Utrecht University

Master Thesis

An Agent Based Approach for Train Traffic Control

Author:
R.N. (Richard) van Wieringen
3996492

Utrecht University Supervisor:
Prof. dr. M.M. (Mehdi) Dastani

ProRail Supervisor:
Emdzad Sehic

Second Examiner:
Prof. dr. F.P.M. (Frank) Dignum

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence

in the

Faculty of Science
Graduate School of Natural Sciences

ProRail Universiteit Utrecht

March 2019

ABSTRACT

The Traffic Control department of ProRail manages the train traffic on the railway network, especially in cases of calamities. Train Traffic Controllers (referred to as TRDLs) are the people at this department that are responsible for safely guiding everything in an assigned area. A TRDL achieves this mainly by controlling signs, switches and railway settings. These TRDLs are supported by an Automatic Railway Setting (referred to as ARI. The main shortcoming of the ARI is that many real world traffic management problems are not supported by the functionality. The goal of this research project is to specify, design and implement a multi-agent system within ProRails distributed simulated environment to support TRDLs in their work that is not handled by the ARI. The specification and design of this multi-agent system were modeled using the Prometheus methodology. The implementation of the MAS is created with the OO2APL Java library. I was able to create a working proof of concept of a multi-agent system within a distributed simulation environment that notifies when a Train Handling Documents (referred to as, TAD(s)) should be applied to ProRail. The system not only does it at the decision point, but it gives the TRDLs a heads up to possibly prepare a TAD.

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude and appreciation to the follow great people. Without the support of these people this master thesis would not have been the same.

To start, I would like to thank my awesome ProRail supervisor Emdzad Sehic for all the support and freedom that he gave to me in order to successfully implement a multi-agent system at ProRail. He also guided me through this big organization and to all the right people within ProRail to achieve my goals. Two of these amazing people I would like to thank are Wilco Tielman and Arie van den Berg. They helped me a lot with the implementation of my multi-agent system and with connecting it to ProRail's simulators.

I want to thank my brilliant Utrecht University supervisor Mehdi Dastani for all the support and his critical appraisal. He gave me the opportunity to conduct my research at ProRail and gave me the guidance to create this multi-agent system. This also brought me the possibility to put theory into practice.

I would like to thank my parents Nico van Wieringen and Yrene van Wieringen for always supporting me and believing in me. Without their support I would not have made it this far. I also want to thank the rest of my family: my brother Fabian Schweichler, my sister Nicole van Wieringen, and her boyfriend Nadir Mea for all the moral support.

I would like to specially thank my partner Zoe van den Aardweg for always being by my side and helping me with every step of the process mentally but also substantively. You always motivated me to keep going and I will always be grateful for all your support.

Many thanks to everybody who stood by me and checked in with me now and again.

CONTENTS

Part I

INTRODUCTION

1 INTRODUCTION

This thesis provides an overview of the research project at the Innovation department of ProRail, where a multi-agent system of train traffic controllers within a distributed simulated environment is specified, designed and implemented. I will start with a short description of ProRail, then the problem and motivation, followed by the research question and the approach. At the end of this chapter will be the outline of this thesis.

1.1 PRORAIL

The company ProRail is responsible for the railway network in the Netherlands and is with over 3.3 million train rides per year one of the busiest railway networks in Europe. Their goal is to make the railway network safer, more reliable, punctual and more sustainable enabling a safe transport of passengers and goods to arrive on time at their destination. They do all this with attention to their influence on the environment and society in the process. ProRail is an independent party that is responsible for the construction, maintenance, management and safety of the railway network. They divide the space on the tracks among transporters, regulate all train traffic, create new tracks and manage and build train stations. Putting it in perceptive, this involves maintaining 404 train stations, 7.219 kilometers of existing tracks, 7.006 switches, 12.093 signals and 2.368 crossings. [23, 24, 26, 28, 29]

1.2 PROBLEM AND MOTIVATION

The Traffic Control department of ProRail manages the train traffic on the railway network, especially in cases of calamities. Train Traffic Controllers (referred to as `TRDLs`, Dutch: *treindienstleider(s)*) are the people at this department that are responsible for safely guiding everything in an assigned area. A `TRDL` achieves this mainly by controlling signs, switches and railway settings. These `TRDLs` are supported by an Automatic Railway Setting (referred to as `ARI`, Dutch: *Automatische Rijweginstelling*) which is an important functionality in the control system to facilitate them to automate railway settings. In the optimal scenario where all trains are riding according to the schedule the control systems such as `ARI` do all the traffic management work automatically. In the real world this optimal scenario is not alway the case. The main shortcoming of the `ARI` is that many real world traf-

fic management problems are not supported by the functionality. For instance, when a train has a delay which deviates too much from the schedule ARI stops working and the TRDL must intervene.

