A Multiagent Approach to Automating Railway Capacity Request Approval

Master Thesis

MSc. Technical Artificial Intelligence
Information and Computing Sciences

Supervisors:
Dr. Mehdi Dastani, Utrecht University
Emdzad Sehic, ProRail

Arie van den Berg
February 2018

ProRail

Abstract

Ongoing growth of the Dutch railway passenger and cargo transporting system demands that the current infrastructure is utilised at utmost efficiency. Next to creating a train schedule well in advance, traffic operators regularly have to schedule new trains in a current, dynamic planning. When the request to create a path for the new train is sent by the transporter company, it is handled by the different parties responsible for the execution of the train schedule. A top-down approach is taken where the operator with the most global view of the network first creates a draft of the train. After that he notifies those operators that have a smaller but more detailed view on the infrastructure, all the way down to the level of switches, signals and sections. This already modular process was subsequently built using a multi-agent system, closely mimicking the roles and responsibilities seen in practice. The system was built in PRL-Game, a mockup of the operator systems used in reality, to ensure a timely realisation and provide a safe playground for the agents. Experimentation to validate the agent system was done on a minimal scale, although preliminary results were exciting and inspiring.

Acknowledgements

I wish to greatly thank my daily supervisor, Emdzad, for giving me the freedom to pave my own path, for clearing the many hidden obstacles along the way, for believing in me and for always taking the time to challenge my poor planning. I feel privileged for having had a former stranger become a dad at work.

I am also immensely grateful for the kind haven I experienced at the Innovation department during my research. I always felt part of the team, building together for the better, smarter and more efficient. Instead of considering the new as being inferior, they harbour the potential in each colleague and stimulate them to become greater.

I want to thank Mehdi, my supervising professor, for giving me the opportunity to land at ProRail, for being scientifically strict, yet considering what is possible and necessary. Each time I told him why my thesis was not scientific enough, he would explain the scientific value and urge me to continue.

Finally, I wish to mention my parents as my perpetual stimulators to achieve not the best, nor the most expensive, but the results that provide that little edge in the world, that spark my curiosity. It never mattered to them how long it would take, or what the result would be, but rather that it was done.

Contents

List of Figures

Chapter 1

Introduction

In the Netherlands, ProRail is in charge of the railway infrastructure. It manages tracks, maintains signals, builds stations but also creates the train schedules. The physical trains, the rolling stock of the rail companies, are not part of ProRail's assets. The rolling stock is used to service customers, be it passengers, or industry and logistics. Instead of directly servicing passengers, ProRail sells its capacity on the railway infrastructure to the train companies.

Due to recent improvements of the nation's economy, demand for rail traffic will rise in the coming years. At the same time, the Dutch government wants to stimulate the use of public transportation and has therefore created a growth challenge until 2020. This means that ProRail will have to anticipate a rising demand in overall infrastructure capacity and that it must discover ways to either intensify the use of its infrastructure, or invest in expanding the existing infrastructure, or both. However, the Dutch government has made clear that intensification of the infrastructure use is not entirely optional: it has created a high-frequency programme that stipulates that key cities be connected using 10 minute services, in contrast to the current quarterly service.

Having one of the most intensively used railway systems of the world already, ProRail faces the challenge of increasing its infrastructure use even more or adding additional infrastructure. Clearly, simply doubling the existing infrastructure is too costly, and easy options to intensify are already exhausted. It is necessary to apply genuine innovative solutions in order to reach the goals. One of the tools available for this job is *simulation*.

The Innovation department of ProRail is continuously searching for ways to make the railway infrastructure management task safer, faster, more efficient and cheaper. One of the tools that are used is simulation. It is employed to replay past situations, to practise situations in the present, but perhaps most importantly to execute future scenarios. Using simulation, one can evaluate the effects of protocols for handling for example disruptions,

or to educate and train personnel, but also to assist decision makers. All of these combinations can be and are indeed used to solve current challenges.

Another project that Innovation is actively persuing is the question of how to effectively use agent-based modeling in a company like ProRail. Agent-based modeling, like big data, looks like a promising field of research and several initiatives are embracing these new techniques. Because difficult operational tasks have gradually become more complex in effective structure, i.e. more operators work together with complex procedures to solve these tasks, agent-based modeling which by nature employs an organisational structure is used to simplify or automate these tasks. Often, simulation is used to develop and test these new ideas. Similarly, simulations profit from the use of multi-agent systems that automate certain tasks because that way, simulation studies require fewer participants.

A suitably constrained scenario for applying multi-agent programming is for the so-called 'order accepting' task: train companies place requests for a new slot in the time table, after which the request is handled by a procedure involving many operators. Order acceptation happens in the last 36 hours before the train is supposed to run, precluding the use of the default planning tools that, due to regulations and procedures, are used further in advance. These orders arrive at the most unfortunate moments, take a long time to be processed by the whole chain of operators, and are prone to repetition, which happens when any operator in the chain rejects a proposal and another often similar proposal is made. The main and sub research questions are formulated as follows:

1. In what way can agent-based modeling be used to facilitate the work of rail traffic operators?
   (a) What is a suitable scope of the project?
   (b) What is a natural and efficient approach to building a supporting system?
   (c) What is a good choice of agent internals?
   (d) How to implement the agents in the available architecture?
   (e) How to find proper sources of data?
   (f) How can the result be verified?

Chapter 2 describes ProRail, their activities, tasks and responsibilities. It also lays the foundation for this research in terms of the problem statement and the context of this research in terms of scheduling and simulation. Chapter 3 describes the process of order acceptation, from operator hierarchy to technical requirements. Chapter 4 takes a step sidewards from the order acceptation process in reality and describes how to use the simulation software around which the rest of the thesis will center. In chapter 5 the

agent and multi-agent models are detailed. Chapter 6 describes how the agent models were built. In chapter 7 the experimentation is written down. Chapter 8 concludes with what has been learnt from this study and lists several developments for the future.

Chapter 2

Background Information

2.1 ProRail

As the manager of the railway infrastructure in the Netherlands, ProRail has three main responsibilities [1, 5]. Firstly, it maintains the infrastructure by for example renovating old tracks and building new stations. Secondly, it implements the train schedule in a safe manner. Thirdly, it allocates the remaining infrastructure capacity not covered by the regular schedule amongst the train operating companies in a neutral way.

Over the years, the train schedule is getting tighter due to the ongoing growth challenge until 2020 [1, 2], but also because of the PHS (Program High-frequency transport) [6, 7], and economic causes [8] in general. Consequently, the rail infrastructure will be stretched to its maximum capacity, requiring innovative solutions to allow more trains on the same number of tracks while guarding the safety of the system.

The Dutch railway network is among the busiest in the world, at least being the most highly utilised in the EU[39, 43]. ProRail manages around $4,000km$ track which is being used by approximately 4 to 5 thousand trains a day [41, 40]. For the *Beter en Meer* (Better and More) programme, the timetable on gradually more sections is being changed to a *high-frequency* schedule [8]. This involves having 6 or more trains of both types of services per hour, so at least 6 intercity services and 6 stopping services, per direction. There should also be room for freight trains, which gets us to the formula $6 + 6 + 2$, giving a good indication of the complexity that this creates. A simple calculation shows that if these 14 trains have to be fit in 60 minutes, each train has an average slack of a little more than 4 minutes. Most of the sections in the Netherlands are double tracked, seldomly triple, rarely and then only partly quadruple tracked (for reference, see figure 2.1).

[1]Door Treinfan op de Nederlandstalige Wikipedia (Originele tekst: T Houtdijk voor versie 1.3, ik heb deze bewerkt tot versie 1.4) - Eigen werk + File:Meersporigheid-v1.3.png, Publiek domein, https://commons.wikimedia.org/w/index.php?curid=2256994

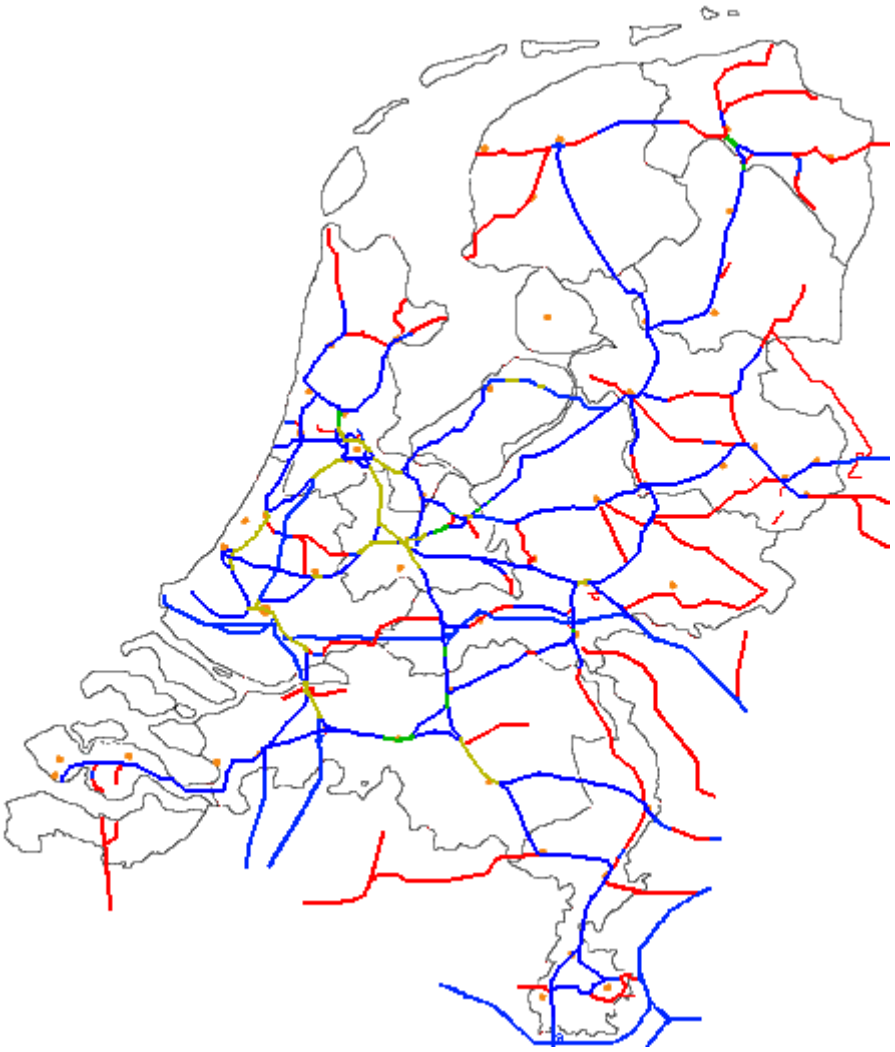


Figure 2.1: Situation of the Dutch railroad network in 2012. Red represents single track, blue double track, green triple track and yellow is quadruple track.
Image obtained from Wikipedia[1].

ProRail is a company of which the Dutch government is the only shareholder. It is subdivided into several departments, e.g. V&D (Transport & Timetable), Projects, Asset management and Traffic management[9][2]. V&D is responsible for creating the train schedule, while Traffic management is responsible for executing this schedule and dealing with incidents and extra capacity requests during the day. This separation of the central task of

[2]See the linked document *Organogram* on this website for a graphical representation of the organisation, listing all departments.

running the train schedule makes for a complicated situation.

2.1.1 Rail Innovations

This project was conducted at the department of Innovation. More specifically, it was done in the context of the RailwayLAB[3]. The RailwayLAB [23] aims to contribute to the better functioning of the railway sector by testing *logistic and infrastructure related* concepts in a safe environment. For this, it uses a combination of games and simulations. By playing serious games (both analog and digital) with operators and policy makers and executing simulations, processes in the railway sector are mimicked and their performance measured. The results help managers of the policy makers create guidelines that improve the performance and capacity of the railway.

2.2 Traffic Management

Of the different departments, ProRail's *traffic management* has the main task of monitoring the traffic flow. The train schedule is central in this task. Every train has a sequence of activities which determine *when* the train does *what*, *where*. An example stopping service performs all activity types: first, it departs (V) from the first station. Along the route, it performs short stops (K), which take less than a minute. At the football stadion, it passes through (D) because the station is not in use when there's no match. At its final station, it arrives (A). At last, it is shunted (R) away from the platform and set aside for future use. Along with each of these activities comes a *route*, which defines what tracks the train will use for that activity. Altogether, the schedules for all trains form the complete time table.

If there would be no disturbances in the train schedule and given that the time table were implemented in such a way that there are no route conflicts between trains, it would be possible to let the system execute by itself: it would only be necessary to change the switches to the correct position at the right time and set the safety signals to the correct (safe) state for the next train. For that, ARI [4] (Automatic Route Booking, Dutch: Automatische RijwegInstelling) is used. It automatically sets routes for trains as defined in the schedule, relying on the underlying safety system to handle switches and signals. In this situation, traffic managers are only monitoring the system. However, this situation can continue only for so long, because the time table is easily disturbed by for example delays, technical difficulties with trains or personnel scheduling problems. Depending on the severity of the situation, traffic controllers can be busy for hours regulating traffic.

[3]Description directly translated from [23]

2.3 Rail Infrastructure and Scheduling

For our purposes, the whole infrastructure can roughly be divided into *TCA*s and *free track* connecting them. TCAs, short for Terminal Control Areas, represent every point of interest along the tracks. Key examples of these are bridges, train stations, shunting yards and other locations where switches are located. All of these areas can be controlled by traffic managers by changing the position of switches, changing scheduled platform tracks or opening and closing bridges. Free track is what connects TCAs and is by definition uninterrupted and uncontrollable. By their very nature, switches cannot be part of a stretch of free track, as the switch would belong to a TCA and the free track would be broken up: one part going to and another continuing from that TCA. Related TCAs are functionally grouped into PPLGs, which are Primary Areas of Process Control.

In order to carry out their task as train traffic flow monitors and occasional planners, traffic managers are assigned different roles with differing scopes. Those at the higher level of perspective are the regional traffic controllers, or DVLs[4]. (For a full task overview, consult [11].) They monitor trains travelling between TCAs, ignoring what exactly goes on inside those places. This model of the state of the network is usually visualised in a time-distance graph (see also figure 2.2), where TCAs lie on the horizontal axis and time is on the vertical axis. To be slightly more precise, TCAs happen to lie on the horizontal axis while the *free track* represents the actual distance. In this way, trains going from A to B travel diagonally, with fast trains having a smaller slope than the slower trains, who take a lot of time to cover the same distance. The lines show the schedule for each train and judging from them, the traffic operator can see potential contention issues, ordering issues and other problems that arise at this level of detail. From his view, the operator can easily spot free space between subsequent trains, something that is harder to do with the systems of the local operators.

Local traffic controllers, or TRDLs[5], have a lower level view of the infrastructure. (See [12] for a detailed task description.) Usually, they manage several logical units of control called PPLGs, whose numbers cannot exceed 9 for a single controller. This handful of PPLGs is conceptually designated as a *workplace*, and each traffic controller operates one of them. Running a workplace involves several different systems which all have to do with tracks. For example, the train schedule is displayed in the *plan screen* (see figure 2.3), showing for each train from which track it will go to which other track

[4]Dutch: Decentraal Verkeersleider. For the sake of keeping with English terms, the terminology of *regional* operator or controller will be preferred. When the use of the abbreviation is beneficial, later on, it will be reintroduced.

