speed of the train whenever needed.

3.2 Interruptions of Service

As mentioned in Section 1.3 our focus is on normal state operation, why we only model the most normal interruptions of service. These all have in common that they cause a delay. Hence, we will only consider interruptions as delay-causing incidents. We have considered two ways of modelling this:

1. Interruptions are associated with a variable-length delay and an indication of the type of the interruption
2. Interruptions are associated with a variable-length delay

In the real system there are many types of interruptions, each falling in one of the three categories mentioned in Section 2.4. The first possibility would give VR the possibility of reacting differently on different types of interruptions.

The second possibility would be simpler than the first, because all possible interruptions fall in one category with a variable-length delay. This, of course, gives e.g. VR no opportunity of reacting differently on different types of interruptions.

We choose the second possibility, because it simplifies the model, and because we have not found any documentation indicating that VR in the real system has access to information on the reasons for delays.

We have to decide where the delay should be effectuated in our model, and our considerations cover the following two possibilities:

1. In a specific track circuit
2. In a specific train

It is obvious that possibility 1 will allow us to model track circuit failure, and that possibility 2 morely will allow us to model train failure, reckoning that we only consider failures causing minor delays, according to our focus on normal state operation.

As we are interested in monitoring how VR reacts to a delay, it is not of great importance that the delay is given to a specific train. It is more important where the delay is inserted into the system, as this can have influence on the traffic flow. Therefore, we choose possibility 1.

For instance, possibility 1 can be used to insert a delay caused by a passenger blocking a door at a station, an event that is likely to occur.

3.2.1 Statistics of interruptions of service

When modelling interruptions of service, statistics from the real system should ideally be used. This would make the model more realistic. Statistics dealing with the size of the delay of prior reasons to interruptions would be useful for this purpose. That is, we cannot use statistics dealing solely with train departure delays, as these might be the results of propagated delays from a specific event. And the character of the propagation might depend on the traffic control strategy used.

3.3 Automatic Train Control

In order to make a realistic model, the modelling of the Automatic Train Control (ATC) will be comprised of the modelling of Central and the subsystems, i.e. ATP, ATO and ATS in simplified versions.

3.3.1 Automatic Train Protection

The responsibilities of ATP are described in Section 2.5, and in an outline ATP

1. prevents train collisions
2. imposes speed restrictions in track circuits
3. intervenes when safety restrictions are violated
4. berths trains and controls door opening and closing

Re point 1 and 2 We find 1 and 2 of greatest interest, because modelling these ATP functionalities would make train movements realistic. Our model's fulfillment of these functionalities are discussed in the following.

Re point 3 In the following we will discuss ATP Vehicle's intervention in ATO decisions in the context of speed limits.

We will not model the intrusion detection part of ATP, but the intrusion can be modelled as a delay, refer to Section 3.2.

Re point 4 We will not model the communication that takes place in connection with train berthing and door opening/closure, because our focus is on train operation. A train berthing problem or a door opening or closing problem could be modelled as a delay, refer to Section 3.2.

Train collision prevention and speed restriction imposition

ATP's prevention of train collisions (point 1 of Section 3.3.1) and ATP's speed restriction imposition (point 2) can be modelled in several ways. We have considered three approaches:

1. Simply disallowing trains to enter an occupied track circuit, and letting trains run at a constant speed troughout a track circuit with an associated upper speed limit. The speed restriction is received by the train when entering a track circuit.
2. Using the control line-method but with autogenerated control line info values based on track circuit lengths and speed zones.
3. Using the control line-method, described in Section 2.5.1, with the actual control line info values from the real system, shown in [CLNDS] and [CLNDX].

Approach 1 is the simplest approach. But it is also the approach farthest from the method of the real system. The approach would imply that trains running at full speed should stop instantaneous at the border of an occupied track circuit.

Approach 2 seems rather realistic, as the model reflects the method of the real system. Realism is only compromised with the fact that control line info values are autogenerated, but this also makes the approach more simple than approach 3. The approach can be be enhanced to approach 3, if the implementation allows for an easy replacement of control line info.

Approach 3 is of course the most realistic approach, as it models the real system very closely, but it is complex and very time-consuming, because the exact control line info values unfortunately only are available to us in paper format, not in any kind of electronic format. And there are 182 track circuits with perhaps two possible ways of traversal each having about five associated control line info values. Refer to [CLNDS] and [CLNDX] for further information.

Because of our time frame we discard approach 3. The question is now whether or not there is a noticeable difference in effect (seen in the outcome from a simulation based on the model) between approach 1 and approach 2. If not, there is no need to choose the more complex approach 2. In Section 3.1.2 the conclusion was that acceleration and deceleration *did* matter for the effect. This conclusion means that in a situation, as illustrated in Figure 3.1, where T1 and T2 run closely, there will be a difference in effect between approach 1 and approach 2.

The situation shown in Figure 3.1 can occur when the frequency of trains is high and a minor interruption propagates through the system. And because VR is sensitive to minor delays, we find that the control line method is the only method that matches the effect of the real system closely enough. The occurrence of a propagating minor delay is within the limits of normal state operation.

Another important argument is that approach 2 allows future insertion of the actual control lines, an enhancement that will put our effect closer to the effect of the real system.

We therefore choose approach 2 and decide the following:

- Control lines will be set to a length of five track circuits.
- Control line info will be autogenerated from track circuit length and an overall civil speed zone of 80 kmph, instead of using the actual values in [CLNDS] and [CLNDX].

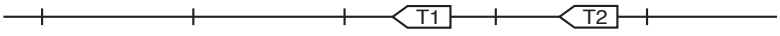

Figure 3.1: Two trains running closely

Having chosen to model control lines, others decisions are implied from this. We must divide the main line into track circuits in order to be capable of sending control line info and ATP Vehicle must calculate a SBD profile upon every reception of control line info.

Control line info The considered values from Table 2.2 are: Track Circuit ID, Line Speed, Target Speed, and Distance-to-Go. Direction Control is a security check, which we will disregard in the model, assuming the trains always run in the correct direction. The Next Carrier Frequency-field is left out, because we will not model changes in carrier frequencies, see later when track circuits are discussed.

Figure 3.2 shows the control line info values in our model. We use a fixed control line length of five, as mentioned, and simply sum the track circuit lengths to obtain the desired Distance-to-Go. The Distance-to-Go value in the rightmost triple is always 2900 metres, as it often is the case in the real system.

ATP Vehicle's intervention

A model of the train could either consist of an ATP Vehicle module and an ATO module or of one single module. An argument for the first solution is to keep the conceptual division of the real system. An argument for the latter solution is that we do not intend to model a misbehaving ATO system, because it certainly is not something that happens in normal state operation, which leaves out the need for ATP intervention. We choose to model a train in two modules, but we will not model ATP Vehicle's intervention in a misbehaving ATO system. Therefore, the ATP Vehicle part in our model will only calculate SBD profiles.

Interlocking

Because crossovers, as mentioned, are discarded, the only interlocking area is the bifurcation. The bifurcation interlocking has two parts. A bifurcation interlocking at track 1 and a bifurcation interlocking at track 2. We will model the bifurcation interlockings by maintaining queues for the settings requested by ATS. The settings are:[1]

- 'normal' (the direction to and from LGP)
- 'reverse' (the direction to and from VEA)

[1]The terms 'normal' and 'reverse' are used in the documentation in two contexts. One context uses the terms to describe the position of a switch (like we do here) and another uses the terms to refer to right-hand sided respectively left-hand sided driving.

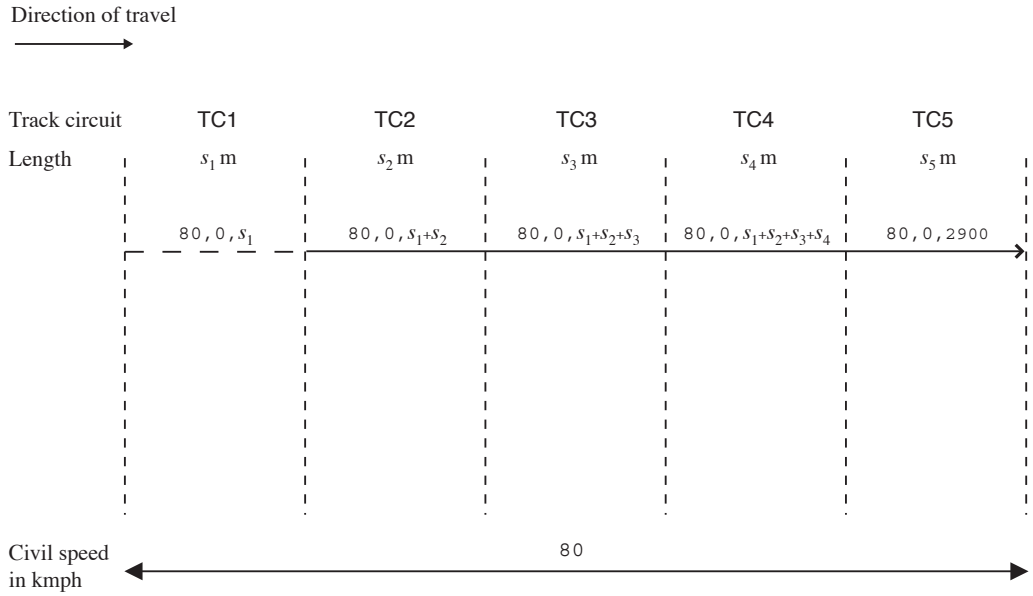

Figure 3.2: Modelled control line info

For the bifurcation at track 2, where the two lines merges, we will prevent two trains, departed from LGP respectively VEA, to collide at the bifurcation by setting a pseudo occupancy in one of the two tracks that merges the lines. The pseudo occupancy is set in either track circuit KHC2-6T or track circuit 3T, refer to [CLNDS], sheet CL010N for the exact position of these track circuits.

The bifurcations are illustrated on Figure 3.3 and Figure 3.4.

Equipment

The equipment included in our model are the results of the decisions taken in the above sections.

ATP Wayside equipment Instead of dividing the ATP Wayside system into several computers, we will look upon ATP Wayside as one unit. It is a simplification that cuts out all internal ATP Wayside communication between ATP Wayside computers situated at the SERs.

Track circuits We do not intend to model transmitters, receivers, antennas, and the use of carrier frequencies. These objects are merely of engineering interest, according to Section 1.3. In our model track circuits can transmit control line info and sense occupancy. Control line info is assumed to be received instantaneous.

3.3.2 Automatic Train Operation

As described in Section 2.6, ATO handles the operations that would otherwise be performed by a train driver, i.e.

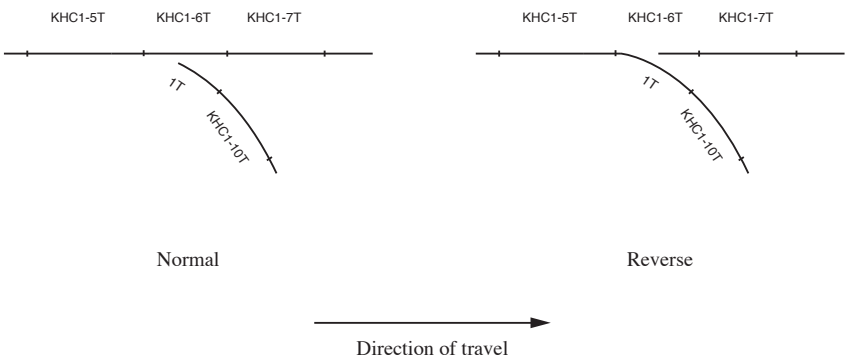

Figure 3.3: The bifurcation at track 1

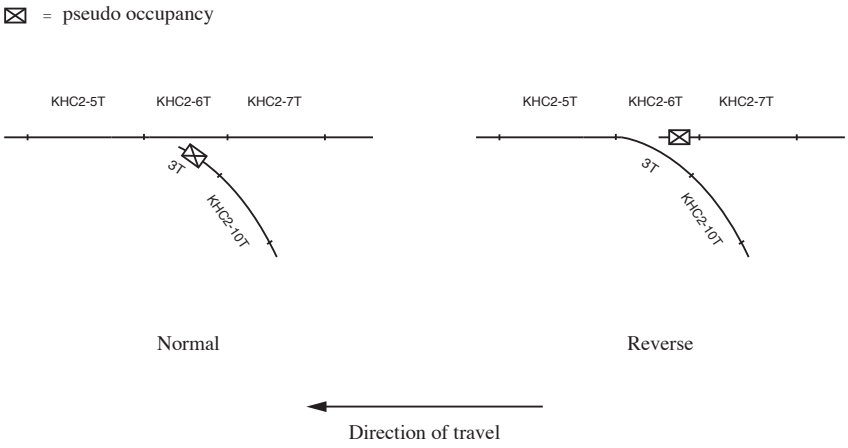

Figure 3.4: The bifurcation at track 2

- Speed regulation
- Programmed stopping
- Door handling
- Station departure

In the following we will discuss which of these operations we will model and how.

Speed regulation

From Section 3.1.2 it follows, that ATO must regulate the speed of the train by performing real speed adjustment, i.e. acceleration and deceleration, whenever indicated by the SBD profile calculated by ATP Vehicle. We will hence model speed regulation.

For the sake of simplicity we will assume that the train always obeys the SBD profile. Furthermore, we will assume the deceleration of the train is always exactly as prescribed by the SBD profile.

Programmed stopping

As described in Section 2.6.5, ATO must initiate a programmed stop when it approaches a station, if indicated by the DID. The programmed stop is initiated when the train passes the corresponding beacon track circuit and is performed in three stages. Since we have decided to model train speed adjustments by real adjustment, it is possible to model all three stages of a programmed stop.

The last stage of a programmed stop, crawl and final braking, primarily ensures that the train stops as close to the desired station stopping point as possible, i.e. minimise undershoot/overshoot. In most cases the consequence of an overshoot or undershoot is only a small delay of the final arrival of the train. Overshoot and undershoot can hence in most be modelled as delay of the final arrival of the train. We will hence assume that the trains always stop exactly at the desired stopping point and model overshoot and undershoot as a delay of the final arrival of the train, that is, we will not model crawl and final braking. Notice that we will assume that major overshoots, where the train continues to the next station, never occurs.

The modelling of a programmed stop is thus comprised of the modelling of the first two stages, i.e. constant speed and constant deceleration. For the sake of simplicity, we will consider these two stages as only one deceleration stage, by letting the train decelerate with the deceleration rate provided by the current performance level, such that it stops at the desired station stopping point. We will set the station stopping point to the end of the station track circuit.

The difference between modelling a programmed stop by only one deceleration stage rather then all three stages is very small and only time-wise. The effect of modelling a programmed as one stage is thus more or less the same as the effect of modelling all three stages. Notice however that the second stage of a programmed stop may be modelled by appropriately delaying the final arrival time of the train.

Under normal circumstances, the SBD profile calculated by ATP Vehicle is calculated by using the emergency braking rate. For the sake of simplicity we will model programmed stopping by calculating the SBD profile from the performance level deceleration rate and current distance to the end of the station track circuit. The train will hence be forced by the SBD profile to come to full stop at the end of the station track circuit.

We will not model the possibility to send a 'Skip Stop' to train, i.e. a train will always perform a programmed stop if indicated in the station stopping table. And once a programmed stop has been initiated it will be performed.

Door handling

In accordance with Section 2.5, we will not model door handling. Instead we will model the arrival and departure of trains at the stations solely by direct communication between ATO and ATS. When the train has come to a full stop at the station it sends a message to ATS indicating its arrival. The train then waits for ATS to send a 'departure' signal to the train indicating that it must leave the station. Since the train might be delayed by e.g. a passenger blocking the doors, the train must send a message to ATS when it leaves the station indicating the actual departure time.

The simplification of not modelling door handling should not influence on the effect of a station stop.

Station departure

We will assume that ATS always assures that all of the departure conditions presented in Section 2.6 are satisfied. Station departure is hence modelled as a simple command send from ATS to ATO. This assumption should not influence on effect of a station stop.

3.3.3 Automatic Train Supervision

From Section 2.7 it follows that ATS gathers information about the current state of the system, controls and coordinates overall traffic movement, provides a schematic overview of the entire main line for the TDs at CMC, and provides data to stations about each individual train.

In accordance with our focus, see Section 1.3, our model of ATS will not include the last two services. Hence we will model ATS as a traffic control system with the following responsibilities:

1. Gathering the information needed for traffic control
2. Planning future traffic movement based on the gathered information
3. Executing the generated plan
    a. using the plan as a basis for deciding what information to send to ATO and ATP Wayside
    b. sending information to ATO and ATP Wayside

Here responsibilities 1 and 3b basically consist of communication with non-ATS parts of our model, and responsibilities 2 and 3a are responsibilities of VR.

To accomplish our project goal, we wish to model ATS in a way that allows any given traffic control subsystem to replace our model of VR—as long as it requires only means that are currently at the disposal to the existing VR. This makes demands both of modularity of the ATS model and of a realistic interface between VR and the rest of the system.

We will meet the demand of modularity by dividing the model of ATS into the following parts:

- An information gathering part with responsibility 1
- A VR part with responsibilities 2 and 3a
- An information delivery part with responsibility 3b

The information gathering and information delivery parts will then be responsible for all communication between VR and the non-ATS part of the model. Hence they will form an interface.

We will meet the demand of realism of the interface by letting the two communication parts provide VR with the greatest possible subset of the means described in Section 2.7.5.

In the following we will first go into detail with the communication parts of ATS. Subsequently we will deal with the modelling of VR.

Information gathering

From 2.7.5 we have that the information available to VR in the real system is:

- The topography of the permanent way
- Speed restrictions
- The DIDs and trips supported by the system
- The performance levels supported by the system
- Configurations made by the TD
- A hastus plan
- The actual arrival and departure times of all trains in operation

We will hence require that the information gathering part of our model gathers this information. We will make one exception, though. Due to the fact that we have chosen only to simulate fully automatic operation the only thing in our model the TD can configure is therefore the different VR-parameters mentioned in Section 2.7.4. And the TD is not allowed to interfere during a simulation run. In consequence we choose not to include the TD as an actual part of the model. Instead we will simply make the VR-parameters input-parameters to the model and make these available from Central.

The actual arrival and departure times should be measured. The rest of the information is in the real system kept at Central. We will therefore also place it there. Hence in practice the gathering simply consists of giving access to information at Central and measuring arrival and departure times.

Information delivery

According to Section 2.7.5 VR should be able to

- put trains into operation from either CMC or a pocket track
- assign DIDs and trips to the trains, i.e. choose the routes they shall follow and at what stations they shall stop at
- control the dwell times of the individual trains in operation
- control the performance levels of the individual trains in operation
- reserve interlockings along the route of any train in operation
- take trains out of operation to either CMC or a pocket track

The information delivery part should thus make these possibilities available. As we have chosen only to model the interlocking at the bifurcation, see Section 3.3.1, the reserving is limited to the bifurcation.

The assignment of trip, DID and performance level to a train positioned at a station does not have any effect until the train leaves the station. And a train is not allowed to leave a station before having a trip, a DID and a performance level assigned to it. Therefore we find it appropriate to combine these assignments and the departure command into one single command. Hence the delivery part should make it possible for VR to send

- a 'put-in'-command to a train with a specification of a location
- a 'departure'-command to ATO containing info on trip, DID and performance level.
- a request to ATP wayside for reserving the bifurcation.
- a 'lay up'-command to a train with a specification of a location

Vehicle Regulation

Simulating VR realisticly is not directly a part of our goal. However some sort of model is needed in order to make a later simulation based on the model executable. And if a realistic model was made it would give a basis for comparison when developing new traffic control strategies.

Unfortunately we do not have sufficient information to make a fully realistic model of VR. Also it would be a rather time-consuming task. Therefore, instead we choose to make two rather simple VR models. One model containing the necessary functionality and nothing but this. And one model containing a bit more functionality.

The first one is meant simply to make an executable simulation. The second one is included partly to give us the opportunity to show whether a replacement of our VR model is possible and partly to give us the possibility to show that a more realistic VR algorithm is possible to make within the models limitations.

Both models will to some extend be based on a hastus schedule.

A very simple Vehicle Regulation model

A model of VR in Schedule Mode containing only the necessary functionality follows the hastus plan's put-in times. At each station the DID, the dwell time and performance level planned by hastus is used. And when the last entry in the hastus plan for a specific train is reached, the train is laid up.

VR reserves the bifurcation for a train, when arrives at or passes the station just before the bifurcation.

With no delays and a good hastus plan this should lead to the hastus plan being followed correctly.

A simple Vehicle Regulation model

A more realistic, but still simple model of the Vehicle Regulation subsystem reacts on delays. Hence such a model should contain a simplyfied version of VR's planning.

Planning We will simplify this phase by only producing base plans for the trains. This should not make our model much less realistic, since the base plans under normal circumstances are valid, see [CASS] p. 126. When making the base plans we will assume that a plan can always be completed at the current station. And we will only make plans for the current station departure.

This makes storing station plans unnecessary, since a plan will be used immediately after it has been generated. Therefore we will not have station plans.

Neither will we have a route plan. First of all because we have chosen not to model other interlockings than the one at the bifurcation. Secondly because we do not have much knowledge of when this reservation is made in the real system. Therefore we will simply reserve the bifurcation for a train, when it is one station away from it, just as in the very simple model, discussed in the previous section.

Plan execution The plans made by VR are followed. We will not model the check for abnormal conditions, as these will not occur during normal operation.

3.4 Central

From the way we have chosen to model the ATC system, it follows that we have to model Central containing information about

1. The performance levels supported by the system (needed by ATO and ATS)
2. Descriptions of geographical train movement (needed by ATO and ATS)
3. TD configurations (needed by ATS)
4. The topography of the permanent way (needed by ATS)
5. Speed limits (needed by ATS)
6. A hastus plan (needed by ATS)

Performance levels The performance level concept we will model by having nine triples containing information on acceleration rate, speed limit and deceleration rate corresponding to the nine performances levels in the real system.

Description of geographical train movement As described in Section 2.2 the geographical train movement in the real system is described by a DID, specifying a route and the stations to stop at on the route. Furthermore, a 'trip' is used to specify whether the DID is to be followed once or the DID is to be followed back and forth repeatedly until a new DID is specified.

We wish to provide VR, or any replacement of this, with the means that VR has in the real system. Therefore we choose to make our description of the geographical movement of a train, contain more

or less the same information as it does in the real system. However, we will change the description a bit to accomodate it to our model and to make the model as simple as possible without changing the effect.

In order to simplify the way we describe geographical movement we choose not to include 'trip'. Trip can be modelled explicitly by putting together DIDs and repeating these as many times as wanted. Thus leaving out trip should not affect neither the amount of information included in the description nor the expressiveness of it.

Also, we will simplify the station stopping tables, leaving out the information about beacon tracks as this information is only included in the station stopping tables for double checking that the train should stop. A simple 'stop' or 'do not stop' for each station on the route should suffice.

Furthermore we will not specify routes by specifying every single track circuit in it. We will only specify the stations that are passed along the route. Since have chosen not to include crossovers and to make the tracks one-way tracks, that specification should be sufficient to unambigiously identify the route.

We do not know how VR, in the real system, is informed about how a given route traverses the bifurcation. We will make this information a part of the DIDs. The consequence of this decision is that a DID can not specify a route that traverses the bifurcation more than once. However, this is not an actual limitation as any route traversing the bifurcation more than once can be expressed as two or more shorter routes.

We will gather the entire description of a geographicel movement in a model DID. And we will work with this as a unit. Hence a station stopping table is given to ATO as a whole DID.

The TD configurations The TD configurations consists of information for each station about:

- The performance levels VR is allowed to command a train leaving the station to use
- The minimum dwell time VR is allowed to use for a train situated on the station
- The maximum dwell time VR is allowed to use for a train situated on the station

The topography The information on the topography that is needed at Central is the positions of the stations and pocket tracks relative to each other and the position of each station relative to the bifurcation and the terminal stations. Also the distance between any two stations should be available. We will model this by for each station (and each pocket track) having information about

- the name of the track circuit
- the name of the following station track circuit (/pocket track circuit)
- the length of the stretch to the next station (/pocket track circuit)
- the number of stations to the bifurcation
- the number of stations to the terminal station

Notice that for some track circuits the above information might be ambiguous. For instance KHC1-2T (the track 1 platform on KHC) the next station track circuit is either ISB1-3T (the track 1 platform on ISB) or LGP1-3T (the track 1 platform on AMB), depending on whether the train is headed for VEA or for LGP. In cases like that all information should of course be available.

Speed limits The information about the speed limits serve the purpose of making it possible for VR to calculate the time it takes for a train to travel between any two stations using a given performance level. Unfortunately we do not know the exact character of the information in the real system. Therefore we have to model the information based on a qualified guess.

There are at least three different ways of estimating the time of travel between two stations. One way is to use time measures based on actual driving—this is according to [KJG] done by HASTUS.

Another way is to calculate the estimation based on the dominating speed limit on the stretch between the stations. A third way is to calculate the estimations based on the exact speed zones on the stretch.

The first one requires knowledge of how the a train actually moves in practice, hence we will not use this. The third one is not relevant in our case, as we only have one overall speed zone, see Section 3.3.1. Therefore we choose to use the second way that bases the calculations on the dominating speed limit, which is 80 kmph in all cases.

Yet, we do not want to prohibit others to use another strategy. Therefore we will model information on speed limits as being information for each station about:

- The length of the stretch to next station
- The dominating speed limit for the stretch to next station
- The individual speed zones the stretch to next station is divided into, the length of these and the speed limit in them

And we will if time allows make time measures available at Central.

Hastus plan

Since we will not have the exact same control lines and civil speed zones as the real system does, we can not just copy real hastus plans. Yet, we are interested in making it possible to use real hastus plans if the control lines are replaced by the real control lines.

Therefore we will make the model of hastus plans include more or less the same information as real hastus plans.

To make it easier to produce hastus plans we will model HASTUS in a very simplified version. Basically, we will seek to make it possible to generate a hastus plan for a train given the DIDs, the dwell times and the performance levels to use. In this way, experimenting with the simulation later on will be a lot easier, since one does not have to type in a new plan every time a new plan is needed. Though, as HASTUS is not the focus of our project, we will not have our HASTUS make any decisions on how to plan, this is up to the user. Notice that we will not include HASTUS itself in Central.

3.5 The Permanent Way and the Trains

In this section we describe the modelling of the permanent way and the trains.

3.5.1 The permanent way

The modelling of the permanent way will be divided into the modelling of the main line and CMC.

The main line

From Section 2.1.1 we have that the main line consists of the following parts

- Track circuits
- Crossovers
- Pocket tracks
- Stations
- The bifurcation
- Beacon track circuits

From Section 3.3 it follows that in order to model ATC and the operation of the Metro, we will as a minimum have to model the track circuits, the stations, the bifurcation, and the beacon track circuits. The track circuits provide the means for modelling ATP and the stations, the bifurcation and the beacon track circuits provides the means for modelling the general operation of the Metro.

To provide the means for VR to perform train frequency adjustments we will also model pocket tracks. Notice however that we will not model pocket tracks as track circuits but rather just as places along the main line where trains can be either stored or fetched.

From Section 3.1.1 it follows that we will not be modelling crossovers.

CMC

The area at CMC can be modelled as a place where trains can be either stored or fetched. We will not model train maintenance, i.e. washing, repairing, etc., see Section 2.6 p. 19. There is an upper bound of 34 for the number of trains that can be fetched from CMC.

3.5.2 The trains

To make a realistic model, we will model each train as a part of the system. From Section 3.3.1 it follows that a train must be modelled to have an ATO part and an ATP Vehicle part and that the two parts must be able to communicate directly. In order to model the functioning of the Vehicle Regulation function, we will model that a train is able to communicate with ATS at stations. To provide ATP Vehicle, ATO and ATS with the necessary information, the train must constantly know its current speed and position within the current track circuit. The position of the train is measured relative to the track circuit boundary behind the train.

For the sake of simplicity we will assume that a train has no length. We will hence not need to calculate how long it takes for train to get from one track circuit to another, but instead assume that this can be done in zero seconds. This may seem as an unrealistic assumption, but since a train is only 39 metres long, the impact on the total simulation time will be negligible.

3.6 Communication between ATC Subsystems

The modelling of the communication between the different subsystems of ATC is illustrated on Figure 3.5.

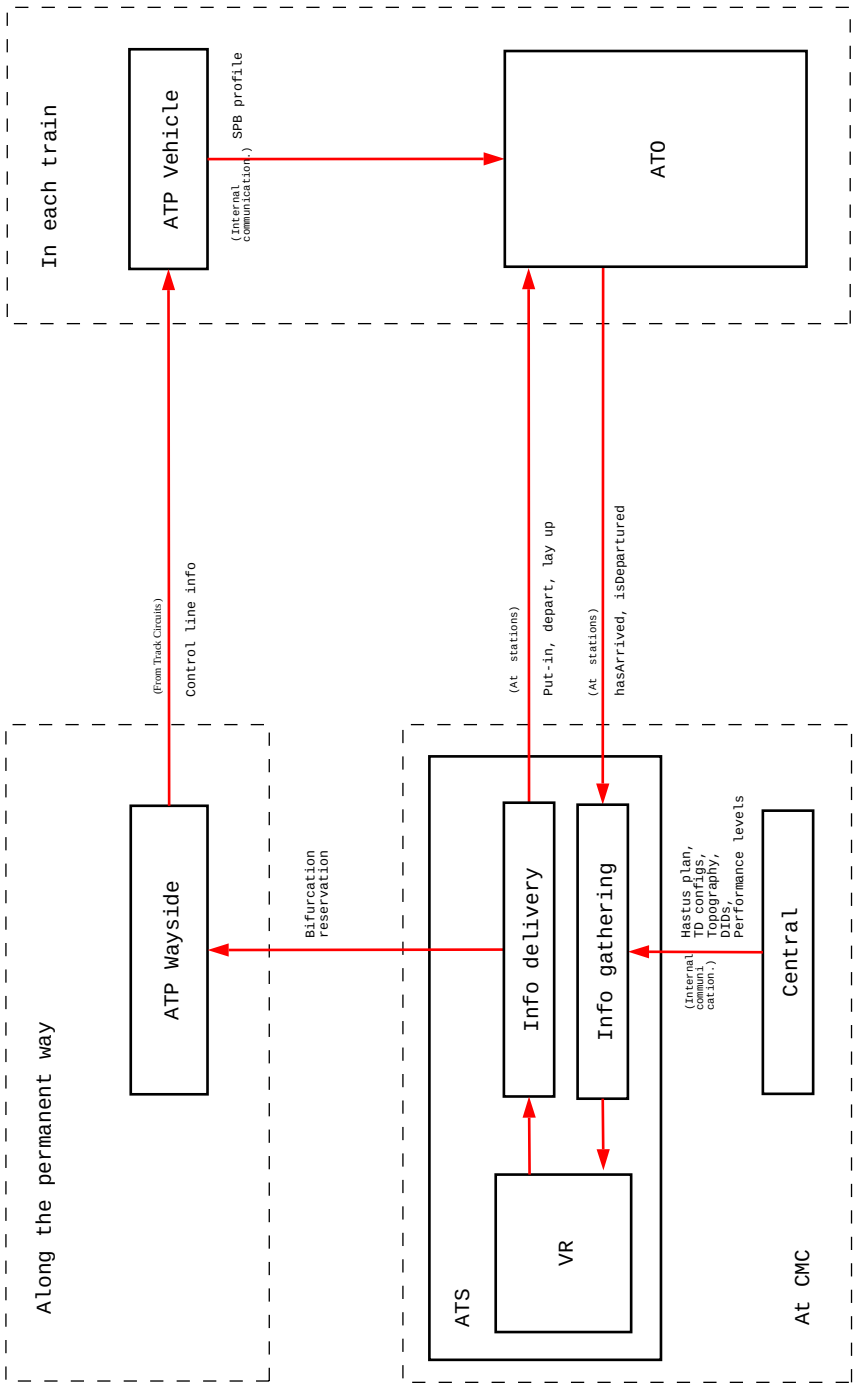

Figure 3.5: Model of the communication between ATC subsystems

3.7 Evaluation of Traffic Control Strategies

In order to make sense to testing different traffic control strategies a metric is necessary. Else comparison of different strategies is not possible.

Service Availability, described in Section 2.9, is one possible metric. Yet, it is far from the only reasonable metric.

The reasonableness of a given metric depends highly of the point of view the operation is seen. The owner of the Metro, Ørestad Development Corporation, is interested in keeping the train arrival frequencies they have promised to the passengers. The passengers might find it just as important that the number of train changes they need to make on their way to their destination is minimised. And the company in charge of the operation of the Metro, Metro Service, might find it of greatest importance to minimise the number of trains in operation, since less trains and less personnel cost less.

The different criterias for optimality gives occasion for different metrics. And so does a mixture of them.

As we do not want to force the use of one specific criteria of optimality on the model, we will not model the Service Availability evaluation. Instead we will strive to make a series of information on the simulation runs available. Then the developers of other traffic strategies can choose their own criteria for optimality and use the available information in accordance with that criteria.

As a minimum, all arrival times of all stations should be available after a simulation run making it possible to produce the Service Availablity score.

Chapter 4

Simulation of the Copenhagen Metro

In this chapter we will address the implementation of the simulation of the Copenhagen Metro. Our intend is to implement a process-based discrete event simulation.

In Section 4.1 we will discuss the overall process design. In Section 4.2 we will describe the interaction between the processes and hereby the event-flow that makes up the simulation. And finally in Section 4.3 we will give a description of our implementation of the simulation.

4.1 Entity Design

As mentioned our intent is to make a process-based discrete event simulation, see [DESJ] for further information on this type of simulation. More specifically we will use the SimJava 2.0 simulation package for Java. SimJava is described in [SJT].

SimJava uses the term 'entity' to refer to a process in the simulation. Entities in SimJava communicate through communication channels that are connected by 'ports'.

In the following we discuss which entities to include in the simulation.

4.1.1 ATP Wayside

We will discuss three possibilities for implementing ATP Wayside in simulation:

1. **Track circuit entities and an ATP Wayside entity.** The ATP Wayside entity maintains information about track circuit occupancy and the sequence of track circuits. The track circuit entities do only communicate with the trains. ATS sends interlocking reservations to the ATP Wayside entity that passes on the information to the relevant track circuit entities.
2. **Track circuit entities only.** Each track circuit has knowledge of its nearest neighbours. It must know the neighbours ahead in order to check for track circuit occupancy. Whenever a occupancy change occurs in the track circuit entity, the neighbours behind must be informed in order to send updated control line info. The trains communicate with the track circuit entities. ATS sends interlocking reservations to the relevant track circuit entities. Each train and ATS must have a communication channel to every track circuit.
3. **An ATP Wayside entity only.** The ATP Wayside entity maintains information about track circuit occupancies and the sequence of track circuits on the main line. The trains and ATS communicates only with the ATP Wayside entity.

Possibility 1 is closest to our model of the real system, but it will give an unnecessary link causing unnecessary communication.

Possibility 2 decentralises the tasks of ATP Wayside, which could be a simplification. Interlocking is divided, and the track circuit entities must communicate on this task, which probably will make interlocking difficult to implement.

Possibility 3 will gather all occupancy surveillance at one place, which might make it complex to implement. Interlocking is controlled from one place.

All three possibilities will give the same effect, but 2 and 3 will also leave out unnecessary communication, and we therefore discard possibility 1.

We find that the possible simplicity of possibility 2 gained by the decentralisation is more than lost by the complexity of the interlocking control. Thus, we choose to implement possibility 3, i.e. ATP Wayside is implemented as one entity, WAYSIDE.

4.1.2 ATP Vehicle and ATO

From Section 3.5.2 we have that a train must have an ATO part and an ATP Vehicle part and that the two parts must be able to communicate directly. We also have from Section 3.3.1 that the only responsibility ATP Vehicle has in our model is to provide ATO with the current SBD profile. And we have from Section 3.3.2 that our model ATO always obeys the SBD profile and that the deceleration of the train is always exactly as prescribed by the SBD profile.

Therefore, we see no reason to keep the parts seperated, and we find it simpler two treat them as one.

Since the responsibility of ATO is to run the train, a train can not be simulated without the existence of an ATO part. For the sake of simplicity we will hence simulate each train as an entity with the necessary ATO and ATP Vehicle functionalities. ATO and ATP Vehicle will hence not be simulated as entities but rather as functionalities within each train entity.

4.1.3 Central

Central shall provide data for ATS, hence we wish to include it in our design. However since the information that Central is to provide is static we see no reason to make Central an entity.

4.1.4 ATS

Concerning ATS we have considered the following three possibilities:

1. Implementing ATS as three entities, one for each part of our ATS model
2. Implementing ATS as two entities, one for the two communication parts and one for VR
3. Implementing ATS as one entity containing all three parts

As mentioned, communication between entities in SimJava is implemented through message-sending. The information gathering hence consists of giving access to Central and of receiving messages from the trains. Likewise, the information delievery consists of sending messages to WAYSIDE and to the trains.

If ATS is divided into more than one entity a lot of unnecessary message-sending will take place. All of the messages ATS is to gather from the trains would be sent to the information delivery or communication entity by the train and then forwarded to VR. And all of the messages that ATS is to deliver for VR would be sent to the information delivery or communication delivery entity and then forwarded to a train or to WAYSIDE. On the other hand a division would make the simulation reflect our model closely.

If ATS is implemented as a single entity, much less message-sending will take place. But the model will not be reflected as closely. Yet, the effect of the division made in the model of ATS can still be obtained. ATS can be realised as an abstract class implementing the communication part as public accessor methods. And VR can correspondingly be realised as a class that extends ATS by implementing only the actual entity actions. Hereby it is possible to obtain the wanted interface and the replaceability.

Therefore, we choose to have ATS as a single entity, ATS.

4.1.5 Interruption

In order to be able to insert interruptions in the simulation at any simulation time, the interruption class must be an entity and thereby gain access to the simulation time. Hence, we will model interruptions in an interruption entity, INTERRUPTION.