The Innovation department of ProRail tests new logistic concepts within the railway network using a distributed simulation environment. In this environment a number of human-in-the-loop simulators are linked via a High Level Architecture (referred to as HLA). Currently a number of human operators such as TRDLs are a requirement to run a simulation. In practice it is very difficult to get TRDLs together in order to run a simulation. In the long run ProRail wants an intelligent system that can replace the need for TRDLs as human-in-the-loop simulators enabling the possibility to tests new logistic concepts faster and more often.

A multi agent based approach is selected as a solution for this intelligent system because it offers powerful modeling techniques and is noted to considerably improve the way in which people conceptualize and implement various kinds of systems [3, 16]. These powerful techniques are used for the development of large-scaled distributed systems to deal with uncertainty in a dynamic environment [7]. Many problems in traffic control are inherently distributed. It is often the case that it is not possible to provide additional capacity. Therefore, a more efficient use of the available infrastructure is a necessity [2]. This agent based approach has been successfully applied in various (simulated) traffic control systems for vehicles [21], airplanes [39] and trains [30].

The goal of this research project is to specify, design and implement a multi-agent system within ProRails distributed simulated environment to support TRDLs in their work that is not handled by the ARI. This multi-agent system will be a proof of concept that sets the foundation to eventually replace the need of TRDLs as human-in-the-loop simulators.

1.3 RESEARCH QUESTION

In order to develop this multi agent system the following research question is formulated from the goal described in the previous section:

> How to specify, design, and implement a multi-agent system within a simulated environment that assists train traffic controllers (TRDLs) in their work that is not handled by ProRail's Automatic Railway Setting (ARI)?

Subquestions:

1. How do Train Traffic Controllers (TRDLs) work with ARI in everyday practice?

2. What are the limitations of ARI that TRDLs encounter in practice?
3. How can a multi-agent system be modeled to overcome these limitations?
4. What is an efficient and effective agent based design method for a multi-agent system?
5. How to implement a multi-agent system onto ProRail's distributed simulation environment?

1.4 APPROACH

A literature study should give insights into the diverse aspects involving: Train Traffic Control at ProRail, Simulation, Multi Agent Systems and 002APL. The specification, design and implementation of the agent-based system will be in the Object Oriented Agent Programming Language called 002APL (developed by M. Dastani and B. Testerink) that is capable of communicating with the distributed simulation environment (of the Innovation department of ProRail). The agent reasoning will be based on the theoretical background and interviews with TRDLs on how they tackle a specified problem that can not be handled by the ARI. The information from the latter will lead to the protocols that drives the multi-agents system.

1.5 OUTLINE

In Chapter 2 I will provide background information necessary to understand the multi-agent system developed in this thesis. This theoretical background consist of four sections (2.1-2.4). In section 2.1 the internal structure of the traffic control department of ProRail will be described and will elaborate on the function of a TRDL. In section 2.2 the background theory and actual simulation environment that is used by ProRail is described. In section 2.3 elaborates on background theory of agents and multi-agent systems. In section 2.4 background information and short description of the 002APL is presented.

In Chapter 3 I will provide the methodology that I used for specifying, designing and implementing a multi-agent system within a simulated environment. This methodology consist of three sections (3.1-3.3). In section 3.1 the specification phase of the multi-agent system is described. In section 3.2 the design phase of the multi-agent system is provided. In section 3.3 the implementation phase is described.

In Chapter 4 I will provide the results of this study that carried out the methodology. The results consist of three sections (4.1-4.3). In section 4.1 the results of the multi-agent system of train traffic control are given. In section 4.2 system setup is described. In section 4.3 a real

world simulation run of the multi-agent system of train traffic control is presented.