[5]Dutch: TReinDienstLeider. Again, for the sake of using English terms the use of *local* operator or controller will be used throughout this thesis, except for later, when using the abbreviation becomes beneficial. It will then be reintroduced.

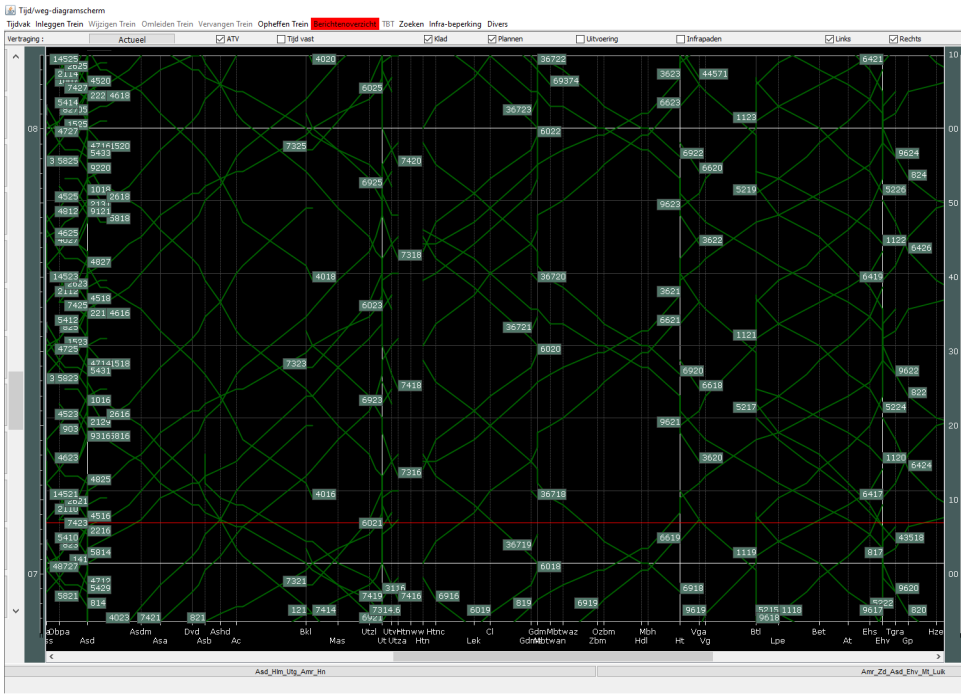


Figure 2.2: Example of a time-distance graph. All trains travel diagonally, one faster than the other. For example, at Ashd to the left, one of the trains is significantly faster than the other two trains, going almost horizontally while the others display a steep incline. Between Gdm and Ht in the middle, one can see that one train gains on the other, but this potential conflict is solved because trains can pass at Ht (signified by the white vertical line: Ht is a major station). Between Ut and Ht there is some significant free space in both directions.

(displayed in figure 2.4). The *signalling screen*s show a detailed map of the PPLGs, including track names and the location of any trains in the area (shown in figure 2.5). The *detail screen* shows names for infra elements that are not shown in the signalling screens. The *track occupation graph* (SBG) shows when and how long certain tracks will be occupied by trains.

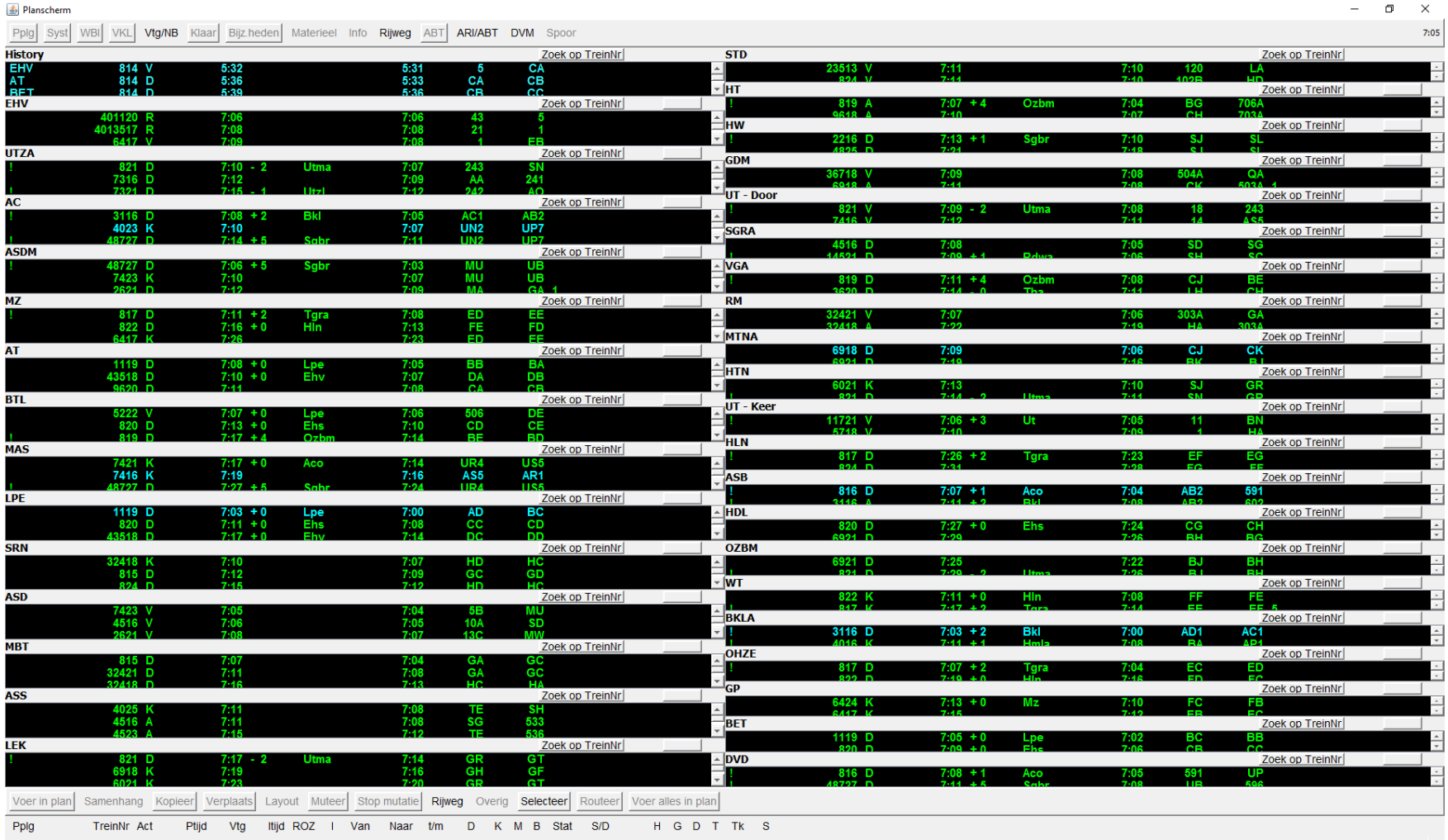


Figure 2.3: The plan screen showing all PPLGs on the A2 corridor, from Amsterdam to Sittard. Each PPLG has its own plan rules. See figure 2.4 for a closeup of a PPLG panel.

Figure 2.4: A detailed view of the PPLG panel in the plan screen. Plan rules for in this case ASD have train number, activity type, planned time and to the right the route.

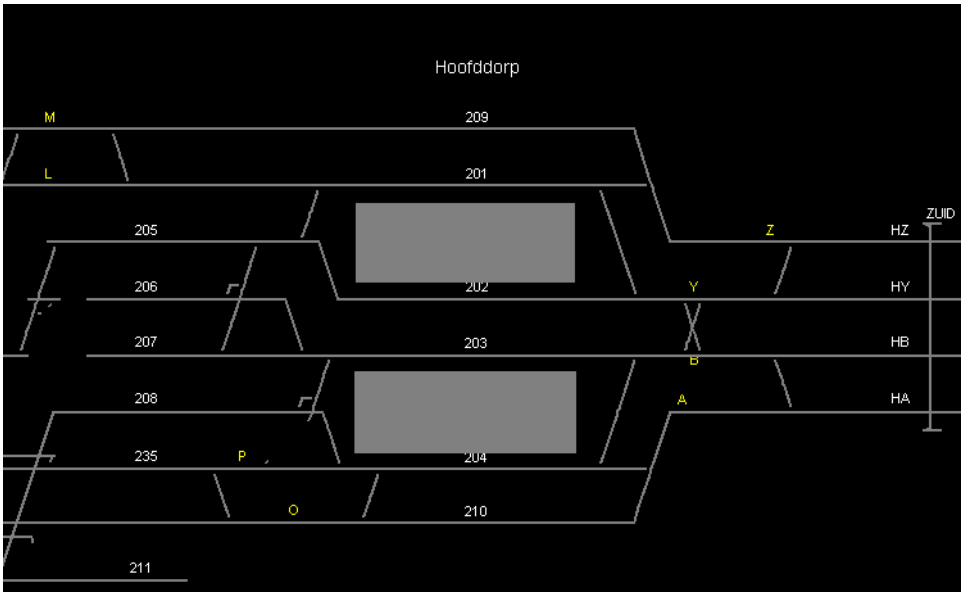


Figure 2.5: A selected piece of the signalling screen, in this case displaying an area in the workplace of Amsterdam.

The two roles complement each other. The regional controller can spot potential conflicts on a higher level and is able to direct traffic on the high level of TCAs. The local operators have full control over the situation inside their PPLGs. They have the power to shunt trains, have them arrive at different tracks, or otherwise control the precise flow on track level. This distinction will be retained in the software solution that is being explained in this thesis. There will be the concept of the PRL-Agent, the local controller agent that knows everything about the tracks and their occupation. Their colleague, the VKL-Agent, has information that is available to the regional operator. The agents will work together, given the local information available to them, in a way that resembles the steps that the human operators would take.

Note: the term *route* is ambiguous. It either means the train's TCA sequence, which is the main responsibility of the regional operator. Conversely, it can also mean the specific tracks the train will follow in a PPLG. This last route is of importance only to the local controllers. Hence, when a train is created, first its route is created by the DVL, after which the TRDLs

will themselves create routes in their respective areas of control. Throughout this thesis, this term will be used to signify *both* concepts, although the aim is not to use the term for both in the *same* section.

2.4 Simulation Software

Simulation is a beneficial tool for safety critical systems such as the rail network. Hence, ProRail uses multiple simulators to test implications of future situations. In this thesis the FRISO simulator [18] is used. Its focus lies on the micro level of the rail infrastructure, detailing train schedule, exact travel times and train speed and precise switch changing times rather than modeling security layers and fallback systems. FRISO can be run standalone, where it can be used for research in, for example, robustness of a train schedule, effect of the management variant in a specific area (FCFS, fixed order or following plan), analysing the cause and relationships between delays and quantifying effects of changes in the train schedule. One application of FRISO Standalone that was used in this thesis is its ability to export a detailed report on which elements of the infrastructure a train has ridden. This information coupled with data on exact train speeds are further used by ProRail to compute possible conflicts between trains. This application will be explained later.

But more interesting things can be done besides simply analysing the results of a simulation. FRISO has been designed to support external influence on the running simulation. The first tool that was coupled with FRISO was TMS [19], ProRail's Traffic Management System. It was already being used to influence the driving behaviour of real trains. TMS receives train positions and their speed and computes the desired speed of the trains, such that they won't be blocked by each other. Both systems, FRISO and TMS, were retrofitted to support the High Level Architecture[6] [20, 19] of which the implementation used is the RTI (Real Time Interface). For the RTI, every client is called a federate, so there would be the FRISO federate and the TMS federate. The RTI's function is to synchronise the systems, making sure that neither of them can outrun the other.

Now that FRISO supported the HLA, PRL-Game [22, 23] was developed at the TU-Delft. It was written in JAVA and it was designed from its inception to exclusively couple with the FRISO federate. PRL-Game acts as a shell around the simulator, enabling users to change how the simulation will carry on similarly to how traffic operators influence the situation in reality. The name is derived from the name of the local traffic control system, called PRL in Dutch, and the fact that the software is used for simulation or serious gaming, making it PRL-Game. Currently, PRL-Game can be used to (re)play past, current or future scenarios using any real-world

[6] Refer to [21] for a book about the HLA

configuration. FRISO is able to work with official documents describing infrastucture, timetable, train relationships and material configurations and PRL-Game does the same, heavily relying on the information that comes from FRISO. Together, they are used to test measures that were designed to reduce congestion and delays when specific calamities occur, to train traffic controllers before using a future train schedule, to replay scenarios in order to scientifically perform psychological tests on users, or to analyse the performance of multiple variants of a time table before proposing one of them to be used later on in the real world.

Chapter 3

Order acceptation: the process

During the day, train operating companies may place requests to ride additional trains. Freight trains are a prime example, for their departure depends on factors that are out of control of the operating companies. Perhaps the overseas cargo is delayed, or the weather is bad in the Alps, to name a few [17]. Empty passenger trains and single locomotives are two other causes where train companies request capacity from ProRail [38]. These requests must be manually processed by traffic management. Berends & van Smeden [3] presents a summary of daily changes to the train schedule, listing an average of 431 new trains a day, while van Smeden [38] later performs a more exhaustive study and finds 500 orders per day. Summarising the most prominent results from the last source, they find that the orders arrive as a constant stream throughout the day, roughly $20 - 40$ per hour, while 40% of the orders are issued less than 30 minutes before departure. Planning trains is hard, time consuming and therefore interferes with the other tasks of traffic management [3]. The current procedure for processing these new train requests is unsatisfactory and ProRail is researching different approaches to tackle this problem. [17] mentiones that freight companies must check in on their granted time slot 90 minutes in advance, such that ProRail can reuse those paths for other trains if they would otherwise be left unused. Berends & van Smeden [3] introduce a tool called RijwegGenerator (RouteGenerator) that helps the local traffic controller with specifying the route of the train.

3.1 Formal Process

The order acceptation process is a stepwise process that includes train operators, local traffic controllers [13, 12], regional controllers [11] and occasionally national traffic managers. Besides being officially documented, the task that

the operators perform has been verified using interviews ([46, (Appendix B)] has the questions, [45, (Appendix A)] has the transcriptions). First, some train company decides that it needs to ride a train that not yet exists in the train schedule. It then has to request a path from ProRail which needs information about the new train. What follows is an incomplete list of information that is commonly provided [16]:

Date Determines the day on which the new train must ride

Train number Coined by the transporter, it obviously may not already exist in the schedule on *date*

From TCA The TCA of departure

To TCA The TCA that is the destination of the new train

From track Defines on which track the company will have the train start

To track Defines on what track the company wants the train to end

Ready time The earliest time on which the train may depart, which is when the cargo is loaded and the driver is present

Deadline *(Optional)* The latest time of arrival at *to TCA*

Traction Defines what type of locomotive will be used

Length The total length of the train

Weight The weight of the train

Special Whether this train transports any special or dangerous cargo

Form Whether this train conforms to the default sizing of the wagons or whether it needs more care when planning

Extra stops *(Optional)* Needed for changing drivers or other special needs of the transporter

Description *(Optional)* A useful description for ProRail about this new train

Train companies have an internet-based connection with ProRail through the ISVL system [16]. As soon as ProRail receives the request, ProRail starts its internal process. The regional operator (DVL) that is responsible for the departure PPLG first handles the request. He creates an initial path for the train by fitting it between existing trains, roughly sketching those parts that are beyond his area of control. He then asks the local traffic controllers (TRDLs) that manage the parts of the route for which he is responsible verbally for verification of the route. The TRDLs will test the train on criteria that will later be discussed and reply whether the train can safely run through his workplace. As soon as all TRDLs replied positively, the DVL sends the draft train to his colleagues along the train's route. This communication is done inside the ISVL system. They in turn will perform the same steps, fitting the draft train correctly through their area of control

and asking the TRDLs responsible for his part for ratification. Only if all DVLs responded positively to the order request, the first DVL is allowed to accept the new route, after which a notification is sent to the transporter. A notification is also sent if there is no such path, after which the transporter company must make sure that his next request is more likely to fit.