4.2 Entity Interaction

In the following section we will describe the interaction between the chosen entities and hereby the event-flow that will make up the simulation.

Based on our model of the Metro operation and on the fact that we intend to make a process-based discrete event simulation, we will simulate the operation of the Metro as described in the following. The term 'message' we will use meaning an event holding some information. And the term 'sleep' we will use, to indicate when an entity is to be considered busy for a period of simulation time.

Put-in

When ATS wants to put-in a train on the main line, it sends a put-in message to that train specifying the location, where the train should be put in. The train reacts by sending a put-in request-message to WAYSIDE.

If it is safe to put in the train on the specified location WAYSIDE updates the track circuit, the train is to be put into, as occupied. WAYSIDE then sends a message with control line info to all the train on main line. The train is then in operation.

If it is not safe to put in the train, the train will sleep for a second and then send a another put-in request.

Train movement

When a train T is in operation, it sends a message to WAYSIDE each time it reaches the end of a track circuit. This message contains information on which track circuit the train has reached the end of. In this way, WAYSIDE knows, when a change in track circuit occupancy happens. WAYSIDE reacts on such a message by sending out appropriate information to every train in operation. This information includes information on the track circuit the individual train is in: the track circuit name, the track circuit type and possibly a track circuit delay. Also it includes control line info decided by possible track occupancy in the train's control line.

If T is allowed to continue, WAYSIDE updates the track circuit occupancy state before sending out messages. The message that T receives from WAYSIDE in this case contains the track circuit name of the next track circuit in the track circuit sequence.

If T is not allowed to continue the message will contain the name of the track circuit that the train was supposed to leave. Due to the control line concept this can only happen if the train has zero speed at the track circuit border. The train will in this case not move again until it receives control line info indicating that movement is allowed.

Train movement from one track circuit border to the next, is simulated by making the train sleep for the period of time it would take for the train to traverse the track circuit in the real system.

For making sure that T follows its assigned DID, ATS is to handle the reservation of the bifurcation switches for T. ATS reserves a bifurcation switch by sending a message to WAYSIDE with an indication of which switch and of how this switch should be positioned.

Station stopping

When a train enters a beacon track circuit, it checks its DID to check whether or not it should stop on the approaching station. If it is not supposed to stop, it proceeds as described above. If it is supposed to stop, it initiates a programmed stop. When the train has performed the programmed stop and has reached the station, it sends a message to ATS indicating that it has arrived to that specific station. After an appropriate dwell ATS send a depart-message to the train including DID and performance level specifications. When the train leaves the station it sends a message to ATS indicating its departure.

Lay up

When ATS wants to lay up a train, it sends a lay up-message to the train in question. The train then sends a message to WAYSIDE indicating that it no longer wishes to be on the main line. WAYSIDE then removes the track circuit occupancy caused by the train.

Interruptions of service

Any interruptions of service will be effectuated as a message specifying the duration of a delay, X, and a track circuit name, TC. The delay is sent from INTERRUPTION to WAYSIDE. WAYSIDE responds by sending the delay to the next train that occupies TC. That is, the next time WAYSIDE sends a message for a train in TC, the message will include a specification of the delay duration X. The train will then stay in TC for X seconds longer than it would otherwise have stayed. After the delay is given to the train, the delay is deleted.

An overview of the communication

The communication between the entities of the simulation is shown in Figure 4.1. Also communication with Central is shown.

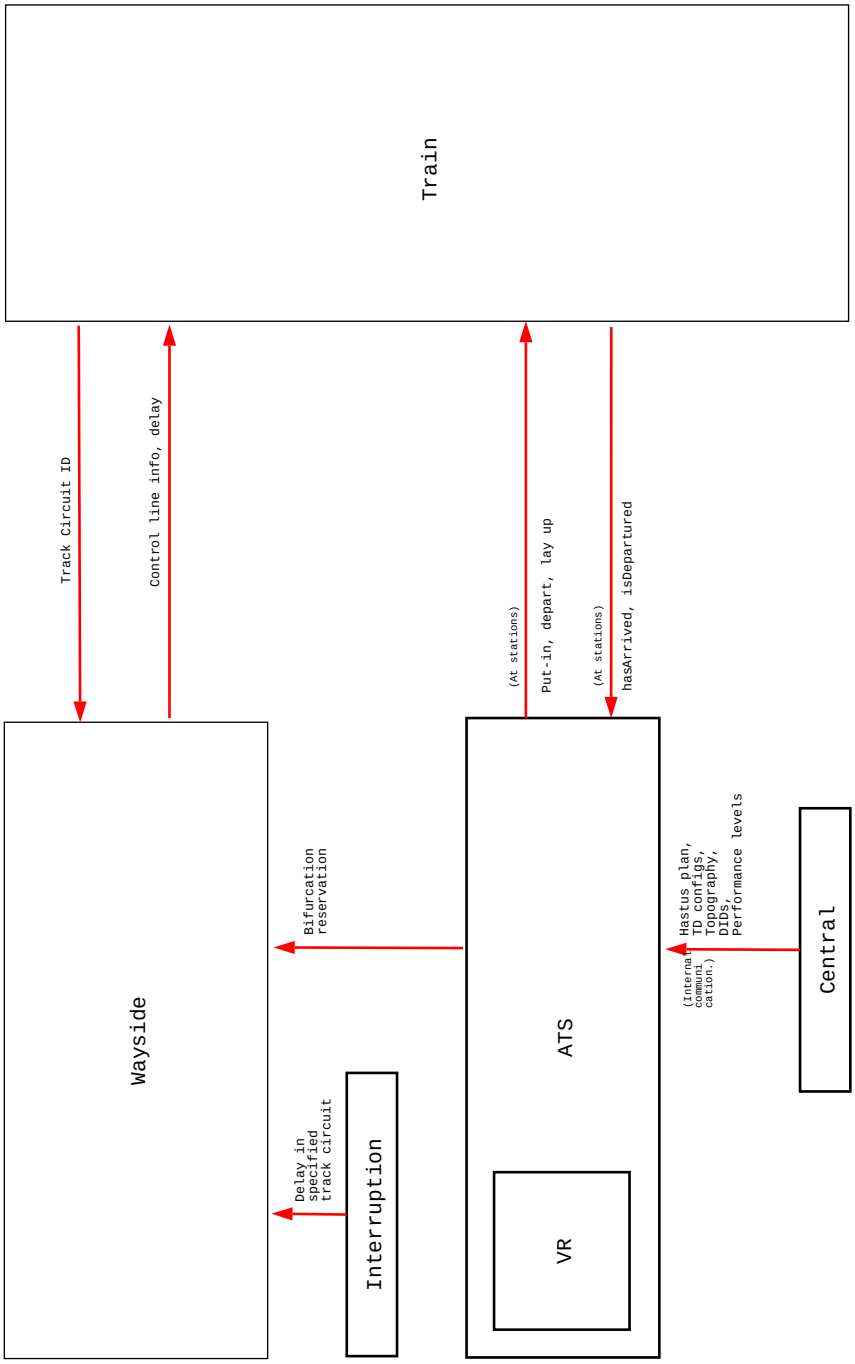

Figure 4.1: Entity communication

4.3 The Programme

The programme we have developed, consists of a Java applet providing the means for performing an animation of the simulation.

The programme is commented in the source code, hence in this section we will only describe the essential parts: the implementation of the four entities and CENTRAL.

Notice that since the runtime for the simulation has no impact on the simulation time, we have not strived to optimise the use of algorithms and data structures.

The source code can be seen in Appendix B.

4.3.1 The ATP Wayside entity

In the following we will describe our implementation of the ATP Wayside entity.

`Wayside`

For the `Wayside` entity we will describe the base information used, and the procedures that manipulates this information.

To keep track of track circuit occupancy and to define the sequence of the track circuits on the main line `Wayside` maintains four arrays, and a hash table, `map`, that maps track circuit names to an index and an array number in one of the arrays of instances of the `TrackCircuit` class. `TrackCircuit` objects holds data on a specific track circuit such as name, length, and control lines. The arrays are:

- `VeaToKn`, containing all track circuits from VEA to KN on track 2
- `KnToVea`, containing all track circuits from KN to VEA on track 1
- `LgpToKn`, containing all track circuits from LGP to KN on track 2
- `KnToLgp`, containing all track circuits from KN to LGP on track 1

The arrays references the same `TrackCircuit` entities on the joined stretches.

Control lines are implemented as arrays of `ControlLineInfo` objects containing Line Speed, Target Speed, and Distance-to-Go as integers.

`Wayside` also maintains a queue for requested bifurcation settings for track 1 and for track 2 sent from ATS.

`InitialiseWayside`

The information in `Wayside` is initialised by the static class `InitialiseWayside`. The initialisation is done outside `Wayside` in order to allow an easier substitution of control line info or addition of track circuits.

Control line length and control line info values can be changed by rewriting the last part of the `init`-method in `InitialiseWayside`. For example, the actual control line info values could be inserted.

The main line can be extended beyond KN and beyond LGP by adding track circuits in the `init`-method of the class `InitialiseWayside`. The track circuits should be added to both the hash table and to one or two of the track circuit arrays. Extension beyond VEA with new stations is not immediately possible, but a such extension would also be unthinkable in the real system.

Deficiencies

We did not find the time to implement pocket tracks, hence we do only have the possibility to put-in and lay up trains from CMC, with starting station VEA.

The end stations do only use the outgoing platform. This means that only one train can be situated at an end station at a time, and it will influence on the arrival time of train, if it arrives to an end

station with an occupied platform. It is a simplification of the needed crossover move at end stations, and can be improved by allowing the use of both platforms at end stations.

4.3.2 The train entity

We have chosen to implement the train entity as a single class named `Train`. Each train in the system will hence be an instance of the train class. The `Train` class implements the required ATO and ATP Vehicle funtionalities. In the following we will describe how the train entity calculates the time that it should sleep in a given track circuit.

Track circuit occupancy time

The calculation of the occupancy time is subject to the speed restrictions provided by the performance level and the current SBD profile. As described in Section 2.6, the train may receive new control line info while still in the current track circuit. In this case ATP Vehicle must calculate a new SBD profile, and ATO must react accordingly.

The calculation of the new SBD profile is based on the new control line info, the current speed of the train and distance within the current track circuit at the time of the reception of the control line.

When new control line info is received, several different situations might occur, dependent on the current speed of the train. The current speed of the train might be lower than the speed limit provided by either the performance level or the new SBD profile. In order to obtain the desired speed, the train must hence commence an acceleration. During this acceleration, the train might however reach the end of the track circuit. In the real system this does not mean anything—the train will just continue its acceleration until reaching the desired speed. However, since we have decided to implement an event simulation, the train must, in this case, temporarily stop its acceleration and wait for an acceptance from ATP Wayside to proceed to the following track circuit. This situation is illustrated i figure 4.2.

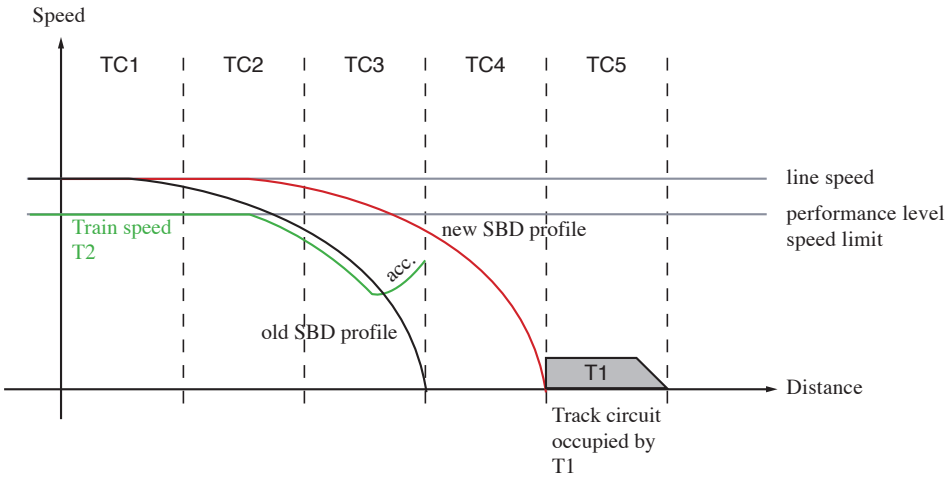

Figure 4.2: A train approching an occupied track circuit. T1 moves from TC4 to TC5 before T2 has fully stopped, why T2 must commence acceleration to regain the desired speed. T2 reaches the end of the track circuit before having obtained the desired speed.

When considering the above situation, it is not obvious what will happen if the train is not

allowed to proceed to the next track circuit. In this case, the train thus must come to a full stop within zero metres which may seem as an unrealistic situation. However, this situation will under normal circumstances never occur. If the following track circuit had been occupied, the SBD profile would not have allowed the train to accelerate in the first place.

4.3.3 The ATS entity

In the following, the classes we have made in order to implement the ATS entity is described.

`ATS`

`ATS` is the implementation of the communication part of ATS. It is an abstract class. All means that ATS is to make available to traffic control subsystems are implemented as public accessor methods. All communication ports are made private so that an implementations of a traffic control strategy can only communicate with the ATC system through `ATS`.

The body method of `ATS` is empty. Any implementation of a traffic control subsystem shall extend `ATS` overriding its body. Hereby the wanted interface between the traffic control subsystem and the rest of the system is obtained.

`ATSWithVerySimpleVR`

`ATSWithVerySimpleVR` is an implementation of the Very Simple VR model, described in Section 3.3.3. It extends `ATS` and overrides its body method with traffic control actions.

The calculations used by `ATSWithVerySimpleVR` for estimating travel time is implemented in a seperate class, `Calc`. This we have done because the same calculations are needed by the implementation of HASTUS and because the seperation is practical in case the estimation method later is subject to improvements.

Deficiencies

We did not find the time to implement the Simple VR algorithm described in Section 3.3.3.

4.3.4 Central

In the following, the classes we have made to implement Central and its contents are described.

`Central`

Central itself is implemented as a static class, `Central`. The class has the responsibility of having and giving access to knowledge about the system's performance levels, the system's DIDs, the current HASTUS plan, the TDs configurations, the topography of the permanent way and the speed limit of the permanent way. In the following the implementations of the essential information classes are described.

DIDs We have implemented the DID concept by having an abtract class `DID`. And five classes implementing actual DIDs, `DID1`, `DID2`, `DID3`, `DID4` and `DID5`.

`DID` has the attributes route, consisting of a list of consecutive stations, station stopping table, indicating which stations are to be stopped at, and bifurcation setting, indicating how the train should traverse the bifurcation. The class provides accessor and iterator functionalities for both the route and the station stopping table. The none-abstract DID classes extends `DID`, adding an appropriate constructor for initialisation of the attributes of `DID`.

Each train is given an instance of the DID they are to follow.

The idea behind this choice of design is that several train should be able to follow the same DID at the same time only a bit staggered. Hence, it should be possible for two or more trains to iterate

the DID simultaniously, but staggered. Also, since DIDs only differ in the choice of route, station stopping table and bifurcation setting, we found it convenient to gather the iterator and accessor implementation in `DID`.

`DID1` describes the route from KN to VEA with stops on all stations on the way. `DID2` describes the route from KN to LGP with stops on all stations on the way. `DID3` describes the route from LGP to KN with stops on all stations on the way. `DID4` describes the route from VEA to KN with stops on all stations on the way. And `DID5` describes the route from KN to VEA with stops on all stations on the way, except for KHS, UNI and ISB.

The current hastus plan For clarity, we have chosen to divide the hastus plan into several 'hastus train plans', one for each train that is to be in operation. These train plans are represented as instances of the `HASTUSTrainPlan` class. Each train plan consists of and a number of consecutive hastus plan entries each with more or less the same information as in a real hastus plan entry: information about the planned DID, a station on the route described in the DID, planned arrival time, planned departure time (and thereby planned dwell), planned performance level to use and planned run time to next station. This entrances are implemented as instances of the `HASTUSPlanEntry` class.

The TD Configurations, the topography of the permanent way and speed restrictions All three types of information is station/pocket track based in the sense that the information can be divided intuitively between the stations/pocket tracks. Therefore we have chosen to keep most of the information offered for `ATS` as a whole. For this purpose we have made the class `StationAndStretchInfo`. This class has the following attributes:

- Topography
  - The name of the track circuit
  - The name of the following station/pocket track circuit
  - The length of the stretch to the next station/pocket track
  - The number of stations to the bifurcation
  - The number of stations to the terminal station
- Speed restrictions
  - The length of the stretch to next station/pocket track
  - The dominating speed limit for the stretch to the next station/pocket track
  - The individual speed zones the stretch to the next station/pocket track is divided into, the length of these and the speed limit in them
- The TD configurations
  - The performance levels VR is allowed to tell a train leaving the station/pocket track to use
  - The minimum dwell time VR is allowed to use for a train situated on the station/in the pocket track
  - The maximum dwell time VR is allowed to use for a train situated on the station/in the pocket track.

InitialiseCentral

Since we wish to be able to change control lines and expand the permanent way, we have chosen to put most of the initialisation of the information contained in `Central` into a seperate class called `InitialiseCentral`. By replacing this entire class or just some of its methods the data contained in `Central` can thus be changed without having to change `Central` itself.

As parametres to the `init`-method, the TDs configurations and a hastus plan must be given. That is, for each station, the allowed performance levels and minimum and maximum dwell times are to be specified together with a hastus plan.

Notice that the information about the topography, as it is currently, does not include information about pocket tracks, since `Wayside` does not yet support the use of these.

`HASTUS`

A hastus plan can be generated by using the methods of the class `HASTUS`. This class' sole responsibilty is to generate hastus plans based on info about the trains that are to be in operation, what DIDs that are to be used for each of the trains, the dwell time and the performance level that are to be used and the time that each change of DID should take.

4.3.5 The interruption entity

The `Interruption` entity can insert a delay of X seconds to a specific `TrackCircuit` entity. The delay is inserted directly, i.e. not by SimJava communication ports. This is done as a simplification of the port communication, and because we see the interruption entity as an artificial unit that, of course, is not a part of the real system.

Improvements

The `Interruption` entity merely a skeleton for an insertion of a more sophisticated interruption generation. It will be improved by using actual statistics of interruptions of service from the daily operation of the Metro. This would increase the simulation's degree of realism.

4.3.6 Evaluation of traffic control strategies

In order to provide the means for evaluating different traffic control strategies, we have decided to maintain a log of the operation. We log the actual arrival and departure times for the trains at the stations. It is however possible to add logging wherever needed. Especially, it is possible to log additional information in any implementation of a traffic control strategy.

To perform the logging we have implemented a static class called `Print` providing a method called `log`. The `log` method is really just a wrapping of the Java `System.out.println()`-method, but we have added the possibility to turn the logging on and off. Also the wrapping gives the possibility of changing the output media. Log messages can be piped from stdout to a file.

4.3.7 Animation

To illustrate the simulation of the Metro system, we have used the SimJava package `simanim`.

Description of the animation

The animation consists of two parts. The first part animates the permanent way and the movement of the trains. The permanent way is animated by the track circuits and the trains are animated as traingles indicating their direction of travel. In order to be able to differentiate between the trains, the PVID for each train is showed.

The second part animates the communication between the trains, ATS and ATP Wayside. Communication is animated as a small dot moving between the communicating parts when the communication takes place. To illustrate what information is sent between the different parts of the system, the dot is in some cases associated with a text showing the relevant parts of the contents of the message sent between the two parts of the system.

Communication between the trains and ATP Wayside The communication between each train and ATP Wayside is illustrated by showing the values of the Line Speed-, Target Speed- and Distance-To-Go fields of the control line infos sent from ATP Wayside to the trains.

Communication between the trains and ATS The communication between each train and ATS is illustrated by showing the actual arrival and departure times of the trains at the stations. When a train stops at a station, it turns red.

Communication between ATS and ATP Wayside The communication between ATS and ATP Wayside is illustrated by showing the bifurcation setting requests, ATS sends to ATP Wayside.

Comments

When using the `simanim` package only entities are animated. In order to animate the permanent way, we therefore decided to let each track circuit be an entity.

When running the simulation it may seem as if the trains are constantly moving, without performing the necessary stops implied by e.g., station stops or delays. However the movement of the trains is only used to illustrate what is going on. The actual simulation is indicated by the simulation time showed in the applet.

It may also seem as if the bifurcation is not set correctly at all times during the simulation. Especially it may seem as if parts of the bifurcation sometimes disappear. Fortunately, this is not the case. It is the consequence of how the generated GIF-images are used. And due to the time frame of the project, we have decided not to improve this part of the simulation.

We have made it possible to choose via the applet, which test run to run. A possible future extention is to make it possible to specify all of the parameters needed for the testing of different traffic control strategies, e.g. max and min dwell times, the number of trains, etc.

Delimitations

Since it is a very time-requiring task to generate all of the GIF-images needed to animate the complete system, we have only generated enough GIF-images to support three trains in the system. Notice however that it is possible to run a simulation with more trains, but it will not be possible to differentiate between the trains except by their positions.

Chapter 5

Evaluation

In this chapter we will shortly evaluate our work on the project. In Section 5.1 we will test the basic functionalities of the simulation. In the Summary on page 4 a conclusion on the project is given.

5.1 Testing

The creation of the simulation programme has been supported by internal testing. This section describes the external testing of the simulation. Because of the time frame of the project, the testing performed on the implemented simulation programme is not a complete testing. Rather, we would like to make probable that the simulation is working as intended to.

The functionalities we would to like make probable that our simulation covers, are:

1. VR can
   (a) put-in trains
   (b) lay up trains
2. VR can route the trains usings DIDs
   (a) stopping at all stations
   (b) stopping at a subset of the stations
3. VR can read and use a hastus plan
4. VR can decide dwell time and performance level
5. VR can reserve the bifurcation switches
6. Correct control line info is sent to the trains
7. The trains obey the control line info
8. The bifurcation switches work as intended
9. Delays can be inserted to a specific track circuit and the trains reacts appropriately to this.

Testing will be carried out as a visual approval of the animation of five simulation runs and an approval of the logs from these runs. We have tested the following cases and their coverage of the above functionalities is shown in Table 5.1.

1. A train can run and stop at stations according to a hastus plan via DIDs. The train reserves the bifurcations correctly, i.e. it doesn't stop before one of the bifurcations.
2. A train can run and stop only at a subset of the stations.
3. Two trains can be put-in, run according to hastus plans, and be laid up.
4. A train T2 stays at least one track circuit behind another train T1 on the same track. If T1 stops, T2 cannot pass.
5. If the bifurcation at track 2 is reserved for normal setting, trains coming from VEA cannot pass.

Test case	**Functionalities covered**
1	1a, 1b, 2a, 3, 4, 5, 8
2	1a, 1b, 2a, **2b**, 3, 4, 5, 8
3	1a, 1b, 2a, 3, 4, 5, 8
4	1a, 1b, 2a, 3, 4, 5, **6**, **7**, 8, 9
5	1a, 1b, 2a, 3, 4, 5, **6**, **7**, **8**, 9

Table 5.1: Functionalities covered by test cases

The five test cases are described in Appendix A.1. Screenshots from the test runs can be found in Appendix A.3. Logs from the test runs can be found in Appendix A.4.

5.1.1 Results

The test results can be seen in Appendix A.2. The results showed no errors, except for test case 5. This test case revealed an error (most probably in the `Train` entity) that caused the train to stop permanently after it had to wait for a pseudo occupancy to move.

Furthermore, we observed that he trains occupied the track circuits longer than expected. While investigating the problems we discovered an error in the calculations of the track circuit occupancy time. More specifically, the train erroneously presumes that its current speed is 0 at each entry to a track circuit. This could be the source of the problems, but unfortunately we have not had the time to solve it.

As all other test cases went well, we will conclude that we have made probable that the programme is in a state, where it, with a little effort, can be corrected to work as intended to.

Bibliography

[VASS] Allen, Gregory: *Vehicle ATC System Specification*,
KBB V 00 PE03 2B3 06100, Rev. 07
Union Switch & Signal, 2001

[WASS] Ananthashankaran, Girish: *Wayside ATC System Specification*,
KBB V 00 PE01 2B4 07002, Rev. 07
Union Switch & Signal, 2000

[CNAOM] Barker, Michael: *Central & NVLE ATC Operator Manual*,
KBB V 00 PE02 2B2 05004, Rev. 05,
Union Switch & Signal, 2002

[Mek] Christiansen, Gunnar, Erik Both, Preben Østergaard Sørensen: *Mekanik*,
Institut for Fysik, Danmarks Tekniske Universitet, 2000

[DESJ] Helsgaun, Keld: *Discrete Event Simulation in Java*,
`http://www.dat.ruc.dk/~keld/research/JAVASIMULATION/JAVASIMULATION-1.0/docs/Report.pdf`,
Roskilde University, 2000

[BCLDC] Repasi, Roger: *Block and Control Line Design Criteria*,
KBB V 00 PE00 2BX 01012, Rev. 2.0,
Union Switch & Signal, 1999

[SJT] Simatos, Costas: *The SimJava Tutorial*,
`http://www.dcs.ed.ac.uk/home/simjava/tutorial/`,
The University of Edinburgh, 2002

[CASS] Theriault, David: *Central ATC System Specification*,
KBB V 00 PE02 2B2 05003, Rev. 02,
Union Switch & Signal, 1998

[CLNDS] *Block and Control Line Design Document: Control Lines Normal Direction Straight*,
KBB V 00 PE01 2B4 07301, Rev. 03,
Union Switch & Signal, 2002

[CLNDX] *Block and Control Line Design Document: Control Lines Normal Direction X-over*,
KBB V 00 PE01 2B4 07301, Rev. 03,
Union Switch & Signal, 2002

[CLRDS] *Block and Control Line Design Document: Control Lines Reverse Direction Straight*,
KBB V 00 PE01 2B4 07301, Rev. 03,
Union Switch & Signal, 2002

[CLRDX] *Block and Control Line Design Document: Control Lines Reverse Direction X-over*,
KBB V 00 PE01 2B4 07301, Rev. 03,
Union Switch & Signal, 2002

[Hastus] The file `P12ASAT-1.csv`, Ørestad Development Corporation

[Intro] *An introduction to Copenhagen's new traffic system*, Ørestad Development Corporation, 2002

[TSATC] *Technical Specifications: Automatic Train Control (ATC)*, Ørestad Development Corporation

[TS] *Track Schematic, Phase 1, 2 and 3*, COWI, 2002

[GSF] Personal communication with Gunni S. Frederiksen, M.Sc. Computer Science, Project Manager Railway Technique, Ørestad Development Corporation

[KJG] Personal communication with Kjeld Jørgensen, M.Sc. Elec. Eng., Senior Engineer, ATC (Automatic Train Control), Metro Construction Management

[Op] Opinion on Chapter 2 from Gunni S. Frederiksen, M.Sc. Computer Science, Project Manager Railway Technique, Ørestad Development Corporation

Acronyms

ATC Automatic Train Control

ATP Automatic Train Protection

ATO Automatic Train Operation

ATS Automatic Train Supervision

Central A "database" containing the necessary information for the operation of the Metro

CMC Control and Maintenance Centre

DID Destination ID

HASTUS Application for generating train plans

PVID Permanent Vehicle ID

SAP Service Availability Point

SER Service Equipment Room

SBD profile Safe Braking Distance profile

VR Vehicle Regulation

Appendix A

Testing

A.1 Test cases

The input to the test cases are described in the following and the source code for the input to the test cases can found in Appendix B, `Metro.java`. The two versions of the interruption entity are `Interruption1.java` and `Interruption2.java`. The expected outcome describes, what we expect to see in the animation and from the log.

Test case 1

The test case is selected by entering '1' in the applet window.

Test case input A hastus plan is generated for one train. The hastus plan specifies that the train should be put in at VEA at time 0, and that the train is to follow the DIDs: `DID4`, `DID2`, `DID3`, `DID1`. That is, the train is to drive VEA–KN, KN–LGP, LGP–KN, and KN–VEA, i.e. a butterfly loop and stop at all stations. The hastus plan also specifies that the train should dwell for 30 seconds at each in-line stations and 40 seconds at each terminal station and that the train should use performance level 2.

Expected outcome The train is put-in at time 0 on VEA, runs the butterfly loop, stopping at all stations, dwells for 30 seconds at in-line stations and 40 seconds at terminal stations and the train is then laid up. All control line info should contain the values (80,0,2900).

Test case 2

The test case is selected by entering '2' in the applet window.

Test case input As case 1, but with `DID4` replaced by `DID5`, indicating that the train should not stop at KHS, UNI, and ISB on its way from VEA to KN.

Expected outcome As case 1, though, the train should not stop at KHS, UNI, and ISB on its way from VEA to KN.

Test case 3

The test case is selected by entering '3' in the applet window.

Test case input A hastus plan is generated for two trains. The first hastus plan is identical with the hastus plan from case 1. The second one differs only in that the put-in time is 500 instead of 0.

Expected outcome The first train should behave like the train in case one. The second likewise only it should not be put in until time 500.

Test case 4

The test case is selected by entering '4' in the applet window. The class `Interruption1` is used in this test case.

Test case input A hastus plan is generated for two trains. The hastus plan specifies that the first train, T0, should be put in at VEA at time 0, and that T0 should follow the DIDs: `DID4`, `DID2`, `DID3`, `DID1`. That is, the T0 is to drive VEA–KN, KN–LGP, LGP–KN, and KN–VEA, i.e. a butterfly loop, and stop at all stations. The hastus plan also specifies that T0 should dwell for 30 seconds at each in-line station and 40 seconds at each terminal station and that the train should use performance level 2.

For the second train, T1, the hastus plan specifies that it should be put in at VEA at time 500, and that it should follow the DIDs: `DID4`, `DID2`, `DID3`, `DID2`, `DID3`, `DID2`, `DID3`, `DID1` That is, T1 is to drive VEA–KN, KN–LGP, LGP–KN, KN–LGP, LGP–KN, KN–LGP, LGP–KN and KN–VEA stopping at all stations. The hastus plan also specifies that T1 should dwell for 30 seconds at each in-line station and 40 seconds at each terminal station and that the train should use performance level 2, like T0 train.

T0 is given a delay of 1000 seconds in the track circuit ORE2-3T, between ORE and BC.

Expected outcome T0 will enter VEA at time 0 and will proceed to ORE2-3T, where it will sleep for 1000 seconds. T1 will enter VEA at time 500 and will proceed to the track with control line info (80,0,2900) until it is five track circuits away. Then the control line info should indicate that the train should stop just before entering ORE2-3T. The train should stop accordingly. Both trains will stand still until the delay is over. When the delay is over T1 stays closely behind T0 until T0 heads for VEA and T1 continues to circulate between KN and LGP. Whenever the trains are closer than five track circuits the control lines for T1 should indicate this.

Test case 5

The test case is selected by entering '5' in the applet window. The strategy for this test case is to make a train wait for another train at the bifurcation switch at track 2. This is done by giving a train a delay right after it has reserved the bifurcation switch, but before it has traversed the switch. Another train requesting the switch will then have to wait. The class `Interruption2` is used in this test case.

Test case input The test case is identical to test case 3 except for two things: the second train is put-in at time 5000 instead of time 500, and a delay of 10000 is set in track circuit KHC2-9T, between AMB and the bifurcation.

Expected outcome The first train starts its butterfly loop but stops in KHC2-9T. The second train reaches the track circuit five track circuit before the bifurcation switch. Here the control lines should start indication that it is not to enter the bifurcation. After the first train has completed its delay, both trains complete their butterfly loops.

A.2 Test results

Table A.1 shows the results of the test cases.

Test case	**Comments**
1	As expected
2	As expected
3	As expected
4	As expected
5	As expected except for the fact that the train stopped by the pseudo occupancy did not start running again after the pseudo occupancy was removed.