In Chapter 5 I will provide the conclusion of this study. In this chapter I will answer the research questions.

In Chapter 6 I will provide the discussion of this study. This discussion consist of four sections (6.1-6.4). In section 6.1 the interpretation of results will be presented. In section 6.2 the limitations of this study will be elaborated. In section 6.3 future research possibilities will provided. In section 6.4 some concluding remarks are given.

2 THEORETICAL BACKGROUND

2.1 TRAIN TRAFFIC CONTROL AT PRORAIL

Train Traffic Control at ProRail is carrying out the final preparations for letting trains ride according to schedule. The schedule is supplied by the logistical chain of capacity allocation and asset management makes sure the needed infrastructure is available. Finally, Train Traffic Control ensures that everything is being executed as well as possible with safety as the priority [25].

2.1.1 *Traffic Control Department*

The traffic control department is hierarchically structured, consisting of three levels: at the top there is the National Traffic Control (referred to as, LVL, Dutch: *Landelijke Verkeersleiding*), then there are the Decentralized Traffic Controllers (referred to as, DVLs, Dutch: *Decentrale Verkleersleider(s)*), and at the lowest level there are the TRDLs [25]. LVL is situated in the Operational Control Center Rail (referred to as, OCCR) located in Utrecht and are the beating heart of the department. They handle major disruption in close cooperation with the national control centers of the transporters and with the DVLs at the 13 posts spread across the country see the red dots in Figure 1. These 13 posts are divided in 5 regions (demarcated by blue lines in Figure 1):

- **Randstad North:** Amsterdam en Alkmaar
- **Randstad South:** Rotterdam en Rotterdam Goederen (Kijfhoek)
- **Region Northeast:** Groningen, Zwolle, Arnhem
- **Central Region:** Den Haag, Utrecht, Amersfoort
- **Region South:** Eindhoven, Roosendaal, Maastricht

Every Traffic Control post is responsible for safely managing everything within a demarcated area. At a post the DVL works together with TDRLs. A TRDL has an assigned area within the demarcated area of a post [27].

Figure 1: Overview of the process management areas [27]

In a nutshell, the hierarchy of the Traffic Control Department works as follows (see Figure 2): The LVL can make changes to the schedule because they have a national overview and therefore they can overrule DVLs. A DVL can make changes to the schedule because they have an overview of the post area and therefore they can overrule TRDLs. When they cannot solve a post area issue they report this to the LVL. The TDRL uses the schedule and ARI to set the railway settings for the trains. When they cannot solve a local issue they report this to the DVL. The next subsection will elaborate on the TRDLs and will also clarify what an assigned area looks like for a TRDL.

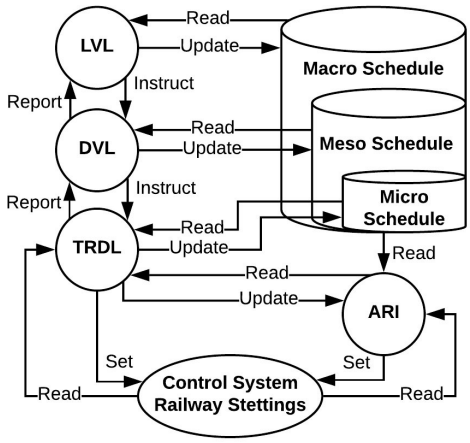

Figure 2: Traffic Control Schematic Overview

2.1.2 *Train Traffic Controller (TRDL)*

A TRDL monitors and manages the train traffic within an assigned area. The rapidity at which a TRDL acts in case of a calamity or deviation of the schedule is of great influence for all the train travelers and transporters. To be able to handle these stressful situations a TRDL needs to be sharp and stress-resistant. When there are multiple problems on the railway that need to be handled, a TRDL must be able to prioritize which one is most urgent. Handling these problems usually happens with close coordination and cooperation with other TRDLs and the DVL.