3.1.1 Planning norms

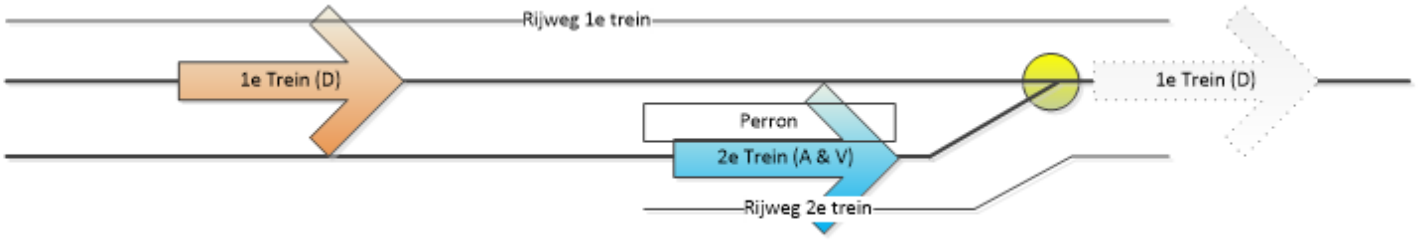
ProRail handles a specific agreement amongst itself and rail transporters called the *network agreement* [5] (Dutch: netverklaring). It states how delays are calculated, what tariffs exist for newly created trains, how the train schedule is designed and which rules are applied when scheduling new trains, amongst many other things. These last rules are called *generic planning norms*. These are in use since two fatal accidents in Barendrecht and Amsterdam.After careful study, it was found that these were caused by *unplanned* red signal approaches. When a train runs on schedule, the train driver expects that its route will be clear between stops. In constrast to traffic lights on public roads, which[1] follow a fixed cyclic pattern with fixed duration, railroad signals on the free track generally display non-safe *only* if there is a train ahead. Hence, train drivers do not expect to see a signal that is not green. If there is enough distance between two trains, signals can safely be set to green. The distance is directly proportional to the type of *activity* the trains have at a certain TCA. If the first train passes through a TCA and a second train will stop there, three minutes is generally enough time to ensure signals will be green along the second train's way. Conversely, if the first train were to stop at a TCA while a second train would pass through, much more than three minutes would be needed between the activities. Again, otherwise the second train would get a red signal and stop before the TCA, a highly unplanned situation for the driver. To summarise, the planning norms have been devised to quantify how much time there must be between activities of two trains in order for there to be enough room between them.

Planning norms only come into play if the routes of two trains *conflict*. There are three variants of conflicts, but they share the fact that both trains have one rail section in common. An example of a case with the longest conflict is simply two trains after each other, taking the same route. The smallest conflict is on a double slip switch, where both trains cross each other's paths. Their only conflict is in the middle of the switch. As the last example shows, it is not enough to consider only full blown tracks, as stretches of track between switches may not have a name, or simply because there is no sensical meaning of track in the case of the switch conflict. Therefore, as already alluded to, route conflicts are computed on the basis

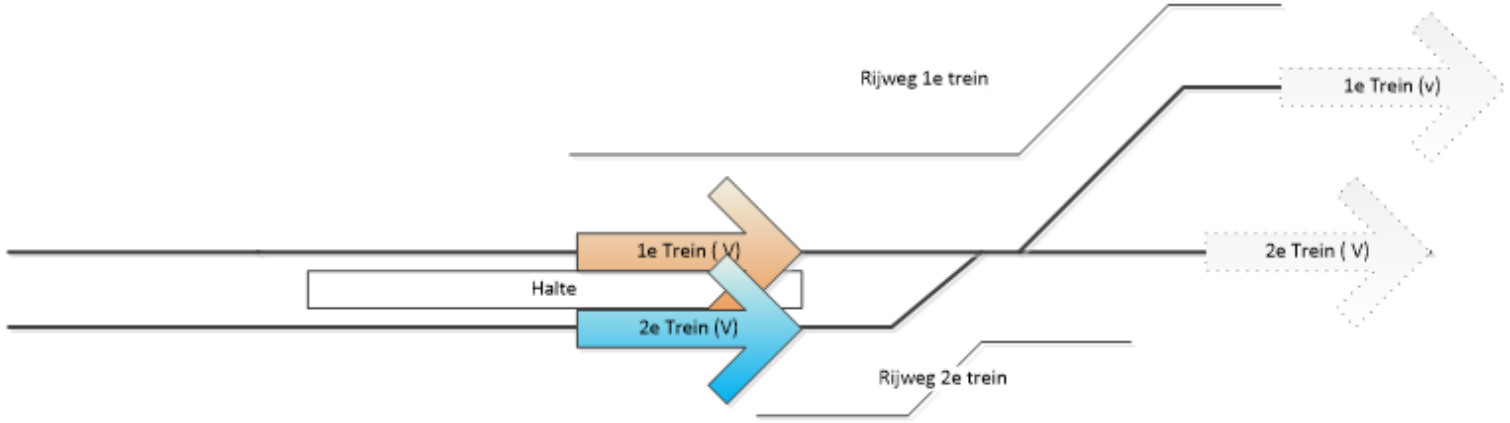
[1] in my experience

of sections.

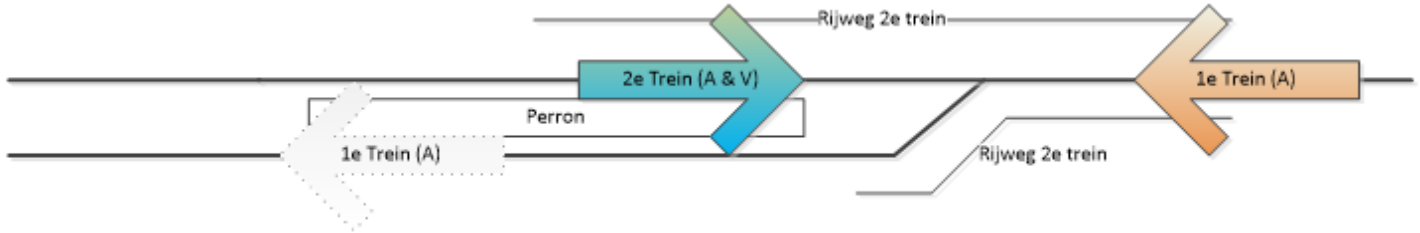
There is an obvious difference between trains going in the same direction and that have a conflict, and conflicts in opposite direction. In fact, there are three flavours. First, the so called *follow-up time*s (Dutch: opvolgtijden) apply to route conflicts of trains going in the same direction and end up going to the same track. Second, *same direction crossing*s (Dutch: overkruistijden in zelfde richting) are similar to the follow up times, but now the trains each end up on their own track. Third, *opposite direction crossing*s (Dutch: overkruistijden tegengestelde richting) are the cases where trains have an opposite direction. For conciseness, from now on the first type of conflict will be called *followup*, the second *cross to*, the last *cross back*. The rationale for this is easier explained from Dutch, but it also works in English: if both trains go *to* the same place, they go in the same direction. Otherwise, the other train is coming *back* from where we go. See figure 3.1 for a visualisation of the different types of conflicts.



(a) The situation where the first (red) train passes through a station first, after which the second (blue) train departs. Both trains will then follow the same piece of track, making this a followup situation.



(b) The situation where the first (red) train departs from a station. It has a common route with a second (blue) train although they end up going to separate tracks. This situation is a crossing in the same direction.



(c) The situation where the first train (red) arrives at a station and the second train departs in the opposite direction from the station. As both trains have a partly common route and they travel in different directions, this situation is a crossover in opposite direction.

Figure 3.1: Visualisation of different conflicts.

Several things can be noted about these definitions. First, there seems to be no fixed and more importantly, *unambiguous* definition of when each category is applicable. It looks like routes are follow-up if they end in the same track. However, what happens if a same direction crossing has a *very* long common route? In this case, although the table is mostly the same, some values differ for the worse. If we'd take this common route as a follow-up, we'd be safe because we now have at most one minute slack, although the definition seems to suggest that we'd need to look at the crossing table. Now what would happen if we have two trains that diverge? They have differing end points, but it probably should be categorised as a follow-up. This becomes even more curious if we consider that if the diverging train, who is *not* going straight, is first, he will likely keep his speed in check until he's passed the switch. The second train can go full speed through the straight switch. Nevertheless, the follow-up norms are slightly stronger which would more likely fit this case. Judging from the above, it might be best to define every conflict of trains going in the *same* direction as follow-up, *unless both* end points of this conflicting route lead to different tracks. Conflicts in opposing direction are easy to categorise.

Another observation is that there is no distinction between train types. Now this is the very nature of *generic* planning norms, but it does raise some concerns about the validity and workability of them. For instance, I have witnessed a long and loaded cargo train leave a station, taking roughly three minutes to even leave the platform. Even then it was not yet up to full speed. How would it be valid to plan a passenger train with a crossing conflict behind this freight train? After two minutes, the first train wasn't even gone, let alone that the whole route of the second train was safe, even worse, the passenger train would be faster and would definately be hindered. Of course, this is a typical example of when generalisation does not work, but it does give us some context about the tools we're working with.

One activity is not listed in the tables. Shunting (R) moves, however, are of particular interest to the order acceptation process. In fact, shunting moves are not visible to the regional operator, which makes sense because shunting takes place inside a TCA or at most a PPLG so that no free track is involved. Therefore it is the sole responsibility of the local operators to validate that any proposed new train does not conflict with any shunting moves. How is he supposed to verify that this is the case?

The most general rule is that trains should be three minutes apart (verified in interviews (see [45, (Appendix A)]) for a full transcription of the interviews).Regional controllers usually consider this norm when planning new trains by making sure that the new path stays three minutes from bordering trains. However, local traffic controllers have a much harder time. Contrary to the regional operators, they have a perspective that includes every combination of tracks a train can follow through switches. So in order to apply the generic planning norms, they should consider every route of

		Activity 2nd train			
		A	D	K	V
Activity 1st train	Arrival (A)	3	2	3	n/a
	Passthrough (D)	3	3	3	2
	Short Stop (K)	4	4	4	3
	Departure (V)	4	4	4	3

(a) Planning norms for follow-up conflicts.

		Activity 2nd train			
		A	D	K	V
Activity 1st train	Arrival (A)	3	2	3	1
	Passthrough (D)	3	3	3	2
	Short Stop (K)	3	3	3	2
	Departure (V)	4	3	3	2

(b) Planning norms of conflicts for same direction crossings.

		Activity 2nd train			
		A	D	K	V
Activity 1st train	Arrival (A)	3	2	1	1
	Passthrough (D)	4	3	4	1
	Short Stop (K)	6	5	6	1
	Departure (V)	6	5	6	2

(c) Planning norms of conflicts for opposite direction crossings.

Table 3.1: Generic planning norms.

every train in his area that might interfere with the new train. He must do so based on track information for each train. Despite holding all needed information of a train's route, track and route information does not give a clear image of how a train is going to travel. Hence, for every route the traffic controller must look it up on his overview screen, find out whether there is an overlap and then apply the generic norms. In practice, it is hardly ever the case that all local operators test the whole train.

3.2 Informal Process: Interviews With Operators

Although a formal description of the process is useful for grasping the initial rationale of the problem and for future review when the problem has been fully understood, it was also possible to talk to the experts in the field. Because the official documents were already at hand, it was not necessary to ask them *what* exactly they do, but rather *how* they do it. The initial idea was to perform a full Protocol Analysis with the operators such that their complete reasoning patterns could be understood and used. Explorations of the subject [26] and some specific applications [24, 25] revealed how to extract useful information from subjects. There has to be a very well thought-out script of how the interview will go. One or more scenarios that will be executed by the subjects must be completely specified. Then, subjects must be familiarised with the method of talking aloud while thinking. During the run of a scenario, the verbal report of the subject is written down as faithfully as possible. Post processing steps involve categorising the possible responses and determining how to create a useful protocol from the analysis.

There were numerous obstacles, many of them known beforehand. First of all, it was not possible to fit the whole order acceptation process into a couple of scenarios. At the point when the interviews would be conducted, it was not known how to distinguish realistic situations that would cover the whole deliberation process without introducing confounds. The creation of the scenarios would simply be too time consuming. Second, it is apparent that subjects must perform and know how to perform the task under question. In our case, the most important part of the process is the testing of an order. At the locations where the interviews would be held it was known that order acceptation was rarely performed, if at all. Therefore, the results would not contain valuable information of how operators perform the task. Combined with other minor issues, including difficulty to roster operators free, the lack of expected utility to perform such a big research and the relevance of doing Protocol Analysis, this led to a shift in interest and instead of a full-blown interview, informal, orientating and concept verification interviews were held (transcriptions in [**?**]).

The main advantage of this approach was that the operators could an-

swer questions while working, that they could show what they were doing in their own systems and that they did not need extensive initial training. However, this meant that the research question shifted from creating an as-faithful-as-possible system to a system that performs the same task as the human operators, but likely not in the same manner.

3.3 Summarising the Facets of the Process

The previous sections explained in detail what the process is formally about and what steps were taken to fully understand it. From this knowledge, we can state the following aspects:

Hard the process is hard in the sense that it's not a trivial task to make sure the new train's route is conflict free. The regional operator must know which parts of his section allow for one or more trains at the same time, while local traffic controllers should consider each train in his plan when testing the new train.

Communication Mostly informal, the main chunk of communication where the safety of the rail system in concerned happens between local traffic controller and DVL. Despite the DVLs notifying each other using the ISVL system, the lack of formalised communication makes this setup prone to error and misunderstanding.

Time pressure Monitoring trains goes relatively well in combination with planning trains. However, the moment that traffic control has to make changes to the current schedule due to any incident, checking the safety of new train's paths becomes unduly time costly.

Repetitive Rejection of one colleague triggers another round of checking a possibly slightly moved train. The space of possibilities is so large, that controllers cannot use a remembered reasoning cycle and must hence do the checking all over again.

Safety The systems that are in use do not assist the user in this task. Their representation space stops at the level of tracks, while safety of the system depends on the occupancy status of sections, one level below tracks. This means that operators either omit a large portion of the information, or they must go to great lengths looking up relevant sections for different routes, information that is not represented in a way that makes order checking easier.

Live situation Extrapolating current delays into the future is very hard. Delays may propagate several hours from now, diminishing or getting

greater depending on the schedule and the human factor of the operators. Planning a new train while keeping track of delays of relevant trains is nigh impossible.

Chapter 4

Information needed for gaming an Order Accepting Scenario

4.1 Using PRL-Game

Being the front-end for playing rail operators, PRL-Game behaves as if it was the real system. PRL-Game is started using a configuration file with extension .prl containing data necessary to create a workplace. A typical workplace for a local traffic controller has four screens.

Operate screen can be used to display the current infrastructure state at the most detailed level. It also provides low-level tools to manipulate the infrastructure, for example by revoking routes, changing switch positions or placing hindrances on signals or switches.