Table A.1: Test results

A.3 Screenshots

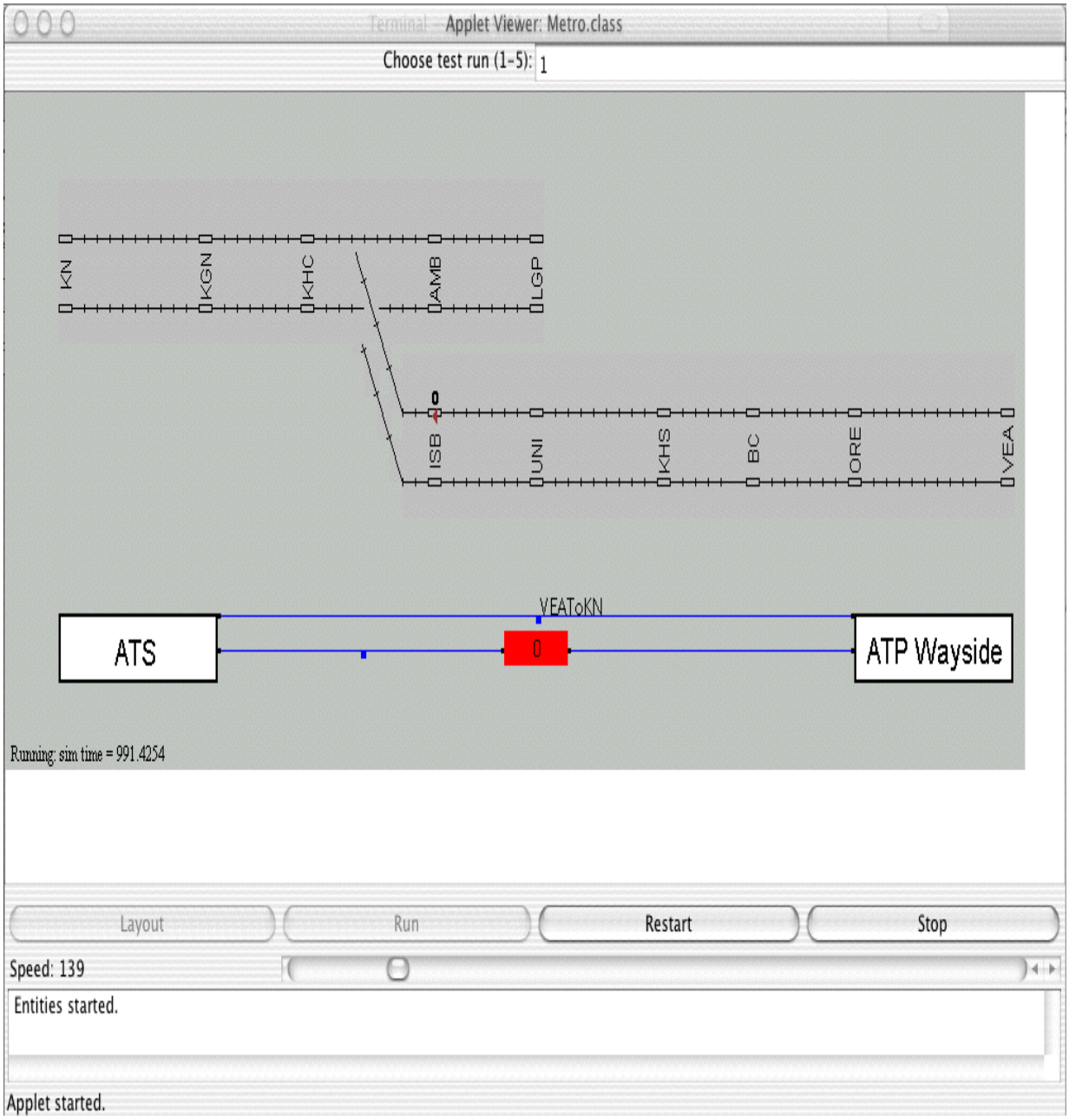

Figure A.1: Screenshot from test case 1

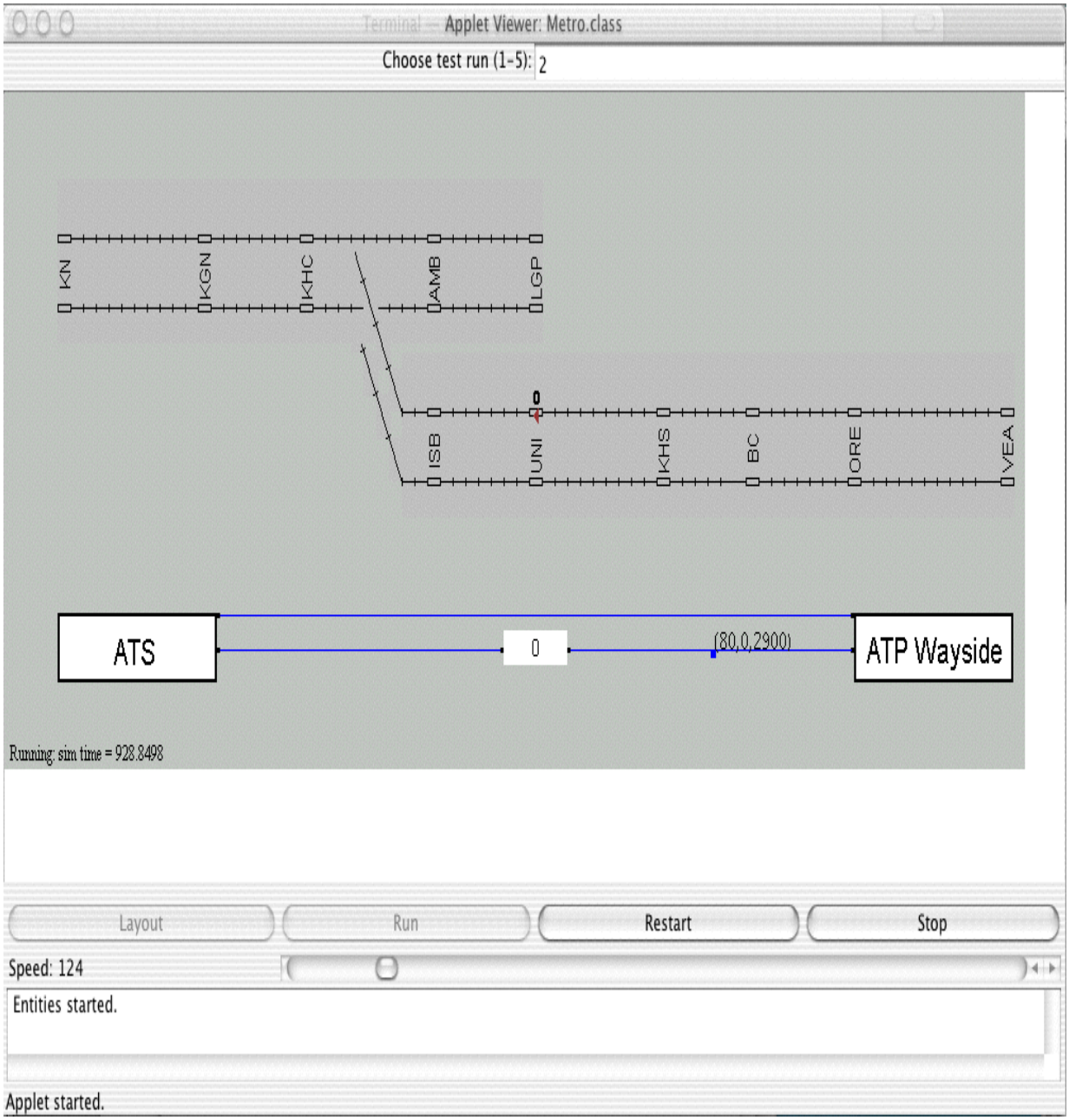

Figure A.2: Screenshot from test case 2

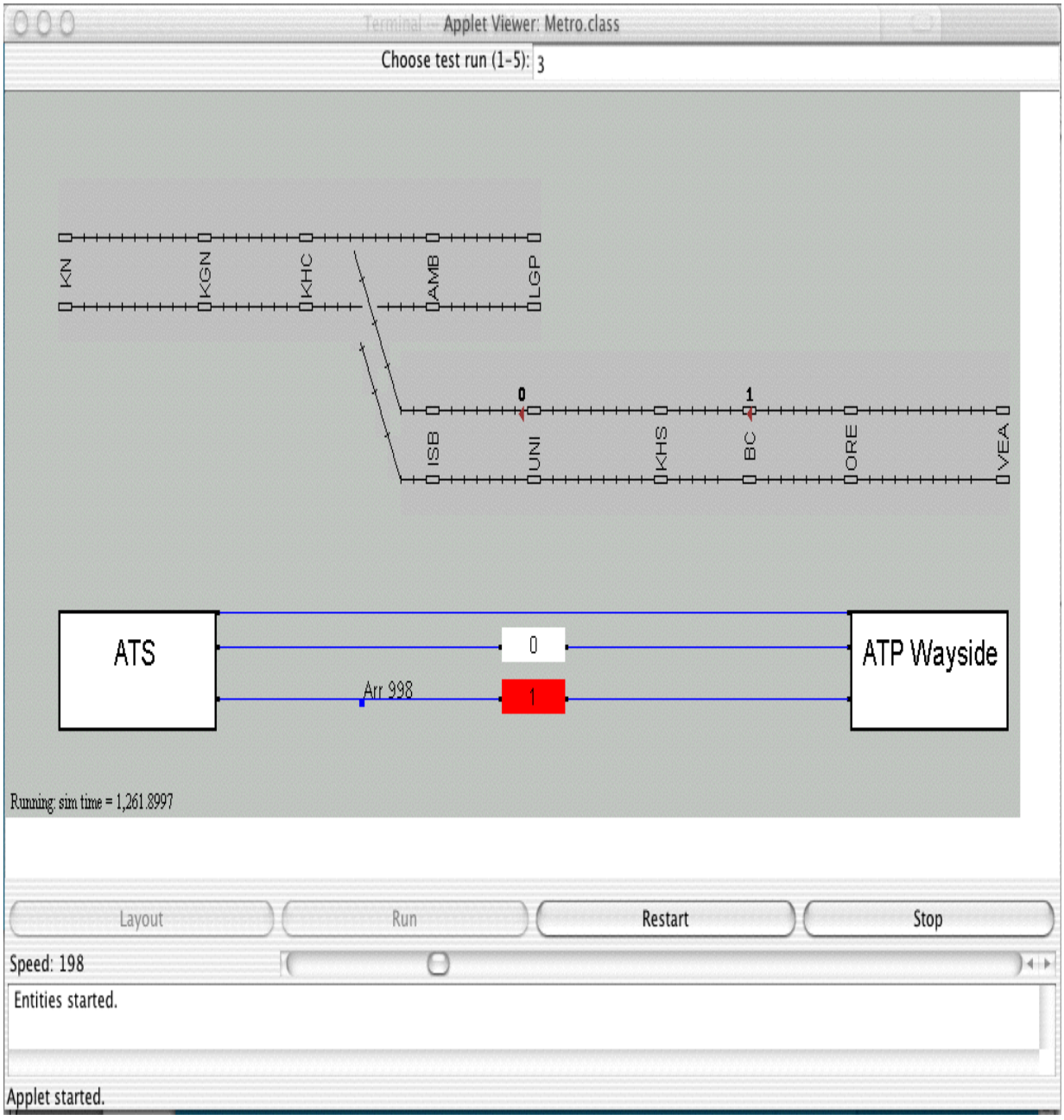

Figure A.3: Screenshot from test case 3

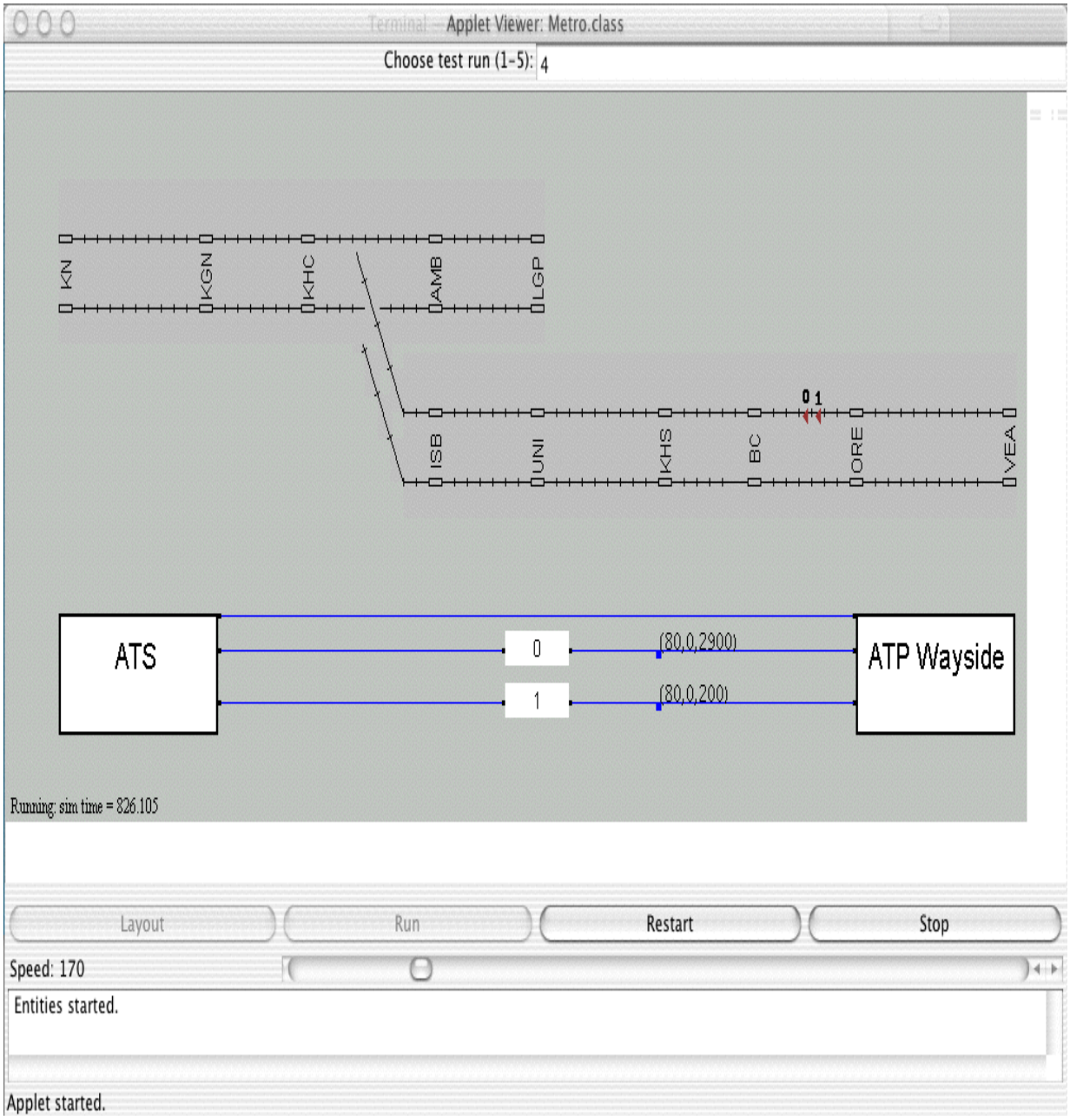

Figure A.4: Screenshot from test case 4

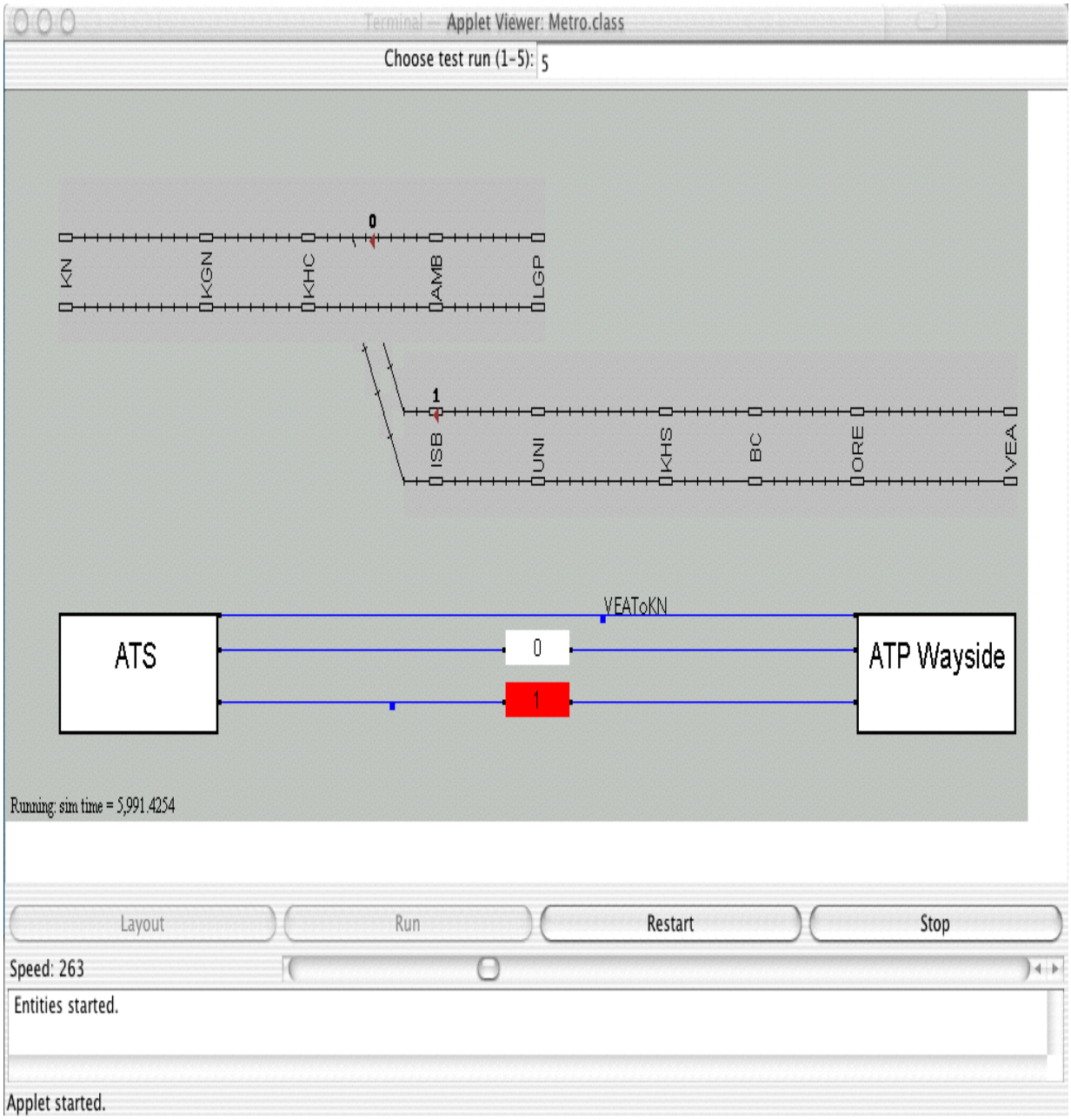

Figure A.5: Screenshot from test case 5

A.4 Logs

Log from test case 1

```
Actual arr. time:0 Train:0 Station:VEA2-8T
Actual dep. time:30 Train:0 Station:VEA2-8T
Actual arr. time:244 Train:0 Station:ORE2-7T
Actual dep. time:274 Train:0 Station:ORE2-7T
Actual arr. time:456 Train:0 Station:KHS2-10T
Actual dep. time:486 Train:0 Station:KHS2-10T
Actual arr. time:616 Train:0 Station:KHS2-3T
Actual dep. time:646 Train:0 Station:KHS2-3T
Actual arr. time:818 Train:0 Station:UNI2-2T
Actual dep. time:848 Train:0 Station:UNI2-2T
Actual arr. time:991 Train:0 Station:ISB2-3T
Actual dep. time:1021 Train:0 Station:ISB2-3T
Actual arr. time:1224 Train:0 Station:KHC2-2T
Actual dep. time:1254 Train:0 Station:KHC2-2T
Actual arr. time:1410 Train:0 Station:KGN2-3T
Actual dep. time:1440 Train:0 Station:KGN2-3T
Actual arr. time:1641 Train:0 Station:KN1-2T
Actual dep. time:1681 Train:0 Station:KN1-2T
Actual arr. time:1894 Train:0 Station:KGN1-3T
Actual dep. time:1924 Train:0 Station:KGN1-3T
Actual arr. time:2092 Train:0 Station:KHC1-2T
Actual dep. time:2122 Train:0 Station:KHC1-2T
Actual arr. time:2346 Train:0 Station:LGP1-3T
Actual dep. time:2376 Train:0 Station:LGP1-3T
Actual arr. time:2558 Train:0 Station:LGP2-11T
Actual dep. time:2598 Train:0 Station:LGP2-11T
Actual arr. time:2779 Train:0 Station:LGP2-3T
Actual dep. time:2809 Train:0 Station:LGP2-3T
Actual arr. time:3014 Train:0 Station:KHC2-2T
Actual dep. time:3044 Train:0 Station:KHC2-2T
Actual arr. time:3200 Train:0 Station:KGN2-3T
Actual dep. time:3230 Train:0 Station:KGN2-3T
Actual arr. time:3431 Train:0 Station:KN1-2T
Actual dep. time:3471 Train:0 Station:KN1-2T
Actual arr. time:3683 Train:0 Station:KGN1-3T
Actual dep. time:3713 Train:0 Station:KGN1-3T
Actual arr. time:3881 Train:0 Station:KHC1-2T
Actual dep. time:3911 Train:0 Station:KHC1-2T
Actual arr. time:4123 Train:0 Station:ISB1-3T
Actual dep. time:4153 Train:0 Station:ISB1-3T
Actual arr. time:4345 Train:0 Station:UNI1-2T
Actual dep. time:4375 Train:0 Station:UNI1-2T
Actual arr. time:4583 Train:0 Station:KHS1-3T
Actual dep. time:4613 Train:0 Station:KHS1-3T
Actual arr. time:4785 Train:0 Station:KHS1-9T
Actual dep. time:4815 Train:0 Station:KHS1-9T
Actual arr. time:4997 Train:0 Station:ORE1-7T
Actual dep. time:5027 Train:0 Station:ORE1-7T
Actual arr. time:5236 Train:0 Station:VEA2-8T
```

Log from test case 2

```
Actual arr. time:0 Train:0 Station:VEA2-8T
Actual dep. time:30 Train:0 Station:VEA2-8T
Actual arr. time:244 Train:0 Station:ORE2-7T
Actual dep. time:274 Train:0 Station:ORE2-7T
Actual arr. time:456 Train:0 Station:KHS2-10T
Actual dep. time:486 Train:0 Station:KHS2-10T
Actual arr. time:2058 Train:0 Station:KHC2-2T
Actual dep. time:2088 Train:0 Station:KHC2-2T
Actual arr. time:2244 Train:0 Station:KGN2-3T
Actual dep. time:2274 Train:0 Station:KGN2-3T
Actual arr. time:2475 Train:0 Station:KN1-2T
Actual dep. time:2515 Train:0 Station:KN1-2T
Actual arr. time:2728 Train:0 Station:KGN1-3T
Actual dep. time:2758 Train:0 Station:KGN1-3T
Actual arr. time:2925 Train:0 Station:KHC1-2T
Actual dep. time:2955 Train:0 Station:KHC1-2T
Actual arr. time:3180 Train:0 Station:LGP1-3T
Actual dep. time:3210 Train:0 Station:LGP1-3T
Actual arr. time:3392 Train:0 Station:LGP2-11T
Actual dep. time:3432 Train:0 Station:LGP2-11T
Actual arr. time:3613 Train:0 Station:LGP2-3T
Actual dep. time:3643 Train:0 Station:LGP2-3T
Actual arr. time:3848 Train:0 Station:KHC2-2T
Actual dep. time:3878 Train:0 Station:KHC2-2T
Actual arr. time:4034 Train:0 Station:KGN2-3T
Actual dep. time:4064 Train:0 Station:KGN2-3T
Actual arr. time:4265 Train:0 Station:KN1-2T
Actual dep. time:4305 Train:0 Station:KN1-2T
Actual arr. time:4517 Train:0 Station:KGN1-3T
Actual dep. time:4547 Train:0 Station:KGN1-3T
Actual arr. time:4715 Train:0 Station:KHC1-2T
Actual dep. time:4745 Train:0 Station:KHC1-2T
Actual arr. time:4957 Train:0 Station:ISB1-3T
Actual dep. time:4987 Train:0 Station:ISB1-3T
Actual arr. time:5179 Train:0 Station:UNI1-2T
Actual dep. time:5209 Train:0 Station:UNI1-2T
Actual arr. time:5417 Train:0 Station:KHS1-3T
Actual dep. time:5447 Train:0 Station:KHS1-3T
Actual arr. time:5618 Train:0 Station:KHS1-9T
Actual dep. time:5648 Train:0 Station:KHS1-9T
Actual arr. time:5831 Train:0 Station:ORE1-7T
Actual dep. time:5861 Train:0 Station:ORE1-7T
Actual arr. time:6069 Train:0 Station:VEA2-8T
```

Log from test case 3

```
Actual arr. time:0 Train:0 Station:VEA2-8T
Actual dep. time:30 Train:0 Station:VEA2-8T
Actual arr. time:244 Train:0 Station:ORE2-7T
Actual dep. time:274 Train:0 Station:ORE2-7T
Actual arr. time:456 Train:0 Station:KHS2-10T
```

```
Actual dep. time:486 Train:0 Station:KHS2-10T
Actual arr. time:500 Train:1 Station:VEA2-8T
Actual dep. time:530 Train:1 Station:VEA2-8T
Actual arr. time:690 Train:0 Station:KHS2-3T
Actual dep. time:720 Train:0 Station:KHS2-3T
Actual arr. time:800 Train:1 Station:ORE2-7T
Actual dep. time:830 Train:1 Station:ORE2-7T
Actual arr. time:1223 Train:0 Station:UNI2-2T
Actual dep. time:1253 Train:0 Station:UNI2-2T
Actual arr. time:1262 Train:1 Station:KHS2-10T
Actual dep. time:1292 Train:1 Station:KHS2-10T
Actual arr. time:2081 Train:1 Station:KHS2-3T
Actual arr. time:2081 Train:0 Station:ISB2-3T
Actual dep. time:2111 Train:0 Station:ISB2-3T
Actual dep. time:2111 Train:1 Station:KHS2-3T
Actual arr. time:3029 Train:1 Station:UNI2-2T
Actual dep. time:3059 Train:1 Station:UNI2-2T
Actual arr. time:3078 Train:0 Station:KHC2-2T
Actual dep. time:3108 Train:0 Station:KHC2-2T
Actual arr. time:3333 Train:1 Station:ISB2-3T
Actual dep. time:3363 Train:1 Station:ISB2-3T
Actual arr. time:3372 Train:0 Station:KGN2-3T
Actual dep. time:3402 Train:0 Station:KGN2-3T
Actual arr. time:3832 Train:1 Station:KHC2-2T
Actual dep. time:3862 Train:1 Station:KHC2-2T
Actual arr. time:3895 Train:0 Station:KN1-2T
Actual dep. time:3935 Train:0 Station:KN1-2T
Actual arr. time:4014 Train:1 Station:KGN2-3T
Actual dep. time:4044 Train:1 Station:KGN2-3T
Actual arr. time:4279 Train:0 Station:KGN1-3T
Actual dep. time:4309 Train:0 Station:KGN1-3T
Actual arr. time:4446 Train:1 Station:KN1-2T
Actual dep. time:4486 Train:1 Station:KN1-2T
Actual arr. time:4505 Train:0 Station:KHC1-2T
Actual dep. time:4535 Train:0 Station:KHC1-2T
Actual arr. time:4826 Train:1 Station:KGN1-3T
Actual dep. time:4856 Train:1 Station:KGN1-3T
Actual arr. time:4875 Train:0 Station:LGP1-3T
Actual dep. time:4905 Train:0 Station:LGP1-3T
Actual arr. time:5109 Train:1 Station:KHC1-2T
Actual dep. time:5139 Train:1 Station:KHC1-2T
Actual arr. time:5147 Train:0 Station:LGP2-11T
Actual dep. time:5187 Train:0 Station:LGP2-11T
Actual arr. time:5814 Train:0 Station:LGP2-3T
Actual arr. time:5814 Train:1 Station:LGP1-3T
Actual dep. time:5844 Train:0 Station:LGP2-3T
Actual dep. time:5844 Train:1 Station:LGP1-3T
Actual arr. time:6469 Train:1 Station:LGP2-11T
Actual dep. time:6509 Train:1 Station:LGP2-11T
Actual arr. time:6591 Train:0 Station:KHC2-2T
Actual dep. time:6621 Train:0 Station:KHC2-2T
Actual arr. time:6712 Train:1 Station:LGP2-3T
Actual dep. time:6742 Train:1 Station:LGP2-3T
```

```
Actual arr. time:6914 Train:0 Station:KGN2-3T
Actual dep. time:6944 Train:0 Station:KGN2-3T
Actual arr. time:7091 Train:1 Station:KHC2-2T
Actual dep. time:7121 Train:1 Station:KHC2-2T
Actual arr. time:7409 Train:0 Station:KN1-2T
Actual dep. time:7449 Train:0 Station:KN1-2T
Actual arr. time:7458 Train:1 Station:KGN2-3T
Actual dep. time:7488 Train:1 Station:KGN2-3T
Actual arr. time:8029 Train:0 Station:KGN1-3T
Actual dep. time:8059 Train:0 Station:KGN1-3T
Actual arr. time:8068 Train:1 Station:KN1-2T
Actual dep. time:8108 Train:1 Station:KN1-2T
Actual arr. time:8291 Train:0 Station:KHC1-2T
Actual dep. time:8321 Train:0 Station:KHC1-2T
Actual arr. time:8509 Train:1 Station:KGN1-3T
Actual dep. time:8539 Train:1 Station:KGN1-3T
Actual arr. time:8558 Train:0 Station:ISB1-3T
Actual dep. time:8588 Train:0 Station:ISB1-3T
Actual arr. time:8763 Train:1 Station:KHC1-2T
Actual dep. time:8793 Train:1 Station:KHC1-2T
Actual arr. time:8801 Train:0 Station:UNI1-2T
Actual dep. time:8831 Train:0 Station:UNI1-2T
Actual arr. time:9189 Train:0 Station:KHS1-3T
Actual arr. time:9189 Train:1 Station:ISB1-3T
Actual dep. time:9219 Train:0 Station:KHS1-3T
Actual dep. time:9219 Train:1 Station:ISB1-3T
Actual arr. time:9583 Train:0 Station:KHS1-9T
Actual dep. time:9613 Train:0 Station:KHS1-9T
Actual arr. time:9622 Train:1 Station:UNI1-2T
Actual dep. time:9652 Train:1 Station:UNI1-2T
Actual arr. time:9827 Train:0 Station:ORE1-7T
Actual dep. time:9857 Train:0 Station:ORE1-7T
Actual arr. time:9927 Train:1 Station:KHS1-3T
Actual dep. time:9957 Train:1 Station:KHS1-3T
Actual arr. time:10043 Train:0 Station:VEA2-8T
Actual arr. time:10386 Train:1 Station:KHS1-9T
Actual dep. time:10416 Train:1 Station:KHS1-9T
Actual arr. time:10598 Train:1 Station:ORE1-7T
Actual dep. time:10628 Train:1 Station:ORE1-7T
Actual arr. time:10837 Train:1 Station:VEA2-8T
```

Log from test case 4

```
Actual arr. time:0 Train:0 Station:VEA2-8T
Actual dep. time:30 Train:0 Station:VEA2-8T
Actual arr. time:244 Train:0 Station:ORE2-7T
Actual dep. time:274 Train:0 Station:ORE2-7T
Actual arr. time:500 Train:1 Station:VEA2-8T
Actual dep. time:530 Train:1 Station:VEA2-8T
Actual arr. time:744 Train:1 Station:ORE2-7T
Actual dep. time:774 Train:1 Station:ORE2-7T
Actual arr. time:1415 Train:0 Station:KHS2-10T
Actual dep. time:1445 Train:0 Station:KHS2-10T
```

```
Actual arr. time:1463 Train:1 Station:KHS2-10T
Actual dep. time:1493 Train:1 Station:KHS2-10T
Actual arr. time:2263 Train:0 Station:KHS2-3T
Actual dep. time:2293 Train:0 Station:KHS2-3T
Actual arr. time:2371 Train:1 Station:KHS2-3T
Actual dep. time:2401 Train:1 Station:KHS2-3T
Actual arr. time:2641 Train:0 Station:UNI2-2T
Actual dep. time:2671 Train:0 Station:UNI2-2T
Actual arr. time:2851 Train:1 Station:UNI2-2T
Actual dep. time:2881 Train:1 Station:UNI2-2T
Actual arr. time:2899 Train:0 Station:ISB2-3T
Actual dep. time:2929 Train:0 Station:ISB2-3T
Actual arr. time:3149 Train:1 Station:ISB2-3T
Actual dep. time:3179 Train:1 Station:ISB2-3T
Actual arr. time:3339 Train:0 Station:KHC2-2T
Actual dep. time:3369 Train:0 Station:KHC2-2T
Actual arr. time:3514 Train:1 Station:KHC2-2T
Actual dep. time:3544 Train:1 Station:KHC2-2T
Actual arr. time:3563 Train:0 Station:KGN2-3T
Actual dep. time:3593 Train:0 Station:KGN2-3T
Actual arr. time:3730 Train:1 Station:KGN2-3T
Actual dep. time:3760 Train:1 Station:KGN2-3T
Actual arr. time:4039 Train:0 Station:KN1-2T
Actual dep. time:4079 Train:0 Station:KN1-2T
Actual arr. time:4214 Train:1 Station:KN1-2T
Actual dep. time:4254 Train:1 Station:KN1-2T
Actual arr. time:4385 Train:0 Station:KGN1-3T
Actual dep. time:4415 Train:0 Station:KGN1-3T
Actual arr. time:4434 Train:1 Station:KGN1-3T
Actual dep. time:4464 Train:1 Station:KGN1-3T
Actual arr. time:4709 Train:0 Station:KHC1-2T
Actual dep. time:4739 Train:0 Station:KHC1-2T
Actual arr. time:4748 Train:1 Station:KHC1-2T
Actual dep. time:4778 Train:1 Station:KHC1-2T
Actual arr. time:5149 Train:0 Station:LGP1-3T
Actual dep. time:5179 Train:0 Station:LGP1-3T
Actual arr. time:5197 Train:1 Station:LGP1-3T
Actual dep. time:5227 Train:1 Station:LGP1-3T
Actual arr. time:5383 Train:0 Station:LGP2-11T
Actual dep. time:5423 Train:0 Station:LGP2-11T
Actual arr. time:5476 Train:1 Station:LGP2-11T
Actual dep. time:5516 Train:1 Station:LGP2-11T
Actual arr. time:5542 Train:0 Station:LGP2-3T
Actual dep. time:5572 Train:0 Station:LGP2-3T
Actual arr. time:5725 Train:1 Station:LGP2-3T
Actual dep. time:5755 Train:1 Station:LGP2-3T
Actual arr. time:5976 Train:0 Station:KHC2-2T
Actual dep. time:6006 Train:0 Station:KHC2-2T
Actual arr. time:6039 Train:1 Station:KHC2-2T
Actual dep. time:6069 Train:1 Station:KHC2-2T
Actual arr. time:6209 Train:0 Station:KGN2-3T
Actual dep. time:6239 Train:0 Station:KGN2-3T
Actual arr. time:6271 Train:1 Station:KGN2-3T
```

```
Actual dep. time:6301 Train:1 Station:KGN2-3T
Actual arr. time:6736 Train:0 Station:KN1-2T
Actual dep. time:6776 Train:0 Station:KN1-2T
Actual arr. time:6794 Train:1 Station:KN1-2T
Actual dep. time:6834 Train:1 Station:KN1-2T
Actual arr. time:7255 Train:0 Station:KGN1-3T
Actual dep. time:7285 Train:0 Station:KGN1-3T
Actual arr. time:7304 Train:1 Station:KGN1-3T
Actual dep. time:7334 Train:1 Station:KGN1-3T
Actual arr. time:7579 Train:0 Station:KHC1-2T
Actual dep. time:7609 Train:0 Station:KHC1-2T
Actual arr. time:7618 Train:1 Station:KHC1-2T
Actual dep. time:7648 Train:1 Station:KHC1-2T
Actual arr. time:7999 Train:0 Station:ISB1-3T
Actual dep. time:8029 Train:0 Station:ISB1-3T
Actual arr. time:8047 Train:1 Station:LGP1-3T
Actual dep. time:8077 Train:1 Station:LGP1-3T
Actual arr. time:8190 Train:0 Station:UNI1-2T
Actual dep. time:8220 Train:0 Station:UNI1-2T
Actual arr. time:8247 Train:1 Station:LGP2-11T
Actual dep. time:8287 Train:1 Station:LGP2-11T
Actual arr. time:8593 Train:0 Station:KHS1-3T
Actual dep. time:8623 Train:0 Station:KHS1-3T
Actual arr. time:8648 Train:1 Station:LGP2-3T
Actual dep. time:8678 Train:1 Station:LGP2-3T
Actual arr. time:8762 Train:0 Station:KHS1-9T
Actual dep. time:8792 Train:0 Station:KHS1-9T
Actual arr. time:8967 Train:1 Station:KHC2-2T
Actual dep. time:8997 Train:1 Station:KHC2-2T
Actual arr. time:9071 Train:0 Station:ORE1-7T
Actual dep. time:9101 Train:0 Station:ORE1-7T
Actual arr. time:9120 Train:1 Station:KGN2-3T
Actual dep. time:9150 Train:1 Station:KGN2-3T
Actual arr. time:9310 Train:0 Station:VEA2-8T
Actual arr. time:144848 Train:1 Station:KN1-2T
Actual dep. time:144888 Train:1 Station:KN1-2T
Actual arr. time:145100 Train:1 Station:KGN1-3T
Actual dep. time:145130 Train:1 Station:KGN1-3T
Actual arr. time:145298 Train:1 Station:KHC1-2T
Actual dep. time:145328 Train:1 Station:KHC1-2T
Actual arr. time:145552 Train:1 Station:LGP1-3T
Actual dep. time:145582 Train:1 Station:LGP1-3T
Actual arr. time:145764 Train:1 Station:LGP2-11T
Actual dep. time:145804 Train:1 Station:LGP2-11T
Actual arr. time:145985 Train:1 Station:LGP2-3T
Actual dep. time:146015 Train:1 Station:LGP2-3T
Actual arr. time:146220 Train:1 Station:KHC2-2T
Actual dep. time:146250 Train:1 Station:KHC2-2T
Actual arr. time:146406 Train:1 Station:KGN2-3T
Actual dep. time:146436 Train:1 Station:KGN2-3T
Actual arr. time:146637 Train:1 Station:KN1-2T
Actual dep. time:146677 Train:1 Station:KN1-2T
Actual arr. time:146889 Train:1 Station:KGN1-3T
```

```
Actual dep. time:146919 Train:1 Station:KGN1-3T
Actual arr. time:147087 Train:1 Station:KHC1-2T
Actual dep. time:147117 Train:1 Station:KHC1-2T
Actual arr. time:147329 Train:1 Station:ISB1-3T
Actual dep. time:147359 Train:1 Station:ISB1-3T
Actual arr. time:147552 Train:1 Station:UNI1-2T
Actual dep. time:147582 Train:1 Station:UNI1-2T
Actual arr. time:147789 Train:1 Station:KHS1-3T
Actual dep. time:147819 Train:1 Station:KHS1-3T
Actual arr. time:147991 Train:1 Station:KHS1-9T
Actual dep. time:148021 Train:1 Station:KHS1-9T
Actual arr. time:148203 Train:1 Station:ORE1-7T
Actual dep. time:148233 Train:1 Station:ORE1-7T
Actual arr. time:148442 Train:1 Station:VEA2-8T
```

Log from test case 5

```
Actual arr. time:0 Train:0 Station:VEA2-8T
Actual dep. time:30 Train:0 Station:VEA2-8T
Actual arr. time:244 Train:0 Station:ORE2-7T
Actual dep. time:274 Train:0 Station:ORE2-7T
Actual arr. time:456 Train:0 Station:KHS2-10T
Actual dep. time:486 Train:0 Station:KHS2-10T
Actual arr. time:616 Train:0 Station:KHS2-3T
Actual dep. time:646 Train:0 Station:KHS2-3T
Actual arr. time:818 Train:0 Station:UNI2-2T
Actual dep. time:848 Train:0 Station:UNI2-2T
Actual arr. time:991 Train:0 Station:ISB2-3T
Actual dep. time:1021 Train:0 Station:ISB2-3T
Actual arr. time:1224 Train:0 Station:KHC2-2T
Actual dep. time:1254 Train:0 Station:KHC2-2T
Actual arr. time:1410 Train:0 Station:KGN2-3T
Actual dep. time:1440 Train:0 Station:KGN2-3T
Actual arr. time:1641 Train:0 Station:KN1-2T
Actual dep. time:1681 Train:0 Station:KN1-2T
Actual arr. time:1894 Train:0 Station:KGN1-3T
Actual dep. time:1924 Train:0 Station:KGN1-3T
Actual arr. time:2092 Train:0 Station:KHC1-2T
Actual dep. time:2122 Train:0 Station:KHC1-2T
Actual arr. time:2346 Train:0 Station:LGP1-3T
Actual dep. time:2376 Train:0 Station:LGP1-3T
Actual arr. time:2558 Train:0 Station:LGP2-11T
Actual dep. time:2598 Train:0 Station:LGP2-11T
Actual arr. time:2779 Train:0 Station:LGP2-3T
Actual dep. time:2809 Train:0 Station:LGP2-3T
Actual arr. time:5000 Train:1 Station:VEA2-8T
Actual dep. time:5030 Train:1 Station:VEA2-8T
Actual arr. time:5244 Train:1 Station:ORE2-7T
Actual dep. time:5274 Train:1 Station:ORE2-7T
Actual arr. time:5456 Train:1 Station:KHS2-10T
Actual dep. time:5486 Train:1 Station:KHS2-10T
Actual arr. time:5616 Train:1 Station:KHS2-3T
Actual dep. time:5646 Train:1 Station:KHS2-3T
```

```
Actual arr. time:5818 Train:1 Station:UNI2-2T
Actual dep. time:5848 Train:1 Station:UNI2-2T
Actual arr. time:5991 Train:1 Station:ISB2-3T
Actual dep. time:6021 Train:1 Station:ISB2-3T
Actual arr. time:13014 Train:0 Station:KHC2-2T
Actual dep. time:13044 Train:0 Station:KHC2-2T
Actual arr. time:13200 Train:0 Station:KGN2-3T
Actual dep. time:13230 Train:0 Station:KGN2-3T
Actual arr. time:13431 Train:0 Station:KN1-2T
Actual dep. time:13471 Train:0 Station:KN1-2T
Actual arr. time:13683 Train:0 Station:KGN1-3T
Actual dep. time:13713 Train:0 Station:KGN1-3T
Actual arr. time:13881 Train:0 Station:KHC1-2T
Actual dep. time:13911 Train:0 Station:KHC1-2T
Actual arr. time:14123 Train:0 Station:ISB1-3T
Actual dep. time:14153 Train:0 Station:ISB1-3T
Actual arr. time:14345 Train:0 Station:UNI1-2T
Actual dep. time:14375 Train:0 Station:UNI1-2T
Actual arr. time:14583 Train:0 Station:KHS1-3T
Actual dep. time:14613 Train:0 Station:KHS1-3T
Actual arr. time:14785 Train:0 Station:KHS1-9T
Actual dep. time:14815 Train:0 Station:KHS1-9T
Actual arr. time:14997 Train:0 Station:ORE1-7T
Actual dep. time:15027 Train:0 Station:ORE1-7T
Actual arr. time:15236 Train:0 Station:VEA2-8T
```

Appendix B

Source code

Metro.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Metro.java"
//=========================================================================