A TRDL has two main tasks: execution of the schedule and making sure everything happens safely. In a nutshell, the execution of the schedule works as follows: Most of the time the TRDL executes the schedule by timely controlling signs, switches and railway settings for train traffic. During this process they are supported by the Process Management System that enables him/her to manage train traffic. This system contains the routing schedule for every emplacement of the assigned area. Usually the railway settings for arriving trains will be set by ARI. This happens when trains, with the corresponding train number according to the schedule, arrive within a certain time frame. When this is the case, ARI will give the railway settings as assignments to the Control System. The Control System sets the order and passes it along to the Security Systems which control the Elements of the Track (switches, signs, etc.). This all happens just before the departure time of the train, but when the train diverges too much from the schedule (time frame) ARI stops working and will do nothing. In this case the TRDL has to make adjustments to the schedule [33, 37]. Figure 3 provides a schematic overview of the process management.

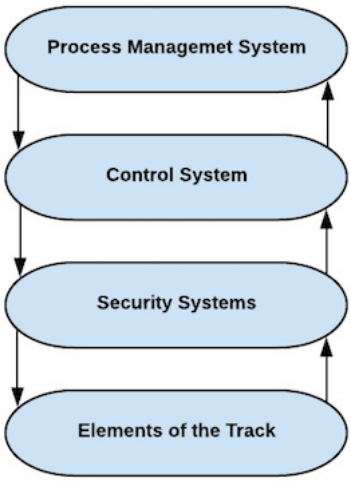

Figure 3: Schematic representation of the process management [37]

An assigned area of a TRDL is called a Process Management Area (referred to as, PLG, Dutch: *ProcesLeidingGebied*). This is the control area that one TRDL is responsible for. Figure 4 provides an overview of PLGs demarcated in black boxes at the post Amsterdam. Every PLG consists of one or more Primary Process Management Areas (referred to as, PPLG, Dutch: *Primair ProcesLeidingGebied*). This is the control area of a train station called emplacement. The connection between two emplacements is referred to as free track. An example of a PLG in Figure 4 is for instance the bottom left demarcated black box named TRDL Schiphol. An example of PPLGs within the PLG of TRDL Schiphol are: Asdl, Asra, Schiphol Airport and Hoofddrop.