Plan screen shows *plan rule*s for all PPLGs that are operated at this workplace. For our purposes, plan rules tell which route a train follows and at which time the switches should be prepared and the route given free.

Signalling screen displays a schematic view of the PPLGs that belong to this workplace. The approximate train positions are painted on the abstract layout of the PPLG. There are usually two screens dedicated to showing this information.

The regional traffic manager workplace that will be used during simulations only sports one screen, called the *Diagram Screen*, which is a *time-distance graph*. Its only use is for dynamically changing the train schedule. It offers an interface to delete (parts of) a train, create new trains (potentially based on an existing train), but also change a train's activities, schedule (a part of) a train to some other time and reroute trains.

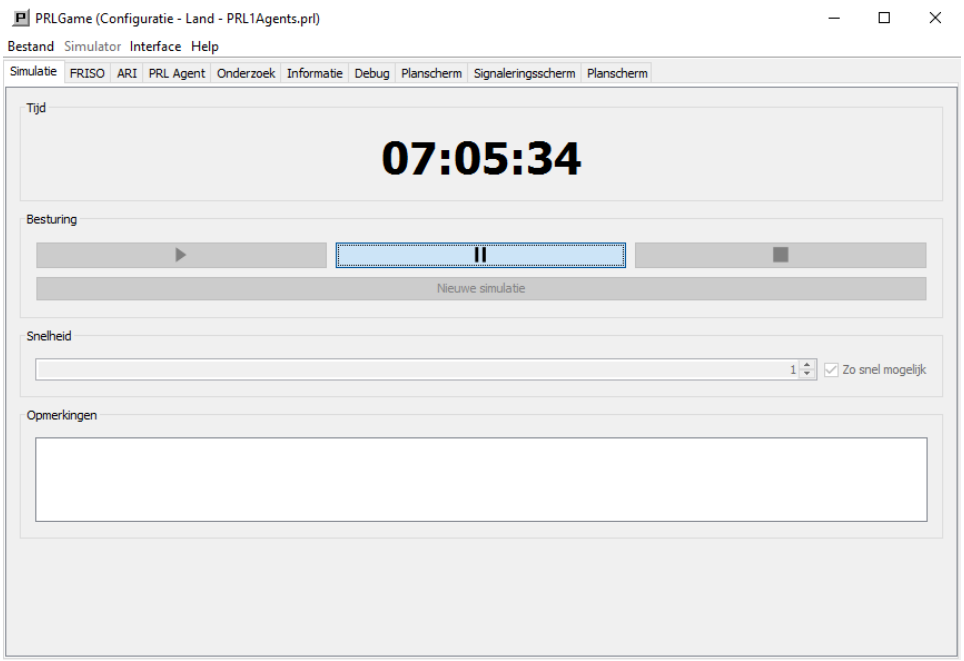


Figure 4.1: Example configuration screen. Along the top are the different tabs for the screens: Planscherm (twice) and the Signaleringsscherm. The tab FRISO is for configuring the simulator, the others are for the modules. The fourth tab is for the PRL Agent. The main panel belonging to the selected "Simulatie" tab shows the wall clock, as well as buttons for playing, pausing, stopping and resetting the simulation.

Because PRL-Game is designed to be able to control past, current and future scenarios alike, it is highly configurable. This mostly happens in a dedicated configuration window, generally simply called *The Configuration* (see figure 4.1). It contains tabs for each screen, simulator and module in use. Every part of the simulation has different parameters, so each tab displays different ways to change the settings. During the creation of a scenario this configuration screen is heavily used, but it still comes in handy during a simulation as the simulator tab functions as a wall clock.

Each workplace is started as an individual application. After loading the configuration, PRL-Game will try to connect to the HLA on a pre-specified port. In HLA terminology, one workplace is called a *federate* and the whole federation is assumed to exist in a single local internet network (LAN). During initial testing, it's well possible to run multiple PRL-Game federates on a single machine, but during full blown experimenting, every user has his own computer. The one special federate is the FRISO federate. It currently functions as the source of the simulation, holding all necessary characteristics of the state of the world and continuously publishing that state on the

HLA. After starting, FRISO will start simulating without publishing. This simulation period is called the *warm-up time*, in which trains are loaded into the model and the simulation creates state. After this fixed time period, FRISO sends its state to the other federates, at which point the simulation begins.

Federates are inherently modular. Conceptually, one workplace handles no more than a handful of PPLGs. The regional controller only knows high level information and is not concerned about what happens in other federates. However, this model is not upheld in the underlying model. Internally, PRL-Game knows everything, everywhere. It simply displays the relevant bits of information and lets the user access only local information. It would be tempting to forgo the idea of modularity and rather design a system that has access to all information.

During the simulation, PRL-Game federates and FRISO communicate with each other. FRISO sends messages containing current updated train positions, amongst many others, while PRL-Game sends messages about changes to the plan. Of particular intererst here is the message flow between federates when a new train is created. First, the DVL creates the superficial route of the train consisting of a sequence of TCAs. This route is already located in time, preferably without hindering other planned trains. As soon as the DVL marks this concept train final, two types of messages are sent. The first message marks the creation of a new train. The second message type denotes activities of the new train. Obviously, at least one message of this last type is sent for each TCA, possibly two if the train performs an arrival + departure move there.

Every federate receives these messages except for the sender. Federates store the received sequence and await the response from FRISO. FRISO checks the consistency of the route and we can assume it accepts the route. On reception of these messages, all PRL federates store these messages as 'global' VKL messages. Only those 'global' messages that are of importance to this federate because the TCA is operated are also made 'local'. The local message are then displayed to the user. For these local messages the user can create routes. The train will ride only if a correct and consistent route is created. As soon as the user enters the routes into the system, PRL-Game sends these messages to all other federates. Again, these messages are cached locally until FRISO responds to them. If FRISO accepts the new routes, they are entered into the system. As a result, they are then displayed in the plan screen of the local operator who operates those rules.

4.2 Sources for a Scenario

For FRISO to run, a database with relevant simulation data must be provided. The database is built from information from DONNA which includes

trains, schedule and possible combination/split data. A complete description of the data itself and their format is out of scope of this thesis and will not be discussed. DONS contains the material types. InfraAtlas is used as the data source of the infrastructure. For our purposes, a working database can simply be requested from the RailwayLAB. This database must then be restored into an SQL server. Currently, *SQL Server 2012 Express* is used. The restoration process itself happens through the *Management Studio* which is a separate program. Because FRISO depends on certain updated libraries from the studio, version 2016 must be used. The studio can talk to the older 2012 server. Finally, FRISO must be installed and a licence must be obtained. Using the *FRISO Incontrol Center*, a connection can be made to the database and a federation can be configured and started.

PRL-Game federates need their own configuration. It is usually best to ask the RailwayLAB for a working Game Leader configuration and tweak it. All configurations have information about which TCAs exist and which PPLG each one belongs to and whether it is operated. PRL configurations need additional data on which PPLGs are visible in the Plan Screen, which visuals it needs to display on the Signalling Screens and which ARI settings it must use. VKL federates need other additional information. VKL functionality needs to know which sections (TCA sequences) is being displayed. This information cannot be generated run-time because FRISO does not send infra data, so it must be generated offline. Furthermore, information of material types, for example name, weight, length, minimum, regular and maixmum speed, etc., must be loaded into VKL.

4.3 Data for New Trains

If we temporarily ignore the separation of tasks of traffic control, the data needed during the complete process of creating a new train is this:

Route This item is twofold. First, the route through the country must be known. This includes all TCAs that the train will encounter. There generally exist multiple possible national routes of which one must be chosen. Usually only the preferred or most common route for the current type of train is used. Second, local operators have a broad choice of routing trains through a station. When submitting a train into the system, one of them must be chosen. The possibilities may nevertheless all need to be considered when choosing a path for the train.

Time Different types of trains have varying travel times between TCAs. These determine the times for each of the train's activities. We must also take into account what braking characteristics the train has when determining travel times before stops. Together with acceleration

times, these depend on the length and weight of the train, as well as the number of locomotives.

Time table The schedule of all other trains along the route must be known, including delay information. From this, we can deduce a route that does not conflict with any existing route.

Conflict margin This determines what criteria exist for the new train relative to the existing trains. It is an interesting measure: it can be set to a static 3 minutes, corresponding to a test by the DVL alone. If it's set to the general norms, it resembles the ideal state where all operators test the train. Nevertheless, if it were set to 'optimal', it could be possible to test the new train based on its own characteristics, based on the infrastructure on hand, leading to the tightest margin possible. In other words, this measure represents the capabilities of the testing procedure.

Train and route characteristics All information that is sent by the ISVL system, c.f. train weight, length, axels, preferred stops, identification number, traction, etc. For the route planning system, the from and to tracks that are specified in the ISVL message are important.

Final route The translation of TCAs to conflict free routes to plan rules is not that trivial. Plan rules must be generated as the train is being entered into the system.

Chapter 5

Towards building an order accepting agent

In the following, a software agent is developed that performs the task of order acceptation. To describe the term *agent*, the definition of [27] is used: an agent is *"anything that can be viewed as percieving its environment through sensors and acting upon that environment through actuators"*. Even more informal, an agent can *notice* things and can *do* things.

5.1 Agent Technology

ProRail is conducting research to learn how to apply agent technology to various tasks. [28] developed an agent-based train driver model based on real driving data. [29] built a multi-agent system for determining future train positions. The benefit of using multi-agent technology lies in its properties. Wooldridge [30] explains that "agents are computer programs that act independently on behalf of their owner". They do this solely by sensing, deliberating and acting. A good agent program is modular, decentralised, changeable, ill-structured and complex [31]. Modularity is to be taken as meaning that the agents are not themselves part of the environment. Changeable means that agents have state, while ill-structured here means that the a-priori design of the agent is not fixed. In other words, agents can be defined by their superficial characteristics or behaviour without needing to go into detail of their internals.

When designing an agent that needs to take over a task originally done by a human, it is useful to make it intelligent [30]. Characteristics of intelligence are *reactiveness*, *proactiveness* and *social skills*. Intelligent agents know how to react to a current situation, take initiative to perform actions, without the need for external stimuli, and they're social, in the sense that they can communicate with fellow agents (human or computer system). It is no requirement that the agent is strictly cooperative in its social skills. It is

only required for intelligent agents to perform those actions that further their way to their goals.

One of the ways to implement the internal program of the agent is by using the *Intentional stance*[30]. It advocates to describe agents using *mental state*s: *beliefs*, *desires* and *intentions* [32] (BDI). Beliefs represent the state of the environment as judged by the agent. Desires are those situations that the agent wants realised. Another way to think of desires is to call them *goal*s. Intentions are strong goals. As soon as an intention is adopted, it will not easily be dropped. Interstingly, goals may conflict with each other, while intentions cannot. Of the (possibly many) intentions, actions are generated. These actions together form the *plan* of the agent.

There exists research into scheduling trains using agent tecnhology, yet the agents are so simple that there is no need to design them using BDI. A striking first example is the German Train Coupling and Sharing (TCS) system [33]. Here, the goal of the system is to combine smaller cargo trains such that this resulting train only occupies one slot in the infrastructure. This is done by grouping trains that have a related travel time slot. Not single trains but groups are modeled as agents, where a single train is a singleton group. When a train needs a path, one is acquired from the net manager. The agent then auctions its path with other agents. The highest bidder is the agent with the longest common route. To optimise the solution, a simulated trading protocol is used.

Other examples are [34], where train drivers are represented by agents. When scheduling conflicts arise, other agents volunteer to take over the duty of the initiator. If this leads to further problems, other agents recursively add themselves to the group. The solution with the lowest cost (c.f. lunch breaks, working overtime, etc.) is chosen. [35] describes a multi-agent system where train agents receive the speed and location of fellow agents and adapt their own speed to optimise the distance between the next and previous train. The agent's actions are determined using fuzzy control.

5.2 Model

Before the proposed model is explored, let us formally define the requirements of the system.
As an initial requirement for this project [37], we have

Agent-based The solution should make use of techniques from Agent Technology.

The Multi-Agent system should perform the act of order acceptation under the following technical prerequisites:

Environment Agents work from inside PRL-Game.

Communication Agents make use of existing messaging framework (HLA).

Modularity Agents use only local information. If the agent functions as the DVL, it only uses DVL-specific information available only to that federation.

Performing order acceptation is considered correct if the following practical considerations are met:

Testing Agents must test the new train. It must minimally abide by the general planning norms as found in [5].

Route creation Agents that operate at the workplace of the local operator should be able to enter correct routes for the new train.

Finally, there are some situations that need mentioning.

Dynamic norms The software should be designed such that the norms can be easily adjusted.

Automating The software should ideally be able to run without user interaction.

5.3 Model Description

Each federate is equipped with its own agent. The goal of the agent is to respond to order events (in the case of the VKL agent) or to respond to test requests (for PRL agents). In the following, the models for the separate agents is described. Treatise of the specific state that is being kept by agents is delayed and filled in when the communication protocol is explained. This is the part where the individual agents are integrated into the final multi-agent system.

5.3.1 Agent's context

The software agents are situated at the same level as the human user. Depending on the level of automation (discussed later), the agent can be used as a decision support system or as a fully automatic train planner. As support system, the agent should propose routes based on information that is also available to the traffic controller. If the system were to be built with a sophisticated reasoning engine, it could also convey its reasoning to the user on request. In this case, it could be important that the traffic controller can verify this train of thought. On the other hand, if the agent were to perform fully autonomously, its capabilities should be on par with the human user. For these reasons, the agent must be designed such that it receives the same information as the human operators do while also being able to act on the system in a similar fashion.

The agent has several sources of information and two communication channels. All of these are modelled as contexts. The agent's *knowledge* also fits in here, but its description will follow later. The first category is *knowledge*. It is further subdivided into data on the current plan and of the current planning norms. With the plan, the whole train schedule including delays and other relevant information for the execution of the time table is meant. These contexts do not have to be shared by all agents. It is likely that information in the plan of one federate is different from other federates. Likewise, agents do not necessarily use the same planning norms. If a real-world scenario would be played, individual agent's motivation could influence how the norms are being applied.

Next, the two communication channels are both modelled as separate contexts. For agents to have an effect on the simulation, for instance by automatically creating plan rules for the new train, there must be a context to do so. The context should roughly use the same code functionality as the user would trigger. Specifically, the agent must be able to create a train, send the TCA sequence and send complete plan rules. In addition, it must make sure that the PRL-Game federate is aware that it sent these messages. In order to communicate with fellow agents, another context is used. The incentive for modeling this as a separate context is to distinguish communication among agents themselves and with PRL-Game. Messages about creating a new train and creating plan rules might be time constrained and should therefore be managed by the HLA, but communication about when it should ride are not time critical. Hence, they should not belong to the same messenger.

5.3.2 Communication protocol

All communication resolves around the possibility of having a train at a certain time. The main ingredient of the protocol is the *time window*. It defines when a train can have a certain activity on a TCA. One can imagine a station serving two trains per hour only. Trains will stop here only shortly. Now, considering a fixed planning norm of three minutes, the available time window for another train is exactly 24 minutes long, having three minutes space between the endpoints and the already scheduled trains. In general, the messages flow as follows. The DVL agent creates a series of time windows for each TCA that the train encounters. The endpoints of these windows follow the travel time between TCAs, ignoring possible time that is spent standing still. Hence, the first window would start at the ready time and continue on through time up to the deadline minus the total travel time. Visually, the time window 'bar' would start at the bottom but not continue to the deadline. Similarly, the final bar for the last TCA would float, hanging from the deadline time. These windows are sent to fellow agents. They incrementally update the windows of their operated TCAs until a valid

route can be created.