//=========================================================================
// Imported files
//=========================================================================

import java.applet.*;
import java.awt.*;
import eduni.simanim.*;
import eduni.simjava.*;

//=========================================================================
// The Metro Applet class definition
//=========================================================================

public class Metro extends Anim_applet {

    // Constants
    private int NUM_TRAINS;
    final int L_MARGIN = 50;
    final int T_MARGIN = 50;
    final int V_ALIGN  = 40;
    final int V_OFFSET = 300;

    // Applet input
    private Panel inputs;
    private Label msgLabel;
    private TextField msgField;

    public void anim_init () {

        // Applet input
        inputs = new Panel();
        inputs.setLayout(new GridLayout(0,2));

        msgLabel = new Label("Choose test run (1-5):",Label.RIGHT);
        inputs.add(msgLabel);
        msgField = new TextField("1",3);
        inputs.add(msgField);

        this.add ("North",inputs);

    }

    public void anim_layout () {

        // Read input
        int testrun = Integer.parseInt(msgField.getText());

        // Set interruption class
        int interruption = 0;

        /* TDs configurations of VR */

        // Performance levels VR can use for train at the stations
        int[] allowedPerformanceLevels = {};

        // Max and min dwell VR can use on the stations
        double minDwell = 0;
        double maxDwell = 0;

        /* Parameters given to HASTUS */

        // The trains
        int[] trainIds = {};

        // The DIDs the trains are to follow
        int[] DIDsTrain0 = {};
        int[] DIDsTrain1 = {};
        int[][] DIDsForEachTrain = {{}};

        // The period of time the trains are to pause between DIDs
        double[] timeBetweenDIDsTrain0 = {};
        double[] timeBetweenDIDsTrain1 = {};
        double[][] timeBetweenDIDsForEachTrain = {{}};

        // The performance level the dwell that hastus is to use
        int PL = 0;
        double dwellTime = 0;

        // The times the plans are to start
        double[] startTimes = {};


        switch (testrun) {

        case 1:
            NUM_TRAINS = 1;

            int[] tmp1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
            allowedPerformanceLevels = tmp1;

            minDwell = 20;
```

```
    maxDwell = 40;

    int[] tmp2 = {0};
    trainIds = tmp2;

    int[] tmp3 = {4,2,3,1};
    DIDsTrain0 = tmp3;

    int[][] tmp4 = {DIDsTrain0};
    DIDsForEachTrain = tmp4;

    double[] tmp5 = {40, 40, 40, 40};
    timeBetweenDIDsTrain0 = tmp5;

    double[][] tmp6 = {timeBetweenDIDsTrain0};
    timeBetweenDIDsForEachTrain = tmp6;

    PL = 2;
    dwellTime = 30;

    double[] tmp7 = {0};
    startTimes = tmp7;

    interruption = 0;
    break;

case 2:
    NUM_TRAINS = 1;

    int[] tmp8 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    allowedPerformanceLevels = tmp8;

    minDwell = 20;
    maxDwell = 40;

    int[] tmp9 = {0};
    trainIds = tmp9;

    int[] tmp10 = {5,2,3,1};
    DIDsTrain0 = tmp10;

    int[][] tmp11 = {DIDsTrain0};
    DIDsForEachTrain = tmp11;

    double[] tmp12 = {40, 40, 40, 40};
    timeBetweenDIDsTrain0 = tmp12;

    double[][] tmp13 = {timeBetweenDIDsTrain0};
    timeBetweenDIDsForEachTrain = tmp13;

    PL = 2;
    dwellTime = 30;

    double[] tmp14 = {0};
    startTimes = tmp14;

    interruption = 0;
    break;

case 3:
    NUM_TRAINS = 2;

    int[] tmp15 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    allowedPerformanceLevels = tmp15;

    minDwell = 20;
    maxDwell = 40;

    int[] tmp16 = {0,1};
    trainIds = tmp16;

    int[] tmp17 = {4,2,3,1};
    DIDsTrain0 = tmp17;

    int[] tmp18 = {4,2,3,1};
    DIDsTrain1 = tmp18;

    int[][] tmp19 = {DIDsTrain0,DIDsTrain1};
    DIDsForEachTrain = tmp19;

    double[] tmp20 = {40, 40, 40, 40};
    timeBetweenDIDsTrain0 = tmp20;

    double[] tmp21 = {40, 40, 40, 40};
    timeBetweenDIDsTrain1 = tmp21;

    double[][] tmp22 = {timeBetweenDIDsTrain0,timeBetweenDIDsTrain1};
    timeBetweenDIDsForEachTrain = tmp22;

    PL = 2;
    dwellTime = 30;

    double[] tmp23 = {0,500};
    startTimes = tmp23;

    interruption = 0;
    break;

case 4:
    NUM_TRAINS = 2;

    int[] tmp24 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    allowedPerformanceLevels = tmp24;

    minDwell = 20;
    maxDwell = 40;

    int[] tmp25 = {0,1};
    trainIds = tmp25;

    int[] tmp26 = {4,2,3,1};
    DIDsTrain0 = tmp26;

    int[] tmp27 = {4,2,3,2,3,2,3,1};
    DIDsTrain1 = tmp27;

    int[][] tmp28 = {DIDsTrain0,DIDsTrain1};
    DIDsForEachTrain = tmp28;

    double[] tmp29 = {40, 40, 40, 40};
    timeBetweenDIDsTrain0 = tmp29;

    double[] tmp30 = {40, 40, 40, 40, 40, 40, 40, 40};
    timeBetweenDIDsTrain1 = tmp30;
```

```
    double[][] tmp31 = {timeBetweenDIDsTrain0,timeBetweenDIDsTrain1};
    timeBetweenDIDsForEachTrain = tmp31;

    PL = 2;
    dwellTime = 30;

    double[] tmp32 = {0,500};
    startTimes = tmp32;

    interruption = 1;
    break;

case 5:
    NUM_TRAINS = 2;

    int[] tmp33 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    allowedPerformanceLevels = tmp33;

    minDwell = 20;
    maxDwell = 40;

    int[] tmp34 = {0,1};
    trainIds = tmp34;

    int[] tmp35 = {4,2,3,1};
    DIDsTrain0 = tmp35;

    int[] tmp36 = {4,2,3,1};
    DIDsTrain1 = tmp36;

    int[][] tmp37 = {DIDsTrain0,DIDsTrain1};
    DIDsForEachTrain = tmp37;

    double[] tmp38 = {40, 40, 40, 40};
    timeBetweenDIDsTrain0 = tmp38;

    double[] tmp39 = {40, 40, 40, 40};
    timeBetweenDIDsTrain1 = tmp39;

    double[][] tmp40 = {timeBetweenDIDsTrain0,timeBetweenDIDsTrain1};
    timeBetweenDIDsForEachTrain = tmp40;

    PL = 2;
    dwellTime = 30;

    double[] tmp41 = {0,5000};
    startTimes = tmp41;

    interruption = 2;
    break;


}

/*-------------------------------------------------------------*/
/*                                                             */
/* Initialise and add entities                                 */
/*                                                             */
/*-------------------------------------------------------------*/

//-------------------------------------------------------------
```

```
// Initialise central
//-------------------------------------------------------------


/* Initialisation of central */

InitialiseCentral.init(allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       allowedPerformanceLevels,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       minDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       maxDwell,
                       trainIds,
                       DIDsForEachTrain,
                       timeBetweenDIDsForEachTrain,
                       PL,
                       dwellTime,
                       startTimes);


//-------------------------------------------------------------
// ATS
//-------------------------------------------------------------

ATSWithVerySimpleVR ATS =
    new ATSWithVerySimpleVR ("ATS",
                             NUM_TRAINS,
                             "ATS_"+NUM_TRAINS,
                             L_MARGIN,
                             V_OFFSET);


//-------------------------------------------------------------
// Initialise Wayside
//-------------------------------------------------------------
```

```
Wayside wayside = new Wayside("Wayside",
                              NUM_TRAINS,
                              "Wayside_"+NUM_TRAINS,
                              L_MARGIN+750,
                              V_OFFSET);

InitialiseWayside.init(wayside,
                       L_MARGIN,
                       T_MARGIN,
                       V_ALIGN);

//-------------------------------------------------------------
// Initialise Interruption
//-------------------------------------------------------------

switch (interruption) {
case 1:
    Interruption1 interruption1 =
        new Interruption1("Interruption1", wayside);
    break;
case 2:
    Interruption2 interruption2 =
        new Interruption2("Interruption2", wayside);
    break;
}

//-------------------------------------------------------------
// Trains
//-------------------------------------------------------------

Train t;

for (int i=0 ; i < NUM_TRAINS ; i++) {

    t = new Train ("train"+i,
                   i,
                   "train"+i,
                   L_MARGIN+420,
                   V_OFFSET+10+i*30);

}

/*-------------------------------------------------------------*/
/*                                                             */
/* Link ports                                                  */
/*                                                             */
/*-------------------------------------------------------------*/

//-------------------------------------------------------------
// ATS <--> Wayside
//-------------------------------------------------------------

Sim_system.link_ports ("ATS",
                       "fromATStoWayside",
                       "Wayside",
                       "toWaysideFromATS");

//-------------------------------------------------------------
// Train <--> ATS
// Train <--> Wayside
//-------------------------------------------------------------

        for (int j=0 ; j < NUM_TRAINS ; j++) {

            // Communication with ATS
            Sim_system.link_ports ("train"+j,
                                   "train"+j+"ToATS",
                                   "ATS",
                                   "toATSfromTrain"+j);

            Sim_system.link_ports ("ATS",
                                   "fromATStoTrain"+j,
                                   "train"+j,
                                   "toTrain"+j+"FromATS");

            // Communication with Wayside
            Sim_system.link_ports ("train"+j,
                                   "train"+j+"ToWayside",
                                   "Wayside",
                                   "toWaysideFromTrain"+j);

            Sim_system.link_ports ("Wayside",
                                   "fromWaysideToTrain"+j,
                                   "train"+j,
                                   "toTrain"+j+"FromWayside");

        }

    }

    public void sim_setup () {
        // Set termination condition
        Sim_system.set_termination_condition(Sim_system.EVENTS_COMPLETED,
                                             "ATS", 0, 1, false);
    }


}
```

Metro.html

```
<html>
  <head>
    <title>Animation of the Copenhagen Metro System</title>
  </head>
  <body bgcolor="#FFFFFF">
    <h1>Animation of the Copenhagen Metro System</h1>
    <p>
      <center>
        <applet code="Metro.class"
                archive="simjava.jar"
                width="1000"
                height="600">
        </applet>
      </center>
    </p>
  </body>
</html>
```

B.1 The ATP Wayside entity

Wayside.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Wayside.java"
//=====================================================================

//=====================================================================
// Imported files
//=====================================================================

import eduni.simjava.*;
import eduni.simanim.*;
import java.util.LinkedList;
import java.util.Hashtable;

//=====================================================================
// The Wayside class definition
//=====================================================================

class Wayside extends Sim_entity {

    // Arrays containing track circuit objects. Note that a track
    // circuit object is referenced from two arrays, when situated at
    // a joint stretch.
    private TrackCircuit[] VeaToKn; // array number 0
    private TrackCircuit[] KnToVea; // array number 1
    private TrackCircuit[] LgpToKn; // array number 2
    private TrackCircuit[] KnToLgp; // array number 3

    // Hashtable containing the correspondance between track circuit
    // names and a pair consisting of array number and array index
    private Hashtable map;

    // The upper and lower bifurcation switches. True is normal
    // direction for the bifurcation switch, false is reverse
    // direction
    private boolean upperBifurcation;
    private boolean lowerBifurcation;

    // The queues for the upper and lower bifurcation
    private LinkedList upperBifurcationQueue;
    private LinkedList lowerBifurcationQueue;

    // Flag raised when the bifurcation can be changed directly
    private boolean upperAllowChange;
    private boolean lowerAllowChange;

    // Logging
    private String controlLineLog = "";

    //---------------------------------------------------------------------
    // Constructor
    //---------------------------------------------------------------------
    Wayside(String name, int numTrains, String image, int x, int y) {
        super(name, image, x, y);

        String portName;
        // Add outgoing ports to all trains
        for (int i = 0; i < numTrains; i++) {
            portName = "fromWaysideToTrain"+i;
            add_port(new Sim_port(portName,
                                  "port",
                                  Anim_port.LEFT,
                                  i*30+20));
        }

        // Add incoming port to receive messages from ATS
        add_port(new Sim_port("toWaysideFromATS",
                              "port",
                              Anim_port.LEFT,
                              0));


        // Add incoming ports to receive messages from trains
        for (int i=0 ; i < numTrains ; i++) {

            add_port(new Sim_port("toWaysideFromTrain"+i,
                                  "port",
                                  Anim_port.LEFT,
                                  i*30+20));

        }


        upperBifurcation = true;
        lowerBifurcation = true;
        upperBifurcationQueue = new LinkedList();
        lowerBifurcationQueue = new LinkedList();
        upperAllowChange = true;
        lowerAllowChange = true;
    }


    //---------------------------------------------------------------------
    // initTrackCircuits:
    //
    // Initialises the Wayside object's track circuits.
    // Should be called before the object is used.
    //---------------------------------------------------------------------
    public void initTrackCircuits(TrackCircuit[] KnToVea,
                                  TrackCircuit[] VeaToKn,
                                  TrackCircuit[] KnToLgp,
                                  TrackCircuit[] LgpToKn,
                                  Hashtable map) {

        this.KnToVea = KnToVea;
        this.VeaToKn = VeaToKn;
        this.LgpToKn = LgpToKn;
        this.KnToLgp = KnToLgp;
        this.map = map;

    }


    //---------------------------------------------------------------------
    // setDelay:
    //
    // Set a delay for the next train entering a track circuit
    //---------------------------------------------------------------------
    public void setDelay(String tcName, int delay) {
```

```
    HashItem lookup = (HashItem) map.get(tcName);
    int arrayNum = lookup.getArrayNum();
    int arrayIndex = lookup.getArrayIndex();

    switch (arrayNum) {
    case 0: // VeaToKn
        VeaToKn[arrayIndex].setDelay(delay);
        break;
    case 1: // KnToVea
        KnToVea[arrayIndex].setDelay(delay);
        break;
    case 2: // LgpToKn
        LgpToKn[arrayIndex].setDelay(delay);
        break;
    case 3: // KnToLgp
        KnToLgp[arrayIndex].setDelay(delay);
        break;
    }
}


//---------------------------------------------------------------------
// trackCircuitChange:
//
// Move a train to the next track circuit
//---------------------------------------------------------------------

private void trackCircuitChange(String tcName, int trainId) {
    HashItem lookup = (HashItem) map.get(tcName);
    TrackCircuit tcCurrent = VeaToKn[0]; // This will be overwritten
    TrackCircuit tcNext = VeaToKn[0];    // This will be overwritten
    int arrayNum = lookup.getArrayNum();
    int arrayIndex = lookup.getArrayIndex();

    switch (arrayNum) {

    case 0: // VeaToKn
        tcCurrent = VeaToKn[arrayIndex];
        // If the train is one track circuit from the end
        if (arrayIndex+1 == VeaToKn.length-1) {
            // Try to move the train to the outgoing platform
            tcNext = KnToVea[0];
        }
        // else move to the next track circuit
        else {
            tcNext = VeaToKn[arrayIndex+1];
        }
        break;

    case 1: // KnToVea
        tcCurrent = KnToVea[arrayIndex];
        // If the train is one track circuit from the end
        if (arrayIndex+1 == KnToVea.length-1) {
            // Try to move the train to the outgoing platform
            tcNext = VeaToKn[0];
        }
        // else move to the next track circuit
        else {
            if (lowerBifurcation && tcName == "KHC1-5T") {
                tcNext = KnToLgp[arrayIndex+1];
            }
            else {
                tcNext = KnToVea[arrayIndex+1];
            }
        }
        break;

    case 2: // LgpToKn
        tcCurrent = LgpToKn[arrayIndex];
        // If the train is one track circuit from the end
        if (arrayIndex+1 == LgpToKn.length-1) {
            // Try to move the train to the outgoing platform
            tcNext = KnToLgp[0];
        }
        // else move to the next track circuit
        else {
            tcNext = LgpToKn[arrayIndex+1];
        }
        break;

    case 3: // KnToLgp
        tcCurrent = KnToLgp[arrayIndex];
        // If the train is one track circuit from the end
        if (arrayIndex+1 == KnToLgp.length-1) {
            // Try to move the train to the outgoing platform
            tcNext = LgpToKn[0];
        }
        // else move to the next track circuit
        else {
            if (!lowerBifurcation && tcName == "KHC1-5T") {
                tcNext = KnToVea[arrayIndex+1];
            }
            else {
                tcNext = KnToLgp[arrayIndex+1];
            }
        }
        break;
    }

    // Checking whether the next track circuit is occupied
    if (!tcNext.getOccupied() && !tcNext.getPseudoOccupied()) {
        // Updating occupancy in track circuits

        tcNext.setOccupiedBy(trainId);
        tcNext.setOccupied(true);
        tcCurrent.setOccupiedBy(-1);
        tcCurrent.setOccupied(false);

        // Move a train from the track circuit just before
        // incoming platform to outgoing platform whenever a train
        // leaves the outgoing platform and a train is situated at
        // the track circuit just before the incoming platform
        if (arrayIndex == 0) {
            switch (arrayNum) {

            case 0: // VeaToKn
                if (KnToVea[KnToVea.length-2].getOccupied()) {
                    VeaToKn[0].setOccupiedBy(KnToVea[KnToVea.length-2]
                                             .getOccupiedBy());
                    VeaToKn[0].setOccupied(true);
                    KnToVea[KnToVea.length-2].setOccupiedBy(-1);
                    KnToVea[KnToVea.length-2].setOccupied(false);
                }
                break;
```

```
            case 1: // KnToVea
                if (VeaToKn[VeaToKn.length-2].getOccupied()) {
                    KnToVea[0].setOccupiedBy(VeaToKn[VeaToKn.length-2]
                                             .getOccupiedBy());
                    KnToVea[0].setOccupied(true);
                    VeaToKn[VeaToKn.length-2].setOccupiedBy(-1);
                    VeaToKn[VeaToKn.length-2].setOccupied(false);
                }
                break;

            case 2: // LgpToKn
                if (KnToLgp[KnToLgp.length-2].getOccupied()) {
                    LgpToKn[0].setOccupiedBy(KnToLgp[KnToLgp.length-2]
                                             .getOccupiedBy());
                    LgpToKn[0].setOccupied(true);
                    KnToLgp[KnToLgp.length-2].setOccupiedBy(-1);
                    KnToLgp[KnToLgp.length-2].setOccupied(false);
                }
                break;

            case 3: // KnToLgp
                if (LgpToKn[LgpToKn.length-2].getOccupied()) {
                    KnToLgp[0].setOccupiedBy(LgpToKn[LgpToKn.length-2]
                                             .getOccupiedBy());
                    KnToLgp[0].setOccupied(true);
                    LgpToKn[LgpToKn.length-2].setOccupiedBy(-1);
                    LgpToKn[LgpToKn.length-2].setOccupied(false);
                }
                break;
            }
        }


        // If leaving one of the two track circuits preceding
        // the upper switch
        if (tcName == "KHC2-6T" || tcName == "3T") {
            upperAllowChange = true;
            changeUpperBifurcation();
        }

        // If leaving the track circuit preceding the lower switch
        if (tcName == "KHC1-5T") {
            lowerAllowChange = true;
            changeLowerBifurcation();
        }

    }
    else {
        // Do nothing. The train will wait for a new track circuit
        // id or control line info with a greater distance-to-go.
    }



}


//---------------------------------------------------------------------
// changeLowerBifurcation:
//
// Change lower bifurcation setting
//---------------------------------------------------------------------
private void changeLowerBifurcation() {
    if (lowerBifurcationQueue.size() > 0) {
        if (lowerBifurcationQueue.removeFirst() == "normal") {
            lowerBifurcation = true;
        }
        else {
            lowerBifurcation = false;
        }
        lowerAllowChange = false;
    }
}


//---------------------------------------------------------------------
// changeUpperBifurcation:
//
// Change upper bifurcation setting
//---------------------------------------------------------------------
private void changeUpperBifurcation() {
    HashItem fromVea = (HashItem) map.get("3T");
    int fromVeaIndex = fromVea.getArrayIndex();
    HashItem fromLgp = (HashItem) map.get("KHC2-6T");
    int fromLgpIndex = fromLgp.getArrayIndex();

    if (upperBifurcationQueue.size() > 0) {
        if (upperBifurcationQueue.removeFirst() == "normal") {
            upperBifurcation = true;
        }
        else {
            upperBifurcation = false;
        }

        if (upperBifurcation) {
            // Set pseudo occupancy blocking trains from Vea
            VeaToKn[fromVeaIndex].setPseudoOccupied(true);
            // and remove pseudo occupancy blocking trains from Lgp
            LgpToKn[fromLgpIndex].setPseudoOccupied(false);
        }
        else {
            // Remove pseudo occupancy blocking trains from Vea
            VeaToKn[fromVeaIndex].setPseudoOccupied(false);
            // and set pseudo occupancy blocking trains from Lgp
            LgpToKn[fromLgpIndex].setPseudoOccupied(true);
        }

        upperAllowChange = false;
    }

}


//---------------------------------------------------------------------
// handleBifurcationRequest:
//
// Handle requests for bifurcation switch setting
//---------------------------------------------------------------------

private void handleBifurcationRequest(String direction) {
    HashItem fromVea = (HashItem) map.get("3T");
    int fromVeaIndex = fromVea.getArrayIndex();
    HashItem fromLgp = (HashItem) map.get("KHC2-6T");
```

```
int fromLgpIndex = fromLgp.getArrayIndex();
HashItem fromKn = (HashItem) map.get("KHC1-5T");
int fromKnIndex = fromKn.getArrayIndex();

if (direction == "LGPToKN") {
    if (upperBifurcationQueue.size() > 0
        || VeaToKn[fromVeaIndex].getOccupied()
        || LgpToKn[fromLgpIndex].getOccupied()
        || !upperAllowChange) {
        // Put to upper bifurcation queue
        upperBifurcationQueue.addLast("normal");
    }
    else {
        // Change bifurcation directly
        VeaToKn[fromVeaIndex].setPseudoOccupied(true);
        LgpToKn[fromLgpIndex].setPseudoOccupied(false);
        upperBifurcation = true;
        upperAllowChange = false;
    }
}

if (direction == "VEAToKN") {
    if (upperBifurcationQueue.size() > 0
        || VeaToKn[fromVeaIndex].getOccupied()
        || LgpToKn[fromLgpIndex].getOccupied()
        || !upperAllowChange) {
        // Put to upper bifurcation queue
        upperBifurcationQueue.addLast("reverse");
    }
    else {
        // Change bifurcation directly
        VeaToKn[fromVeaIndex].setPseudoOccupied(false);
        LgpToKn[fromLgpIndex].setPseudoOccupied(true);
        upperBifurcation = false;
        upperAllowChange = false;
    }
}


if (direction == "KNToVEA") {
    if (lowerBifurcationQueue.size() > 0
        || KnToLgp[fromKnIndex].getOccupied()
        || !lowerAllowChange) {
        // Put to upper bifurcation queue
        lowerBifurcationQueue.addLast("reverse");
    }
    else {
        // Change bifurcation directly
        lowerBifurcation = false;
        lowerAllowChange = false;
    }
}

if (direction == "KNToLGP") {
    if (lowerBifurcationQueue.size() > 0
        || KnToLgp[fromKnIndex].getOccupied()
        || !lowerAllowChange) {
        // Put to upper bifurcation queue
        lowerBifurcationQueue.addLast("normal");
    }
    else {
        // Change bifurcation directly
        lowerBifurcation = true;
        lowerAllowChange = false;
    }
}

}


//---------------------------------------------------------------------
// broadcastControlLineInfo:
//
// Send control line info to every train
//---------------------------------------------------------------------
private void broadcastControlLineInfo() {
    int obstruction;
    ControlLineInfo cli;
    int train;
    WaysideToTrain msg;
    String outPort;
    HashItem beforeSwitch = (HashItem) map.get("KHC1-5T");
    int beforeSwitchIndex = beforeSwitch.getArrayIndex();


    // VeaToKn
    for (int i = 0; i < VeaToKn.length; i++) {
        obstruction = -1;
        if (VeaToKn[i].getOccupied()) {
            // Check upper bifurcation switch direction
            if (upperBifurcation) {
                // Normal switch position
                for (int j = 1;
                     j < VeaToKn[i].getControlLineNormalLength()
                         && i+j < VeaToKn.length;
                     j++) {
                    if (VeaToKn[i+j].getOccupied()
                        || VeaToKn[i+j].getPseudoOccupied()) {
                        obstruction = j;
                        break; // Nearest for-loop
                    }
                }
                // Lookup correct control line info for the train
                if (obstruction > 0) {
                    cli = VeaToKn[i]
                        .getControlLineInfoNormal(obstruction-1);
                }
                else {
                    // If no obstructions, send last control line
                    // info
                    cli = VeaToKn[i]
                        .getControlLineInfoNormal(VeaToKn[i]
                                .getControlLineNormalLength()-1);
                }
            }
            else {
                // Reverse switch position
                for (int j = 1;
                     j < VeaToKn[i].getControlLineReverseLength()
                         && i+j < VeaToKn.length;
                     j++) {
                    if (VeaToKn[i+j].getOccupied()
                        || VeaToKn[i+j].getPseudoOccupied()) {
                        obstruction = j;
```

```
                    break; // Nearest for-loop
                }
            }
            // Lookup correct control line info for the train
            if (obstruction > 0) {
                cli = VeaToKn[i]
                    .getControlLineInfoReverse(obstruction-1);
            }
            else {
                // If no obstructions, send last control line
                // info
                cli = VeaToKn[i]
                    .getControlLineInfoReverse(VeaToKn[i]
                            .getControlLineReverseLength()-1);
            }

        }
        // Send control line info to the train
        train = VeaToKn[i].getOccupiedBy();
        msg = new WaysideToTrain(VeaToKn[i].getTcName(),
                                 VeaToKn[i].getType(),
                                 VeaToKn[i].getLength(),
                                 cli.getLineSpeed(),
                                 cli.getTargetSpeed(),
                                 cli.getDistanceToGo(),
                                 VeaToKn[i].getDelay());
        // Reset delay when used
        if (VeaToKn[i].getDelay() > 0) {
            VeaToKn[i].setDelay(0);
        }
        outPort = "fromWaysideToTrain" + train;
        sim_schedule(outPort, 0.0, 1, msg);
        controlLineLog =
            "("+cli.getLineSpeed()+","+
            cli.getTargetSpeed()+","+cli.getDistanceToGo()+")";
        sim_trace(1, "S " + outPort + " "+controlLineLog);
    }
}
```

```
// KnToVea
for (int i = 0; i < KnToVea.length; i++) {
    obstruction = -1;
    if (KnToVea[i].getOccupied()) {
        // Check lower bifurcation switch direction
        if (lowerBifurcation) {
            // When in normal switch position, track circuit
            // occupancy shall be checked in Lgp-direction
            // when the train is situated before the switch
            if (i <= beforeSwitchIndex) {
                for (int j = 1;
                     j < KnToLgp[i].getControlLineNormalLength()
                         && i+j < KnToLgp.length;
                     j++) {
                    if (KnToLgp[i+j].getOccupied()
                        || KnToLgp[i+j].getPseudoOccupied()) {
                        obstruction = j;
                        break; // Nearest for-loop
                    }
                }
                // Lookup correct control line info for the train
                if (obstruction > 0) {
                    cli = KnToLgp[i]
                        .getControlLineInfoNormal(obstruction-1);
                }
                else {
                    // If no obstructions, send last control
                    // line info
                    cli = KnToLgp[i]
                        .getControlLineInfoNormal(KnToLgp[i]
                                .getControlLineNormalLength()-1);
                }
            }
            else {
                // else the track occupancy check shall be in
                // Vea-direction
                for (int j = 1;
                     j < KnToVea[i].getControlLineNormalLength()
                         && i+j < KnToVea.length;
                     j++) {
                    if (KnToVea[i+j].getOccupied()
                        || KnToVea[i+j].getPseudoOccupied()) {
                        obstruction = j;
                        break; // Nearest for-loop
                    }
                }
                // Lookup correct control line info for the train
                if (obstruction > 0) {
                    cli = KnToVea[i]
                        .getControlLineInfoNormal(obstruction-1);
                }
                else {
                    // If no obstructions, send last control
                    // line info
                    cli = KnToVea[i]
                        .getControlLineInfoNormal(KnToVea[i]
                                .getControlLineNormalLength()-1);
                }
            }

        }
        else {
            // Reverse switch position
            for (int j = 1;
                 j < KnToVea[i].getControlLineReverseLength()
                     && i+j < KnToVea.length;
                 j++) {
                if (KnToVea[i+j].getOccupied()
                    || KnToVea[i+j].getPseudoOccupied()) {
                    obstruction = j;
                    break; // Nearest for-loop
                }
            }
            // Lookup correct control line info for the train
            if (obstruction > 0) {
                cli = KnToVea[i]
                    .getControlLineInfoReverse(obstruction-1);
            }
```

```
            else {
                // If no obstructions, send last control line
                // info
                cli = KnToVea[i]
                    .getControlLineInfoReverse(KnToVea[i]
                              .getControlLineReverseLength()-1);
            }

        }
        // Send control line info to the train
        train = KnToVea[i].getOccupiedBy();
        msg = new WaysideToTrain(KnToVea[i].getTcName(),
                                 KnToVea[i].getType(),
                                 KnToVea[i].getLength(),
                                 cli.getLineSpeed(),
                                 cli.getTargetSpeed(),
                                 cli.getDistanceToGo(),
                                 KnToVea[i].getDelay());
        // Reset delay when used
        if (KnToVea[i].getDelay() > 0) {
            KnToVea[i].setDelay(0);
        }
        outPort = "fromWaysideToTrain" + train;
        sim_schedule(outPort, 0.0, 1, msg);
        controlLineLog =
            "("+cli.getLineSpeed()+","+
            cli.getTargetSpeed()+","+cli.getDistanceToGo()+")";
        sim_trace(1, "S " + outPort + " "+controlLineLog);
    }
}



// Control line info triples are already sent to track
// circuits on the joint stretch
HashItem khc26t = (HashItem) map.get("KHC2-6T");
int khc26tIndex = khc26t.getArrayIndex();

// LgpToKn
for (int i = 0; i <= khc26tIndex; i++) {
    obstruction = -1;
    if (LgpToKn[i].getOccupied()) {
        // Check upper bifurcation switch direction
        if (upperBifurcation) {
            // Normal switch position
            for (int j = 1;
                 j < LgpToKn[i].getControlLineNormalLength()
                     && i+j < LgpToKn.length;
                 j++) {
                if (LgpToKn[i+j].getOccupied()
                    || LgpToKn[i+j].getPseudoOccupied()) {
                    obstruction = j;
                    break; // Nearest for-loop
                }
            }
            // Lookup correct control line info for the train
            if (obstruction > 0) {
                cli = LgpToKn[i]
                    .getControlLineInfoNormal(obstruction-1);
            }
            else {
                // If no obstructions, send last control line
                // info
                cli = LgpToKn[i]
                    .getControlLineInfoNormal(LgpToKn[i]
                              .getControlLineNormalLength()-1);
            }
        }
        else {
            // Reverse switch position
            for (int j = 1;
                 j < LgpToKn[i].getControlLineReverseLength()
                     && i+j < LgpToKn.length;
                 j++) {
                if (LgpToKn[i+j].getOccupied()
                    || LgpToKn[i+j].getPseudoOccupied()) {
                    obstruction = j;
                    break; // Nearest for-loop
                }
            }
            // Lookup correct control line info for the train
            if (obstruction > 0) {
                cli = LgpToKn[i]
                    .getControlLineInfoReverse(obstruction-1);
            }
            else {
                // If no obstructions, send last control line
                // info
                cli = LgpToKn[i]
                    .getControlLineInfoReverse(LgpToKn[i]
                              .getControlLineReverseLength()-1);
            }

        }
        // Send control line info to the train
        train = LgpToKn[i].getOccupiedBy();
        msg = new WaysideToTrain(LgpToKn[i].getTcName(),
                                 LgpToKn[i].getType(),
                                 LgpToKn[i].getLength(),
                                 cli.getLineSpeed(),
                                 cli.getTargetSpeed(),
                                 cli.getDistanceToGo(),
                                 LgpToKn[i].getDelay());
        // Reset delay when used
        if (LgpToKn[i].getDelay() > 0) {
            LgpToKn[i].setDelay(0);
        }
        outPort = "fromWaysideToTrain" + train;
        sim_schedule(outPort, 0.0, 1, msg);
        controlLineLog =
            "("+cli.getLineSpeed()+","+
            cli.getTargetSpeed()+","+cli.getDistanceToGo()+")";
        sim_trace(1, "S " + outPort + " "+controlLineLog);
    }
}



// Control line info triples are already sent to track
// circuits on the joint stretch
HashItem khc16t = (HashItem) map.get("KHC1-6T");
int khc16tIndex = khc16t.getArrayIndex();
```

```
// KnToLgp
for (int i = khc16tIndex; i < KnToLgp.length; i++) {
    obstruction = -1;
    if (KnToLgp[i].getOccupied()) {
        // Check lower bifurcation switch direction
        if (lowerBifurcation) {
            // Normal switch position
            for (int j = 1;
                 j < KnToLgp[i].getControlLineNormalLength()
                     && i+j < KnToLgp.length;
                 j++) {
                if (KnToLgp[i+j].getOccupied()
                    || KnToLgp[i+j].getPseudoOccupied()) {
                    obstruction = j;
                    break; // Nearest for-loop
                }
            }
            // Lookup correct control line info for the train
            if (obstruction > 0) {
                cli = KnToLgp[i]
                    .getControlLineInfoNormal(obstruction-1);
            }
            else {
                // If no obstructions, send last control line
                // info
                cli = KnToLgp[i]
                    .getControlLineInfoNormal(KnToLgp[i]
                              .getControlLineNormalLength()-1);
            }
        }
        else {
            // When in reverse switch position, track circuit
            // occupancy shall be checked in Vea-direction
            // when the train is situated before the switch
            if (i <= beforeSwitchIndex) {
                for (int j = 1;
                     j < KnToVea[i].getControlLineReverseLength()
                         && i+j < KnToVea.length;
                     j++) {
                    if (KnToVea[i+j].getOccupied()
                        || KnToVea[i+j].getPseudoOccupied()) {
                        obstruction = j;
                        break; // Nearest for-loop
                    }
                }
                // Lookup correct control line info for the train
                if (obstruction > 0) {
                    cli = KnToVea[i]
                        .getControlLineInfoReverse(obstruction-1);
                }
                else {
                    // If no obstructions, send last control
                    // line info
                    cli = KnToVea[i]
                        .getControlLineInfoReverse(KnToVea[i]
                                  .getControlLineReverseLength()-1);
                }
            }
            else {
                // else the track occupancy check shall be in
                // Lgp-direction
                for (int j = 1;
                     j < KnToLgp[i].getControlLineReverseLength()
                         && i+j < KnToLgp.length;
                     j++) {
                    if (KnToLgp[i+j].getOccupied()
                        || KnToLgp[i+j].getPseudoOccupied()) {
                        obstruction = j;
                        break; // Nearest for-loop
                    }
                }
                // Lookup correct control line info for the train
                if (obstruction > 0) {
                    cli = KnToLgp[i]
                        .getControlLineInfoReverse(obstruction-1);
                }
                else {
                    // If no obstructions, send last control
                    // line info
                    cli = KnToLgp[i]
                        .getControlLineInfoReverse(KnToLgp[i]
                                  .getControlLineReverseLength()-1);
                }
            }

        }
        // Send control line info to the train
        train = KnToLgp[i].getOccupiedBy();
        msg = new WaysideToTrain(KnToLgp[i].getTcName(),
                                 KnToLgp[i].getType(),
                                 KnToLgp[i].getLength(),
                                 cli.getLineSpeed(),
                                 cli.getTargetSpeed(),
                                 cli.getDistanceToGo(),
                                 KnToLgp[i].getDelay());
        // Reset delay when used
        if (KnToLgp[i].getDelay() > 0) {
            KnToLgp[i].setDelay(0);
        }
        outPort = "fromWaysideToTrain" + train;
        sim_schedule(outPort, 0.0, 1, msg);
        controlLineLog =
            "("+cli.getLineSpeed()+","+
            cli.getTargetSpeed()+","+cli.getDistanceToGo()+")";
        sim_trace(1, "S " + outPort + " "+controlLineLog);
    }
}


//---------------------------------------------------------------------
// sendToVea:
//
// Sends a train to the starting position at Vea
//---------------------------------------------------------------------
private boolean sendToVea(int trainId) {
    boolean succes;
    if (!VeaToKn[0].getOccupied()) {
        VeaToKn[0].setOccupiedBy(trainId);
        VeaToKn[0].setOccupied(true);
        succes = true;
    }
```

```
    else {
        WaysideToTrain msg = new WaysideToTrain("CMC",1,1,1,1,1,0);
        String outPort = "fromWaysideToTrain" + trainId;
        sim_schedule(outPort, 0.0, 1, msg);
        sim_trace(1, "S " + outPort);
        succes = false;
    }
    return succes;
}


//---------------------------------------------------------------------
// removeTrain:
//
// Removes a train from service
//---------------------------------------------------------------------
private void removeTrain(String tcName) {
    HashItem lookup = (HashItem) map.get(tcName);
    int arrayNum = lookup.getArrayNum();
    int arrayIndex = lookup.getArrayIndex();
    switch (arrayNum) {
    case 0: // VeaToKn
        VeaToKn[arrayIndex].setOccupied(false);
        VeaToKn[arrayIndex].setOccupiedBy(-1);
    case 1: // KnToVea
        KnToVea[arrayIndex].setOccupied(false);
        KnToVea[arrayIndex].setOccupiedBy(-1);
    case 2: // LgpToKn
        LgpToKn[arrayIndex].setOccupied(false);
        LgpToKn[arrayIndex].setOccupiedBy(-1);
    case 3: // KnToLgp
        KnToLgp[arrayIndex].setOccupied(false);
        KnToLgp[arrayIndex].setOccupiedBy(-1);
    }
}



//---------------------------------------------------------------------
// Entity body
//---------------------------------------------------------------------
public void body() {

    boolean end = false;

    while (Sim_system.running() && !end) {
        // Get the next event
        Sim_event e = new Sim_event();
        sim_get_next(e);


        // Taken action depends on tag
        switch (e.get_tag()) {

        case 2: // Message from a train
            TrainToWayside twMsg = (TrainToWayside) e.get_data();
            String tcName = twMsg.getPreviousTc();
            int trainId = twMsg.getTrainId();
            if (tcName == "CMC") {
                if (sendToVea(trainId)) {
                    broadcastControlLineInfo();
                }
                else {
                    trackCircuitChange(tcName,trainId);
                    broadcastControlLineInfo();
                }
                break;

            case 3: // Message from ATS
                ATSToWayside awMsg = (ATSToWayside) e.get_data();
                handleBifurcationRequest(awMsg.getBifurcationReservation());
                broadcastControlLineInfo();
                break;

            case -1: // Remove-me message from a train
                TrainToWayside rmMsg = (TrainToWayside) e.get_data();
                String tcNameRm = rmMsg.getPreviousTc();
                removeTrain(tcNameRm);
                broadcastControlLineInfo();
                break;

            case 0:
                end = true;
                break;
            }

            // The event has completed
            sim_completed(e);
        }
    }
}
```