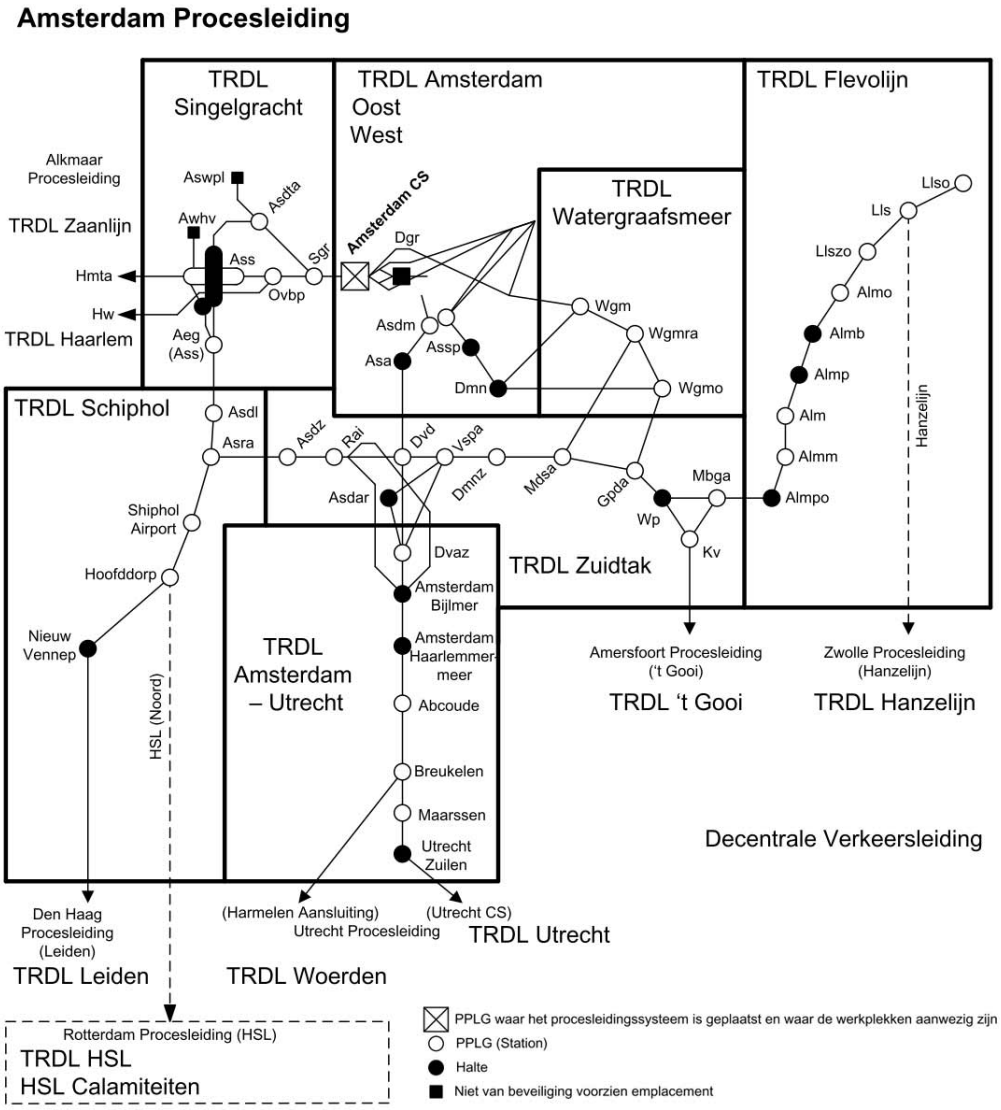

Figure 4: Overview of the process management post Amsterdam [45]

One of the subsystems of the Process Management System is the schedule screen as displayed in Figure 5. This schedule screen is an example of the TRDL Schiphol assigned area in Figure 4 with the four PPLGs. A schedule screen consists of the following components:

A Schedule Menus

B History Window

C Schedule Window

D Mutation Menu Bar

E Mutation Window

In **A** all the functions can be found concerning History Window or Schedule Window. The History Window (**B**) shows the last four railway setting commands. The Schedule Window (**C**) displays for every PPLG the schedule. In **D** are all the functions concerning the Mutation Window. In the Mutation Window (**E**)

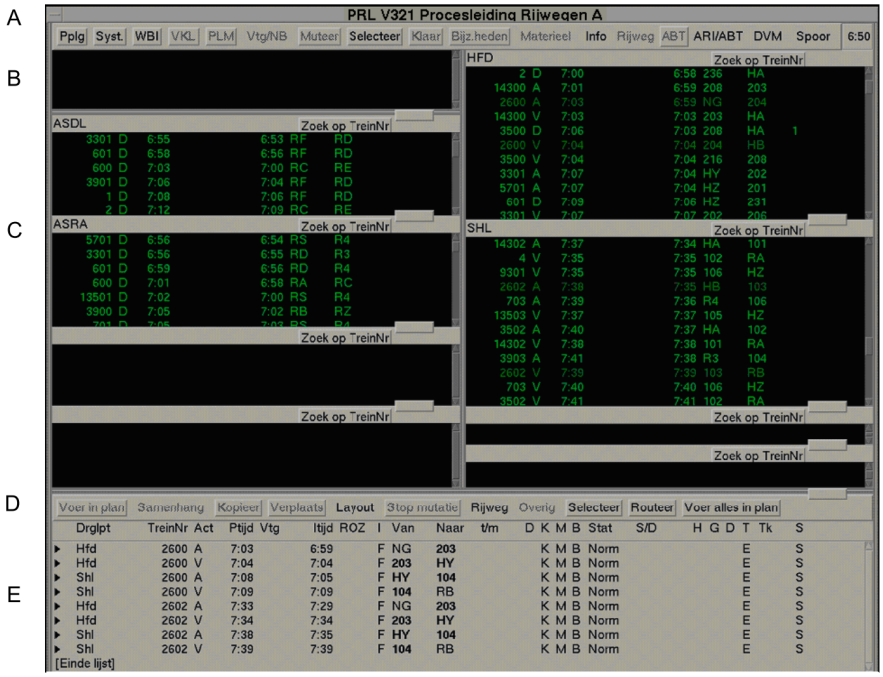

Figure 5: Schedule screen of the process management system [33]