There are many choices a PRL agent can make. Each of these options can be prioritised individually. Taking for example a situation where two tracks A1 and B1 enter the agent's area, cross over at the middle and leave the area on A2 and B2, there are four choices. The most logical one is taking the train from A1 and sending it to A2. As this is a double track situation, trains will probably go from B2 to B1. This means that the time window to send the train to B2 will be quite limited, because we are not allowed to block the trains coming from B2. Similarly, taking the train from B1 will most likely block the other trains already in the schedule.

Not only is there a routing choice for tracks coming from the previous area going to the next, but there is also a multitude of paths inside the area. And not only are there only routing options, there is also a possibility to have trains stop on an unoccupied track and continue on later. Creating these arrival + departure activities has implications for the travel times of the train, but these are ignored here. All of these choices can be prioritised. Continuing with the previous example, the route A1 A2 would have the highest priority. If we now take the part B1 to the switches as having a shorter distance than the switch to B2, B1 A2 would have a higher priority than A1 B2. Finally, B1 B2 is least favourable. Each of these options can have several time windows throughout the planning period.

Even the time windows themselves may be prioritised. Imagining a situation where the train may pass through the station at only one minute, say $9{:}\,13h$. If the train were delayed, it would conflict with a previous train or hinder a train running later. However, at some later point, the time window is $[9{:}\,49h - 9{:}\,56h]$. The agent may even be so witty to split up this window into a preferred window $[9{:}\,51h - 9{:}\,53h]$ and the total window itself. If at all possible, the agent would like the smaller window to be chosen. Only if this doesn't fit with his colleage agents, he will accept the other, larger window. And if that one does not result in a succesful path, the $9{:}\,13h$ window is proposed.

The protocol will be guaranteed to be ending. The limiting factors in this set-up are time (we assume that we cannot plan more than 36 hours into the future, which correlates with the maximum time an order may be given [5, pp. 132], although time may also be limited by the order's deadline), and possible routes. Currently, ProRail plans in minute precision, and we will use this measure for the protocol. Now, the DVL agent first sends the initial time windows. Implicit here are the minimum times. PRL agents generate the possible routes locally, along with the time windows that belong to those routes. This can be done at the first step. If each agent would send its first priority window back to the DVL agent, the DVL agent may try to construct a route out of these windows. If he doesn't succeed, he needs to request more time windows. Based on the latest time window received, he computes the time points of each TCA using the train's travel times. The times are sent

to the agents. The intent of this message is to provoke the agents by saying "if you don't compromise, the train will travel at least *this* late!".

Based on the renewed latest time received from the DVL, agents now send new windows based on their priorities. If the DVL finds no earlier path, he increments all last times and sends them again. This ensures that the algorithm ends. It also offers the most flexibility from the agents. PRL agents may be designed in any way. The friendliest agents do not prioritise their time windows and blindly trust the planning norms. Less collaborative agents may favour usual or smart routes, while keeping the time windows in order. Other agents may mix everything up. They may do this by sending their favourite windows first and rejecting to send more windows until the very last moment (when the updated time is equal to their last minute of the original window sent by the DVL). This strategy hopes to ensure that their favourite windows are chosen regardless of other agent's preferences. Evil agents may only send their favourite windows and be done with the protocol.

As soon as the DVL agent found a route, it surely is based on time windows sent by his colleage agents. In order for the agents to know how to pick routes for the new train, they must know which windows were chosen for this train. Hence, each time window gets a unique identifier that is later used by the DVL to inform the agents which route was chosen. This goes one step further, because the DVL must stitch routes together that match. Considering our simple example above, if one agent sends a window that lets the train exit at B1 while the other agent has a matching window but for A1, it won't work. Ergo, time windows must be accompanied by both an identifier, as well as the entry and exit track names.

The algorithm may be improved in speed by allowing the agents to send back at what time they will proceed to send more windows. This ensures that the DVL does not have to send times in vain. This change does not influence the agent's working, because they can easily deduce from their priorities when the next compromise would occur. Another improvement might be to enforce consective TCAs that belong to the same agent to be matching with regards to tracks. This change conceptually groups the responsibility where it should, namely the single agent responsible for that part of the route. For this to work, however, the protocol must be redesigned. Time windows then no longer apply to one TCA but to a sequence of them. Hence, activity times cannot readily be distilled from these windows anymore.

Extending this protocol to include multiple DVL agents is trivial. Conceptually, agents push the time horizon forward until a suitable path is found. Using more DVL agents only adds one layer of agents that push the horizon, while also executing in parallel. The initiating DVL agent needs to only merge the partial routes together. If that fails, he pushes on and requests a new route from colleage DVLs.

5.3.3 Selecting good routes

Choosing a route through the PPLG is based on a multitude of factors. For instance, the size of platforms or the orientation of track and platform may prevent some wagon forms to use a track. The dimensions of the train's wagons are known beforehand and any deviations from normal are listed as "out of proportion" (Dutch: Buiten Proportie). Throughout the Netherlands, a handful of these restrictions are in order [15]. Moreover, an often heard rule of thumb (personal communication) [45, (Appendix A)] is that cargo trains should not use platform tracks when passing through a station. Furthermore, if a train must halt at a station, the operator must take into account the length of the tracks in order not to block paths of other trains. Finally, forcing a train over different switches might incur a speed loss because different switches may only be used with a certain speed.

Much of the initial priorities among the above factors can be learned from the daily time table. However, as shortcomings in the plan are sometimes corrected by the operators or as a personal preference overrides default behaviour, a more dynamic approach must be taken to find the realisation data. ProRail's *Sherlock* program aggregates this information. It discloses both the static time table as well as the routes as they are executed. A final option for changes to these patterns is when calamities occur. The rules applied for routes of trains may diverge even more. Using historical data of incidents these deviations can be recognised. Machine learning techniques can be used to find patterns in the data, yet another option that can be considered more agent-like is explored here.

Of all possible routes, those that violate the BP conditions are cut. Then, agents may choose to put a high priority on the platform criterium, or on the infrastructure/switch behaviour, or maybe on following scheduled behaviour. These chosen prioritisations can then be matched against known previous behaviour of operators. This information can be used to tweak the behaviour of the agent's choices. However, in order to enable playing future scnario's lacking user information, the agent must evolve such that we can trust it to make realistic choices. This may also lead to the definition of several attitudes to solve the route problem, but the effects of these attitudes must be well defined in order to be useful. Then, research can be done on the effects of user attitude on the routes of trains and the effects on the whole time table realisation.

5.3.4 Levels of automation

The agents can be put to work at different levels of autonomy. These levels are related to the ease of use and conversely to the trust accredited to the system. For example, at the lowest level, an agent performs the planning of the new train and proposes its result to the human user. Its task is that of

a decision support system. The user is fully responsible for the new train and he only uses the agent system to initially guide his actions. The system cannot yet be trusted to do the right thing. When that trust increases, users may decide to only verify the most difficult trains and accept the others as they are presented by the agent. This potentially saves a lot of time, but the user is still responsible for the actions of the system. The last step is when the agent gains full responsibility over his actions, gained from extensive testing and proofs of validation. At this point, the agent enters new trains in the system itself, freeing the user from the burdon to check its every action. Note that it is not necessary that the user loses all control. He may disable the agent and perform the planning himself, just as what happens with other automatic systems like ARI.

5.3.5 Track subscription

On any double track, one side is usually for one direction and the other side for trains coming from the other side. Hence, when creating plan rules for new trains, the computer system may already fill in the correct track names if the direction is known. This process is already present in the operational traffic control systems. However, PRL-Game lacked this funcitonality. This provided a great opportunity to learn from the original design and still being able to improve it. This part is not inherent to the agent's design, but the agent needs it in order to automatically fill plan rules, so it is discussed here. Much of the information found here was acquired by questions in person, so it cannot be referred to.

The route of a train is a sequence of TCAs. These are defined when the infrastructure model is made and are considered static information in this context. Of these TCAs, some are operated and some are not. Because TCAs are simply points of interest, they may not contain switches which renders them useless. Nevertheless, PRL receives all TCAs and filters out the unoperated ones. This leaves the system with a bunch of TCAs. The systems does not care about TCAs but wants plan rules for PPLGs. Thus, TCAs must be converted to plan rules. Because operated TCAs all belong to some PPLG and routes are defined on the level op the PPLG, not every TCA needs a route. Operators may assign a route to every TCA, which causes the route of the train on that specific PPLG to be split up, which gives the system and himself some freedom by not having to free the complete route. Instead, the route is freed and assigned in parts. The conversion from TCA sequence to plan rules is lossy and difficult [3].

Next, for all operated TCAs found in a region, only those are selected which are actually interesting for routing purposes. Note that all of them are theoretically useful for routing so this step seems a bit unecessary. Finally, the regional admin enters track names for common sequences of TCAs. The

format is

$$FromTCA, ToTCA, FromTrackName, ToTrackName$$

. This system is not designed and not used for filling platform tracks. It is deemed that operators will always want to select a platform track manually. This simple system is called the *Track Subscription* system (Dutch: Spoorabonnement). There are several drawbacks and shortcomings of this sytem, taking into account that PRL agents want to create complete plan rules:

Force The format presented above lists only track names. However, a route is slightly more than simply naming tracks. Due to restrictions of the initial time table, or for any other reason, it may be interesting to add the force field to the subscription. For agents this is crucial, because force is a major part of the choices for the route.

Train characteristics The subscription as presented here does not take into account what type of train will follow this route. For instance, in areas where there are four parallel tracks, two out and two back, cargo trains may be sent over the outer track while passenger trains must take the inner track in order to reach the platform (see figure 2.5 where this is the case). Or a distinction may be made between stopping trains and continuing trains taking different tracks. These options are lost by not taking into account on what type of train the rule will be applied. Filling tracks may therefore result in unwanted situations, leading to a situation where these tracks are not filled in in order to avoid accidents. This kind of attitude is unwanted.

Platform tracks Filling in tracks for platforms is not done. However, if there is little choice or if there are clear rules of thumb that are usually followed, it might be a bonus if these tracks were filled. For agents, the ability to do this is crucial, as one of their strengths is that arbitrary stops may be planned for cargo trains, meaning they must be able to use platform tracks.

Unused rules As discussed above, only TCAs that are deemed interesting are used to create routes. For our purposes, we can assume that this encompasses all operated TCAs. This however means that each operated TCA will generate a potential plan rule, while in reality, only a single rule is wanted. There is no way to ignore certain TCAs and fill fitting tracks for the other TCAs.

Route splitting In general, but more specific in the case of cargo trains, train operators can split a plan rule into two parts. This partitioning will decrease the stress of the rail network if done correctly. The

current track subscription system cannot advise the user to create a certain split route. Again, agents may want to opt for split routes if it were advanced enough to compute the advantage of that. For instance, splitting a rule may enable another train to go in front of this train, while a non split rule would prevent the other train from using the tracks to get there. In the former case, there is better use of the infrastructure and more trains can be planned.

The system that was designed for human use and agent use was built on the knowledge described above. Taking into account that it was being created in PRL-Game, it did not have access to all information which steered its design. Two main changes to the above format were made. First, instead of simply naming tracks, complete routes can be defined. Nevertheless, if no full route is wanted, parts can be omitted, rendering the resulting plan rule partly filled in. By allowing such freedom, the generated rules can be made such that the operator is automatically steered into the right direction when choosing the route. Second, the design was made fully modular with regards to train type. The way that is done is by using the train series number. This number gives a good impression what type of material the train will consist of. For example, the 3500 series is an intercity service. Instances of this series are the 3515, 3517 and so on, which ride each half hour. When splitting a 3517, the front part may continue to use the number 3517 while the second part may get the number 403517. When cancelling trains halfway on its route, the original train will ride with its own number until the cancelling point, while the part of the train further along the route will be called 303517. (Obviously, the material itself cannot go and perform the 303517's service because it was cancelled. Local operators will have to juggle in some other train to take over the 303517's duty.)

What follows is a description of all elements to a subscription rule. The model is more advanced than the original.

Train number Default (-1) series, other series number, or train number

PPLG Name of the PPLG for which this rule exists

From TCA Name of the TCA from where the train will come

To TCA Name of the TCA to where the train will go

Sequence number Number that is used for route splitting

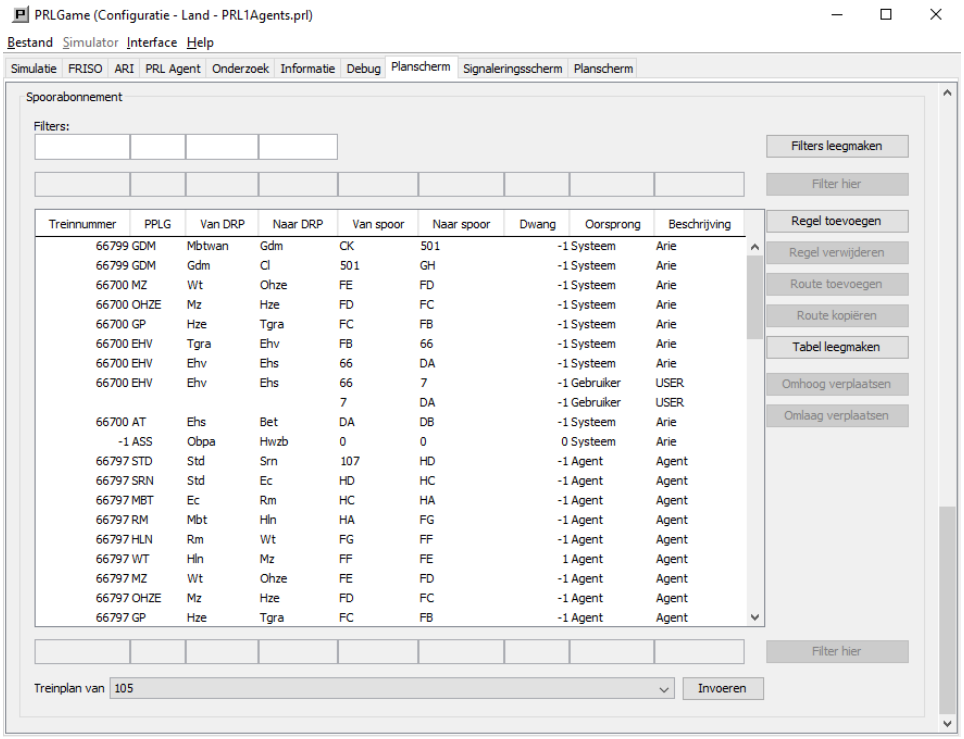
From track Name of the from track of a plan rule

To track Name of the to track of a plan rule

Force Number of the forced route. Empty means default route.