InitialiseWayside.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "InitialiseWayside.java"
//=====================================================================

//=====================================================================
// Imported files
//=====================================================================

import java.util.Hashtable;

//=====================================================================
// The InitialiseWayside class definition
//=====================================================================

class InitialiseWayside {

    static private int BIFU_OFFSET = 32;

    //-----------------------------------------------------------------
    // init:
    //
    // Method for initialisation of Wayside
    //-----------------------------------------------------------------
    public static void init(Wayside wayside,
```

```
                    int L_MARGIN,
                    int T_MARGIN,
                    int V_ALIGN) {

// The length of each autogenerated control line
int controlLineLength = 5;

int arrayNum;
int arrayIndex;
String name;
int length;
int type;
int index;

// The length of the track circuit arrays
int numTcVeaToKn = 75;
int numTcKnToVea = 73;
int numTcLgpToKn = 38;
int numTcKnToLgp = 38;

// Creation of track circuit arrays. See also comment in
// Wayside.java
TrackCircuit[] VeaToKn = new TrackCircuit[numTcVeaToKn];
TrackCircuit[] KnToVea = new TrackCircuit[numTcKnToVea];
TrackCircuit[] LgpToKn = new TrackCircuit[numTcLgpToKn];
TrackCircuit[] KnToLgp = new TrackCircuit[numTcKnToLgp];

// The length of the hash table
int numUniqueTc = 182;

// Creation of the hash table. See also comment in
// Wayside.java
Hashtable map = new Hashtable(numUniqueTc);


//-----------------------------------------------------------------
// Track circuit initialisation
//-----------------------------------------------------------------

// KN --> VEA (KnToVea)
arrayNum = 1;

// KN
index = 0;
name = "KN1-2T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_KN",
                                  L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// KN --> KGN
index++;
name = "KN1-3T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (1 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-4T";
length = 103;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (2  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-5T";
length = 148;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,"track_circuit",
                                  (3  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-6T";
length = 168;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (4  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-7T";
length = 47;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (5  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-8T";
length = 46;
```

```
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (6  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-9T";
length = 183;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (7  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN1-10T";
length = 200;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (8  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-1T";
length = 95;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (9  * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-2T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (10 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));
```



```
// KGN
index++;
name = "KGN1-3T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_KGN",
                                  (11 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// KGN --> KHC
index++;
name = "KGN1-4T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (12 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-5T";
length = 103;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (13 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-6T";
length = 124;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (14 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-7T";
length = 260;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
```

```
                                    "track_circuit",
                                    (15 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-8T";
length = 164;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (16 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN1-9T";
length = 103;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (17 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-1T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (18 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// KHC
index++;
name = "KHC1-2T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "station_KHC",
                                    (19 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// KHC --> Bifurcation
index++;
name = "KHC1-3T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (20 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-4T";
length = 97;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (21 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-5T";
length = 144;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (22 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// The KnToVea array and the KnToLgp array contains the same
// elements until the bifurcation
int uniqueKnToLgpStart = index + 1;
for (int i = 0; i < uniqueKnToLgpStart; i++) {
    KnToLgp[i] = KnToVea[i];
}


// KnToVea continued

// Bifurcation --> ISB
index++;
name = "1T";
length = 107;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "bifu_start",
                                    (23 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+BIFU_OFFSET+(0 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
```

```
name = "KHC1-10T";
length = 186;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "bifu",
                                  (24 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+BIFU_OFFSET+(1 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-11T";
length = 253;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "bifu",
                                  (25 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+BIFU_OFFSET+(2 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-12T";
length = 200;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "bifu_end",
                                  (26 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+BIFU_OFFSET+(3 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-1T";
length = 105;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (27 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-2T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (28 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// ISB
index++;
name = "ISB1-3T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_ISB",
                                  (29 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// ISB --> UNI
index++;
name = "ISB1-4T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (30 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-5T";
length = 66;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (31 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-6T";
length = 56;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (32 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-7T";
length = 291;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
```

```
                                type,
                                "track_circuit",
                                (33 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-8T";
length = 290;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (34 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB1-9T";
length = 111;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (35 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-1T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (36 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// UNI
index++;
name = "UNI1-2T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "station_UNI",
                                (37 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// UNI --> KHS
index++;
name = "UNI1-3T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (38 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-4T";
length = 180;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (39 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-5T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (40 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-6T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (41 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-7T";
length = 247;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit",
                                (42 * 12) + L_MARGIN,
                                T_MARGIN+V_ALIGN+100);
```

```
map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-8T";
length = 236;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (43 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI1-9T";
length = 200;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (44 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-1T";
length = 102;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (45 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-2T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (46 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// KHS
index++;
name = "KHS1-3T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_KHS",
                                  (47 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// KHS --> BC
index++;
name = "KHS1-4T";
length = 135;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (48 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-5T";
length = 128;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (49 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-6T";
length = 289;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (50 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-7T";
length = 177;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (51 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS1-8T";
length = 50;
```

```
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_dobbelt",
                                  (52 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// BC
index++;
name = "KHS1-9T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_BC",
                                  (54 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// BC --> ORE
index++;
name = "KHS1-10T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (55 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-1T";
length = 103;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (56 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-2T";
length = 200;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (57 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-3T";
length = 296;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (58 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-4T";
length = 200;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (59 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-5T";
length = 86;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (60 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-6T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (61 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// ORE
index++;
name = "ORE1-7T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
```

```
                                    "station_ORE",
                                    (62 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

// ORE --> VEA
index++;
name = "ORE1-8T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (63 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-9T";
length = 103;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (64 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-10T";
length = 233;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (65 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE1-11T";
length = 104;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (66 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA1-1T";
length = 50;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (67 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA1-2T";
length = 38;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (68 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA1-3T";
length = 48;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (69 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA1-4T";
length = 203;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (70 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA1-5T";
length = 196;
type = 2;
KnToVea[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit",
                                    (71 * 12) + L_MARGIN,
                                    T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));
```

```
index++;
name = "VEA1-6T";
length = 48;
type = 1;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_dobbelt",
                                  (72 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));


// VEA
index++;
name = "VEA1-7T";
length = 38;
type = 3;
KnToVea[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_VEA",
                                  (74 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN+100);

map.put(name, new HashItem(arrayNum,index));


// KnToLgp continued

arrayNum = 3;

// Bifurcation --> AMB
index = uniqueKnToLgpStart;
name = "KHC1-6T";
length = 107;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (23 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-7T";
length = 165;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (24 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-8T";
length = 288;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (25 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC1-9T";
length = 250;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (26 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-1T";
length = 104;
type = 2;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (27 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-2T";
length = 50;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (28 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// AMB
index++;
name = "LGP1-3T";
length = 38;
type = 3;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_AMB",
                                  (29 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);
```

```
map.put(name, new HashItem(arrayNum,index));

// AMP --> LGP
index++;
name = "LGP1-4T";
length = 50;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (30 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-5T";
length = 103;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (31 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-6T";
length = 200;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (32 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-7T";
length = 272;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (33 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-8T";
length = 200;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (34 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-9T";
length = 104;
type = 2;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (35 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP1-10T";
length = 50;
type = 1;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit",
                                  (36 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));

// LGP
index++;
name = "LGP1-11T";
length = 38;
type = 3;
KnToLgp[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_LGP",
                                  (37 * 12) + L_MARGIN,
                                  T_MARGIN+V_ALIGN);

map.put(name, new HashItem(arrayNum,index));


// VeaToKn

arrayNum = 0;

// VEA
index = 0;
name = "VEA2-8T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (74 * 12) + L_MARGIN,
                                  T_MARGIN+100);
```

```
map.put(name, new HashItem(arrayNum,index));

// VEA --> ORE
index++;
name = "VEA2-7T";
length = 48;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (73 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-6T";
length = 83;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (72 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-5T";
length = 224;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (71 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-4T";
length = 89;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (70 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-3T";
length = 51;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (69 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-2T";
length = 38;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (68 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "VEA2-1T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (67 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-11T";
length = 104;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (66 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-10T";
length = 233;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (65 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-9T";
length = 103;
```

```
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (64 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-8T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (63 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// ORE
index++;
name = "ORE2-7T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (62 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// ORE --> BC
index++;
name = "ORE2-6T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (61 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-5T";
length = 86;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (60 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-4T";
length = 200;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (59 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-3T";
length = 296;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (58 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-2T";
length = 200;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (57 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ORE2-1T";
length = 103;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (56 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-11T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
```

```
                                (55 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// BC
index++;
name = "KHS2-10T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "station_west",
                                (54 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// BC --> KHS
index++;
name = "KHS2-9T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (53 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-8T";
length = 177;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (52 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-7T";
length = 289;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (51 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-6T";
length = 61;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (50 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-5T";
length = 132;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (49 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-4T";
length = 70;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (48 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// KHS
index++;
name = "KHS2-3T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "station_west",
                                (47 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// KHS --> UNI
index++;
name = "KHS2-2T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (46 * 12) + L_MARGIN,
                                T_MARGIN+100);
```

```
map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHS2-1T";
length = 102;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (45 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-9T";
length = 200;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (44 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-8T";
length = 236;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (43 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-7T";
length = 247;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (42 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-6T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (41 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-5T";
length = 49;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (40 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-4T";
length = 185;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (39 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "UNI2-3T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (38 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// UNI
index++;
name = "UNI2-2T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (37 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// UNI --> ISB
index++;
name = "UNI2-1T";
length = 50;
```

```
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (36 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-9T";
length = 109;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (35 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-8T";
length = 290;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (34 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-7T";
length = 291;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (33 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-6T";
length = 56;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (32 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-5T";
length = 66;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (31 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-4T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (30 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// ISB
index++;
name = "ISB2-3T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (29 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

// ISB --> Bifurcation
index++;
name = "ISB2-2T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (28 * 12) + L_MARGIN,
                                  T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "ISB2-1T";
length = 104;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
```

```
                                (27 * 12) + L_MARGIN,
                                T_MARGIN+100);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-12T";
length = 200;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "bifu_end",
                                (26 * 12) + L_MARGIN,
                                T_MARGIN+BIFU_OFFSET+(3 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-11T";
length = 267;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "bifu",
                                (25 * 12) + L_MARGIN,
                                T_MARGIN+BIFU_OFFSET+(2 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-10T";
length = 186;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "bifu",
                                (24 * 12) + L_MARGIN,
                                T_MARGIN+BIFU_OFFSET+(1 * 25)+2);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "3T";
length = 107;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "bifu_start",
                                (23 * 12) + L_MARGIN,
                                T_MARGIN+BIFU_OFFSET+(0 * 25)+2);

map.put(name, new HashItem(arrayNum,index));


// The VeaToKn array and the LgpToKn array contains the same
// elements after the bifurcation
int equalElementsStart = index + 1;
```

```
// Bifurcation --> KHC
index++;
name = "KHC2-5T";
length = 152;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (22 * 12) + L_MARGIN,
                                T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-4T";
length = 97;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (21 * 12) + L_MARGIN,
                                T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-3T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "track_circuit_west",
                                (20 * 12) + L_MARGIN,
                                T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// KHC
index++;
name = "KHC2-2T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
                                "station_west",
                                (19 * 12) + L_MARGIN,
                                T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// KHC --> KGN
index++;
name = "KHC2-1T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                length,
                                type,
```

```
                                    "track_circuit_west",
                                    (18 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-9T";
length = 104;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (17 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-8T";
length = 163;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (16 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-7T";
length = 270;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (15 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-6T";
length = 108;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (14 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-5T";
length = 103;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (13 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-4T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (12 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// KGN
index++;
name = "KGN2-3T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "station_west",
                                    (11 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// KGN --> KN
index++;
name = "KGN2-2T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (10 * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KGN2-1T";
length = 95;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                    length,
                                    type,
                                    "track_circuit_west",
                                    (9  * 12) + L_MARGIN,
                                    T_MARGIN);

map.put(name, new HashItem(arrayNum,index));
```

```
index++;
name = "KN2-10T";
length = 200;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (8  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-9T";
length = 197;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (7  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-8T";
length = 48;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (6  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-7T";
length = 45;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (5  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-6T";
length = 166;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (4  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-5T";
length = 145;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (3  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-4T";
length = 103;
type = 2;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (2  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KN2-3T";
length = 50;
type = 1;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (1  * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// KN
index++;
name = "KN2-2T";
length = 38;
type = 3;
VeaToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));


// LgpToKn

arrayNum = 2;
```

```
// LGP
index = 0;
name = "LGP2-11T";
length = 38;
type = 3;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (37 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// LGP --> AMB
index++;
name = "LGP2-10T";
length = 50;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (36 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-9T";
length = 104;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (35 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-8T";
length = 200;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (34 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-7T";
length = 249;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (33 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-6T";
length = 200;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (32 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-5T";
length = 103;
type = 2;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (31 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-4T";
length = 50;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (30 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// AMB
index++;
name = "LGP2-3T";
length = 38;
type = 3;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "station_west",
                                  (29 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

// AMB --> Bifurcation
index++;
name = "LGP2-2T";
length = 50;
```

```
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (28 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "LGP2-1T";
length = 104;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (27 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-9T";
length = 250;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (26 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-8T";
length = 282;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (25 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));

index++;
name = "KHC2-7T";
length = 159;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (24 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));
```



```
index++;
name = "KHC2-6T";
length = 107;
type = 1;
LgpToKn[index] = new TrackCircuit(name,
                                  length,
                                  type,
                                  "track_circuit_west",
                                  (23 * 12) + L_MARGIN,
                                  T_MARGIN);

map.put(name, new HashItem(arrayNum,index));


// The VeaToKn array and the LgpToKn array contains the same
// elements after the bifurcation
int lastUniqueInLgpToKn = index;
for (int i = 0; i+equalElementsStart < VeaToKn.length; i++) {
    LgpToKn[lastUniqueInLgpToKn + 1 + i]
        = VeaToKn[equalElementsStart + i];
}



//---------------------------------------------------------------
// Autogenerating control line info
//---------------------------------------------------------------

ControlLineInfo[] clis;
int lineSpeed = 80;
int targetSpeed = 0;
int distanceToGo = 0;
int lastDistanceToGo = 2900;

// VeaToKn
for (int i = 0; i < VeaToKn.length; i++) {
    clis = new ControlLineInfo[controlLineLength];
    distanceToGo = 0;
    for (int j = 0; j < controlLineLength-1
             && i+j < VeaToKn.length; j++) {
        distanceToGo += VeaToKn[i+j].getLength();
        clis[j] = new ControlLineInfo(lineSpeed,
                                      targetSpeed,
                                      distanceToGo);
    }
    clis[controlLineLength-1] = new ControlLineInfo(lineSpeed,
                                                    targetSpeed,
                                                    lastDistanceToGo);
    VeaToKn[i].setControlLineNormal(clis);
    VeaToKn[i].setControlLineReverse(clis);
}


// KnToVea
for (int i = 0; i < KnToVea.length; i++) {
    clis = new ControlLineInfo[controlLineLength];
    distanceToGo = 0;
    for (int j = 0; j < controlLineLength-1
             && i+j < KnToVea.length; j++) {
        distanceToGo += KnToVea[i+j].getLength();
        clis[j] = new ControlLineInfo(lineSpeed,
                                      targetSpeed,
```

```
                                        distanceToGo);
            }
            clis[controlLineLength-1] = new ControlLineInfo(lineSpeed,
                                                            targetSpeed,
                                                            lastDistanceToGo);
            KnToVea[i].setControlLineNormal(clis);
            KnToVea[i].setControlLineReverse(clis);
        }


        // LgpToKn
        for (int i = 0; i < LgpToKn.length; i++) {
            clis = new ControlLineInfo[controlLineLength];
            distanceToGo = 0;
            for (int j = 0; j < controlLineLength-1
                     && i+j < LgpToKn.length; j++) {
                distanceToGo += LgpToKn[i+j].getLength();
                clis[j] = new ControlLineInfo(lineSpeed,
                                              targetSpeed,
                                              distanceToGo);
            }
            clis[controlLineLength-1] = new ControlLineInfo(lineSpeed,
                                                            targetSpeed,
                                                            lastDistanceToGo);
            LgpToKn[i].setControlLineNormal(clis);
            LgpToKn[i].setControlLineReverse(clis);
        }

        // KnToLgp
        for (int i = 0; i < KnToLgp.length; i++) {
            clis = new ControlLineInfo[controlLineLength];
            distanceToGo = 0;
            for (int j = 0; j < controlLineLength-1
                     && i+j < KnToLgp.length; j++) {
                distanceToGo += KnToLgp[i+j].getLength();
                clis[j] = new ControlLineInfo(lineSpeed,
                                              targetSpeed,
                                              distanceToGo);
            }
            clis[controlLineLength-1] = new ControlLineInfo(lineSpeed,
                                                            targetSpeed,
                                                            lastDistanceToGo);
            KnToLgp[i].setControlLineNormal(clis);
            KnToLgp[i].setControlLineReverse(clis);
        }


        //---------------------------------------------------------------
        // Initialisation of the track circuits with autogenerated
        // control line info called on the Wayside object passed to the
        // init-method.
        //---------------------------------------------------------------

        wayside.initTrackCircuits(KnToVea, VeaToKn, KnToLgp, LgpToKn, map);

    }

}
```


TrackCircuit.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "TrackCircuit.java"
//=====================================================================

//=====================================================================
// Imported files
//=====================================================================

import eduni.simjava.*;
import eduni.simanim.*;
import java.util.regex.*;

//=====================================================================
// The TrackCircuit class definition
//=====================================================================

class TrackCircuit extends Sim_entity {
    private String name;
    private int length;
    private int type; // 1 is normal, 2 is beacon, 3 is station
    private boolean occupied;
    private boolean pseudoOccupied;
    private int occupiedBy;
    private int delay;
    private String direction; // East or west

    // Objects containing control lines for normal
    // and reverse position of the bifurcation switch
    private ControlLineInfo[] controlLineNormal;
    private ControlLineInfo[] controlLineReverse;

    //-----------------------------------------------------------------
    // Constructor
    //-----------------------------------------------------------------
    TrackCircuit(String name, int length,
                 int type, String image, int x, int y) {
        super(name, image, x, y);
        this.name = name;
        this.length = length;
        this.type = type;
        this.occupied = false;
        this.occupiedBy = -1;
        this.delay = 0;

        // The system starts with upper bifurcation in normal position
        if (name == "3T") {
            this.pseudoOccupied = true;
        }
        else {
            this.pseudoOccupied = false;
        }

        // Initially the track circuit is free
        add_param(new Anim_param(name,Anim_param.STATE,"free",0,0));

        // Subtracting direction information from name. Every track
        // circuit name in the east direction contains a 1 after the
        // letter code, and every track circuit name in the west
        // direction contains a 2 after the letter code.
```

```
    if (Pattern.matches("\\w{2,3}1-\\d{1,2}T", name) || name == "1T") {
        this.direction = "east";
    }
    else {
        this.direction = "west";
    }
}


//---------------------------------------------------------------------
// Entity body
//---------------------------------------------------------------------

// All entities must have a body method
public void body() {}


//---------------------------------------------------------------------
// Accessors and mutators
//---------------------------------------------------------------------

public String getTcName() {
    return name;
}

public int getLength() {
    return length;
}

public int getType() {
    return type;
}

public boolean getOccupied() {
    return occupied;
}

public void setOccupied(boolean occupied) {
    String traceString;
    this.occupied = occupied;
    if (occupied) {
        traceString = "P occupied_" + this.occupiedBy
            + "_" + this.direction;
    }
    else {
        traceString = "P free";
    }

    // Setting trace for the animation
    sim_trace(1, traceString);
}

public boolean getPseudoOccupied() {
    return pseudoOccupied;
}

public void setPseudoOccupied(boolean pseudoOccupied) {
    this.pseudoOccupied = pseudoOccupied;
}

public int getOccupiedBy() {
    return occupiedBy;
}
```



```
    }

    public void setOccupiedBy(int occupiedBy) {
        this.occupiedBy = occupiedBy;
    }

    public int getDelay() {
        return delay;
    }

    public void setDelay(int delay) {
        this.delay = delay;
    }

    public ControlLineInfo getControlLineInfoNormal(int index) {
        return controlLineNormal[index];
    }

    public void setControlLineNormal(ControlLineInfo[] controlLineNormal) {
        this.controlLineNormal = controlLineNormal;
    }

    public ControlLineInfo getControlLineInfoReverse(int index) {
        return controlLineReverse[index];
    }

    public void setControlLineReverse(ControlLineInfo[] controlLineReverse) {
        this.controlLineReverse = controlLineReverse;
    }

    public int getControlLineNormalLength() {
        return controlLineNormal.length;
    }

    public int getControlLineReverseLength() {
        return controlLineReverse.length;
    }

}
```

ControlLineInfo.java

```
//======================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "ControlLineInfo.java"
//======================================================================

//======================================================================
// The ControlLineInfo class definition
//======================================================================

class ControlLineInfo {
    private int lineSpeed;
    private int targetSpeed;
    private int distanceToGo;

    //-----------------------------------------------------------------
    // Constructor
    //-----------------------------------------------------------------
    ControlLineInfo(int lineSpeed, int targetSpeed, int distanceToGo) {
        this.lineSpeed = lineSpeed;
```

```
        this.targetSpeed = targetSpeed;
        this.distanceToGo = distanceToGo;
    }
    public int getLineSpeed() {
        return lineSpeed;
    }
    public int getTargetSpeed() {
        return targetSpeed;
    }
    public int getDistanceToGo() {
        return distanceToGo;
    }
}
```

B.2 The ATS entity

ATS.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "ATS.java"
//=========================================================================

//=========================================================================
// Imported files
//=========================================================================

import eduni.simjava.*;
import eduni.simanim.*;


//=========================================================================
// The ATS class definition
//=========================================================================

abstract class ATS extends Sim_entity {

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    // Ports for communication
    private Sim_port toATS;
    private Sim_port fromATStoWayside;
    private String[] namesOnPortsToTrains;

    // Constant identifiying message tags
    private final int FROM_TRAIN;
    private final int FROM_ATS;
    private final int END_SIM;

    // Number of train in the system
    int numTrains;


    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Constructors                                                      */
    /*                                                                   */
    /*-------------------------------------------------------------------*/


    ATS(String name, int numTrains, String image, int x, int y) {

        // Call to the constructor of Sim_entity
        super(name, image, x, y);


        // Add incoming ports to receive messages from trains
        for (int i=0 ; i < numTrains ; i++) {

            add_port(new Sim_port("toATSfromTrain"+i,
                                  "port",
                                  Anim_port.RIGHT,
                                  i*30+20));

        }

        // Add outgoing ports to send messanges to train
        String portName;
        namesOnPortsToTrains = new String[numTrains];

        for(int i = 0; i < numTrains; i++){
            portName = "fromATStoTrain"+i;
            add_port(new Sim_port(portName,
                                  "port",
                                  Anim_port.RIGHT,
                                  i*30+20));
            namesOnPortsToTrains[i] = portName;
        }


        // Add outgoing port to send messanges to Wayside
        fromATStoWayside = new Sim_port("fromATStoWayside",
                                        "port",
                                        Anim_port.RIGHT,
                                        0);
        add_port(fromATStoWayside);


        // Initialisation of message tags
        FROM_TRAIN = 2;
        FROM_ATS   = 3;
        END_SIM    = 0;

        // Number of train that are to be put into operation
        this.numTrains = numTrains;
    }



    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Methods for communicating with trains                             */
    /*                                                                   */
    /*-------------------------------------------------------------------*/
```

```
//---------------------------------------------------------------------
// getEventFromTrain:
//
// The first message received from a train and the time the
// message arrived is returned. If no messaages has arrived
// the return is delayed until a message arrives.
//---------------------------------------------------------------------

final Sim_event getEventFromTrain(){

    Sim_event msg = new Sim_event();
    Sim_predicate p2 = new Sim_type_p(FROM_TRAIN);

    // Wait for message from a train
    sim_pause_until (p2, msg);
    return msg;
}


//---------------------------------------------------------------------
// eventFromTrainCompleted:
//
// Informs the simulating system that the specified event has been
// dealt with.
//---------------------------------------------------------------------

final void eventFromTrainCompleted(Sim_event e){
    sim_completed(e);
}


// ---------------------------------------------------------------------
// put-in:
//
// After 'delay' seconds the specified train on the specified
// track circuit (CMC or specific pocket track) is informed to be
// put into operation with the specified DID and the specified
// performancelevel
// ---------------------------------------------------------------------

final void putIn(String stationId,
                 int trainId,
                 DID did,
                 PerformanceLevel PL,
                 double delay){


    // Information is packed as message
    ATSToTrain msg = new ATSToTrain(stationId,
                                    trainId,
                                    did,
                                    PL,
                                    false);

    // Information is sent
    String portToUse = namesOnPortsToTrains[trainId];
    sim_schedule(portToUse, delay, FROM_ATS, msg);
    sim_trace(1, "S "+portToUse);

}
```

```
// ---------------------------------------------------------------------
// depart:
//
// After 'delay' seconds the specified train on the specified
// station is informed to depart immidiately with the specified DID
// and the specified performancelevel
// ---------------------------------------------------------------------

final void depart(String stationId,
                  int trainId,
                  DID did,
                  PerformanceLevel PL,
                  double delay){


    // Information is packed as message
    ATSToTrain msg = new ATSToTrain(stationId,
                                    trainId,
                                    did,
                                    PL,
                                    false);

    // Information is sent
    String portToUse = namesOnPortsToTrains[trainId];
    sim_schedule(portToUse, delay, FROM_ATS, msg);
    sim_trace(1, "S "+portToUse);


}




// ---------------------------------------------------------------------
// layUp:
//
// After 'delay' seconds the specified train on the specified
// track circuit (CMS or pocket track) is informed to be layed up
// ---------------------------------------------------------------------

final void layUp(String stationId,
                 int trainId,
                 double delay){


    // Information is packed as message
    ATSToTrain msg = new ATSToTrain(stationId,
                                    trainId,
                                    null,
                                    null,
                                    true);

    // Information is sent
    String portToUse = namesOnPortsToTrains[trainId];
    sim_schedule(portToUse, delay, FROM_ATS, msg);
    sim_trace(1, "S "+portToUse);


}
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Methods for communicating with wayside                            */
/*                                                                   */
/*-------------------------------------------------------------------*/


// ------------------------------------------------------------------
// reserveBifurcation:
//
// After 'delay' seconds a request is send to Wayside for a
// reservation for the bifurcation to be in the specified
// position
// ------------------------------------------------------------------

final void reserveBifurcation(String bifurcationSetting, double delay){

    // Information is packed
    ATSToWayside msg = new ATSToWayside(bifurcationSetting);


    // Information is sent
    String portToUse = "fromATStoWayside";
    sim_schedule(portToUse, delay, FROM_ATS, msg);
    sim_trace(1, "S "+portToUse+" "+bifurcationSetting);
}


// ------------------------------------------------------------------
// endSim:
//
// After 'delay' seconds the simulation is ended
// ------------------------------------------------------------------

final void endSim(double delay){

    int END = 0;
    String portToUseX;

    // End message is sent to wayside
    portToUseX = "fromATStoWayside";
    sim_schedule(portToUseX, delay, END);
    sim_trace(1, "S "+portToUseX);

}



/*-------------------------------------------------------------------*/
/*                                                                   */
/* Methods for communicating with Central                            */
/*                                                                   */
/*-------------------------------------------------------------------*/

// ------------------------------------------------------------------
// getHASTUSTrainPlan:
//
// Gets the HASTUSPlan for the specified train from Central
// ------------------------------------------------------------------

final HASTUSTrainPlan getHASTUSTrainPlan(int trainId){
    return Central.getHASTUSTrainPlan(trainId);
}



// ------------------------------------------------------------------
// getStationAndStretchInfo:
//
// Gets StationAndStretchInfo about the station 'stationTc'
// ------------------------------------------------------------------

final StationAndStretchInfo getStationAndStretchInfo(String stationTc,
                                                     String bfSetting){

    return Central.getStationAndStretchInfo(stationTc, bfSetting);
}



// ------------------------------------------------------------------
// getStationAndStretchInfos:
//
// Gets StationAndStretchInfo about the station 'stationTc' and
// the following 'numStations'-1 stations from Central
// ------------------------------------------------------------------

final StationAndStretchInfo[] getStationAndStretchInfos(int numStations,
                                                        String stationTc,
                                                        String bfSetting){

    return Central.getStationAndStretchInfos(numStations,
                                             stationTc,
                                             bfSetting);
}



// ------------------------------------------------------------------
// getDID:
//
// Gets information on the DID numbered 'DIDno'
// ------------------------------------------------------------------

final DID getDID(int DIDno){
    return Central.getDID(DIDno);
}



/*-------------------------------------------------------------------*/
/*                                                                   */
/* VR                                                                */
/*                                                                   */
/*-------------------------------------------------------------------*/

// Any non-abstract extension of ATS should override this body
// making it include a vehicle regulation algorithm that:
//
//  - handles train put-in
//  - handles dispatching of train from stations including dwell
//    time handling and performance and DID handling. (This is to
```

```
    //    be done using the depart method)
    //  - handles route control by reserving the bifurcation for
    //    train when appropriate
    //  - handles train lay up

    public void body(){}


}
```

ATSWithVerySimpleVR.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "ATSWithVerySimpleVR.java"
//=========================================================================

//=========================================================================
// Imported files
//=========================================================================

import eduni.simjava.*;

//=========================================================================
// The ATSWithSimpleVR class definition
//=========================================================================

class ATSWithVerySimpleVR extends ATS {

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Constructor                                                     */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    public ATSWithVerySimpleVR(String name, int numTrains,
                               String image, int x, int y){

        super(name, numTrains, image, x, y);
    }


    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* An implementation of a Vehicle Regulation algorithm             */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    //-----------------------------------------------------------------
    // body:
    //
    // A very simple Vehicle algorithm. Follows the dwell times and
    // performance levels from HASTUS
    //-----------------------------------------------------------------

    public void body() {
```



```
/* Local variables  */

// Number of trains left to control
int trainsLeft;

// For train put-in
HASTUSTrainPlan trainPlan;
HASTUSPlanEntry firstEntryInPlan;
double startTime;
int startDIDno;
DID startDID;
PerformanceLevel startPL;


// For info on recieved events
Sim_event event;
TrainToATS msg;
int arrivingTrain;
String arrivalStation;
DID arrivalDID;
int arrivalDIDno;

// For info on hastus plan
HASTUSTrainPlan hastusTrainPlan;
HASTUSPlanEntry[] nextTwoHastusEntries;

// For VR planning
HASTUSPlanEntry VRPlan;
int plannedDIDno;
DID plannedDID;
PerformanceLevel plannedPL;
double plannedDwell;

// For bifurcation reservation
String bfSetting;
StationStopInfo infoOnStops;
int numStationsToNextStop;
StationAndStretchInfo[] infoOnStationsToNextStop;
StationAndStretchInfo infoOnCurrentStop;
StationAndStretchInfo infoOnNextStop;
Integer stationsToBifurcationInteger;
int stationsToBifurcation;


/* Train put-in from CMC */

  // For each train
  for(int i = 0; i < numTrains; i++){

    // VR gets the train plan
    trainPlan =  Central.getHASTUSTrainPlan(i);
    firstEntryInPlan = trainPlan.getNextEntry();

    // And according to the first entry in it, VR schedules a message
    // to be delivered to the train the moment it is supposed to
    // arrive at VEA_STATION_2
    startTime  = firstEntryInPlan.getArrivalTime();
    startDIDno = firstEntryInPlan.getDID();
    startDID   = Central.getDID(startDIDno);
    startPL    = firstEntryInPlan.getPL();
    putIn("CMC", i,  startDID, startPL, startTime);
```

```
}

  // The number of train to operate are initialised
  trainsLeft = numTrains;



  /* Traffic control and coordination */

while (Sim_system.running() && trainsLeft > 0) {

    // Get next event from train
    event = getEventFromTrain();
    msg = (TrainToATS) event.get_data();

    // If the event is a train arrival
    if(!msg.isDepartured()){

        // VR gets the message information
        arrivingTrain = msg.getTrainId();
        arrivalStation = msg.getStationId();
        arrivalDID = msg.getDID();
        arrivalDIDno = arrivalDID.getDIDno();

        // VR checks the HASTUS plan
        hastusTrainPlan = getHASTUSTrainPlan(arrivingTrain);
        nextTwoHastusEntries = hastusTrainPlan.getNextEntries(2);

        // VR executes the plan
        VRPlan = nextTwoHastusEntries[0];

        // If it is the last station in the plan
        // VR takes the train out of operation
        if(nextTwoHastusEntries.length == 1){
            plannedDwell = VRPlan.getDwellTime();
            plannedPL = VRPlan.getPL();
            layUp(arrivalStation,
                  arrivingTrain,
                  plannedDwell);
            trainsLeft--;

        }
        // Else the train is given departure info to leave for
        // next station after the planned dwell
        else{
            plannedDIDno = VRPlan.getDID();
            plannedPL = VRPlan.getPL();
            plannedDwell = VRPlan.getDwellTime();

            // If the planned DID is the same as the DID
            // planned on the previous station it is reused
            if(arrivalDIDno == plannedDIDno){
                plannedDID = arrivalDID;

                // If the DID is not the same the new one is
                // used
            }else{
                plannedDID = getDID(plannedDIDno);
            }

            // The train is scheduled to depart
            depart(arrivalStation, arrivingTrain,
                   plannedDID, plannedPL, plannedDwell);


            /* Possible bifurcation reservation */

            // The number of stations from current stop to the
            // bifurcation is determined
            bfSetting   = plannedDID.getBifurcationSetting();
            infoOnStops = plannedDID.getNextStop();
            numStationsToNextStop = infoOnStops.getStationsToNextStop();


            infoOnCurrentStop =
                getStationAndStretchInfo(arrivalStation,
                                         bfSetting);
            stationsToBifurcationInteger
                = infoOnCurrentStop.getStationsToBifurcation(bfSetting);

            stationsToBifurcation
                = stationsToBifurcationInteger.intValue();

            // If the current station is the last stop, but
            // not the last station before the bifurcation VR
            // reserves the bifurcation after the estimated
            // travel time to the last station before the
            // bifurcation plus planned dwell
            if(stationsToBifurcation > 0
               && numStationsToNextStop > stationsToBifurcation){

                infoOnStationsToNextStop =
                    getStationAndStretchInfos(stationsToBifurcation+1,
                                              arrivalStation,
                                              bfSetting);

                double timeToStationJustBeforeBf =
                    Calc.estimatedTimeOfTravel(plannedPL,
                                               infoOnStationsToNextStop,
                                               bfSetting) + plannedDwell;

                reserveBifurcation(bfSetting,timeToStationJustBeforeBf);
            }


            // If the current station is the last station
            // before the bifurcation VR reserves the
            // bifurcation after the planned dwell time
            if(stationsToBifurcation == 0){
                double timeToStationJustBeforeBf = plannedDwell;
                reserveBifurcation(bfSetting,timeToStationJustBeforeBf);

            }

            // The HASTUS plan is updated
            hastusTrainPlan.entryUsed();

        }
    }
    // If the event is a train departure
    else{

        // Do nothing
    }
```

```
            // The event has completed service
            eventFromTrainCompleted(event);
        }

        endSim(100);
    }

}
```

B.3 The train entity

Train.java

```
//===========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Train.java"
//===========================================================================

//===========================================================================
// Imported files
//===========================================================================

import eduni.simanim.*;
import eduni.simjava.*;

//===========================================================================
// The Train class definition
//===========================================================================

class Train extends Sim_entity {

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    // Ports for communication with ATS and Wayside
    private Sim_port trainToATS;
    private Sim_port trainToWayside;
    private Sim_port toTrainFromATS;
    private Sim_port toTrainFromWayside;

    // Predicates
    private Sim_predicate p1 = new Sim_type_p(1); // from wayside
    private Sim_predicate p3 = new Sim_type_p(3); // from ats

    // Event
    private Sim_event e = new Sim_event();

    // Messages
    private ATSToTrain fromATS;
    private TrainToATS msgToATS;
    private TrainToWayside msgToWayside;

    // Global variables needed for programmed stopping
    private boolean programmedStop = false;
    private double distanceToStationStop = 0;
    private double stationStopBrakingRate = 0;

    // Information provided by ATP Wayside
    private String trackCircuitID = "";
    private int trackCircuitType;
    private double trackCircuitLength = 0;
    private double trackCircuitDelay = 0;
    private double lineSpeed = 0;
    private double targetSpeed = 0;
    private double distanceToGo = 0;
    private double targetSpeed_tmp = 0;
    private double distanceToGo_tmp = 0;

    // Information provided by Central
    private double brakingRate = -0.7;

    // Information provided by ATS
    private int PVID = 0;
    private DID DID;
    private PerformanceLevel PL;
    private boolean layUp = false;

    // Logging
    private double time = 0;
    private String logMsg = "";

    // Global variables
    private double currentSpeed = 0;
    private double distWithinTC = 0;
    private double lastOccupancyTimeCalculation = 0;
    private double plAccPeriod = 0;
    private double plPeriod = 0;
    private double occupancyTime = 0;
    private boolean continueOK = true;

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Constructor                                                       */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    Train(String name, int PVID, String image, int x, int y) {
        super(name, image, x, y);

        // Train id
        this.PVID = PVID;

        /* Communication with ATS */

        // Port for sending messages to ATS
        trainToATS = new Sim_port ("train"+PVID+"ToATS",
                                   "port",
                                   Anim_port.LEFT,
                                   10);
        add_port(trainToATS);
```

```
    // Port for receiving messages from ATS
    toTrainFromATS = new Sim_port ("toTrain"+PVID+"FromATS",
                                   "port",
                                   Anim_port.LEFT,
                                   10);
    add_port(toTrainFromATS);

    /* Communication with ATP Wayside */

    // Port for sending messages to ATP Wayside
    trainToWayside = new Sim_port ("train"+PVID+"ToWayside",
                                   "port",
                                   Anim_port.RIGHT,
                                   10);
    add_port(trainToWayside);

    // Port for receiving messages from ATP Wayside
    toTrainFromWayside = new Sim_port ("toTrain"+PVID+"FromWayside",
                                       "port",
                                       Anim_port.RIGHT,
                                       10);
    add_port(toTrainFromWayside);

    // Add animation
    add_param (new Anim_param ("train"+PVID,
                               Anim_param.STATE,
                               "running"));

}


/*-------------------------------------------------------------------*/
/*                                                                   */
/* Body                                                              */
/*                                                                   */
/*-------------------------------------------------------------------*/

public void body() {

    /*---------------------------------------------------------------*/
    /*                                                               */
    /* Train put in ...                                              */
    /*                                                               */
    /*---------------------------------------------------------------*/

    /* Wait for startup request ... */

    // Wait for "train put in" message from ATS ...
    sim_pause_until(p3, e);

    // Read message received from ATS
    fromATS = (ATSToTrain) e.get_data();

    if (fromATS.getTrainId()   == PVID) {

        trackCircuitID = fromATS.getStationId();

        // Set new performance level
        PL = fromATS.getPL();

        // Set new routing information
        DID = fromATS.getDID();
```

```
    }
    // Event completed
    sim_completed(e);

    /* Wait actively for start up acceptance ... */

    // Send "put in request" to Wayside
    msgToWayside = new TrainToWayside ("CMC", PVID);
    sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
    sim_trace (1, "S "+trainToWayside.get_pname());

    // Wait for message from wayside
    sim_get_next(p1,e);

    // Read message from wayside
    readMsgFromWayside(e);
    sim_completed(e);

    while (!continueOK) {

        // Sleep for one simulation clock cycle
        sim_process (1);

        // Send "put in request" to wayside
        msgToWayside = new TrainToWayside ("CMC", PVID);
        sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
        sim_trace (1, "S "+trainToWayside.get_pname());

        // Wait for message from wayside
        sim_get_next(p1, e);

        // Read message from wayside
        readMsgFromWayside(e);
        sim_completed(e);

    }

    /* Stop at VEA ...  */

    // Send "has arrived" message to ATS
    msgToATS = new TrainToATS (trackCircuitID, PVID, DID, PL, false);
    sim_schedule (trainToATS, 0.0, 2, msgToATS);
    sim_trace (1, "S "+trainToATS.get_pname());

    // Log the train arrival
    time = Sim_system.sim_clock();
    logMsg =
        "Actual arr. time:"+Math.round(time)+
        " Train:"+PVID+" Station:"+trackCircuitID;
    Print.log(logMsg);

    // Make sure the train is stopped
    currentSpeed = 0;

    // Update animation
    sim_trace(1, "P stopped");

    // Wait for departure signal from ATS
    sim_process_until (e);

    while (e.get_tag() != 3) {
```

```
    if (e.get_tag() == 1) { // from Wayside
        // Read message from Wayside
        readMsgFromWayside(e);
        sim_completed(e);
    }

    sim_process_until (e);

}

// Read message received from ATS
fromATS = (ATSToTrain) e.get_data();

// Is HASTUS plan terminated ?
layUp = fromATS.getLayUp();

if (!layUp) {

    if (fromATS.getTrainId()   == PVID &&
        fromATS.getStationId() == trackCircuitID) {

        // Set performance level
        PL = fromATS.getPL();

        // Set routing information
        DID = fromATS.getDID();

        // Wait for the specified delay
        sim_process (trackCircuitDelay);

        // Send "has departured" message to ATS
        msgToATS = new TrainToATS (trackCircuitID, PVID, DID, PL, true);
        sim_schedule (trainToATS, 0.0, 2, msgToATS);
        sim_trace (1, "S "+trainToATS.get_pname());

        // Log the train departure
        time = Sim_system.sim_clock();
        logMsg =
            "Actual dep. time:"+Math.round(time)+
            " Train:"+PVID+" Station:"+trackCircuitID;
        Print.log(logMsg);

        // Update animation
        sim_trace (1, "P running");

        // Programmed stop accomplished
        programmedStop = false;

    }

    // Update DID's internal iterator
    DID.stationStoppedAt();
    DID.stationPassed();

    // Send "has reached end of track circuit" message to Wayside
    msgToWayside = new TrainToWayside (trackCircuitID, PVID);
    sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
    sim_trace (1, "S "+trainToWayside.get_pname());

}
else {
    // Update animation
    sim_trace (1, "P running");

    // Send "has reached end of track circuit" message to Wayside
    msgToWayside = new TrainToWayside (trackCircuitID, PVID);
    sim_schedule (trainToWayside, 0.0, -1, msgToWayside);
    sim_trace (1, "S "+trainToWayside.get_pname());


}

sim_completed(e);


/*-------------------------------------------------------------*/
/*                                                             */
/* Train running ...                                           */
/*                                                             */
/*-------------------------------------------------------------*/

while (Sim_system.running()) {

    // Get the next event from wayside
    sim_get_next(p1, e);

    // Read message from Wayside
    readMsgFromWayside(e);
    sim_completed(e);

    // Wait for permission to continue
    while (!continueOK) {

        // Wait for next message from Wayside ...
        sim_get_next(p1, e);

        // Read message from Wayside
        readMsgFromWayside(e);
        sim_completed(e);

    }

    switch (trackCircuitType) {

    case 1: // Normal track circuit

        occupyTrackCircuit(programmedStop);
        distanceToStationStop -= trackCircuitLength;

        // Send "has reached end of track circuit" message to Wayside
        msgToWayside = new TrainToWayside (trackCircuitID, PVID);
        sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
        sim_trace (1, "S "+trainToWayside.get_pname());

        break;

    case 2: // Beacon track circuit

        StationStopInfo stationStopInfo = DID.getNextStation();

        // Check if programmed stop is to be initiated ...
        if (stationStopInfo.isStop()) {
```

```
        // Initiate programmed stop
        programmedStop = true;

        // Read the simulation time
        time = Sim_system.sim_clock();

        // Calculate time since last occupancytime calculation
        double t = lastOccupancyTimeCalculation - time;

        /*-- Perform programmed stop --*/

        // Determine current speed (from last track circuit)
        currentSpeed(t);

        // Determine distance to station stop
        if (trackCircuitID != "KHS2-5T" && trackCircuitID != "VEA1-5T") {
            distanceToStationStop = trackCircuitLength + 50 + 38;
        }
        else if (trackCircuitID == "KHS2-5T") {
            distanceToStationStop = trackCircuitLength + 70 + 38;
        }
        else if (trackCircuitID == "VEA1-5T") {
            distanceToStationStop = trackCircuitLength + 48 + 38;
        }

        // Station stopping braking rate
        stationStopBrakingRate = PL.getDecelerationRate();

        // Programmed stop initiated ...
        occupyTrackCircuit (programmedStop);

    }
    else {
        programmedStop = false;
        occupyTrackCircuit(programmedStop);
    }

    distanceToStationStop -= trackCircuitLength;

    // Send "has reached end of track circuit" message to Wayside
    msgToWayside = new TrainToWayside (trackCircuitID, PVID);
    sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
    sim_trace (1, "S "+trainToWayside.get_pname());

    break;

case 3: // Station track circuit

    if (programmedStop) {

        // Make sure the train is stopped
        currentSpeed = 0;

        // Update animation
        sim_trace(1, "P stopped");

        // Send "has arrived" message to ATS
        msgToATS =
            new TrainToATS (trackCircuitID, PVID, DID, PL, false);
        sim_schedule (trainToATS, 0.0, 2, msgToATS);
        sim_trace (1, "S "+trainToATS.get_pname()+" Arr:"+
                  Math.round(time));

        // Log the train arrival
        time = Sim_system.sim_clock();
        logMsg =
            "Actual arr. time:"+Math.round(time)+
            " Train:"+PVID+" Station:"+trackCircuitID;
        Print.log(logMsg);

        // Wait for departure signal from ATS
        sim_process_until (e);

        while (e.get_tag() != 3) {

            if (e.get_tag() == 1) { // from Wayside
                // Read message from Wayside
                readMsgFromWayside(e);
                sim_completed(e);
            }

            sim_process_until (e);

        }

        // Read message received from ATS
        fromATS = (ATSToTrain) e.get_data();

        // Is HASTUS plan terminated
        layUp = fromATS.getLayUp();

        if (!layUp) {

            if (fromATS.getTrainId()   == PVID &&
                fromATS.getStationId() == trackCircuitID) {

                // Get new performance level
                PL = fromATS.getPL();

                // Get new routing information
                DID = fromATS.getDID();

                // Wait for the specified delay
                sim_process (trackCircuitDelay);

                // Send "has departured" message to ATS
                msgToATS =
                    new TrainToATS (trackCircuitID, PVID, DID, PL, true);
                sim_schedule (trainToATS, 0.0, 2, msgToATS);
                sim_trace (1, "S "+trainToATS.get_pname()+" Dep:"+
                           Math.round(time));

                // Log the train departure
                time = Sim_system.sim_clock();
                logMsg =
                    "Actual dep. time:"+Math.round(time)+
                    " Train:"+PVID+" Station:"+trackCircuitID;
                Print.log(logMsg);

                // Update animation
                sim_trace (1, "P running");

                // Programmed stop accomplished
```

```
                    programmedStop = false;

                }

                // Update DID's internal iterator
                DID.stationStoppedAt();
            }

            sim_completed(e);

        }

        // Get last recieved control line
        targetSpeed  = targetSpeed_tmp;
        distanceToGo = distanceToGo_tmp;

        occupyTrackCircuit(programmedStop);

        // Send "has reached end of track circuit"
        // message to Wayside
        if (layUp) {

            // Update animation
            sim_trace (1, "P running");

            msgToWayside =
                new TrainToWayside (trackCircuitID, PVID);
            sim_schedule (trainToWayside, 0.0, -1, msgToWayside);
            sim_trace (1, "S "+trainToWayside.get_pname());

            break;

        }
        else {

            // Update DID's internal iterator
            DID.stationPassed();

            msgToWayside =
                new TrainToWayside (trackCircuitID, PVID);
            sim_schedule (trainToWayside, 0.0, 2, msgToWayside);
            sim_trace (1, "S "+trainToWayside.get_pname());

        }

        break;

    }

    if (layUp) {
        break;
    }

  }

}


/*------------------------------------------------------------------*/
/*                                                                  */
/* Private methods                                                  */
/*                                                                  */
/*------------------------------------------------------------------*/

//--------------------------------------------------------------------
// occupyTrackCircuit:
//
// Occupies the current track circuit for calculated amount of time
//--------------------------------------------------------------------

private void occupyTrackCircuit (boolean programmedStop) {

    double remainingTime = 0;
    Sim_event e = new Sim_event();

    // Initialisation of global variables
    lastOccupancyTimeCalculation = Sim_system.sim_clock();
    distWithinTC = 0;
    plAccPeriod = 0;
    plPeriod = 0;

    occupancyTime = trackCircuitOccupancyTime(programmedStop);

    remainingTime = sim_process_for (occupancyTime, e);

    // Message received from Wayside ...
    while (remainingTime > 0) {

        if (e.get_tag() == 1) {
            readMsgFromWayside(e);
            sim_completed(e);
        }

        occupancyTime = trackCircuitOccupancyTime(programmedStop);

        remainingTime = sim_process_for (occupancyTime, e);

    }

    /* Track circuit delay ... */

    remainingTime = sim_process_for (trackCircuitDelay, e);

    // Message received from Wayside ...
    while (remainingTime > 0) {

        if (e.get_tag() == 1) {
            readMsgFromWayside(e);
            sim_completed(e);
        }

        remainingTime = sim_process_for (remainingTime+trackCircuitDelay, e);

    }

}


//--------------------------------------------------------------------
// readMsgFromWayside:
//
// Read message from ATP Wayside
//--------------------------------------------------------------------
```

```
private void readMsgFromWayside (Sim_event e) {

    // Get message
    WaysideToTrain fromWayside = (WaysideToTrain) e.get_data();

    continueOK = true;
    String newTrackCircuitID = fromWayside.getTcId();
    int newDistanceToGo = fromWayside.getDistanceToGo();

    if (newTrackCircuitID == trackCircuitID &&
        newDistanceToGo <= distanceToGo_tmp) {

        continueOK = false;

    }

    // track circuit id
    trackCircuitID     = newTrackCircuitID;
    // line speed
    lineSpeed          = fromWayside.getLineSpeed()/3.6;

    if (programmedStop) {
        // target speed
        targetSpeed_tmp  = fromWayside.getTargetSpeed ()/3.6;
        // distance to go
        distanceToGo_tmp = newDistanceToGo;
    }
    else {
        // target speed
        targetSpeed  = fromWayside.getTargetSpeed ()/3.6;
        // distance to go
        distanceToGo = newDistanceToGo;
    }

    // track circuit length
    trackCircuitLength = fromWayside.getLengthOfTc();
    // track circuit delay
    trackCircuitDelay  = fromWayside.getDelayInSeconds();
    // track circuit type
    trackCircuitType   = fromWayside.getTcType();

}


//---------------------------------------------------------------------
// Calculate current speed
//---------------------------------------------------------------------

private void currentSpeed(double t) {

    if (t < plAccPeriod) { // train is accelerating ...

        // Update current speed
        currentSpeed =
            currentSpeed +
            PL.getAccelerationRate() * t;

    }
    else {

        // Update current speed
        currentSpeed =
            currentSpeed +
            PL.getAccelerationRate() * plAccPeriod;

        if (t > plAccPeriod + plPeriod) {

            currentSpeed =
                currentSpeed +
                brakingRate * (t - plAccPeriod - plPeriod);

        }
    }
}


//---------------------------------------------------------------------
// Calculate current distance within track circuit
//---------------------------------------------------------------------

private void currentDistWithinTC (double t) {

    if (t < plAccPeriod) { // train is accelerating ...

        // Update distance travelled within tc
        distWithinTC +=
            currentSpeed * t +
            0.5 * PL.getAccelerationRate() * Math.pow(t,2);

    }
    else {

        // Update distance travelled within tc
        distWithinTC +=
            currentSpeed * plAccPeriod +
            0.5 * PL.getAccelerationRate() * Math.pow(plAccPeriod,2);

        distWithinTC += getPLSpeedLimit()  * plPeriod;

        if (t > plAccPeriod + plPeriod) {

            distWithinTC +=
                getPLSpeedLimit() * (t - plAccPeriod - plPeriod) +
                0.5 * brakingRate * Math.pow((t - plAccPeriod - plPeriod),2);

        }
    }
}


//---------------------------------------------------------------------
// Calculate track circuit occupancy time
//---------------------------------------------------------------------

private double trackCircuitOccupancyTime (boolean programmedStop) {

    // Read the simulation time
    double time = Sim_system.sim_clock();

    // Calculate time since last track circuit occupancy time was
    // calculated ...
    double t = lastOccupancyTimeCalculation - time;

    // Update information about when the last occupancy time was
```

```
// calculated ...
lastOccupancyTimeCalculation = time;

// Update current distance within track circuit
currentDistWithinTC(t);

// Update current speed
currentSpeed(t);

// Determine target speed and distance to go
if (programmedStop) {
    targetSpeed = 0;
    distanceToGo = Math.min(distanceToGo, distanceToStationStop);
    brakingRate = Math.min(brakingRate,stationStopBrakingRate);
}

/*-- Calculate current distance to go --*/
double currentDTG = distanceToGo - distWithinTC;

/*-- Calculate track occupancy time --*/
occupancyTime = 0;

/*-- Calculate desired current speed according to new control line --*/
// (the speed of the train must be 5 kmph below the speed limit
// prescribed by the SBD profile)
double v1 = Math.sqrt(Math.pow(targetSpeed,2) + 2 *
                      -brakingRate *
                      currentDTG) - (5/3.6);

if (v1 > lineSpeed-(5/3.6)) {
    v1 = lineSpeed-(5/3.6);
}

/*-- Desired current speed  --*/
double desiredSpeed = Math.min (getPLSpeedLimit(), v1);

/*-- Calculate possible needed acceleration time --*/
// Condition should always be true ...
if (currentSpeed <= desiredSpeed) {

    // Time for train to accelerate to performance level speed limit
    double t1 =
        (getPLSpeedLimit() - currentSpeed) /
        PL.getAccelerationRate();

    // Time for train to accelerate to the speed limit prescribed by the
    // SBD profile
    double t2_a =
        (lineSpeed - currentSpeed) /
        PL.getAccelerationRate();

    double t2_b =
        (targetSpeed - currentSpeed) /
        (PL.getAccelerationRate() + brakingRate);

    double t2 = Math.min(t2_a,t2_b);

    // Time for acceleration to reach end of track circuit
    double t3 = timeToEndOfTrackCircuit(currentSpeed,
                                        distWithinTC,
                                        PL.getAccelerationRate());

// Update information about acceleration period
plAccPeriod = Math.min (t1, Math.min(t2, t3));

double expectedDistWithinTC = 0;

if (plAccPeriod == t1) { // Performance level is reached first

    occupancyTime += t1;

    /*-- Calculate how long time performance level is followed --*/

    // Expected distance travelled within track circuit during acc
    expectedDistWithinTC =
        distWithinTC +
        currentSpeed * t1 +
        0.5 * PL.getAccelerationRate() * Math.pow(t1,2);

    /*-- Time to track circuit end following performance level --*/
    double d = trackCircuitLength - expectedDistWithinTC;
    // (d ought always to be positive ...)
    double t1_a =  d / getPLSpeedLimit();

    /*-- Time to SBD profile intersection with performance level --*/
    // Time for SBD profile to reach performance level
    // speed from target speed
    double t1_b =
        (getPLSpeedLimit() - targetSpeed) /
        -brakingRate;

    // Distance from target speed target point
    double dist_a =
        targetSpeed * t1_b +
        0.5 * -brakingRate * Math.pow(t1_b,2);

    // Intersection position within track circuit
    double dist_b = distanceToGo - expectedDistWithinTC - dist_a;

    if (dist_b < (trackCircuitLength - expectedDistWithinTC)) {
        // SBD profile reached first

        expectedDistWithinTC += dist_b;
        t1 = dist_b / getPLSpeedLimit();

        // Update information about period where pl is followed
        plPeriod = t1;

        occupancyTime += t1;
        occupancyTime +=
            timeToEndOfTrackCircuit(getPLSpeedLimit(),
                                    expectedDistWithinTC,
                                    brakingRate);

    }
    else {
        // End of track circuit reached first

        // Update information about period where pl is followed
        plPeriod = t1_a;
        occupancyTime += t1_a;

    }
```

```
        }
        else if (plAccPeriod == t2) { // New SBD profile is reached first

            plPeriod = 0;

            occupancyTime += t2;

            // Expected distance travelled within track circuit during acc
            expectedDistWithinTC =
                distWithinTC +
                currentSpeed * t2 +
                0.5 * PL.getAccelerationRate() * Math.pow(t2,2);

            // Expected current speed within track circuit after acc
            double expectedCurrentSpeed =
                currentSpeed +
                t2 * PL.getAccelerationRate();

            occupancyTime +=
                timeToEndOfTrackCircuit(expectedCurrentSpeed,
                                        expectedDistWithinTC,
                                        brakingRate);

        }
        else if (plAccPeriod == t3) { // End of track circuit is reached

            plPeriod = 0;

            occupancyTime += t3;

        }

    }

    return occupancyTime;

}


//-----------------------------------------------------------------
// Time to end of track circuit
//-----------------------------------------------------------------

private double timeToEndOfTrackCircuit (double speed,
                                        double distWithinTrackCircuit,
                                        double accelerationRate) {

    double distToEndOfTC =
        trackCircuitLength - distWithinTrackCircuit;

    double t1 =
        (-1*speed +
         Math.sqrt(Math.pow(speed,2)
              - (4 * 0.5 * PL.getAccelerationRate() *
                 -1 * distToEndOfTC))) /
        PL.getAccelerationRate();

    double t2 =
        (-1*speed -
         Math.sqrt(Math.pow(speed,2)
                   - (4 * 0.5 * PL.getAccelerationRate() *
                      -1 * distToEndOfTC))) /
                          PL.getAccelerationRate();

        double t3 = 0;
        if (t1 < 0 && t2 >= 0) {
            t3 = t2;
        }
        else if (t2 < 0 && t1 >= 0) {
            t3 = t1;
        }
        else if (t1 > 0 && t2 > 0) {
            t3 = Math.min(t1,t2);
        }

        return t3;

    }

    //-----------------------------------------------------------------
    // Read performance level speed limit
    //-----------------------------------------------------------------

    private double getPLSpeedLimit () {
        double plSpeedLimit = PL.getSpeedLimit();
        return (plSpeedLimit/3.6);
    }

}
```

B.4 The interruption entity

Interruption.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Interruption.java"
//=====================================================================

//=====================================================================
// Imported files
//=====================================================================

import eduni.simjava.*;

//=====================================================================
// The Interruption class definition
//=====================================================================

class Interruption extends Sim_entity {

    private Wayside wayside;

    //-----------------------------------------------------------------
    // Constructor
    //-----------------------------------------------------------------
    Interruption(String name, Wayside wayside) {
        super(name);
        this.wayside = wayside;
    }
```

```
    //--------------------------------------------------------------------
    // Entity body
    //--------------------------------------------------------------------
    public void body() {
        wayside.setDelay("ORE2-3T", 1000);
    }

}
```

Interruption1.java

```
//========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Interruption1.java"
//========================================================================

//========================================================================
// Imported files
//========================================================================

import eduni.simjava.*;

//========================================================================
// The Interruption1 class definition
//========================================================================

class Interruption1 extends Sim_entity {

    private Wayside wayside;

    //--------------------------------------------------------------------
    // Constructor
    //--------------------------------------------------------------------
    Interruption1(String name, Wayside wayside) {
        super(name);
        this.wayside = wayside;
    }

    //--------------------------------------------------------------------
    // Entity body
    //--------------------------------------------------------------------
    public void body() {
        wayside.setDelay("ORE2-3T", 1000);
    }

}
```

Interruption2.java

```
//========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Interruption2.java"
//========================================================================

//========================================================================
// Imported files
//========================================================================

import eduni.simjava.*;

//========================================================================
// The Interruption2 class definition
//========================================================================

class Interruption2 extends Sim_entity {

    private Wayside wayside;

    //--------------------------------------------------------------------
    // Constructor
    //--------------------------------------------------------------------
    Interruption2(String name, Wayside wayside) {
        super(name);
        this.wayside = wayside;
    }

    //--------------------------------------------------------------------
    // Entity body
    //--------------------------------------------------------------------
    public void body() {
        wayside.setDelay("KHC2-7T", 10000);
    }

}
```

B.5 Communication between entities

ATSToTrain.java

```
//========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "ATSToTrain.java"
//========================================================================

//========================================================================
// The ATSToTrain class definition
//========================================================================

class ATSToTrain {

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Attributes                                                         */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    // Geographical destination of message
    private String stationId;

    // Destination train
    private int trainId;

    // DID the train is to use
    private DID didToUse;

    // Performance level the train is to use
    private PerformanceLevel plToUse;
```

```
// Indication of terminated HASTUS plan
private boolean layUp;

/*-------------------------------------------------------------------*/
/*                                                                   */
/* Constructor                                                       */
/*                                                                   */
/*-------------------------------------------------------------------*/

ATSToTrain(String stationId,
           int trainId,
           DID didToUse,
           PerformanceLevel plToUse,
           boolean layUp) {

    this.stationId = stationId;
    this.trainId = trainId;
    this.didToUse = didToUse;
    this.plToUse = plToUse;
    this.layUp = layUp;

}



/*-------------------------------------------------------------------*/
/*                                                                   */
/* Accessors                                                         */
/*                                                                   */
/*-------------------------------------------------------------------*/

//-------------------------------------------------------------------
// getStationId:
//
// Returns the name of the track circuit on which the train should
// be when receiving the message
//-------------------------------------------------------------------

public String getStationId(){
    return stationId;
}


//-------------------------------------------------------------------
// getTrainId:
//
// Returns the ID of the train to which the message is sent
//-------------------------------------------------------------------

public int getTrainId(){
    return trainId;
}


//-------------------------------------------------------------------
// getDID:
//
// Returns the DID the train is to use
//-------------------------------------------------------------------

public DID getDID(){
    return didToUse;
}


//-------------------------------------------------------------------
// getPL:
//
// Returns the performance level the train is to use
//-------------------------------------------------------------------

public PerformanceLevel getPL(){
    return plToUse;
}


//-------------------------------------------------------------------
// getHastusTerminated:
//
// Returns whether or not the Hastus plan is terminated
//-------------------------------------------------------------------

public boolean getLayUp() {
    return layUp;
}

}
```

TrainToATS.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "TrainToATS.java"
//=====================================================================

//=====================================================================
// The TrainToATS class definition
//=====================================================================

class TrainToATS {

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    // Geographical source of message
    private String stationId;

    // Train which sent the message
    private int trainId;

    // The train's current DID
    private DID currentDID;

    // The train's current performance level
    private PerformanceLevel currentPL;

    // Attribute indicating whether it is an arrival message or a
    // departure message
    private boolean isDepartured;
```

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Constructor                                                        */
/*                                                                    */
/*--------------------------------------------------------------------*/

TrainToATS (String stationId,
            int trainId,
            DID currentDID,
            PerformanceLevel currentPL,
            boolean isDepartured) {

    this.stationId    = stationId;
    this.trainId      = trainId;
    this.currentDID   = currentDID;
    this.currentPL    = currentPL;
    this.isDepartured = isDepartured;
}



/*--------------------------------------------------------------------*/
/*                                                                    */
/* Accessors                                                          */
/*                                                                    */
/*--------------------------------------------------------------------*/

//--------------------------------------------------------------------
// getStationId:
//
// Returns the name of the track circuit on which the train is
// situated when sending the message
//--------------------------------------------------------------------

public String getStationId(){
    return stationId;
}



//--------------------------------------------------------------------
// getTrainId:
//
// Returns the ID number of the train sending the message
//--------------------------------------------------------------------

public int getTrainId(){
    return trainId;
}



//--------------------------------------------------------------------
// getDID:
//
// Returns the current DID of the train
//--------------------------------------------------------------------

public DID getDID(){
    return currentDID;
}



//--------------------------------------------------------------------
// getPL:
//
// Returns the current performance level of the train
//--------------------------------------------------------------------

public PerformanceLevel getPL(){
    return currentPL;
}



//--------------------------------------------------------------------
// isDeparted:
//
// Returns true if and only if the message is a departure message
//--------------------------------------------------------------------

public boolean isDepartured(){
    return isDepartured;
}


}
```

ATSToWayside.java

```
//====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "ATSToWayside.java"
//====================================================================

//====================================================================
// The ATSToTrain class definition
//====================================================================

class ATSToWayside{

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Attributes                                                         */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    // A reservation for the bifurcation
    // Can be either "KNToVEA", "VEAToKN", "KNToLGP" or "LGPToKN"
    String bifurcationReservation;


    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Constructor                                                        */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    ATSToWayside(String bifurcationReservation) {
```

```
        this.bifurcationReservation = bifurcationReservation;
    }

    /*----------------------------------------------------------------------*/
    /*                                                                      */
    /* Accessors                                                            */
    /*                                                                      */
    /*----------------------------------------------------------------------*/

    //----------------------------------------------------------------------
    // getBifurcationReservation:
    //
    // Returns the reservation of the bifurcation
    //----------------------------------------------------------------------

    public String getBifurcationReservation(){
        return bifurcationReservation;
    }

}
```

WaysideToTrain.java

```
//======================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "WaysideToTrain.java"
//======================================================================

//======================================================================
// The WaysideToTrain class definition
//======================================================================

class WaysideToTrain {

    /*----------------------------------------------------------------------*/
    /*                                                                      */
    /* Attributes                                                           */
    /*                                                                      */
    /*----------------------------------------------------------------------*/

    // Info about track circuit
    private String nameOfTc;
    private int typeOfTc;
    private int lengthOfTc;

    // Control line info
    private int lineSpeed;
    private int targetSpeed;
    private int distanceToGo;

    // Delay info
    private int delayInSeconds;



    /*----------------------------------------------------------------------*/
    /*                                                                      */
    /* Constructor                                                          */
    /*                                                                      */
    /*----------------------------------------------------------------------*/

    WaysideToTrain(String nameOfTc,
                   int typeOfTc,
                   int lengthOfTc,
                   int lineSpeed,
                   int targetSpeed,
                   int distanceToGo,
                   int delayInSeconds) {

        this.nameOfTc = nameOfTc;
        this.typeOfTc = typeOfTc;
        this.lengthOfTc = lengthOfTc;
        this.lineSpeed = lineSpeed;
        this.targetSpeed = targetSpeed;
        this.distanceToGo = distanceToGo;
        this.delayInSeconds = delayInSeconds;
    }



    /*----------------------------------------------------------------------*/
    /*                                                                      */
    /* Accessors                                                            */
    /*                                                                      */
    /*----------------------------------------------------------------------*/

    //----------------------------------------------------------------------
    // getTcId:
    //
    // Returns the name of the track circuit
    //----------------------------------------------------------------------

    public String getTcId(){
        return nameOfTc;
    }



    //----------------------------------------------------------------------
    // getTcType:
    //
    // Returns the type of the track circuit
    //----------------------------------------------------------------------

    public int getTcType(){
        return typeOfTc;
    }



    //----------------------------------------------------------------------
    // getLengthOfTc:
    //
    // Returns the length of the track circuit
    // Measure : meters
    //----------------------------------------------------------------------

    public int getLengthOfTc(){
        return lengthOfTc;
    }
```

```
    //--------------------------------------------------------------------
    // getLineSpeed:
    //
    // Returns the line speed parameter of the control line info
    // Measure : km/h
    //--------------------------------------------------------------------

    public int getLineSpeed(){
        return lineSpeed;
    }



    //--------------------------------------------------------------------
    // getDistanceToGo:
    //
    // Returns the distance-to-go parameter of the control line info
    // Measure : meters
    //--------------------------------------------------------------------

    public int getDistanceToGo(){
        return distanceToGo;
    }



    //--------------------------------------------------------------------
    // getTargetSpeed:
    //
    // Returns the target speed parameter of the control line info
    // Measure : km/h
    //--------------------------------------------------------------------

    public int getTargetSpeed(){
        return targetSpeed;
    }



    //--------------------------------------------------------------------
    // getDelayInSeconds:
    //
    // Returns the distance to go parameter of the control line info
    // Measure: seconds
    //--------------------------------------------------------------------

    public int getDelayInSeconds(){
        return delayInSeconds;
    }


}
```

TrainToWayside.java

```
//======================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "TrainToWayside.java"
//======================================================================

//======================================================================
// The TrainToWayside class definition
//======================================================================

class TrainToWayside {

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Attributes                                                         */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    // The Tc that is about to be left
    private String previousTc;

    // The train that is about to leave 'previousTc'
    private int trainId;

    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Constructor                                                        */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    TrainToWayside(String previousTc, int trainId) {
        this.previousTc = previousTc;
        this.trainId = trainId;
    }



    /*--------------------------------------------------------------------*/
    /*                                                                    */
    /* Accessors                                                          */
    /*                                                                    */
    /*--------------------------------------------------------------------*/

    //--------------------------------------------------------------------
    // getPrevious:
    //
    // Returns the name of the track circuit that is about to be left
    //--------------------------------------------------------------------

    public String getPreviousTc(){
        return previousTc;
    }



    //--------------------------------------------------------------------
    // getTrainId:
    //
    // Returns the Id of the train that is about to leave 'previousTc'
    //--------------------------------------------------------------------

    public int getTrainId(){
        return trainId;
    }


}
```

B.6 Central

Central.java

```
//===========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Central.java"
//===========================================================================

//===========================================================================
// Imported files
//===========================================================================

import java.util.*;

//===========================================================================
// The Central class definition
//===========================================================================

class Central{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Available info                                                    */
    /*                                                                   */
    /*-------------------------------------------------------------------*/


    //-------------------------------------------------------------------
    // Topography, speed restrictions and the TDs VR configurations
    //-------------------------------------------------------------------

    private static Hashtable stationAndStretchInfos;



    //-------------------------------------------------------------------
    // HASTUS plans for all trains
    //-------------------------------------------------------------------

    private static HASTUSTrainPlan[] HASTUSPlan;



    //-------------------------------------------------------------------
    // Performance levels
    //-------------------------------------------------------------------

    final static PerformanceLevel PL1 =
        new PerformanceLevel(1, 80, 1.2, -0.9);

    final static PerformanceLevel PL2 =
        new PerformanceLevel(2, 70, 1.0, -0.9);

    final static PerformanceLevel PL3 =
        new PerformanceLevel(3, 60, 0.8, -0.8);

    final static PerformanceLevel PL4 =
        new PerformanceLevel(4, 55, 0.8, -0.8);

    final static PerformanceLevel PL5 =
        new PerformanceLevel(5, 50, 0.6, -0.7);

    final static PerformanceLevel PL6 =
        new PerformanceLevel(6, 60, 1.0, -0.5);

    final static PerformanceLevel PL7 =
        new PerformanceLevel(7, 30, 0.8, -0.5);

    final static PerformanceLevel PL8 =
        new PerformanceLevel(8, 15, 0.8, -0.5);

    final static PerformanceLevel PL9 =
        new PerformanceLevel(9,  5, 0.8, -0.5);



    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Initialisers                                                      */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    //-------------------------------------------------------------------
    // initTopographySpeedRestrictionsAndTDconfigs:
    //
    // Sets the hastus plan of Central to be the specified plan.
    // Should be called before the system starts up if the system is
    // in schedule mode. Should be called before Central is used if
    // the system is to run in schedule mode
    //-------------------------------------------------------------------

    static void initTopographySpeedRestrictionAndTDconfig(Hashtable info){
        stationAndStretchInfos = info;
    }



    //-------------------------------------------------------------------
    // initHASTUSPlan:
    //
    // Sets the hastus plan of Central to be the specified plan.
    // Should be called before the system starts up if the system is
    // in schedule mode. Should be called before Central is used if
    // the system is to run in schedule mode
    //-------------------------------------------------------------------

    public static void initHASTUSPlan(HASTUSTrainPlan[] plan){
        HASTUSPlan = plan;
    }



    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Accessors                                                         */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    //-------------------------------------------------------------------
    // getStationAndStretchInfo:
    //
```

```
// Returns info about the specified station and the stretch of
// rail between that station and the next
//---------------------------------------------------------------------

public static StationAndStretchInfo
    getStationAndStretchInfo(String stationTc,
                             String bifurcationSetting){

    StationAndStretchInfo wantedInfo = null;

    if(stationAndStretchInfos.containsKey(stationTc)){
        wantedInfo =
         (StationAndStretchInfo) stationAndStretchInfos.get(stationTc);
    }

    return wantedInfo;
}



//---------------------------------------------------------------------
// getStationAndStretchInfos:
//
// Returns info about the specified station and the following
// 'numStations' - 1 stations and the stretches of rail
// following each of these.
//---------------------------------------------------------------------

public static StationAndStretchInfo[]
    getStationAndStretchInfos(int numStations,
                              String stationTc,
                              String bifurcationSetting){

    String station = stationTc;
    ArrayList wantedInfos = new ArrayList() ;
    StationAndStretchInfo wantedInfo;

    // For each station info is found in StationAndStretchInfos
    // and is added to the list of wanted infos
    for(int i = 0; i < numStations; i++){

        if(stationAndStretchInfos.containsKey(station)){
          wantedInfo =
           (StationAndStretchInfo) stationAndStretchInfos.get(station);
          wantedInfos.add(wantedInfo);

          station =
           wantedInfo.getFollowingStation(bifurcationSetting);

          if(station == null){
              break;
          }
        }
        else{
            break;
        }
    }

    // The list is turned into an array and returned
    int numInfos = wantedInfos.size();

    StationAndStretchInfo[] infos
        = new StationAndStretchInfo[numInfos];

    for(int i=0; i < numInfos; i++){
        infos[i] = (StationAndStretchInfo) wantedInfos.get(i);
    }

    return infos;
}



//---------------------------------------------------------------------
// getHASTUSTrainPlan:
//
// Returns the HASTUS plan for the specified train
//---------------------------------------------------------------------

public static HASTUSTrainPlan getHASTUSTrainPlan(int trainId){
    return HASTUSPlan[trainId];
}



//---------------------------------------------------------------------
// getDID:
//
// Returns a DID object of the type DID'DIDno'
//---------------------------------------------------------------------

public static DID getDID(int DIDno){
    DID newDID = null;

    switch(DIDno){
    case 1:
        newDID = new DID1();
        break;
    case 2:
        newDID = new DID2();
        break;
    case 3:
        newDID = new DID3();
        break;
    case 4:
        newDID = new DID4();
        break;
    case 5:
        newDID = new DID5();
        break;

    // New DIDs should be here...