Cause Identifies who created this subscription rule

Description Optional description for this rule



Treinnummer	PPLG	Van DRP	Naar DRP	Van spoor	Naar spoor	Dwang	Oorsprong	Beschrijving
66799	GDM	Mbtwan	Gdm	CK	501	-1	Systeem	Arie
66799	GDM	Gdm	Cl	501	GH	-1	Systeem	Arie
66700	MZ	Wt	Ohze	FE	FD	-1	Systeem	Arie
66700	OHZE	Mz	Hze	FD	FC	-1	Systeem	Arie
66700	GP	Hze	Tgra	FC	FB	-1	Systeem	Arie
66700	EHV	Tgra	Ehv	FB	66	-1	Systeem	Arie
66700	EHV	Ehv	Ehs	66	DA	-1	Systeem	Arie
66700	EHV	Ehv	Ehs	66	7	-1	Gebruiker	USER
				7	DA	-1	Gebruiker	USER
66700	AT	Ehs	Bet	DA	DB	-1	Systeem	Arie
-1	ASS	Obpa	Hwzb	0	0	0	Systeem	Arie
66797	STD	Std	Srn	107	HD	-1	Agent	Agent
66797	SRN	Std	Ec	HD	HC	-1	Agent	Agent
66797	MBT	Ec	Rm	HC	HA	-1	Agent	Agent
66797	RM	Mbt	Hln	HA	FG	-1	Agent	Agent
66797	HLN	Rm	Wt	FG	FF	-1	Agent	Agent
66797	WT	Hln	Mz	FF	FE	1	Agent	Agent
66797	MZ	Wt	Ohze	FE	FD	-1	Agent	Agent
66797	OHZE	Mz	Hze	FD	FC	-1	Agent	Agent
66797	GP	Hze	Tgra	FC	FB	-1	Agent	Agent

Figure 5.1: The track subscription screen. It belongs to the Planscreen tab, as that is where the tracks are being entered. Not only are all subscribed rules displayed neatly, but the user can also filter on selected columns. The two filter rows above and below the table indicate the previous and next subscription rule along the train's route, regardless of the filter. Moreover, the bottom row enables the user to automatically create rules for an existing train, generalising them for use by other trains of its series.

Inherent in the design of this system is *specificality*. Rules for the default train series, -1, are applicable to every train. However, if there is also a rule for the 5700 series, that rule will be taken rather than the -1 rule if the train belongs to this series. More specifically, if there were a rule for the 305700 series and the train had a number in that series, that rule would have priority. The most specific rule has a series number equal to the train number and will therefore only be applicable for one specific train. Yet this specificality continues beyond series numbers. TCA names need not both be filled in, as are track names. For any new train, the sequence of TCAs is filtered into only the operated ones. These are then grouped by PPLG. Now, if there exist no rules specifically for this train and PPLG combination, all TCAs must be displayed to the user and their route must be empty. Otherwise, each TCA that has a matching (chain of) rule is being displayed with the correct route. TCAs that are not found are considered to be not interesting.

Throughout a PPLG there may be several TCAs. Yet before these and after are the neighbouring TCAs. Subscription rules may be defined on any of these TCAs. For each TCA in this PPLG, rules are searched. Each previous TCA may be used in combination with any next TCA. However, if any next TCA is skipped, it cannot create a plan rule anymore. If rules are found, they are sorted based on the cause, where SYSTEM (statically defined) rules have lower priority than USER rules, and AGENT rules have the highest priority. The remaining rules are then sorted based on their specificality. The first chain of rules is finally applied to the TCA.

The only thing that has not been addressed so far are defining platform tracks. However, the definition of the subscription rules follows directly from the TCA sequence. If a train performs a stop somewhere, its activities on the same TCA will be an Arrival and Departure. The TCA sequence will reflect this, so the same TCA will be listed twice. This is not the case for route splits, because there we manually create two routes for one activity (effectively splitting it into two activities with the same name). The rule for arriving will have as next TCA itself. Likewise, the departure rule has itself as previous TCA. The To Track of the arriving rule can now be a platform track name and the plan rule will be correctly filled.

Chapter 6

Implementation of the model

6.1 Initial Explorations

In order to get familiar with the way PRL-Game enables changing the train schedule, the train removal "Script" was developed. Changes to the time table are done in the VKL system. Being an independent JAVA application, the source code for VKL had been incorporated into PRL-Game. It had already been noted by the maintainers of PRL-Game that the code was messy and incredibly hard to edit [22] and this was verified in this small project. The state of the code can technically be best described as a tangled ball of static classes, of which most are singletons, that all use a handful of stateful static classes that themselves allow other static classes to set their variables. The Script project has succeeded but is entirely not maintainable. The most viable option for finding out what a handful of code does, is setting a breakpoint at some interesting line and tirelessly stepping through the code flow, making mental notes about which paths are chosen and due to which variables, because the program flow as displayed by the code itself does not reveal any purpose or intent.

VKL is a client-server application. Changes done on one client are propagated to the server for verification and further distribution. Instead of implementing the interface between client and server, mimicking the server but plugging a connection to FRISO in its stead, VKL's source code has been directly changed to support FRISO's messages. An important observation is that it was done in such a manner that it disallowed using more than a single VKL federate. Specifically, FRISO creates a new train and *then* sends its activities one by one, while VKL assumes that a train without activities means a removal. Although the solution looks simple enough, by caching the created train and pushing it to VKL as soon as its activities are received, the code still uses a full system refresh path instead of smoothly incorporating the changes in the displayed model. The integration of VKL into PRL-Game was deemed sufficient for the time being and hence did not

necessitate implementing VKL to allow multiple VKL federates. If a second VKL federate would dynamically change a train, the first federate would not synchronise that change.

6.2 Model Creation

The main trouble that appeared during the development of the agent software was that PRL-Game does not have access to infrastructure data. This means that the PRL agent cannot plan routes on its own. Neither can the DVL agent compute exact travel times. The visualiser for the infrastructure in PRL-Game was using a slightly disjunct set of data already, leading to subtle yet persisting bugs. Although there exist infrastructure deliverables, preliminary research concluded that the data was difficult to read and often incomplete. Furthermore, FRISO changes the data when it imports it into its own database, so this approach was not guaranteed to work. A workaround was needed.

6.2.1 ROBERTO

FRISO offers a run configuration called "ROBERTO" that rides each train in the schedule individually, noting every infrastructure element it passes, its speed on each of those points, the signals that are encountered and routing information. From this data, conflicting routes can be computed. So to use this feature, the train that we want to create dynamically using the agent must initially be created in FRISO. To enable multiple routes to choose from, multiple trains must be created in FRISO. Then, the whole time table can be executed with ROBERTO which gives us the exact routing information.

There was no documentation available for the output of the ROBERTO computations, so empirical analysis was needed to figure out how to use the data. It appears that as soon as the train is placed in the model, the infra elements are listed beginning from the *back* of the train. This was evident because the speed on the first several sections was zero. Then, during its journey, all signals, switches, sections, isolators and other elements are listed. For our purposes, only the section names are of importance and the direction of which the train passes the section.

Listing 6.1: Example ROBERTO input data.

```
<ROBERTO>
  <ROBERTOINPUT ROBERTOINPUT_VERSIE="1.2">
    <TREINEN>
      <TREIN id="1" naam="1011-T-1" materieelTypeId="
          56" beveiligingsType="NS54" treinSerie="1011"
           treinType="HS" treinGebruik="R" treinLengteM
```

```
 ="177" bronDienstregeling="Donna BasisDagen
 20150907-BD-002 van 2015-07-03">
<TREINPAD>
  <SPOORTAKSTUKBEGRENZER id="13347" naam="
      Hsghtn_HART1" type="DIENSTREGELPUNT_HART"
       lengteVanafVorigeBegrenzerM="185"
      volgnummer="1" richting="T" kilometerLint
      ="Rtd-Hfd" kilometrering="137300"
      sectieNaam1="1133AT-7"
      dienstregelpuntCode="Hsghtn"
      voorkantPassageTijdstipS="11.000225"
      voorkantSnelheidMS="44.4444444444444"
      voorkantVersnellingMS2="0"
      voorkantIntervalS="-99999" remAfstandM="
      2582.10279996772"
      achterkantPassageTijdstipS="14.982725"
      achterkantSnelheidMS="44.4444444444444"
      achterkantVersnellingMS2="0"
      achterkantIntervalS="-99999" />
  <SPOORTAKSTUKBEGRENZER id="13331" naam="
      HSL16" type="PERMISS_SEIN"
      lengteVanafVorigeBegrenzerM="1675"
      volgnummer="2" richting="M" kilometerLint
      ="Rtd-Hfd" kilometrering="138975"
      sectieNaam1="1133AT-7"
      dienstregelpuntCode="Hsghtn"
      voorkantPassageTijdstipS="48.687725"
      voorkantSnelheidMS="44.4444444444444"
      voorkantVersnellingMS2="0"
      voorkantIntervalS="37.6875" remAfstandM="
      2582.10279996772" seinbeeld="GR"
      eindeRodeGolf="false" isHoogSein="true"
      achterkantPassageTijdstipS="52.670225"
      achterkantSnelheidMS="44.4444444444444"
      achterkantVersnellingMS2="0"
      achterkantIntervalS="37.6875" />
  <SPOORTAKSTUKBEGRENZER id="13302" naam="1133
      AT-7" type="SECTIE"
      lengteVanafVorigeBegrenzerM="11"
      volgnummer="3" richting="T" kilometerLint
      ="Rtd-Hfd" kilometrering="138986"
      sectieNaam1="1133AT-7"
      dienstregelpuntCode="Hsghtn"
      voorkantPassageTijdstipS="48.935225"
```

```
voorkantSnelheidMS="44.4444444444444"
voorkantVersnellingMS2="0"
voorkantIntervalS="0.247500000000002"
remAfstandM="2582.10279996772"
achterkantPassageTijdstipS="52.917725"
achterkantSnelheidMS="44.4444444444444"
achterkantVersnellingMS2="0"
achterkantIntervalS="0.247499999999995" />
<SPOORTAKSTUKBEGRENZER id="13301" naam="
LAS14:45:15" type="LAS"
lengteVanafVorigeBegrenzerM="12"
volgnummer="4" richting="M" kilometerLint
="Rtd-Hfd" kilometrering="138998"
sectieNaam1="1133AT-6" sectieNaam2="1133
AT-7" dienstregelpuntCode="Hsghtn"
voorkantPassageTijdstipS="49.205225"
voorkantSnelheidMS="44.4444444444444"
voorkantVersnellingMS2="0"
voorkantIntervalS="0.270000000000003"
remAfstandM="2582.10279996772"
achterkantPassageTijdstipS="53.187725"
achterkantSnelheidMS="44.4444444444444"
achterkantVersnellingMS2="0"
achterkantIntervalS="0.270000000000003" />
<SPOORTAKSTUKBEGRENZER id="7623" naam="FV"
type="SPOOR" lengteVanafVorigeBegrenzerM=
"1040" volgnummer="5" richting="T"
kilometerLint="Rtd-Hfd" kilometrering="
140038" sectieNaam1="1133AT-6"
vrijebaanCode="Ght_Hfdght"
voorkantPassageTijdstipS="72.605225"
voorkantSnelheidMS="44.4444444444444"
voorkantVersnellingMS2="0"
voorkantIntervalS="23.4" remAfstandM="
2582.10279996772"
achterkantPassageTijdstipS="76.587725"
achterkantSnelheidMS="44.4444444444444"
achterkantVersnellingMS2="0"
achterkantIntervalS="23.4" />
<SPOORTAKSTUKBEGRENZER id="13332" naam="
HSL15" type="PERMISS_SEIN"
lengteVanafVorigeBegrenzerM="1009"
volgnummer="6" richting="M" kilometerLint
="Rtd-Hfd" kilometrering="141047"
```

```
    sectieNaam1="1133AT-6" vrijebaanCode="
    Ght_Hfdght" voorkantPassageTijdstipS="
    95.307725" voorkantSnelheidMS="
    44.4444444444444" voorkantVersnellingMS2=
    "0" voorkantIntervalS="22.7025"
    remAfstandM="2582.10279996772" seinbeeld=
    "GR" eindeRodeGolf="false" isHoogSein="
    true" achterkantPassageTijdstipS="
    99.290225" achterkantSnelheidMS="
    44.4444444444444"
    achterkantVersnellingMS2="0"
    achterkantIntervalS="22.7025" />
<SPOORTAKSTUKBEGRENZER id="13299" naam="1133
    AT-6" type="SECTIE"
    lengteVanafVorigeBegrenzerM="13"
    volgnummer="7" richting="T" kilometerLint
    ="Rtd-Hfd" kilometrering="141060"
    sectieNaam1="1133AT-6" vrijebaanCode="
    Ght_Hfdght" voorkantPassageTijdstipS="
    95.600225" voorkantSnelheidMS="
    44.4444444444444" voorkantVersnellingMS2=
    "0" voorkantIntervalS="0.292500000000004"
     remAfstandM="2582.10279996772"
    achterkantPassageTijdstipS="99.582725"
    achterkantSnelheidMS="44.4444444444444"
    achterkantVersnellingMS2="0"
    achterkantIntervalS="0.29249999999999" />
<SPOORTAKSTUKBEGRENZER id="13298" naam="
    LAS14:40:49" type="LAS"
    lengteVanafVorigeBegrenzerM="15" ...

  ⋮
```

Looking at the data, every SPOORTAKSTUKBEGRENZER, which is a thing along a track and will be called *separator* in this part, is listed in order of encounter. It lists TCA hearts, permissive, automatic and standard signals, sections, isolators, tracks and more. Every such item has a unique id. Elements have many attributes which cannot be displayed on a single sheet of paper. Those include speed of the train, acceleration, location, time stamp, TCA or free track code, distance from previous separator, sequence number, direction, and more.

For our current purposes, only the actual route of the train is important. We define it here as the sections that are encountered. We want the train's route on each TCA, but not on free tracks. Free track is not considered because the entry section uniquely determines on what section the train will

end up. This will always be in another TCA. Ergo, it is only needed to extract the route up to the next free track. Judging from the input data, it is possible that two TCAs connect directly. TCA name is defined in the key *dienstregelpuntcode* (TCA code) and free track names are defined in the key *vrijebaancode* (free track code) and they are mutually exclusive. Because these keys are not always filled with a non-null value, a sequence of section names is defined to lie in one TCA if the TCA code remains the same, excluding empty values. If the section name becomes empty, it only signals the end of the TCA if the element has a non-empty vrijebaancode.

Isolator elements connect two sections. This means that such an element has two *sectieNaam* elements. These are not ordered, however, so special logic is needed to deal with this. It appears that there is no fixed order of infrastructure elements, so in theory it is possible to encounter an isolator followed by another isolator. In this case the section name in between is missing, but this is not a problem. In order to find the very first section the train is on, either the first section element that is read is used, or the isolator combined with the next section name determines the first section, or the second isolator determines what the very first section would be. Consider that the first isolator has sections A and B, and the next section is named A, we know that the first section was B. Likewise, if the second isolator has D and E, there is an error. Otherwise, if it is A and C, we know the first section was B, because the common section was A.

The route of the train is deduced from the above. Because either the section elements or the isolator elements could be used to list all sections that were encountered, it was decided that this reduncandy would be used to verify the correctness of the route. Hence, the section elements would forcefully set the current section, while isolator elements would verify that the current section was indeed the section that was isolated. If the isolator element found that the iteration state was still correct, it would add the last seen section to the route. As soon as the current TCA was lost, the last seen section was also added to the list, regardless of whether a final isolator was encountered.

Armed with the routes for each train, how should the appropriate planning norm be selected? First off, checking norms is only relevant if the route of two trains conflict. This can easily be verified by iterating through all sections of both routes at the same TCA and checking if at least one of them has the same name. However, based solely on the sequence of sections, finding the train's direction is more complicated. If the number of common sections is more than one, the process is straightforward: if the sequence of one train is the other's reverse, the conflict is *cross back*. Otherwise, if both endpoints are the same, it is *followup*, otherwise it is *cross to*. However, if both trains only have one section in common, this approach is not viable.