    }

    return newDID;
}



//---------------------------------------------------------------------
```

```
    // getPerformanceLevel:
    //
    // Returns a reference to a performance level 'PLno'
    //---------------------------------------------------------------------

    public static PerformanceLevel getPerformanceLevel(int PLno){

        // Default PL is PL2 (normal)
        PerformanceLevel PL = PL2;

        // If a specific existing performance level is
        // asked for PL is set to that

        switch(PLno){
        case 1:
            PL = PL1;
            break;
        case 2:
            PL = PL2;
            break;
        case 3:
            PL = PL3;
            break;
        case 4:
            PL = PL4;
            break;
        case 5:
            PL = PL5;
            break;
        case 6:
            PL = PL6;
            break;
        case 7:
            PL = PL7;
            break;
        case 8:
            PL = PL8;
            break;
        case 9:
            PL = PL9;
            break;
        }

        // The chosen performance level is returned
        return PL;
    }


}
```

InitialiseCentral.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "InitialiseCentral.java"
//=====================================================================

//=====================================================================
// Imported files

//=====================================================================

import java.util.*;


//=====================================================================
// Class for initialising Central
//=====================================================================

class InitialiseCentral{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Permanent way info                                                */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    //-------------------------------------------------------------------
    // Names of station track circuits
    //-------------------------------------------------------------------

    //---------
    // Track 1
    //---------

    /* Shared part */
    final static String KN_STATION_1  = "KN1-2T";
    final static String KGN_STATION_1 = "KGN1-3T";
    final static String KHC_STATION_1 = "KHC1-2T";

    /* VEA part */
    final static String ISB_STATION_1 = "ISB1-3T";
    final static String UNI_STATION_1 = "UNI1-2T";
    final static String KHS_STATION_1 = "KHS1-3T";
    final static String BC_STATION_1  = "KHS1-9T";
    final static String ORE_STATION_1 = "ORE1-7T";
    final static String VEA_STATION_1 = "VEA1-7T";

    /* LGP part */
    final static String AMB_STATION_1 = "LGP1-3T";
    final static String LGP_STATION_1 = "LGP1-11T";

    //---------
    // Track 2
    //---------

    /* VEA part */
    final static String VEA_STATION_2 = "VEA2-8T";
    final static String ORE_STATION_2 = "ORE2-7T";
    final static String BC_STATION_2  = "KHS2-10T";
    final static String KHS_STATION_2 = "KHS2-3T";
    final static String UNI_STATION_2 = "UNI2-2T";
    final static String ISB_STATION_2 = "ISB2-3T";

    /* LGP part */
    final static String LGP_STATION_2 = "LGP2-11T";
    final static String AMB_STATION_2 = "LGP2-3T";

    /* Shared part */
    final static String KHC_STATION_2  = "KHC2-2T";
    final static String KGN_STATION_2  = "KGN2-3T";
    final static String KN_STATION_2   = "KN2-2T";
```

```
//-------------------------------------------------------------------
// Distances between stations
//-------------------------------------------------------------------'

//---------
// Track 1
//---------

/* Shared part */
final static int distKNToKGN  = 1078;
final static int distKGNToKHC = 854;


/* VEA part */
final static int distKHCToISB = 1230;
final static int distISBToUNI = 952;
final static int distUNIToKHS = 1203;
final static int distKHSToBC  = 817;
final static int distBCToORE  = 1023;
final static int distOREToVEA = 1111;
final static int distOutOfVEA = 0;

/* LGP part */
final static int distKHCToAMB = 1293;
final static int distAMBToLGP = 1017;
final static int distOutOfLGP = 0;


//---------
// Track 2
//---------

/* VEA part */
final static int distVEAToORE = 1111;
final static int distOREToBC  = 1023;
final static int distBCToKHS  = 817;
final static int distKHSToUNI = 1207;
final static int distUNIToISB = 950;
final static int distISBToKHC = 1251;

/* LGP part */
final static int distLGPToAMB = 994;
final static int distAMBToKHC = 1289;

/* Shared part */
final static int distKHCToKGN = 886;
final static int distKGNToKN  = 1137;
final static int distOutOfKN  = 0;



//-------------------------------------------------------------------
// Speed restriction info
//-------------------------------------------------------------------

//---------
// Track 1
//---------

/* Shared part */
private static Stretch stretchKNToKGN;
private static Stretch stretchKGNToKHC;
private static Stretch stretchKHCToISB;


/* VEA part */
private static Stretch stretchISBToUNI;
private static Stretch stretchUNIToKHS;
private static Stretch stretchKHSToBC;
private static Stretch stretchBCToORE;
private static Stretch stretchORETOVEA;
private static Stretch stretchOutOfVEA;

/* LGP part */
private static Stretch stretchKHCToAMB;
private static Stretch stretchAMBToLGP;
private static Stretch stretchOutOfLGP;


//---------
// Track 2
//---------

/* VEA part */
private static Stretch stretchVEAToORE;
private static Stretch stretchORETOBC;
private static Stretch stretchBCToKHS;
private static Stretch stretchKHSToUNI;
private static Stretch stretchUNIToISB;
private static Stretch stretchISBToKHC;

/* LGP part */
private static Stretch stretchLGPToAMB;
private static Stretch stretchAMBToKHC;

/* Shared part */
private static Stretch stretchKHCToKGN;
private static Stretch stretchKGNToKN;
private static Stretch stretchOutOfKN;




/*-------------------------------------------------------------------*/
/*                                                                   */
/* TD specified constants                                            */
/*                                                                   */
/*-------------------------------------------------------------------*/

//-------------------------------------------------------------------
// VR configurations
//-------------------------------------------------------------------

// The performance levels VR is allowed to use


/* Shared part */
private static PerformanceLevel[] PLsKN;
private static PerformanceLevel[] PLsKGN;
private static PerformanceLevel[] PLsKHC;

/* VEA part */
```

```
private static PerformanceLevel[] PLsISB;
private static PerformanceLevel[] PLsUNI;
private static PerformanceLevel[] PLsKHS;
private static PerformanceLevel[] PLsBC;
private static PerformanceLevel[] PLsORE;
private static PerformanceLevel[] PLsVEA;

/* LGP part */
private static PerformanceLevel[] PLsAMB;
private static PerformanceLevel[] PLsLGP;


// The minimum dwell time VR is allowed to use

/* Shared part */
private static double minDwellKN;
private static double minDwellKGN;
private static double minDwellKHC;

/* VEA part */
private static double minDwellISB;
private static double minDwellUNI;
private static double minDwellKHS;
private static double minDwellBC;
private static double minDwellORE;
private static double minDwellVEA;

/* LGP part */
private static double minDwellAMB;
private static double minDwellLGP;


// The maximum dwell time VR is allowed to use

/* Shared part */
private static double maxDwellKN;
private static double maxDwellKGN;
private static double maxDwellKHC;

/* VEA part */
private static double maxDwellISB;
private static double maxDwellUNI;
private static double maxDwellKHS;
private static double maxDwellBC;
private static double maxDwellORE;
private static double maxDwellVEA;

/* LGP part */
private static double maxDwellAMB;
private static double maxDwellLGP;



/*-------------------------------------------------------------------*/
/*                                                                   */
/* Initialisers                                                      */
/*                                                                   */
/*-------------------------------------------------------------------*/


//-------------------------------------------------------------------
```

```
// initSpeedRestrictions:
//
// Initialises info on speed restrictions
// Should be consistent with speed restriction info used in Wayside.
//-------------------------------------------------------------------

private static void initSpeedRestrictions(){

    int dominatingSpeedLimit = 80;
    int[] speedLimits = {80};
    int lengthOfStretch;

    //---------
    // Track 1
    //---------

    /* Shared part */

    // KN --> KGN
    lengthOfStretch         = distKNToKGN;
    int[] speedLimitKNToKGN = { lengthOfStretch };
    stretchKNToKGN          =  new Stretch(dominatingSpeedLimit,
                                           lengthOfStretch,
                                           speedLimits,
                                           speedLimitKNToKGN);

    // KGN --> KHC
    lengthOfStretch         = distKGNToKHC;
    int[] speedLimitKGNToKHC= { lengthOfStretch };
    stretchKGNToKHC         =  new Stretch(dominatingSpeedLimit,
                                           lengthOfStretch,
                                           speedLimits,
                                           speedLimitKGNToKHC);


    /* VEA part */

    // KHC --> ISB
    lengthOfStretch         = distKHCToISB;
    int[] speedLimitKHCToISB= { lengthOfStretch };
    stretchKHCToISB         =  new Stretch(dominatingSpeedLimit,
                                           lengthOfStretch,
                                           speedLimits,
                                           speedLimitKHCToISB);

    // KHC --> ISB
    lengthOfStretch         = distISBToUNI;
    int[] speedLimitISBToUNI= { lengthOfStretch };
    stretchISBToUNI         =  new Stretch(dominatingSpeedLimit,
                                           lengthOfStretch,
                                           speedLimits,
                                           speedLimitISBToUNI);


    // ISB --> KHS
    lengthOfStretch         = distUNIToKHS;
    int[] speedLimitUNIToKHS= { lengthOfStretch };
    stretchUNIToKHS         =  new Stretch(dominatingSpeedLimit,
                                           lengthOfStretch,
                                           speedLimits,
                                           speedLimitUNIToKHS);
```

```
// KHS --> BC
lengthOfStretch         = distKHSToBC;
int[] speedLimitKHSToBC = { lengthOfStretch };
stretchKHSToBC          =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitKHSToBC);

// BC --> ORE
lengthOfStretch         = distBCToORE;
int[] speedLimitBCToORE = { lengthOfStretch };
stretchBCToORE          =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitBCToORE);

// ORE --> VEA
lengthOfStretch         = distOREToVEA;
int[] speedLimitOREToVEA= { lengthOfStretch };
stretchOREToVEA         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitOREToVEA);

// VEA -->
lengthOfStretch         = distOutOfVEA;
int[] speedLimitOutOfVEA= { lengthOfStretch };
stretchOutOfVEA         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitOutOfVEA);


/* LGP part */

// KHC --> AMB
lengthOfStretch         = distKHCToAMB;
int[] speedLimitKHCToAMB= { lengthOfStretch };
stretchKHCToAMB         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitKHCToAMB);

// AMB --> LGP
lengthOfStretch         = distAMBToLGP;
int[] speedLimitAMBToLGP= { lengthOfStretch };
stretchAMBToLGP         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitAMBToLGP);

// LGP -->
lengthOfStretch         = distOutOfLGP;
int[] speedLimitOutOfLGP= { lengthOfStretch };
stretchOutOfLGP         =  new Stretch(dominatingSpeedLimit,
                                        lengthOfStretch,
                                       speedLimits,
                                        speedLimitOutOfLGP);
//---------
// Track 2
//---------
```



```
/* VEA part */

// VEA --> ORE
lengthOfStretch         = distVEAToORE;
int[] speedLimitVEAToORE= { lengthOfStretch };
stretchVEAToORE         =  new Stretch(dominatingSpeedLimit,
                                        lengthOfStretch,
                                        speedLimits,
                                        speedLimitVEAToORE);

// ORE --> BC
lengthOfStretch         = distOREToBC;
int[] speedLimitOREToBC= { lengthOfStretch };
stretchOREToBC          =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitOREToBC);

// BC --> KHS
lengthOfStretch         = distBCToKHS;
int[] speedLimitBCToKHS= { lengthOfStretch };
stretchBCToKHS          =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitBCToKHS);

// KHS --> UNI
lengthOfStretch         = distKHSToUNI;
int[] speedLimitKHSToUNI= { lengthOfStretch };
stretchKHSToUNI         =  new Stretch(dominatingSpeedLimit,
                                        lengthOfStretch,
                                        speedLimits,
                                        speedLimitKHSToUNI);

// UNI --> ISB
lengthOfStretch         = distUNIToISB;
int[] speedLimitUNIToISB= { lengthOfStretch };
stretchUNIToISB         =  new Stretch(dominatingSpeedLimit,
                                        lengthOfStretch,
                                        speedLimits,
                                        speedLimitUNIToISB);

// ISB --> KHC
lengthOfStretch         = distISBToKHC;
int[] speedLimitISBToKHC= { lengthOfStretch };
stretchISBToKHC         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitISBToKHC);

/* LGP part */

// LGP --> AMB
lengthOfStretch         = distLGPToAMB;
int[] speedLimitLGPToAMB= { lengthOfStretch };
stretchLGPToAMB         =  new Stretch(dominatingSpeedLimit,
                                       lengthOfStretch,
                                       speedLimits,
                                       speedLimitLGPToAMB);

// AMB --> KHC
lengthOfStretch         = distAMBToKHC;
```

```
    int[] speedLimitAMBToKHC= { lengthOfStretch };
    stretchAMBToKHC        =  new Stretch(dominatingSpeedLimit,
                                          lengthOfStretch,
                                          speedLimits,
                                          speedLimitAMBToKHC);

    /* Shared part */

    // KHC --> KGN
    lengthOfStretch         = distKHCToKGN;
    int[] speedLimitKHCToKGN= { lengthOfStretch };
    stretchKHCToKGN        =  new Stretch(dominatingSpeedLimit,
                                          lengthOfStretch,
                                          speedLimits,
                                          speedLimitKHCToKGN);

    // KGN --> KN
    lengthOfStretch        = distKGNToKN;
    int[] speedLimitKGNToKN= { lengthOfStretch };
    stretchKGNToKN        =  new Stretch(dominatingSpeedLimit,
                                         lengthOfStretch,
                                         speedLimits,
                                         speedLimitKGNToKN);

    // KN -->
    lengthOfStretch        = distOutOfKN;
    int[] speedLimitOutOfKN= { lengthOfStretch };
    stretchOutOfKN        =  new Stretch(dominatingSpeedLimit,
                                         lengthOfStretch,
                                         speedLimits,
                                         speedLimitOutOfKN);


}


//---------------------------------------------------------------------
// initTDConfigs:
//
// Initialises the TDs configuration for VR.
// Should be called before Central is used
//---------------------------------------------------------------------


static void initTDConfigs(int[] allowedPLsKN,
                          int[] allowedPLsKGN,
                          int[] allowedPLsKHC,
                          int[] allowedPLsISB,
                          int[] allowedPLsUNI,
                          int[] allowedPLsKHS,
                          int[] allowedPLsBC,
                          int[] allowedPLsORE,
                          int[] allowedPLsVEA,
                          int[] allowedPLsAMB,
                          int[] allowedPLsLGP,
                          double minDwellTimeKN,
                          double minDwellTimeKGN,
                          double minDwellTimeKHC,
                          double minDwellTimeISB,
                          double minDwellTimeUNI,
                          double minDwellTimeKHS,
                          double minDwellTimeBC,
                          double minDwellTimeORE,
                          double minDwellTimeVEA,
                          double minDwellTimeAMB,
                          double minDwellTimeLGP,
                          double maxDwellTimeKN,
                          double maxDwellTimeKGN,
                          double maxDwellTimeKHC,
                          double maxDwellTimeISB,
                          double maxDwellTimeUNI,
                          double maxDwellTimeKHS,
                          double maxDwellTimeBC,
                          double maxDwellTimeORE,
                          double maxDwellTimeVEA,
                          double maxDwellTimeAMB,
                          double maxDwellTimeLGP){


// Configuration of the performance levels VR
// is allowed to use

int numPLs;
int PLno;

/* Shared part */

// KN
numPLs = allowedPLsKN.length;
PLsKN = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsKN[i];
    PLsKN[i] = Central.getPerformanceLevel(PLno);
}


// KGN
numPLs = allowedPLsKGN.length;
PLsKGN = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsKGN[i];
    PLsKGN[i] = Central.getPerformanceLevel(PLno);
}


// KHC
numPLs = allowedPLsKHC.length;
PLsKHC = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsKHC[i];
    PLsKHC[i] = Central.getPerformanceLevel(PLno);
}


/* VEA part */

// ISB
numPLs = allowedPLsISB.length;
PLsISB = new PerformanceLevel[numPLs];
```

```
for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsISB[i];
    PLsISB[i] = Central.getPerformanceLevel(PLno);
}


// UNI
numPLs = allowedPLsUNI.length;
PLsUNI = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsUNI[i];
    PLsUNI[i] = Central.getPerformanceLevel(PLno);
}


// KHS
numPLs = allowedPLsKHS.length;
PLsKHS = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsKHS[i];
    PLsKHS[i] = Central.getPerformanceLevel(PLno);
}


// BC
numPLs = allowedPLsBC.length;
PLsBC = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsBC[i];
    PLsBC[i] = Central.getPerformanceLevel(PLno);
}


// ORE
numPLs = allowedPLsORE.length;
PLsORE = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsORE[i];
    PLsORE[i] = Central.getPerformanceLevel(PLno);
}


// VEA
numPLs = allowedPLsVEA.length;
PLsVEA = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsVEA[i];
    PLsVEA[i] = Central.getPerformanceLevel(PLno);
}

/* LGP part */

// AMB
numPLs = allowedPLsAMB.length;
PLsAMB = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsAMB[i];
    PLsAMB[i] = Central.getPerformanceLevel(PLno);
}


// LGP
numPLs = allowedPLsLGP.length;
PLsLGP = new PerformanceLevel[numPLs];

for(int i = 0; i < numPLs; i++){
    PLno = allowedPLsLGP[i];
    PLsLGP[i] = Central.getPerformanceLevel(PLno);
}



// Configuration of minimum dwell time VR is allowed to use
minDwellKN  = minDwellTimeKN;
minDwellKGN = minDwellTimeKGN;
minDwellKHC = minDwellTimeKHC;
minDwellISB = minDwellTimeISB;
minDwellUNI = minDwellTimeUNI;
minDwellKHS = minDwellTimeKHS;
minDwellBC  = minDwellTimeBC;
minDwellORE = minDwellTimeORE;
minDwellVEA = minDwellTimeVEA;
minDwellAMB = minDwellTimeAMB;
minDwellLGP = minDwellTimeLGP;

// Configuration of maximum dwell time VR is allowed to use
maxDwellKN  = maxDwellTimeKN;
maxDwellKGN = maxDwellTimeKGN;
maxDwellKHC = maxDwellTimeKHC;
maxDwellISB = maxDwellTimeISB;
maxDwellUNI = maxDwellTimeUNI;
maxDwellKHS = maxDwellTimeKHS;
maxDwellBC  = maxDwellTimeBC;
maxDwellORE = maxDwellTimeORE;
maxDwellVEA = maxDwellTimeVEA;
maxDwellAMB = maxDwellTimeAMB;
maxDwellLGP = maxDwellTimeLGP;


}



//---------------------------------------------------------------------
// initTopography:
//
// Initialises the topography.
// Should be called before Central is used
//---------------------------------------------------------------------


public static Hashtable initTopography(){

    Hashtable stationAndStretchInfos = new Hashtable();
    StationAndStretchInfo info;

    //----------
```

```
// Track 1
//----------

/* Shared part */

// KN
info = new StationAndStretchInfo(KN_STATION_1,
                                 minDwellKN,
                                 maxDwellKN,
                                 PLsKN,
                                 KGN_STATION_1,
                                 KGN_STATION_1,
                                 stretchKNToKGN,
                                 stretchKNToKGN,
                                 new Integer(2),
                                 new Integer(2),
                                 new Integer(8),
                                 new Integer(4));

stationAndStretchInfos.put(KN_STATION_1,info);


// KGN
info = new StationAndStretchInfo(KGN_STATION_1,
                                 minDwellKGN,
                                 maxDwellKGN,
                                 PLsKGN,
                                 KHC_STATION_1,
                                 KHC_STATION_1,
                                 stretchKGNToKHC,
                                 stretchKGNToKHC,
                                 new Integer(1),
                                 new Integer(1),
                                 new Integer(7),
                                 new Integer(3));

stationAndStretchInfos.put(KGN_STATION_1,info);



// KHC
info = new StationAndStretchInfo(KHC_STATION_1,
                                 minDwellKHC,
                                 maxDwellKHC,
                                 PLsKHC,
                                 ISB_STATION_1,
                                 AMB_STATION_1,
                                 stretchKHCToISB,
                                 stretchKHCToAMB,
                                 new Integer(0),
                                 new Integer(0),
                                 new Integer(6),
                                 new Integer(2));

stationAndStretchInfos.put(KHC_STATION_1,info);


/* VEA part */

// ISB
info = new StationAndStretchInfo(ISB_STATION_1,
                                 minDwellISB,
                                 maxDwellISB,
                                 PLsISB,
                                 UNI_STATION_1,
                                 null,
                                 stretchISBToUNI,
                                 null,
                                 new Integer(-1),
                                 null,
                                 new Integer(5),
                                 null);

stationAndStretchInfos.put(ISB_STATION_1,info);


// UNI
info = new StationAndStretchInfo(UNI_STATION_1,
                                 minDwellUNI,
                                 maxDwellUNI,
                                 PLsUNI,
                                 KHS_STATION_1,
                                 null,
                                 stretchUNIToKHS,
                                 null,
                                 new Integer(-2),
                                 null,
                                 new Integer(4),
                                 null);

stationAndStretchInfos.put(UNI_STATION_1,info);


// KHS
info = new StationAndStretchInfo(KHS_STATION_1,
                                 minDwellKHS,
                                 maxDwellKHS,
                                 PLsKHS,
                                 BC_STATION_1,
                                 null,
                                 stretchKHSToBC,
                                 null,
                                 new Integer(-3),
                                 null,
                                 new Integer(3),
                                 null);

stationAndStretchInfos.put(KHS_STATION_1,info);



// BC
info = new StationAndStretchInfo(BC_STATION_1,
                                 minDwellBC,
                                 maxDwellBC,
                                 PLsBC,
                                 ORE_STATION_1,
                                 null,
                                 stretchBCToORE,
                                 null,
                                 new Integer(-4),
                                 null,
                                 new Integer(2),
                                 null);
```

```
stationAndStretchInfos.put(BC_STATION_1,info);


// ORE
info = new StationAndStretchInfo(ORE_STATION_1,
                                 minDwellORE,
                                 maxDwellORE,
                                 PLsORE,
                                 VEA_STATION_1,
                                 null,
                                 stretchOREToVEA,
                                 null,
                                 new Integer(-5),
                                 null,
                                 new Integer(1),
                                 null);

stationAndStretchInfos.put(ORE_STATION_1,info);


// VEA
info = new StationAndStretchInfo(VEA_STATION_1,
                                 minDwellVEA,
                                 maxDwellVEA,
                                 PLsVEA,
                                 "NO",
                                 null,
                                 stretchOutOfVEA,
                                 null,
                                 new Integer(-6),
                                 null,
                                 new Integer(0),
                                 null);

stationAndStretchInfos.put(VEA_STATION_1,info);


/* LGP part */

// AMB
info = new StationAndStretchInfo(AMB_STATION_1,
                                 minDwellAMB,
                                 maxDwellAMB,
                                 PLsAMB,
                                 null,
                                 LGP_STATION_1,
                                 null,
                                 stretchAMBToLGP,
                                 null,
                                 new Integer(-1),
                                 null,
                                 new Integer(1));

stationAndStretchInfos.put(AMB_STATION_1,info);



// LGP
info = new StationAndStretchInfo(LGP_STATION_1,
                                 minDwellLGP,
                                 maxDwellLGP,
                                 PLsLGP,
                                 null,
                                 "NO",
                                 null,
                                 stretchOutOfLGP,
                                 null,
                                 new Integer(-2),
                                 null,
                                 new Integer(0));

stationAndStretchInfos.put(LGP_STATION_1,info);



//----------
// Track 2
//----------

/* VEA part */

// VEA
info = new StationAndStretchInfo(VEA_STATION_2,
                                 minDwellVEA,
                                 maxDwellVEA,
                                 PLsVEA,
                                 ORE_STATION_2,
                                 null,
                                 stretchVEAToORE,
                                 null,
                                 new Integer(5),
                                 null,
                                 new Integer(8),
                                 null);

stationAndStretchInfos.put(VEA_STATION_2,info);


// ORE
info = new StationAndStretchInfo(ORE_STATION_2,
                                 minDwellORE,
                                 maxDwellORE,
                                 PLsORE,
                                 BC_STATION_2,
                                 null,
                                 stretchOREToBC,
                                 null,
                                 new Integer(4),
                                 null,
                                 new Integer(7),
                                 null);

stationAndStretchInfos.put(ORE_STATION_2,info);


// BC
info = new StationAndStretchInfo(BC_STATION_2,
                                 minDwellBC,
                                 maxDwellBC,
                                 PLsBC,
                                 KHS_STATION_2,
                                 null,
                                 stretchBCToKHS,
```

```
                                null,
                                new Integer(3),
                                null,
                                new Integer(6),
                                null);

stationAndStretchInfos.put(BC_STATION_2,info);



// KHS
info = new StationAndStretchInfo(KHS_STATION_2,
                                minDwellKHS,
                                maxDwellKHS,
                                PLsKHS,
                                UNI_STATION_2,
                                null,
                                stretchKHSToUNI,
                                null,
                                new Integer(2),
                                null,
                                new Integer(5),
                                null);

stationAndStretchInfos.put(KHS_STATION_2,info);



// UNI
info = new StationAndStretchInfo(UNI_STATION_2,
                                minDwellUNI,
                                maxDwellUNI,
                                PLsUNI,
                                ISB_STATION_2,
                                null,
                                stretchUNIToISB,
                                null,
                                new Integer(1),
                                null,
                                new Integer(4),
                                null);

stationAndStretchInfos.put(UNI_STATION_2,info);



// ISB
info = new StationAndStretchInfo(ISB_STATION_2,
                                minDwellISB,
                                maxDwellISB,
                                PLsISB,
                                KHC_STATION_2,
                                null,
                                stretchISBToKHC,
                                null,
                                new Integer(0),
                                null,
                                new Integer(3),
                                null);

stationAndStretchInfos.put(ISB_STATION_2,info);
```



```
/* LGP part */

// LGP
info = new StationAndStretchInfo(LGP_STATION_2,
                                minDwellLGP,
                                maxDwellLGP,
                                PLsLGP,
                                null,
                                AMB_STATION_2,
                                null,
                                stretchLGPToAMB,
                                null,
                                new Integer(1),
                                null,
                                new Integer(4));

stationAndStretchInfos.put(LGP_STATION_2,info);


// AMB
info = new StationAndStretchInfo(AMB_STATION_2,
                                minDwellAMB,
                                maxDwellAMB,
                                PLsAMB,
                                null,
                                KHC_STATION_2,
                                null,
                                stretchAMBToKHC,
                                null,
                                new Integer(0),
                                null,
                                new Integer(3));

stationAndStretchInfos.put(AMB_STATION_2,info);



/* Shared part */

// KHC
info = new StationAndStretchInfo(KHC_STATION_2,
                                minDwellKHC,
                                maxDwellKHC,
                                PLsKHC,
                                KGN_STATION_2,
                                KGN_STATION_2,
                                stretchKHCToKGN,
                                stretchKHCToKGN,
                                new Integer(-1),
                                new Integer(-1),
                                new Integer(2),
                                new Integer(2));

stationAndStretchInfos.put(KHC_STATION_2,info);


// KGN
info = new StationAndStretchInfo(KGN_STATION_2,
                                minDwellKGN,
                                maxDwellKGN,
```

```
                                    PLsKGN,
                                    KN_STATION_2,
                                    KN_STATION_2,
                                    stretchKGNToKN,
                                    stretchKGNToKN,
                                    new Integer(-2),
                                    new Integer(-2),
                                    new Integer(1),
                                    new Integer(1));

    stationAndStretchInfos.put(KGN_STATION_2,info);



    // KN
    info = new StationAndStretchInfo(KN_STATION_2,
                                    minDwellKN,
                                    maxDwellKN,
                                    PLsKN,
                                    "NO",
                                    "NO",
                                    stretchOutOfKN,
                                    stretchOutOfKN,
                                    new Integer(-3),
                                    new Integer(-3),
                                    new Integer(0),
                                    new Integer(0));

    stationAndStretchInfos.put(KN_STATION_2, info);

    return stationAndStretchInfos;

}



//-------------------------------------------------------------------
// init:
//
// Initialises Central with TDs configuration of VR and a
// predefined HASTUS plan
//-------------------------------------------------------------------

public static void init(int[] allowedPLsKN,
                        int[] allowedPLsKGN,
                        int[] allowedPLsKHC,
                        int[] allowedPLsISB,
                        int[] allowedPLsUNI,
                        int[] allowedPLsKHS,
                        int[] allowedPLsBC,
                        int[] allowedPLsORE,
                        int[] allowedPLsVEA,
                        int[] allowedPLsAMB,
                        int[] allowedPLsLGP,
                        double minDwellTimeKN,
                        double minDwellTimeKGN,
                        double minDwellTimeKHC,
                        double minDwellTimeISB,
                        double minDwellTimeUNI,
                        double minDwellTimeKHS,
                        double minDwellTimeBC,
                        double minDwellTimeORE,
                        double minDwellTimeVEA,
                        double minDwellTimeAMB,
                        double minDwellTimeLGP,
                        double maxDwellTimeKN,
                        double maxDwellTimeKGN,
                        double maxDwellTimeKHC,
                        double maxDwellTimeISB,
                        double maxDwellTimeUNI,
                        double maxDwellTimeKHS,
                        double maxDwellTimeBC,
                        double maxDwellTimeORE,
                        double maxDwellTimeVEA,
                        double maxDwellTimeAMB,
                        double maxDwellTimeLGP,
                        HASTUSTrainPlan[] hastusPlan){


//-------------------------------------------------------------------
// Initialisation of the TDs configuration of VR
//-------------------------------------------------------------------

// Configuration of which performance levels VR
// is allowed to use

initTDConfigs(allowedPLsKN,
              allowedPLsKGN,
              allowedPLsKHC,
              allowedPLsISB,
              allowedPLsUNI,
              allowedPLsKHS,
              allowedPLsBC,
              allowedPLsORE,
              allowedPLsVEA,
              allowedPLsAMB,
              allowedPLsLGP,
              minDwellTimeKN,
              minDwellTimeKGN,
              minDwellTimeKHC,
              minDwellTimeISB,
              minDwellTimeUNI,
              minDwellTimeKHS,
              minDwellTimeBC,
              minDwellTimeORE,
              minDwellTimeVEA,
              minDwellTimeAMB,
              minDwellTimeLGP,
              maxDwellTimeKN,
              maxDwellTimeKGN,
              maxDwellTimeKHC,
              maxDwellTimeISB,
              maxDwellTimeUNI,
              maxDwellTimeKHS,
              maxDwellTimeBC,
              maxDwellTimeORE,
              maxDwellTimeVEA,
              maxDwellTimeAMB,
              maxDwellTimeLGP);



//-------------------------------------------------------------------
```

```
    // Initialisation of controlline related info
    //-----------------------------------------------------------------

    initSpeedRestrictions();


    //-----------------------------------------------------------------
    // Initialisation of topography info
    //-----------------------------------------------------------------

    Hashtable info = initTopography();


    //-----------------------------------------------------------------
    // Initialisation of stationAndStretchInfo at Central
    //-----------------------------------------------------------------

    Central.initTopographySpeedRestrictionAndTDconfig(info);


    //-----------------------------------------------------------------
    // Initialisation of the HASTUS plan
    //-----------------------------------------------------------------


    Central.initHASTUSPlan(hastusPlan);


}

//-----------------------------------------------------------------
// init:
//
// Initialises Central with TDs configuration of VR and parameters
// for HASTUS to make a HASTUS plan.
//-----------------------------------------------------------------


public static void init(int[] allowedPLsKN,
                        int[] allowedPLsKGN,
                        int[] allowedPLsKHC,
                        int[] allowedPLsISB,
                        int[] allowedPLsUNI,
                        int[] allowedPLsKHS,
                        int[] allowedPLsBC,
                        int[] allowedPLsORE,
                        int[] allowedPLsVEA,
                        int[] allowedPLsAMB,
                        int[] allowedPLsLGP,
                        double minDwellTimeKN,
                        double minDwellTimeKGN,
                        double minDwellTimeKHC,
                        double minDwellTimeISB,
                        double minDwellTimeUNI,
                        double minDwellTimeKHS,
                        double minDwellTimeBC,
                        double minDwellTimeORE,
                        double minDwellTimeVEA,
                        double minDwellTimeAMB,
                        double minDwellTimeLGP,
                        double maxDwellTimeKN,
                        double maxDwellTimeKGN,
                        double maxDwellTimeKHC,
                        double maxDwellTimeISB,
                        double maxDwellTimeUNI,
                        double maxDwellTimeKHS,
                        double maxDwellTimeBC,
                        double maxDwellTimeORE,
                        double maxDwellTimeVEA,
                        double maxDwellTimeAMB,
                        double maxDwellTimeLGP,
                        int[] trainsToUseInHASTUS,
                        int[][] DIDsToUseInHASTUS,
                        double[][] timeBetweenDIDsToUseInHASTUS,
                        int performanceLevelToUseInHASTUS,
                        double dwellTimeToUseInHASTUS,
                        double[] startTimesToUseInHASTUS){



//-----------------------------------------------------------------
// Initialisation of info at Central
//-----------------------------------------------------------------

init(allowedPLsKN,
     allowedPLsKGN,
     allowedPLsKHC,
     allowedPLsISB,
     allowedPLsUNI,
     allowedPLsKHS,
     allowedPLsBC,
     allowedPLsORE,
     allowedPLsVEA,
     allowedPLsAMB,
     allowedPLsLGP,
     minDwellTimeKN,
     minDwellTimeKGN,
     minDwellTimeKHC,
     minDwellTimeISB,
     minDwellTimeUNI,
     minDwellTimeKHS,
     minDwellTimeBC,
     minDwellTimeORE,
     minDwellTimeVEA,
     minDwellTimeAMB,
     minDwellTimeLGP,
     maxDwellTimeKN,
     maxDwellTimeKGN,
     maxDwellTimeKHC,
     maxDwellTimeISB,
     maxDwellTimeUNI,
     maxDwellTimeKHS,
     maxDwellTimeBC,
     maxDwellTimeORE,
     maxDwellTimeVEA,
     maxDwellTimeAMB,
     maxDwellTimeLGP,
     null);

//-----------------------------------------------------------------
// Initialisation of the HASTUS plan
//-----------------------------------------------------------------
```

```
        /* Generation of a HASTUS */

        HASTUSTrainPlan[] hastusPlan
            = HASTUS.genHastusPlan(trainsToUseInHASTUS,
                                   DIDsToUseInHASTUS,
                                   timeBetweenDIDsToUseInHASTUS,
                                   performanceLevelToUseInHASTUS,
                                   dwellTimeToUseInHASTUS,
                                   startTimesToUseInHASTUS);

        Central.initHASTUSPlan(hastusPlan);

    }

}
```

StationAndStretchInfo.java

```
//=============================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "StationAndStretchInfo.java"
//=============================================================================

//=============================================================================
// The StationAndStretch class definition
//=============================================================================

class StationAndStretchInfo{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    // Info about station
    private String thisStation;
    private double minDwellTime;
    private double maxDwellTime;
    private PerformanceLevel[] allowedPLs;

    // Info about stretch to the following station
    private String followingStationVEA;
    private String followingStationLGP;
    private Stretch stretchToFollowingStationVEA;
    private Stretch stretchToFollowingStationLGP;

    // Info about the stations location on the permanent way
    private Integer stationsToBifurcationVEA;
    private Integer stationsToBifurcationLGP;
    private Integer stationsToEndVEA;
    private Integer stationsToEndLGP;
```

```
    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Constructor                                                       */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    public StationAndStretchInfo(String thisStation,
                                 double minDwellTime,
                                 double maxDwellTime,
                                 PerformanceLevel[] allowedPLs,
                                 String followingStationVEA,
                                 String followingStationLGP,
                                 Stretch stretchToFollowingStationVEA,
                                 Stretch stretchToFollowingStationLGP,
                                 Integer stationsToBifurcationVEA,
                                 Integer stationsToBifurcationLGP,
                                 Integer stationsToEndVEA,
                                 Integer stationsToEndLGP){


        this.thisStation = thisStation;
        this.minDwellTime = minDwellTime;
        this.maxDwellTime = maxDwellTime;
        this.allowedPLs = allowedPLs;
        this.followingStationVEA = followingStationVEA;
        this.followingStationLGP = followingStationLGP;
        this.stretchToFollowingStationVEA = stretchToFollowingStationVEA;
        this.stretchToFollowingStationLGP = stretchToFollowingStationLGP;
        this.stationsToBifurcationVEA = stationsToBifurcationVEA;
        this.stationsToBifurcationLGP = stationsToBifurcationLGP;

    }


    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Accessors                                                         */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    //-------------------------------------------------------------------
    // getThisStation:
    //
    // Returns the station track circuit name
    //-------------------------------------------------------------------

    public String getThisStation(){
        return thisStation;
    }



    //-------------------------------------------------------------------
    // getMinDwellTime:
    //
    // Returns the minimal number of seconds that VR is allowed use on
    // that station track circuit
    //-------------------------------------------------------------------

    public double getMinDwellTime(){
        return minDwellTime;
```

```
}



//---------------------------------------------------------------------
// getMaxDwell:
//
// Returns the maximum number of seconds that VR is allowed use on
// that station track circuit
//---------------------------------------------------------------------

public double getMaxDwellTime(){
    return maxDwellTime;
}



//---------------------------------------------------------------------
// getAllowedPLs:
//
// Returns an array with the performance levels that VR is allowed
// use on the stretch following that station track circuit
//---------------------------------------------------------------------

public PerformanceLevel[] getAllowedPs(){
    return allowedPLs;
}



//---------------------------------------------------------------------
// getFollowingStation:
//
// Returns the name of the following station track circuit.
//---------------------------------------------------------------------

public String getFollowingStation(String bifurcationSetting){
    if(bifurcationSetting.startsWith("VEA")||
       bifurcationSetting.endsWith("VEA")){
        return followingStationVEA;
    }else{
        return followingStationLGP;
    }
}



//---------------------------------------------------------------------
// getStretchToFollowingStation:
//
// Returns the speed zones of the stretch to the next station
//---------------------------------------------------------------------

public Stretch getStretchToFollowingStation(String bifurcationSetting){
    if(bifurcationSetting.startsWith("VEA")||
       bifurcationSetting.endsWith("VEA")){
        return stretchToFollowingStationVEA;
    }else{
        return stretchToFollowingStationLGP;
    }
}
```



```
    //---------------------------------------------------------------------
    // getStationsToBifurcation:
    //
    // Returns the number of stations before the bifurcation
    // If this station is after the bifurcation the number is negative
    //---------------------------------------------------------------------

    public Integer getStationsToBifurcation(String bifurcationSetting){
        if(bifurcationSetting.startsWith("VEA")||
           bifurcationSetting.endsWith("VEA")){
            return stationsToBifurcationVEA;
        }else{
            return stationsToBifurcationLGP;
        }
    }



    //---------------------------------------------------------------------
    // getStationsToEnd:
    //
    // Returns the number of stations before the end of the track is
    // reached
    //---------------------------------------------------------------------

    public Integer getStationsToEnd(String bifurcationSetting){
        if(bifurcationSetting.startsWith("VEA")||
           bifurcationSetting.endsWith("VEA")){
            return stationsToEndVEA;
        }else{
            return stationsToEndLGP;
        }
    }


}
```

Stretch.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Stretch.java"
//=====================================================================

//=====================================================================
// The Stretch class definition
//=====================================================================

class Stretch{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/
```

```
// The dominatin maximum speed allowed on the stretch
private int dominatingSpeedLimit;

// The length of the entire stretch
private int totalLength;

// Info in the speed zones within the stretch
private SpeedZone[] speedZones;



/*------------------------------------------------------------------*/
/*                                                                  */
/* Constructor                                                      */
/*                                                                  */
/*------------------------------------------------------------------*/

Stretch(int dominatingSpeedLimit,
        int totalLength,
        int[] speedLimits,
        int[] zoneLengths){

    this.dominatingSpeedLimit = dominatingSpeedLimit;
    this.totalLength = totalLength;
    int numZones = speedLimits.length;
    speedZones = new SpeedZone[numZones];

    for(int i = 0; i < numZones; i++){
        speedZones[i] = new SpeedZone(speedLimits[i], zoneLengths[i]);
    }
}



/*------------------------------------------------------------------*/
/*                                                                  */
/* Accessors                                                        */
/*                                                                  */
/*------------------------------------------------------------------*/

//------------------------------------------------------------------
// getDominatingSpeedLimit:
//
// Returns the dominatingSpeedLimit
//------------------------------------------------------------------

public int getDominatingSpeedLimit(){
    return dominatingSpeedLimit;
}



//------------------------------------------------------------------
// getTotalLenght:
//
// Returns the total length of the stretch.
//------------------------------------------------------------------

public int getTotalLength(){
    return totalLength;
}
```



```
    //------------------------------------------------------------------
    // getSpeedZones:
    //
    // Returns the speed zones of the stretch.
    //------------------------------------------------------------------

    public SpeedZone[] getSpeedZones(){
        return speedZones;
    }


}
```

SpeedZone.java

```
//======================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "SpeedZone.java"
//======================================================================

//======================================================================
// The SpeedZone class definition
//======================================================================

class SpeedZone{

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Attributes                                                       */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    // Maximum speed in zone
    private int speedLimit;

    // Length of zone
    private int zoneLength;



    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Constructor                                                      */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    public SpeedZone(int speedLimit, int zoneLength){
        this.speedLimit = speedLimit;
        this.zoneLength = zoneLength;
    }



    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Accessors                                                        */
    /*                                                                  */
    /*------------------------------------------------------------------*/
```

```
    //-------------------------------------------------------------------
    // getSpeedLimit:
    //
    // Returns the speedLimit of the speed zone
    //-------------------------------------------------------------------

    public int getSpeedLimit(){
        return speedLimit;
    }


    //-------------------------------------------------------------------
    // getZoneLength:
    //
    // Returns the length of the zone in meters
    //-------------------------------------------------------------------

    public int getZoneLength(){
        return zoneLength;
    }


}
```

B.7 DID

DID.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID.java"
//=====================================================================

//=====================================================================
// The DID class definition
//=====================================================================

abstract class DID{

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Attributes                                                      */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    //-----------------------------------------------------------------
    // Info about the DID
    //-----------------------------------------------------------------

    // DID identification number
    int DIDno;

    // Info about the stations on the route
    StationStopInfo[] route;

    // Info about the stations to be stopped at on the route
    StationStopInfo[] stationStoppingTable;

    // Info about how the route traverses the bifurcation
    String bifurcationSetting;

    // Info used for iterating the DID
    int stationsPassed;
    int stationsStoppedAt;
    int numStations;
    int numStops;

    //-----------------------------------------------------------------
    // Station track circuits (to ease the making of new DIDs)
    //-----------------------------------------------------------------

    //---------
    // Track 1
    //---------

    /* Shared part */
    final String KN_STATION_1  = "KN1-2T";
    final String KGN_STATION_1 = "KGN1-3T";
    final String KHC_STATION_1 = "KHC1-2T";

    /* VEA part */
    final String ISB_STATION_1 = "ISB1-3T";
    final String UNI_STATION_1 = "UNI1-2T";
    final String KHS_STATION_1 = "KHS1-3T";
    final String BC_STATION_1  = "KHS1-9T";
    final String ORE_STATION_1 = "ORE1-7T";
    final String VEA_STATION_1 = "VEA1-7T";

    /* LGP part */
    final String AMB_STATION_1 = "LGP1-3T";
    final String LGP_STATION_1 = "LGP1-11T";

    //---------
    // Track 2
    //---------

    /* VEA part */
    final String VEA_STATION_2 = "VEA2-8T";
    final String ORE_STATION_2 = "ORE2-7T";
    final String BC_STATION_2  = "KHS2-10T";
    final String KHS_STATION_2 = "KHS2-3T";
    final String UNI_STATION_2 = "UNI2-2T";
    final String ISB_STATION_2 = "ISB2-3T";

    /* LGP part */
    final String LGP_STATION_2 = "LGP2-11T";
    final String AMB_STATION_2 = "LGP2-3T";

    /* Shared part */
    final String KHC_STATION_2  = "KHC2-2T";
    final String KGN_STATION_2  = "KGN2-3T";
    final String KN_STATION_2   = "KN2-2T";



    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Accessors                                                       */
    /*                                                                 */
    /*-----------------------------------------------------------------*/
```

```
//-----------------------------------------------------------------
// getDIDno:
//
// Returns the DID's identification number
//-----------------------------------------------------------------

public int getDIDno(){
    return DIDno;
}


//-----------------------------------------------------------------
// getStationsLeft:
//
// Returns the number of station on the route that has not yet
// been passed
//-----------------------------------------------------------------

public int getStationsLeft(){
    return (numStations - stationsStoppedAt);
}


//-----------------------------------------------------------------
// getNextStation:
//
// Returns info about the next station on the route
//-----------------------------------------------------------------

public StationStopInfo getNextStation(){

    StationStopInfo nextStation = null;

    if(numStations > stationsPassed){
        nextStation = route[stationsPassed];
    }
    return nextStation;
}

//-----------------------------------------------------------------
// getNextStations:
//
// Returns info about the next 'numStations' stations on the route
//-----------------------------------------------------------------

public StationStopInfo[] getNextStations(int numStations){

    StationStopInfo[] nextStations = null;
    int actualStations = Math.min(getStationsLeft(), numStations);

    if(actualStations > 0){
        nextStations = new StationStopInfo[actualStations];
        for(int i=0, j=stationsPassed; i < actualStations; i++,j++){
            nextStations[i] = route[j];
        }
    }

    return nextStations;
}
```



```
//-----------------------------------------------------------------
// getStopsLeft:
//
// Returns the number of stops on the route that has not yet
// been stopped at
//-----------------------------------------------------------------

public int getStopsLeft(){
    return (numStops - stationsStoppedAt);
}



//-----------------------------------------------------------------
// getNextStop:
//
// Returns info about the next stop on the route.
// While on a station next stop is the current station.
//-----------------------------------------------------------------

public StationStopInfo getNextStop(){

    StationStopInfo nextStop = null;

    if(numStops > stationsStoppedAt){
        nextStop = stationStoppingTable[stationsStoppedAt];
    }

    return nextStop;
}


//-----------------------------------------------------------------
// getNextStops:
//
// Returns info about the next 'numStops' stops on the route
//-----------------------------------------------------------------

public StationStopInfo[] getNextStops(int numStops){

    StationStopInfo[] nextStops = null;
    int actualStops = Math.min(getStopsLeft(), numStops);

    if(actualStops > 0){
        nextStops = new StationStopInfo[actualStops];
        for(int i=0, j=stationsStoppedAt; i < actualStops; i++,j++){
            nextStops[i] = stationStoppingTable[j];
        }
    }
    return nextStops;
}


//-----------------------------------------------------------------
// getBifurcationSetting:
//
// Returns info about how the route traverses the bifurcation
//-----------------------------------------------------------------

public String getBifurcationSetting(){
```

```
        return bifurcationSetting;
    }


    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Mutators                                                         */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    //------------------------------------------------------------------
    // stationPassed:
    //
    // Moves the stationiterator to next station on the route
    //------------------------------------------------------------------

    void stationPassed(){
        stationsPassed++;
    }

    //------------------------------------------------------------------
    // stationStoppedAt:
    //
    // Moves the stopiterator to next stop
    //------------------------------------------------------------------

    void stationStoppedAt(){
        stationsStoppedAt++;
    }
}
```


DID1.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID1.java"
//=========================================================================

//=========================================================================
// The DID1 class definition
//=========================================================================

class DID1 extends DID{

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Constructor                                                      */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    DID1(){

     // Initialisation of identification number for the DID
     DIDno = 1;

     // Initialisation of the stations on the route
     StationStopInfo KN1  = new StationStopInfo(KN_STATION_1, true,1);
     StationStopInfo KGN1 = new StationStopInfo(KGN_STATION_1,true,1);
     StationStopInfo KHC1 = new StationStopInfo(KHC_STATION_1,true,1);
     StationStopInfo ISB1 = new StationStopInfo(ISB_STATION_1,true,1);
     StationStopInfo UNI1 = new StationStopInfo(UNI_STATION_1,true,1);
     StationStopInfo KHS1 = new StationStopInfo(KHS_STATION_1,true,1);
     StationStopInfo BC1  = new StationStopInfo(BC_STATION_1, true,1);
     StationStopInfo ORE1 = new StationStopInfo(ORE_STATION_1,true,1);
     StationStopInfo VEA1 = new StationStopInfo(VEA_STATION_1,true,1);

     // Initialisation of the route: the series of stations KN --> VEA
     StationStopInfo[] tmp1 = {KN1, KGN1, KHC1, ISB1, UNI1,
                               KHS1, BC1, ORE1, VEA1};
     route = tmp1;

     // Initialisation of the stationStoppingTable: as the entire route
     // hence all stations is to be stopped at
     stationStoppingTable = route;

     // Initialisation of the bifurcationSetting
     bifurcationSetting = "KNToVEA";

     // Initialisation of iterator info
     stationsPassed = 0;
     stationsStoppedAt = 0;
     numStations = route.length;
     numStops = stationStoppingTable.length;

    }

}
```

DID2.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID2.java"
//=========================================================================

//=========================================================================
// The DID2 class definition
//=========================================================================

class DID2 extends DID{

    /*------------------------------------------------------------------*/
    /*                                                                  */
    /* Constructor                                                      */
    /*                                                                  */
    /*------------------------------------------------------------------*/

    DID2(){

      // Initialisation of identification number for the DID
      DIDno = 2;

      // Initialisation of the stations on the route
      StationStopInfo KN1  = new StationStopInfo(KN_STATION_1, true,1);
      StationStopInfo KGN1 = new StationStopInfo(KGN_STATION_1,true,1);
      StationStopInfo KHC1 = new StationStopInfo(KHC_STATION_1,true,1);
      StationStopInfo AMB1 = new StationStopInfo(AMB_STATION_1,true,1);
      StationStopInfo LGP1 = new StationStopInfo(LGP_STATION_1,true,1);

      // Initialisation of the route: the series of stations KN --> LGP
      StationStopInfo[] tmp1 = {KN1, KGN1, KHC1, AMB1, LGP1};
```

```
      route = tmp1;

      // Initialisation of the stationStoppingTable as the entire
      // route, hence all stations is to be stopped at
      stationStoppingTable = route;

      // Initialisation of the bifurcationSetting
      bifurcationSetting = "KNToLGP";

      // Initialisation of iterator info
      stationsPassed = 0;
      stationsStoppedAt = 0;
      numStations = route.length;
      numStops = stationStoppingTable.length;

    }

}
```

DID3.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID3.java"
//=========================================================================

//=========================================================================
// The DID3 class definition
//=========================================================================

class DID3 extends DID{

    /*---------------------------------------------------------------------*/
    /*                                                                     */
    /* Constructor                                                         */
    /*                                                                     */
    /*---------------------------------------------------------------------*/

    DID3(){

     // Initialisation of identification number for the DID
     DIDno = 3;

     // Initialisation of the stations on the route
     StationStopInfo LGP2 = new StationStopInfo(LGP_STATION_2,true,1);
     StationStopInfo AMB2 = new StationStopInfo(AMB_STATION_2,true,1);
     StationStopInfo KHC2 = new StationStopInfo(KHC_STATION_2,true,1);
     StationStopInfo KGN2 = new StationStopInfo(KGN_STATION_2,true,1);
     StationStopInfo KN2  = new StationStopInfo(KN_STATION_2, true,1);

     // Initialisation of the route: the series of stations KN --> VEA
     StationStopInfo[] tmp1 = {LGP2, AMB2, KHC2, KGN2, KN2};
     route = tmp1;

     // Initialisation of the stationStoppingTable: as the entire route
     // hence all stations is to be stopped at
     stationStoppingTable = route;

     // Initialisation of the bifurcationSetting
     bifurcationSetting = "LGPToKN";

     // Initialisation of iterator info
     stationsPassed = 0;
     stationsStoppedAt = 0;
     numStations = route.length;
     numStops = stationStoppingTable.length;

    }

}
```

DID4.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID4.java"
//=========================================================================

//=========================================================================
// The DID4 class definition
//=========================================================================

class DID4 extends DID{

    /*---------------------------------------------------------------------*/
    /*                                                                     */
    /* Constructor                                                         */
    /*                                                                     */
    /*---------------------------------------------------------------------*/

    DID4(){

     // Initialisation of identification number for the DID
     DIDno = 4;

     // Initialisation of the stations on the route
     StationStopInfo VEA2 = new StationStopInfo(VEA_STATION_2,true,1);
     StationStopInfo ORE2 = new StationStopInfo(ORE_STATION_2,true,1);
     StationStopInfo BC2  = new StationStopInfo(BC_STATION_2, true,1);
     StationStopInfo KHS2 = new StationStopInfo(KHS_STATION_2,true,1);
     StationStopInfo UNI2 = new StationStopInfo(UNI_STATION_2,true,1);
     StationStopInfo ISB2 = new StationStopInfo(ISB_STATION_2,true,1);
     StationStopInfo KHC2 = new StationStopInfo(KHC_STATION_2,true,1);
     StationStopInfo KGN2 = new StationStopInfo(KGN_STATION_2,true,1);
     StationStopInfo KN2  = new StationStopInfo(KN_STATION_2, true,1);

     // Initialisation of the route: the series of stations VEA-->KN
     StationStopInfo[] tmp1 = {VEA2, ORE2, BC2, KHS2, UNI2,
                               ISB2, KHC2, KGN2, KN2};
     route = tmp1;

     // Initialisation of the stationStoppingTable: as the entire route
     // hence all stations is to be stopped at
     stationStoppingTable = route;

     // Initialisation of the bifurcationSetting
     bifurcationSetting = "VEAToKN";

     // Initialisation of iterator info
     stationsPassed = 0;
     stationsStoppedAt = 0;
```

```
     numStations = route.length;
     numStops = stationStoppingTable.length;

    }

}
```

DID5.java

```
//=============================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "DID5.java"
//=============================================================================

//=============================================================================
// The DID5 class definition
//=============================================================================

class DID5 extends DID{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Constructor                                                       */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    DID5(){

     // Initialisation of identification number for the DID
     DIDno = 5;

     // Initialisation of the stations on the route
     StationStopInfo VEA2 = new StationStopInfo(VEA_STATION_2,true, 1);
     StationStopInfo ORE2 = new StationStopInfo(ORE_STATION_2,true, 1);
     StationStopInfo BC2  = new StationStopInfo(BC_STATION_2, true, 4);
     StationStopInfo KHS2 = new StationStopInfo(KHS_STATION_2,false,3);
     StationStopInfo UNI2 = new StationStopInfo(UNI_STATION_2,false,2);
     StationStopInfo ISB2 = new StationStopInfo(ISB_STATION_2,false,1);
     StationStopInfo KHC2 = new StationStopInfo(KHC_STATION_2,true, 1);
     StationStopInfo KGN2 = new StationStopInfo(KGN_STATION_2,true, 1);
     StationStopInfo KN2  = new StationStopInfo(KN_STATION_2, true, 1);

     // Initialisation of the route: the series of stations KN --> VEA
     StationStopInfo[] tmp1 = {VEA2, ORE2, BC2, KHS2, UNI2, ISB2,
                               KHC2, KGN2, KN2};

     route = tmp1;

     // Initialisation of the stationStoppingTable: as the route
     // minus KHS, UNI, ISB
     StationStopInfo[] tmp2 = {VEA2, ORE2, BC2, KHC2, KGN2, KN2};

     stationStoppingTable = tmp2;

     // Initialisation of the bifurcationSetting
     bifurcationSetting = "VEAToKN";

     // Initialisation of iterator info
     stationsPassed = 0;
     stationsStoppedAt = 0;
     numStations = route.length;
     numStops = stationStoppingTable.length;

    }

}
```

StationStopInfo.java

```
//=============================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "StationStopInfo.java"
//=============================================================================

//=============================================================================
// The StationStopInfo class definition
//=============================================================================

class StationStopInfo{

    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Attributes                                                        */
    /*                                                                   */
    /*-------------------------------------------------------------------*/

    // Info about the station's track circuit name
    String stationTc;

    // Info about whether the station is a stopping location
    boolean stop;

    // Info about the number of stations on the route till the next
    // stop (including this station)
    int stationsToNextStop;



    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Constructor                                                       */
    /*                                                                   *
    /*-------------------------------------------------------------------*/

    StationStopInfo(String stationTc,
                    boolean stop,
                    int stationsToNextStop){

        this.stationTc = stationTc;
        this.stop = stop;
        this.stationsToNextStop = stationsToNextStop;
    }



    /*-------------------------------------------------------------------*/
    /*                                                                   */
    /* Accessors                                                         */
    /*                                                                   */
    /*-------------------------------------------------------------------*/
```

```
    //-------------------------------------------------------------------
    // getStation:
    //
    // Returns the station track circuit name
    //-------------------------------------------------------------------

    public String getStation(){
        return stationTc;
    }



    //-------------------------------------------------------------------
    // isStop:
    //
    // Returns whether the station is to be stopped at
    //-------------------------------------------------------------------

    public boolean isStop(){
        return stop;
    }



    //-------------------------------------------------------------------
    // getStationsToNextStop:
    //
    // Returns the number of stations on the route till the next
    // stop (including this station)
    //-------------------------------------------------------------------

    public int getStationsToNextStop(){
        return stationsToNextStop;
    }


}
```

B.8 Performance level

PerformanceLevel.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "PerformanceLevel.java"
//=====================================================================

//=====================================================================
// The PerformanceLevel class definition
//=====================================================================

class PerformanceLevel{

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Attributes                                                      */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

private int level;
private int speedLimit;
private double accelerationRate;
private double decelerationRate;



/*-------------------------------------------------------------------*/
/*                                                                   */
/* Constructor                                                       */
/*                                                                   */
/*-------------------------------------------------------------------*/

PerformanceLevel(int level,
                 int speedLimit,
                 double accelerationRate,
                 double decelerationRate){

    this.level = level;
    this.speedLimit = speedLimit;
    this.accelerationRate = accelerationRate;
    this.decelerationRate = decelerationRate;
}



/*-------------------------------------------------------------------*/
/*                                                                   */
/* Accessors                                                         */
/*                                                                   */
/*-------------------------------------------------------------------*/

//-------------------------------------------------------------------
// getLevel:
//
// Returns the level.
//-------------------------------------------------------------------

public int getLevel(){
    return level;
}


//-------------------------------------------------------------------
// getSpeedLimit:
//
// Returns the speedLimit.
//-------------------------------------------------------------------

public int getSpeedLimit(){
    return speedLimit;
}


//-------------------------------------------------------------------
// getAccelerationRate:
//
// Returns the accerationRate.
//-------------------------------------------------------------------

public double getAccelerationRate(){
    return accelerationRate;
```

```
    }


    //-------------------------------------------------------------------
    // getDecelerationRate:
    //
    // Returns the decelerationRate.
    //-------------------------------------------------------------------

    public double getDecelerationRate(){
        return decelerationRate;
    }


}
```

B.9 Hastus

HASTUS.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "HASTUS.java"
//=====================================================================

//=====================================================================
// Imported files
//=====================================================================

import java.util.*;


//=====================================================================
// The HASTUS class definition
//=====================================================================

class HASTUS{

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Methods                                                         */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    //-----------------------------------------------------------------
    // genHastusTrainPlan:
    //
    // Generates a hastus plan for the specified train
    //-----------------------------------------------------------------

    private static HASTUSTrainPlan genHastusTrainPlan(int trainId,
                                                      int[] DIDs,
                                                      double[] timeBetweenDIDs,
                                                      PerformanceLevel PL,
                                                      double dwellTime,
                                                      double startTime){

        /*-- Local variables --*/

        // Info about stations
        StationStopInfo currentStop;
        String currentStation;
        StationStopInfo nextStop;

        // Info about stretches
        StationAndStretchInfo[] stretchInfos;
        StationAndStretchInfo stretchInfo;
        Stretch speedInfo;

        // DID info
        DID did;
        int numDIDs = DIDs.length;
        int numEntries;
        int stationsToNextStop;
        String currentBf;

        // HASTUS plan info
        ArrayList hastusEntries = new ArrayList();
        HASTUSPlanEntry planEntry;
        double runTime;
        double arrivalTime = startTime;
        double departureTime = arrivalTime + dwellTime;


        // For each DID the following is done:
        for(int i = 0; i < numDIDs ; i++){

            // Get info about the DID
            did = Central.getDID(DIDs[i]);
            numEntries = did.getStopsLeft();
            currentBf = did.getBifurcationSetting();

            // Make plan for the first station
            currentStop    = did.getNextStop();
            currentStation = currentStop.getStation();

            stationsToNextStop = currentStop.getStationsToNextStop()+1;

            stretchInfos =
                Central.getStationAndStretchInfos(stationsToNextStop,
                                                  currentStation,
                                                  currentBf);

            runTime = Calc.estimatedTimeOfTravel(PL,
                                                 stretchInfos,
                                                 currentBf);


            // Register this plan
            planEntry = new HASTUSPlanEntry(DIDs[i],
                                            currentStation,
                                            arrivalTime,
                                            departureTime,
                                            runTime,
                                            PL);

            hastusEntries.add(planEntry);

            // Make plan for the rest of the stops minus 1
            for(int j = 1; j < numEntries - 1; j++){
                did.stationStoppedAt();
```

```
            currentStop = did.getNextStop();
            currentStation = currentStop.getStation();
            arrivalTime = departureTime + runTime;
            departureTime = arrivalTime + dwellTime;

            stationsToNextStop = currentStop.getStationsToNextStop()+1;

            stretchInfos =
                Central.getStationAndStretchInfos(stationsToNextStop,
                                                  currentStation,
                                                  currentBf);

            runTime = Calc.estimatedTimeOfTravel(PL,
                                                 stretchInfos,
                                                 currentBf);

            // Register plans
            planEntry = new HASTUSPlanEntry(DIDs[i],
                                            currentStation,
                                            arrivalTime,
                                            departureTime,
                                            runTime,
                                            PL);


            hastusEntries.add(planEntry);

        }

        // Make info about the last stop
        did.stationStoppedAt();
        currentStop = did.getNextStop();
        currentStation = currentStop.getStation();
        arrivalTime = departureTime + runTime;
        departureTime = arrivalTime + timeBetweenDIDs[i];

        // If it is the last DID to be planned the last stop is
        // planned
        if(i == (numDIDs-1)){

            planEntry = new HASTUSPlanEntry(DIDs[i],
                                            currentStation,
                                            arrivalTime,
                                            departureTime,
                                            0,
                                            PL);

            hastusEntries.add(planEntry);

        }
    }


    // The list is turned into an array
    int planSize = hastusEntries.size();
    HASTUSPlanEntry[] trainPlan = new HASTUSPlanEntry[planSize];

    for(int i=0; i < planSize; i++){
        trainPlan[i] = (HASTUSPlanEntry) hastusEntries.get(i);
    }

    // Based on the array a HASTUS train plan is generated
        HASTUSTrainPlan hastusTrainPlan
            = new HASTUSTrainPlan(trainPlan, trainId);
        return hastusTrainPlan;

    }



    //-----------------------------------------------------------------
    // genHastusPlan:
    //
    // generates a Hastus plan for each of the specified trains by use
    // of genHASTUSTrainPlan
    //-----------------------------------------------------------------


    public static HASTUSTrainPlan[] genHastusPlan(int[] trainIds,
                                  int[][] DIDsForEachTrain,
                                  double[][] timeBetweenDIDsForEachTrain,
                                  int performanceLevel,
                                  double dwellTime,
                                  double[] startTimes){


        int numTrains = DIDsForEachTrain.length;
        HASTUSTrainPlan[] plan = new HASTUSTrainPlan[numTrains];
        PerformanceLevel PL;

        for(int i = 0; i < numTrains; i++){
            PL = Central.getPerformanceLevel(performanceLevel);
            plan[i] = genHastusTrainPlan(trainIds[i],
                                         DIDsForEachTrain[i],
                                         timeBetweenDIDsForEachTrain[i],
                                         PL,
                                         dwellTime,
                                         startTimes[i]);
        }

        return plan;
    }


    //-----------------------------------------------------------------
    // printHastusTrainPlan:
    //
    // print the specified Hastus train plan
    //-----------------------------------------------------------------

    public static void printHastusTrainPlan(HASTUSTrainPlan plan){

        int trainId = plan.getTrainId();
        System.out.println("Hastus plan for train: " + trainId);
        System.out.println("DIDno   st.    arr.    dwell   dep.    run     pl");

        HASTUSPlanEntry entry;
        int didno;
        String station;
        double arr;
        double dwell;
        double dep;
        double run;
        PerformanceLevel PL;
```

```
        int PLno;

        entry = plan.getNextEntry();

        while((entry  != null)){

            didno = entry.getDID();
            station = entry.getStation();
            arr = entry.getArrivalTime();
            dwell = entry.getDwellTime();
            dep = entry.getDepartureTime();
            run = entry.getRunTime();

            PL = entry.getPL();
            PLno = PL.getLevel();

            System.out.println( didno + "     " + station + "     " + Math.round(arr) +
                                "     " + Math.round(dwell)  + "     " + Math.round(dep) +
                                "     " + Math.round(run)  + "    " + PLno);

            plan.entryUsed();
            entry = plan.getNextEntry();

        }
    }


    //-----------------------------------------------------------------
    // printHastusPlan:
    //
    // generates the entire Hastus plan from Central
    //-----------------------------------------------------------------

    public static void printHastusPlan(int numTrains){

        HASTUSTrainPlan trainPlan;

        for(int i = 0; i < numTrains; i++){
            trainPlan = Central.getHASTUSTrainPlan(i);
            printHastusTrainPlan(trainPlan);
        }

    }

}
```

HASTUSTrainPlan.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "HASTUSTrainPlan.java"
//=====================================================================

//=====================================================================
// The HASTUSPlanEntry class definition
//=====================================================================

class HASTUSTrainPlan{

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Attributes                                                      */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    // The HASTUS plan
    private HASTUSPlanEntry[] trainPlan;

    // The train the plan is for
    int trainId;

    // Number of entries in the plan
    int numEntries;

    // Info for iteration
    int nextEntry;



    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Constructor                                                     */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    public HASTUSTrainPlan(HASTUSPlanEntry[] trainPlan, int trainId){
        this.trainPlan = trainPlan;
        this.trainId = trainId;
        numEntries = trainPlan.length;
        nextEntry = 0;
    }



    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Accessors                                                       */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    //-----------------------------------------------------------------
    // getNumEntriesLeft:
    //
    // Returns the number of entries not used in the train plan.
    //-----------------------------------------------------------------

    public int getNumEntriesLeft(){
        return numEntries - nextEntry;
    }



    //-----------------------------------------------------------------
    // getNextEntry:
    //
    // Returns the next entry in the train plan.
    //-----------------------------------------------------------------

    public HASTUSPlanEntry getNextEntry(){

        int remainingEntries = numEntries - nextEntry;
        int actualEntries = Math.min(remainingEntries, numEntries);
```

```
    HASTUSPlanEntry nextEntries;

    if(actualEntries < 1){
        nextEntries = null;
    }else{
        nextEntries = trainPlan[nextEntry];
    }
    return nextEntries;
}



//-------------------------------------------------------------------
// getNextEntries:
//
// Returns the next 'numEntries' entries in the train plan.
//-------------------------------------------------------------------

public HASTUSPlanEntry[] getNextEntries(int numEntries){

    int remainingEntries = getNumEntriesLeft();
    int actualEntries = Math.min(remainingEntries, numEntries);
    HASTUSPlanEntry[] nextEntries;

    if(actualEntries == 0){
        nextEntries = null;
    }else{
        nextEntries = new HASTUSPlanEntry[actualEntries];

        for(int i = 0, j = nextEntry; i < actualEntries; i++, j++){
            nextEntries[i] = trainPlan[nextEntry];
        }
    }
    return nextEntries;
}



//-------------------------------------------------------------------
// getTrainId:
//
// Returns the id of the train the plan is made for.
//-------------------------------------------------------------------

public int getTrainId(){
    return trainId;
}



/*-----------------------------------------------------------------*/
/*                                                                 */
/* Mutators                                                        */
/*                                                                 */
/*-----------------------------------------------------------------*/

//-------------------------------------------------------------------
// entryUsed:
//
// Declares the current entry as used
//-------------------------------------------------------------------

    public void entryUsed(){
        nextEntry++;
    }


    //-------------------------------------------------------------------
    // entryUsed:
    //
    // Declares the current entry as the first
    //-------------------------------------------------------------------

    public void restartPlan(){
        nextEntry = 0;
    }

}
```

HASTUSPlanEntry.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "HASTUSPlanEntry.java"
//=====================================================================

//=====================================================================
// The HASTUSPlanEntry class definition
//=====================================================================

class HASTUSPlanEntry{

    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Attributes                                                      */
    /*                                                                 */
    /*-----------------------------------------------------------------*/

    // Planned DID
    private int DIDno;

    // Planned arrival station
    private String stationTc;

    // Planned arrival time
    private double arrivalTime;

    // Planned departure time
    private double departureTime;

    // Planned runtime to next station
    private double runTime;

    // Planned performance level
    private PerformanceLevel PL;



    /*-----------------------------------------------------------------*/
    /*                                                                 */
    /* Constructor                                                     */
    /*                                                                 */
```

```
/*--------------------------------------------------------------------*/

public HASTUSPlanEntry(int DIDno,
                       String stationTc,
                       double arrivalTime,
                       double departureTime,
                       double runTime,
                       PerformanceLevel PL){

    this.DIDno = DIDno;
    this.stationTc = stationTc;
    this.arrivalTime = arrivalTime;
    this.departureTime = departureTime;
    this.runTime = runTime;
    this.PL = PL;
}



/*--------------------------------------------------------------------*/
/*                                                                    */
/* Accessors                                                          */
/*                                                                    */
/*--------------------------------------------------------------------*/

//--------------------------------------------------------------------
// getDID:
//
// Returns the planned DIDs enumeration value
//--------------------------------------------------------------------

public int getDID(){
    return DIDno;
}



//--------------------------------------------------------------------
// getStation:
//
// Returns the planned arrival track circuit
//--------------------------------------------------------------------

public String getStation(){
    return stationTc;
}



//--------------------------------------------------------------------
// getArrivalTime:
//
// Returns the planned arrival time
//--------------------------------------------------------------------

public double getArrivalTime(){
    return arrivalTime;
}



//--------------------------------------------------------------------
```



```
// getDepartureTime:
//
// Returns the planned departure time
//--------------------------------------------------------------------

public double getDepartureTime(){
    return departureTime;
}



//--------------------------------------------------------------------
// getRunTime:
//
// Returns the planned runTime
//--------------------------------------------------------------------

public double getRunTime(){
    return runTime;
}



//--------------------------------------------------------------------
// getDwellTime:
//
// Returns the planned dwell time
//--------------------------------------------------------------------

public double getDwellTime(){
    double dwellTime = departureTime - arrivalTime;
    return dwellTime;
}



//--------------------------------------------------------------------
// getPL:
//
// Returns the planned performance level
//--------------------------------------------------------------------

public PerformanceLevel getPL(){
    return PL;
}


}
```

B.10 Additional classes

Calc.java

```
//=====================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Calc.java"
//=====================================================================
```

```
//=========================================================================
// The Calc class definition
//=========================================================================

class Calc{

    //-------------------------------------------------------------------
    // estimatedTimeOfTravel:
    //
    // Returns an estimate of the time of travel between two specified
    // stations based on the dominating speed restriction on the
    // stretch.
    //-------------------------------------------------------------------

    public static double estimatedTimeOfTravel(PerformanceLevel PL,
                                               StationAndStretchInfo[] S,
                                               String bifurcationSetting){

        /*-- Initialising variables --*/

        // Get first stretch
        Stretch stretch
            = S[0].getStretchToFollowingStation(bifurcationSetting);

        // Get length of first stretch
        int l = stretch.getTotalLength();

        // Get dominating speed limit for first stretch
        double v = stretch.getDominatingSpeedLimit();

        // Determine dominating speed limit
        double lengthOfDominatingStretch = l;
        double dominatingSpeedLimit = Math.min(PL.getSpeedLimit(), v);

        // Total distance between stations
        int distBetweenStations = l;

        for (int i=1 ; i < S.length-1 ; i++) {

            if (S[i] != null) {
                // Get stretch
                stretch = S[i].getStretchToFollowingStation(bifurcationSetting);

                // Get length of stretch
                l = stretch.getTotalLength();

                // Get speed limit for stretch
                v = stretch.getDominatingSpeedLimit();

                if (l >= lengthOfDominatingStretch &&
                    v <= dominatingSpeedLimit) {

                    if (v == dominatingSpeedLimit) {
                        lengthOfDominatingStretch += l;
                    }
                    else {
                        // Update information about dominating speed limit
                        lengthOfDominatingStretch = l;
                        dominatingSpeedLimit      = v;
                    }
                }
                // Update information about total distance between stations
                distBetweenStations += l;
            }
        }

        /*-- Calculate estimated time of travel --*/

        // Distance bestween stations
        int    s = distBetweenStations;

        // Performance level acceleration rate
        double a = PL.getAccelerationRate();

        // Performance level deceleration rate
        double d = PL.getDecelerationRate();

        // Dominating speed limit on the stretch between the stations
        v = dominatingSpeedLimit / 3.6;


        /*-- Calculating estimated time of travel between stations --*/

        // Time to get from zero speed to v
        double t1 = v / a;

        // Time to get from v to zero speed
        double t3 = (0-v) / d;

        // Travelled distance - acceleration
        double s1 = 0.5 * a * Math.pow (t1,2);

        // Travelled distance - deceleration
        double s3 = v * t3 + 0.5 * d * Math.pow (t3,2);

        // Travelled distance - constant speed
        double s2 = s - s1 - s3;

        // Time of travelling - constant speed
        double t2 = s2 / v;

        // Estimated time of travelling
        return t1 + t2 + t3;

    }

}
```

Print.java

```
//=========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "Print.java"
//=========================================================================

//=========================================================================
// The Print class definition
//=========================================================================

class Print {
```

```
    private static boolean allowPrint = true;

    //---------------------------------------------------------------------
    // log:
    //
    // Prints a message to std-out.
    //---------------------------------------------------------------------
    public static void log(String logMessage) {
        if (allowPrint) {
            System.out.println(logMessage);
        }
    }

    //---------------------------------------------------------------------
    // Accessor
    //---------------------------------------------------------------------
    public static void setAllowPrint(boolean onoff) {
        allowPrint = onoff;
    }

}
```

HashItem.java

```
//===========================================================================
// Ida Moltke, Matias Sevel Rasmussen & Tor Justesen - DIKU 2003
//
// Bachelor project: "HashItem.java"
//===========================================================================

//===========================================================================
// The HashItem class definition
//===========================================================================

class HashItem {
    private int arrayNum;
    private int arrayIndex;

    //---------------------------------------------------------------------
    // Constructor
    //---------------------------------------------------------------------
    HashItem(int arrayNum, int arrayIndex) {
        this.arrayNum = arrayNum;
        this.arrayIndex = arrayIndex;
    }

    //---------------------------------------------------------------------
    // Accessors
    //---------------------------------------------------------------------
    public int getArrayNum() {
        return arrayNum;
    }
    public int getArrayIndex() {
        return arrayIndex;
    }
}
```