Because finding the direction of a single section conflict poses a problem, external information is needed. One element of the train route is particularly

useful for this. TCA Hearts (DIENSTREGELPUNT_HART) are located on each track in a TCA and determine where delays are computed. Conceptually, they can be thought of as the centre of the TCA. Now, the route of a train is split between those section that come before the heart and those that come after, relative to the direction of the train. The conflict is now easy to resolve, because if the single section appears in the same list of sections (either both *towards* or *from*) the heart for both routes, the conflict is *cross to*. Otherwise, it is *cross back*. It can never be a *followup* if there is only a single section in common.

Determining the relative position of the heart along the route is done by keeping a flag during parsing, which flags whether the heart has been seen. As soon as the heart element is read, the flag is flipped. Depending on the state of the flag, sections made final by reading the isolator element are placed in the correct list. Dummy section names are inserted in the list if the heart element is the first or last element to be read to prevent dealing with *null* values.

Arrivals and Departure activities complicate the matter further. A train that arrives at a station now may depart after several hours. Potential conflicts during arrival are thus not relevant anymore during departure. However, the routing information does not care about activities and lists the whole route as a continuous sequence. Fortunately, the ROBERTO input file also lists every activity of the train, including time stamp and last seen separator ID. Interestingly, K activities are listed twice there, the first entry being the arrival time and the second the departure time. For our purposes, K activities can be considered to be one activity with a route, and the separator IDs of the A and V activities can be used to create proper routes for each of them.

6.2.2 Train plan for templates

Superficial route information on TCA level was already available in the VKL federate. Because creating a train programmatically from scratch in VKL is a mess, it was decided to persist an existing train and reload it when an agent needs to create a variant of that train. The only change that must be made to this clone is the actual activities, because the agent may introduce stops along the route. The decision to persist the train *outside* of PRL-Game was made because otherwise a dummy train had to be always present in the train schedule. The Train class was changed to support writing its internals to XML. This was done recursively, because train objects have delay objects and activity objects. The XML structure that was created was saved to a file that could be specified in the configuration.

The single dependency for exporting trains is the ROBERTO routing information. So before saving the trains to file, a separate ROBERTO run must be done, yielding the text file containing the route. After linking the

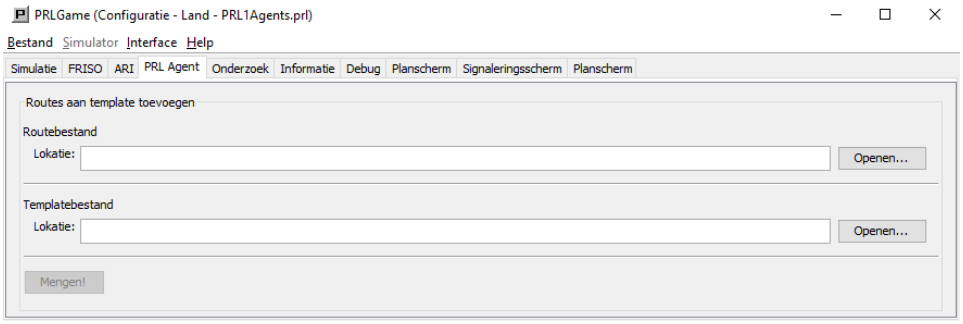


Figure 6.1: The PRL Agent module offers a way to merge the route file with the template file.

file in the configuration, the trains that are needed for creating new trains can be exported. This export is an XML file containing both the train's details from VKL as well as the ROBERTO information extracted from the linked file. This resulting file will be called the *templates file*. It contains different options for a final train: one template may pass through a station (D) on track 705, while another template follows the exact same route, but performs an A+V on track 702.

A detail that is overlooked upto now is the data needed for creating plan rules. While platform tracks might be deduced from the ROBERTO information, free track names are not so easily defined. The final step to creating a workable templates file is by adding plan rule information to each activity. The names of the tracks and the route numbers were extracted from the time table in FRISO. There, each template has its own plan rules and these could be easily copied. An excel document was created containing a sheet per train and the route per activity. It is possible to add split routes for activities. The configuration was then amended with the option to merge both excel file and templates XML into the final templates file.

6.2.3 Modules

Orders are modelled as events that arrive in the system at a specified time. All events are specified in advance in a dedicated excel file. Only the most important fields are given, namely event time, train number, from TCA, to TCA, ready time, deadline, from track, to track, and template numbers. As each order is read at simulation start and should only fire at a certain simulation time, PRL-Game modules were used. The module was registered as a listener to the simulation clock, and it supported access to the orders file. The orders would be parsed when the module was created and stored as event objects in a priority queue. At each time step, the head of the queue would be inspected and if its due time is before the current time, it is executed. This is done for each remaining event with due time in the

past. This method ensures that orders can be dynamically loaded in the system, either by manually creating orders, by loading the orders file while the simulation is running, or by simulating a hickup in the order delivery channel.

Both the PRL agent and the VKL agent are modelled as modules. They both extend from the Order Module. As both agents need information of the template trains, the data model is created in this parent class. However, each module only uses information that is relevant for its job.

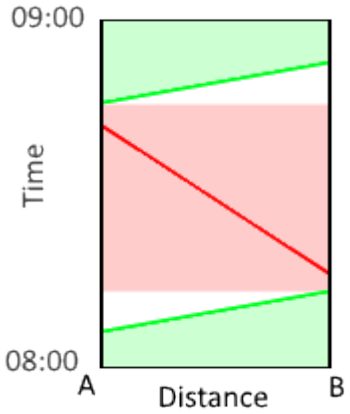
6.2.4 VKL agent module

The VKL agent handles orders on the level of TCAs. From the order, the template train numbers are read. For each of these dependent numbers, the timetable is deduced from the template file. For each of the TCAs in the train's schedule, a time window model is created based on the schedule of all other trains travelling in that TCA. Between each two sequential TCAs of the template train, the agent checks if there are other trains doing that exact crossing.

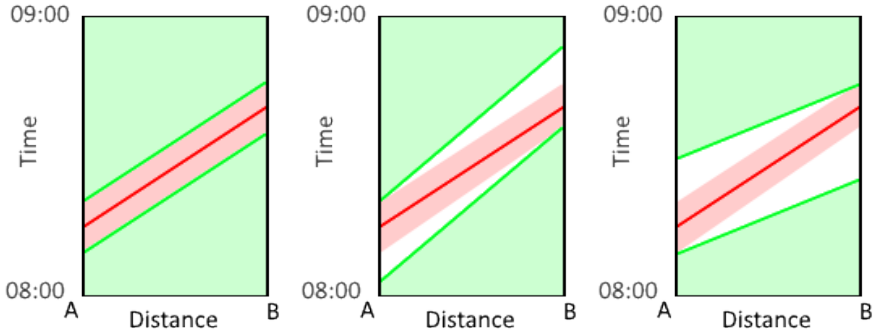
The two TCAs can be named *this* and *that*. The route in this TCA ends with a certain section. If there is any route of the other train that lies in this TCA and contains our last section, we have a conflict. The next step is to create the diagonal line. For that, the non-arriving activity of the other train in this TCA is taken, as well as the non-departing activity in that TCA. The activity times give the slope of the line of the other train. Our time window in this TCA is thus intersected with a time window of the other train's departure time, just like the time window in that TCA is intersected with a time window of the other train's arrival time.

Based on the direction of the conflict, the intersecting time window is different. If the direction is *cross back*, the free track is assumed to be occupied until the other train finishes his route. Assuming a 3 minute norm, the window becomes a block form beginning at three minutes before departure of the other train, until three minutes after arrival. Conversely, *cross to* conflicts need to take the slope of the other train into account. If the other train has the same travel time, the blocked time window runs parallel to the train. Otherwise, for example if the other train is *slower*, the blocked window does not follow the other train's slope. Instead, it ends three minutes after arrival, but it does not have to start three minutes after departure. Rather, it starts as many minutes earlier as is our travel time.

After all these conflicts have been processed, the agent has a model of free space over time. For each template train, it sends a request to check it to all PRL agents. It is assumed that TCA are controlled by only one agent, so the VKL agent waits until for each template, each TCA's time windows are verified. Because these templates all belong to a single order, they are sent as such. Moreover, the message contains the order's *from* and



(a) Train conflict where trains travel in opposite direction. The other (red) train creates a box around it. This box contains a certain margin around the train that represents the generic norm that is applicable in this case. The train that we want to add to the schedule (green) can go anywhere in the green area. The white area is not reachable because of the train's travel time.



(b) Train conflicts where trains travel in the same direction. The other (red) train creates a margin around itself that represents the generic norms that are applicable. Note that these margins do not have to be parallel to either other line, as ordering (as in: first or second train) determines the norm, as well as activity, which are not necessarily the same at A and B. The green area is where the new train can go. The white area poses no conflict, but cannot be utilised because the green train's travel speed must be taken into account.

Figure 6.2: Train conflict types.

to tracks that are needed by the PRL agent. Sending messages is done using PRL-Game's own message handling system. Treatise of this agent's logic will continue after the PRL agent has responded.

6.2.5 PRL agent module

The PRL agent receives multiple trains that each have multiple TCAs that each possibly have multiple time windows to be checked. By looking at the schedule of all trains, the received time windows will be *adapted* to reflect the agent's knowledge. Because this agent only needs to respond for the TCAs that are operated locally, a check is done to see if the TCA is known at all, and if it is, if its PPLG's ARI state is not inactive. For these operated TCAs, the complete timetable is checked to see if any train passes the TCA. If it does, and the routes conflict, the time windows may be adapted.

As the PRL agent considers track occupation, a special case must be made for arrival and departure movements. Whereas trains conflict with all other trains using the same connecting track between TCAs, platform tracks may be unoccupied for a significantly longer period of time. We must therefore make an additional distinction between arrivals and departures, which were called *stops*. Any ongoing train intersects the arriving window, while the stopping window remains intact as long as no other train uses that track. The train may arrive at its track if its arriving window is clear, waiting at its stopping track as long as the stopping window is clear, and departing in any clear departure time window.

Special care must be taken when dealing with train relations. When a train combines with another train, the track occupation of the first train should be merged with the occupation of the second. Likewise, whereas the second train only has a relatively small occupation, the occupation of the first train must also be taken into account. This special case only occurs however on train arrival and departures.

Depending on the type of conflict detected, the type of activity the new train will perform and the other train's relations, the time windows are updated. For conflict handling, the generic planning norms are used. As soon as all other trains have been checked and all windows adapted, the PRL agent sends the new schedule to the requesting DVL agent. Note that the stopping activities are also transmitted, as they will be needed for creating the final plan.

6.2.6 Aggregating the results

The initiating DVL has the grand task of consolidating all responses into one single train. There are time windows for each of the template trains, created independently by each PRL agent. The DVL agent attempts to find a result by building a graph. The templates differ on a few key points. There, one

template may perform a passthrough while another halts on track A, while another train halts at platform B. These are called the *choice points* of the graph. Another choice lies in a single template. The agent is free to choose the staying time of a train on a station. It might pick a departure time that further along the route leads to an impossible situation. It can then backtrack to this branching location and choose to depart at a later time.

It makes sense to aim for a train schedule that results in as small travel time as possible. That way, the infrastructure is used most optimally. However, the DVL agent must also adhere to the suggested departure time as found in the order specification. Hence, taking the earliest possible departure time in consideration, the earliest arrival of all possible paths through the graph is not guaranteed to be the best route. From the earliest arrival, the latest possible departure time should be found. This last path is the shortest and therefore preferred path. In order to find this option, the agent naively computes all paths and finds the shortest one.

When the DVL agent settles on a specific option, or is unable to find any path, it informs his fellow agents of the result. The other agents are interested in what exact route the DVL agent has chosen. It may be the route over track A, but also over B. The trdl agents can create track subscription rules for this specific new train such that, as soon as the system has processed the creation of the new train and the usual messages are passed asking to create tracks for the new train, the correct route information is present.

Chapter 7

Experiments, validation and results

Ultimately, a single experiment was conducted. This had everything to do with the scale of manual operations needed to create useful initial data. For the scenario, the then-current SCHIPHOL00 was chosen. It is an extract of the current infrastructure and train schedule of 2017. The use-case for the scenario was to test the effects of an innovative way to handle emergency calls in the Schiphol Airport tunnel. The effects studied were located around Schiphol, Amsterdam and specifically the so-called A2 corridor, passing through Utrecht and Geldermalsen into Den Bosch, Eindhoven and further. It was on this stretch of the network that the MAS was tested.

On a practical note it should be said that the MAS experimentation happened in close connection with the development of the above scenario. Therefore, at the time that a suitable freight train had been chosen to base 'new' trains on, no cargo trains had been tested yet. These tests would change the train schedule such that the simulation would execute it properly. Moreover, due to issues in the scenario that would later be solved, certain trains ((1)3500 series) were taken out of the model because their train relationships were not correctly configured. It was practically impossible to merge the model in progress with the changes made to the model for use with the multi-agent system. Hence, the experiment was done in the face of some missing trains and with untested cargo trains.

One such cargo train runs from Germany, through Venlo on the A2 corridor, passing through Amsterdam and then follows its way to the North Sea coast. It was this train that was chosen to recreate. The first thing that had to be done is figure out possible routes. It was deemed unnecessary to consider multiple global routes to get to the destination, so only the route along the A2 corridor was taken. Along that route, however, are many choice points. For instance, it can pass through Geldermalsen or utilise a stopping track. If it does so, dos it go there by track X or track Z? It can get as

specific as choosing which switches to set. But not only does this apply to Geldermalsen, the same choices can be made for Den Bosch and Utrecht.

For this experiment three template trains were made based on the original train. The original passes through all stations, so two extra stops were created. One of them in Eindhoven, the other in Geldermalsen. Each of those stops could happen on two separate tracks, yielding no more than two extra templates in total. To create these schedules, the scenario's train schedule was consulted using FIC (FRISO Incontrol Center), an application that holds all data needed for a simulation. The train had to be copied twice, after which for each activity the planned times and claim times had to be adapted depending on the length of the stop. It was decided to set the scheduled stopping time at 2 minutes, knowing that the agents would change the final schedule due to safety and feasibility considerations.

In addition to those two templates, 20 others were made using the same strategy. These 20 were not even all imaginable routes for Eindhoven. Then all trains were run individually, yielding ROBERTO information about track utilisation. This data was read into the application in advance and coupled with a manual file that held the tracks of each templates. Finally, the newly created trains were removed from the scenario except the original. An order was created that required the new train to be created at 1 minute past 7 in the morning. The deadline was three in the afternoon, a large enough time frame to place the train. As the initial attempt to use all 23 options for templates resulted in a too long computation time, it was settled to use only three templates.

The workplaces of the operators were divided in only two. One operated as DVL, the other as trdl. As this workplace operated all places, it could also be called the gameleader workplace. This set-up was refined until no bugs were present and was presented to current traffic controllers, colleagues of the Innovation department at ProRail, but also ICT specialists and other office staff working with planning trains. This proof of concept was enough to stirr interest, although many were critical as to its use in real systems as-is at the time. The most visible improvement of the MAS technique was that the creation of a new train took only 6 seconds, while otherwise 20 minutes would have been needed on average.

There are three main results of this experiment and therefore this thesis. First, this multi-agent system shows that it is possible to automate on a small scale using relatively simple software. This system that is not nearly as complex as otherwise used software is already able to show promising results. It is not necessary to spend a lot of money to have a system built that performs this task. Second, it shows that is is possible to use personal agents to assist in the operator's tasks. Users are surprised that it is possible to show a system that proposes a train schedule in their systems. They still hold responsibility for the final schedule and this is reinforced by allowing the operator to change the proposal. Nevertheless, the steps that are taken

automatically relieve the users of having to do repetitive actions.

The third result is that it is possible to create a planning system in a live and dynamic environment. By restricting the agents to only plan between existing trains and preventing them to change existing schedules, the planning problem they solve becomes easier. Obviously, generating an optimal planning for each new train takes far too much time and does not yield a tractable time table. Is it also not necessary to have a correct or safe or executable time table to start with. The agents only search the free space as dictated by the current situation.

Chapter 8

Conclusion and future research

Solving the problem of creating a safe path for a new train in a dynamic, currently existing train schedule was done naturally using a multi-agent system. Because the original procedure for planning trains was already based on the interaction between multiple operators along the hierarchy of train traffic control, the modular setup of the MAS proved to be an elegant solution. Moreover, using this modular approach stretches further than simply the architectual design part: separating the interests of the agents leads to the desirable situation where agents can be freely turned on or off, tuned or otherwise changed based on the required functionality. For example, if one wished to research the impact of the task of train planning for a local operator, a simulation could be run where the other operators were simulated using agents. That way, only a single participant has to be in the simulation. Another possibility is to vary the levels of automation of the agents, or the strictness of their following of the official rules. The MAS provides an easy paradigm where such interactions are inherent in the design, they come for free, so to say.

The main research question is 'In what way can agent-based modeling be used to facilitate the work of rail traffic operators?'. By following the subquestions it can be determined whether this research answered the main question.

(a) What is a suitable scope of the project?

One of the tasks of ProRail's rail traffic monitors is to create schedules for new trains in a live, dynamic train schedule. This task deviates from their other work, which mainly involves problem solving during conflicts or incidents. Manually planning trains is a difficult task which is also not fully supported by the operators' tools. The procedure is subdivided into subtasks for multiple operators, which unfortunately translates into a dependence between operators. Busy operators have even more work to do, while colleagues have to wait long for the whole process to be finished. Because the task can be easily defined and is mostly separated from the other work of the operators, it is most suitable for this project.

(b) What is a natural and efficient approahc to building a supporting system?
A multi-agent system provides a unique solution for automating the task of dynamic train planning. Its multi-module nature closely resembles the hierarchy of the process under study. Each operator can be represented by a single agent, that handles the planning, checking and communication between the other participants. Agents can also be used as a temporary substitution for a final integral planning solution, where humans gradually entrust their tasks to the agents until the task can be fully automated.

(c) What is a good choice for agent internals?
The agents must be rational. Their task is largely safety-related and must therefore be precisely specified. The advantage of using rational agents is that human operators can verify the reasoning of the agents. This makes it possible for operators to quickly correct a proposed action of the agents, while also allowing operators to gain insight in decisions that were automatically taken.

(d) How to implement the agents in the available architecture?
The traffic operators' workplace are simulated in PRL-Game. Due to the intimate data sharing protocols employed between PRL-Game and the simulator FRISO, it was considered best to build the agent code directly into PRL-Game. This however prevented the use of more sophisticated multi-agent platforms. The communication capabilities of PRL-Game were sufficient because the agent communication protocol was kept very basic for this proof of concept.

(e) How to find proper sources of data?
There is not yet a standardised set of interfaces to get realistic ProRail data from. Worse, the simulation used data that was slightly modified in order to create a working model. Unfortunately, retrieving that information was not trivial at all. Although the simulator did not expose simple ways to retrieve model data, the FRISO simulator was solely chosen as the source of data. This meant that the data could at least be considered consistent.

(f) How can the result be verified?

Given the difficulty of obtaining a large workable dataset to experiment with, it was impossible to showcase the potential of the multi-agent system. A workable demo was created and the results were manually verified. The demo was then shown to staff and traffic operators, who agreed that the results were feasible. The demo was also shown to managers at nearly all levels of ProRail management, who were all enthousiastic about the potential of this multi-agent solution.

The question of how multi-agent programming can help automate the order acceptation process can be answered with a resounding "*trivially*", as long as the process remains in the current hierarchy of operator responsibility. It is debatable whether the solution to an automatic scheduling system should be sought in a multi-agent system, instead of a linear constraint model or a path finder through a graph of possibilities. However, as long as multiple operators each control separate areas of the total solution space *and* they remain responsible for the safe outcome of the planning process, the use of agents is advantageous in multiple ways. (1) The inherent modularity of the multi-agent system closely resembles the hierarchical model of the problem domain. (2) The rationality employed by the agents can be used to prove that the agent will execute correctly and obeys all safety constraints, and can be used by the operator to understand the reasoning underlying the agent's proposal. (3) Different levels of automation can be set, enabling a smooth transition from a system that proposes solutions to a fully automatic multi-agent system that operates autonomously.

A great difficulty of the scope of the problem involved is the effort it costs to create a working scenario. Because the environment is based largely on legacy software and other proprietary software, streamlining the generation of test data was so involving that only a single scenario could be tested. Despite that, the modularity of the agent system lends itself to easy testing nevertheless. Each module has a strictly defined set of possible interactions so each of those could be individually tested. The lack of integral data made this automatic testing far more difficult and it has not been tried in this research.

The very existence of PRL-Game provided a magnificent kickstart to the research. It was not necessary to create a mockup interface for participants, nor was it needed to convince the ICT department to grant an entry into their systems. The source code of the PRL-Game project was directly accessible and it was straightforward to build a custom processing and messaging system that this MAS is in it. Because of its portability, the whole concept could be presented in any location without the hassle to access resources in private networks. Using an environment like PRL-Game greatly speeds up the time needed to create such a proof of concept.

The hardest part of building the multi-agent system was its internals. There was so much specialised knowledge that needed to be built into the system that the effort greatly overshadowed the initial explorations or the

design and implementation of the messaging system, for example.

In the future, multiple parts of this thesis will be revisited and extended. Firstly, work is already on its way to revise the design of this system. As it currently stands, the program is highly interconnected with existing source and does not use original software for tasks that have previously been solved. For instance, there currently are advancements in creating a working and timely interface with the ROBERTO component that is used to precisely compute data on conflicting trains. A very basic implementation was done for this thesis, but working with the real thing is far more beneficial.

Secondly, the current implementation lacks almost all advanced features described in the module design chapter. Those points should be studied in more detail and built into the system. Only then will the program start to behave as advertised. During that process, more detailed output should be given. For instance, use cases show that it is far more relevant to see why a path cannot be created at a certain time than the very fact that it did not succeed.

Another improvement would be to devise an interface between the agent's internals or behaviour and a potential user. An example of the functionality envisoned herewith is a popup to the user detailing a set of path options, after which the user may give a first choice. This choice should then be communicated to the rest of the agent system. Another possiblity is to decide the robustness of the chosen route, requesting user interaction if the robustness falls below a certain threshold. Allowing the user to interact with the agent system provides both a feeling of control for the operators, as well as a fallback if the system fails to decide on an outcome.

In parallel of this research, ProRail is conducting and already has conducted more and more research using agents. One of those deals with the automatic replanning of trains in the face of a disturbance in the infrastructure. For these systems to communicate and cooperate in a structured manner, this system must be retrofitted to communicate using a fixed communication protocol. This is a good moment to further explore the use of OO2APL [44] and its framework of agent interactions.

Bibliography

[1] Middelkoop, D., Meijer, S., Steneker, J., Sehic, E., & Mazzarello, M. (2012, December). Simulation backbone for gaming simulation in railways: a case study. In *Proceedings of the Winter Simulation Conference* (p. 287). Winter Simulation Conference.

[2] Meijer, S. (2012). Introducing gaming simulation in the Dutch railways. *Procedia-Social and Behavioral Sciences, 48*, 41-51.

[3] Berends, H., van Smeden, K. (2015). Project Start Architectuur, Snel en Veilig Plannen (SVP), (in de bijsturing). *ProRail ICT / CC Be- en Bijsturing* (Concept) (Dutch!).

[4] VL Vakopleidingen (2015). Systemen ARI en ABT versie 4.1 definitief. *ProRail: basisleertraject treindienstleider* (Dutch!).

[5] ProRail (2016). Netverklaring 2016. *http://www.prorail.nl/vervoerders/netverklaring* (Dutch!).

[6] ProRail (2016). Programma Hoogfrequent Spoorvervoer. *http://www.prorail.nl/programma-hoogfrequent-spoorvervoer* (Dutch!).

[7] Rijksoverheid (2016). Spoorboekloos reizen. *https://www.rijksoverheid.nl/onderwerpen/spoor/inhoud/spoorboekloos-reizen* (Dutch!).

[8] ProRail & NS (2013). Beter en meer; Werkdocument: Concept operationele uitwerking van de Lange Termijn Spooragenda. *teamsites.prorail.nl: Operatie ICT LTVL* (Dutch!).

[9] ProRail (2016). Organisatie - Reizigers - ProRail. *http://www.prorail.nl/reizigers/wie-zijn-we/organisatie* (Dutch!).

[10] ProRail (2016). Beheerplan. *http://www.prorail.nl/sites/default/files/prorail_beheerplan_2016.pdf* (Dutch!).

[11] ProRail, staf Verkeersleiding Bureau Productieservices (2010). Werkwijze Verkeersleider. *https://teamsites.prorail.nl/teams/Operatie_VL_RW/oude%20documentatie/verkeersregie/Werkwijze%20Verkeersleider%20uitgave%20december%202011.pdf* (Dutch!).

[12] ProRail, Verkeersleiding (2015). Werkwijze treindienstleider. *https://teamsites.prorail.nl/teams/Operatie_VL_PE/Gedeelde%20documenten/Werkwijze%20en%20Handboek/Werkwijze%20treindienstleider%20uitgave%20november%202015%20definitief.pdf* (Dutch!).

[13] ProRail, Verkeersleiding (2014). Handboek treindienstleider. *https://teamsites.prorail.nl/teams/Operatie_VL_RW/wwvl/Werkwijzen/Handboek%20treindienstleider%20versie%20november%202014.pdf* (Dutch!).

[14] RailwayLAB, de Rijk, R., Bekius, F. (2015). Spelsimulatie Veiliger plannen. *https://teamsites.prorail.nl/projecten/Directie_IenO_SG/Gedeelde%20documenten/Rapport%20game%20Veiliger%20plannen.docx* (Dutch!).

[15] Standaard voorwaarden Buitenprofielcodes 1-2-3 - versie 2018-01 definitief. *ProRail intern, available on SharePoint* (Dutch!).

[16] ProRail Verkeersleiding, Steensma, P., Swaans, K. (2012). ISVL, gebruikershandleiding.

[17] ProRail (2016). Goederentreinen checken voortaan in op het spoor. *http://www.prorail.nl/nieuws/goederentreinen-checken-voortaan-in-op-het-spoor* (Dutch!).

[18] Steneker, J., van Schayk, M., Cunes, B. (2015). Conceptueelmodel FRISO. *Provided on installation* (Dutch!).

[19] van den Berg, T. W., Jansen, R. E. J., & Middelkoop, D. (2008, June). Application of HLA in the Optimization of Rail Transport. In *Proceedings of the 2008 Summer Computer Simulation Conference* (p.3). Society for Modeling & Simulation International.

[20] van Lieshout, F., Cornelissen, F., Neuteboom, J., & Möller, B. (2008, June). Simulating Rail Traffic Safety Systems using HLA 1516. In *Proceedings of 2008 Euro Simulation Interoperability Workshop, 08E-SIW-069, Simulation Interoperability Standards Organization.*

[21] Kuhl, F., Weatherly, R., & Dahmann, J. (1999). *Creating computer simulation systems: an introduction to the high level architecture.* Prentice Hall PTR.

[22] Wouda, B., PRLGame Technical Manual. RailwayLAB.

[23] RailwayLAB (2016). Gebruikershandleiding PRL-game simulator RailwayLAB ProRail Innovatie.

[24] Barber, A. E., & Roehling, M. V. (1993). Job postings and the decision to interview: A verbal protocol analysis. *Journal of Applied Psychology, 78*(5), 845.

[25] Bettman, J. R., & Park, C. W. (1980). Effects of prior knowledge and experience and phase of the choice process on consumer decision proccesses: A protocol analysis. *Journal of consumer research, 7*(3), 234-248.

[26] Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. *Psychological review, 87*(3), 215.

[27] Russell, S. & Norvig, P. (1995). *A modern approach*.

[28] Tielman, W. (2015). An Agent-based Approach to Simiulating Train Driver Behaviour. *Master's thesis*.

[29] van der Lof, J. (2015). Multi Agent Based Train Simulation. *Master's thesis*.

[30] Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.

[31] Parunak, H. V. D. (1999). Industrial and practical applications of DAI. *Multiagent Systems: a modern approach to distributed artificial intelligence*, 337-421.

[32] Rao, A. S., & Georgeff, M. P. (1995, June). BDI agents: From theory to practice. In *ICMAS* Vol. 95, pp. 312-319).

[33] Böcker, J., Lind, J., & Zirkler, B. (2001). Using a multi-agent approach to optimise the train coupling and sharing system. *European Journal of Operational Research, 131*(2), 242-252.

[34] Abbink, E. J., Mobach, D. G., Fioole, P. J., Kroon, L. G., van der Heijden, E. H., & Wijngaards, N. J. (2009, May). Actor-agent application for train driver rescheduling. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1* (pp. 513-520). International Foundation for Aunomous Agents and Multiagent Systems.

[35] Siahvashi, A., & Moaveni, B. (2010). Automatic train control based on the multi-agent control of cooperative systems. *The Journal of Mathematics and Computer Science, 1*(4), 247-257.

[36] Dastani, M. (2015). Programming multi-agent systems. *The Knowledge Engineering Review, 30*(04), 394-418.

[37] ProRail (2015). Afstudeeropdracht ProRail: Ontwerpen en implementeren van een multi-agent systeem in railsimulatie omgeving. *Project description.*

[38] van Smeden, K. (2016). Rapport Verwerking Orders Vervoer en Beheer na Overdracht Plan. *ProRail* (Dutch!).

[39] Ramaekers, P., de Wit, T., & Pouwels, M. (2009). Hoe druk is het nu werkelijk op het Nederlandse spoor? *CBS* (Dutch!).

[40] Treinreiziger.nl. Cijfers Nederlandse spoor. *www.treinreiziger.nl/kennisnet/kerngetallen.* Retrieved: 19-12-2016.

[41] WillemWever. Hoeveel treinen heeft NS? *willemwever.kroncrv.nl/vraag_antwoord/wetenschap-techniek/hoeveel-treinen-heeft-ns.* Retrieved: 19-12-2016.

[42] ProRail. ProRail in cijfers. *https://www.prorail.nl/over-prorail/wat-doet-prorail/prorail-in-cijfers.* Retrieved: 17-12-2016.

[43] DG MOVE, A. Kroon. (2016). *Fifth report on monitoring development of the rail market.* European Commission {SWD(2016) 427 final}

[44] Dastani, M., & Testerink, B. (2014, May). From multi-agent programming to object oriented design patterns. In *International Workshop on Engineering Multi-Agent Systems* (pp. 204-226). Springer, Cham.

[45] Appendix A, transcription of interviews with ProRail rail traffic operators. Available on demand.

[46] Appendix B, interview questions for both local and regional traffic operators.