A	⌐ B ¬	CD	⌐E ¬	F G	H	⌐ I ¬	J	K	L	M	N	O	P
!	3824	A1	8: 08	V +3	Dv	8:06	3	BA	14	1	K	M	CHL>

Figure 6: An overview of a schedule line of the schedule screen [33]

The schedule contains lines and every line provides information of one specific train that a TRDL needs for the process management. Figure 6 provides an example of a schedule line that consists of the following elements:

A Indicator

B Train Number

C Activity Type

D Railway Track Number

E Schedule Time

F Delay Type

G Delay Duration

H Control Point

I Scheduled Set Time

J Setting Method

K From (Railway Destination)

L To (Railway Destination)

M Forced Railway Number

N Ready Notification Indicator

O Material Relationship Indicator

P Particularities

All in all, the task for every TRDL is to execute the schedule with safety as the priority. In this process the effects of a calamity or deviation of the schedule on travelers and transporters has to be minimized. During this task a TRDL is initially not allowed to discriminate among transporters. This means that for example a passenger-train does not automatically have priority over a goods-train.

2.1.3 *Automatic Railway Setting (ARI)*

ARI is a subsystem of the process management system that runs at every workplace of a TRDL. ARI gives assignments to the safety systems to automatically set the railway routes for a train according to the route planning process (the schedule). For every PPLG there are two mutually exclusive modes in which ARI can be activated. The two modes are called "Plan is leading" and "First Come First Served" (referred to as, FCFS). In the "Plan is leading" mode the order in which ARI handles train route settings is based on the schedule. Trains have to notify ARI within a certain time frame. In the "FCFS" mode the order in which ARI handles train route settings is based on when trains

notify ARI. Incoming trains do not have to abide by the time frame of the schedule to be handled by ARI [1, 22, 40, 46].

There a few important conditions that need to be met in order for ARI to automatically set routes for trains. The conditions are as follows [46]:

- The railway that has to be set needs to be available.
- The train to be handled has to have the correct train number.
- The set-up time needs to be within the time frame or the waiting frame.

In order for ARI to work all the conditions need to be met. If one (ore more) of the conditions is not met ARI will be disabled for that particular schedule line. The status of ARI for a schedule line can be derived from the color of the line within the schedule screen (see Figure 7). There are different color codings for an activated or deactivated status of ARI. What these colors entail is displayed in Figure 8.

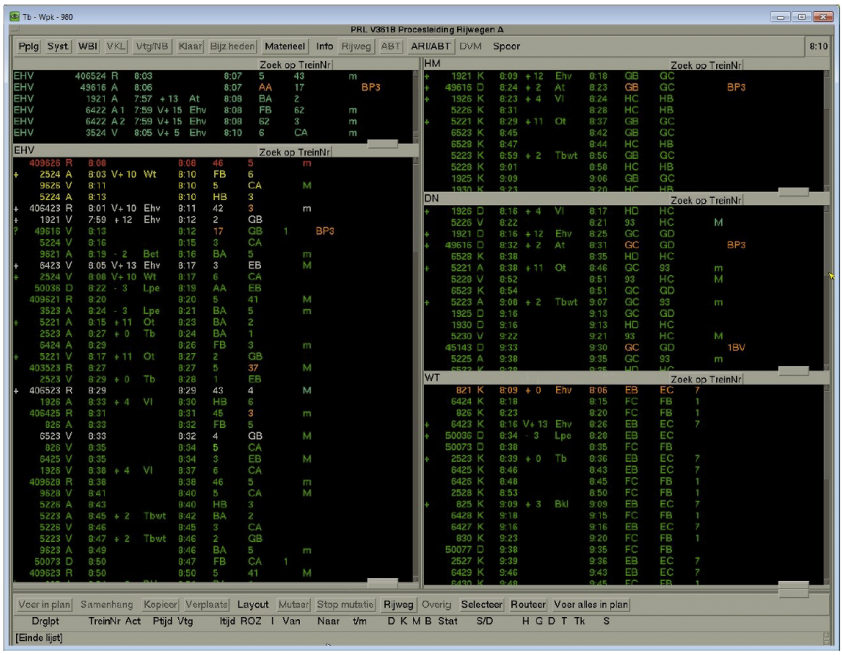

Figure 7: Schedule screen of the process management system [46]

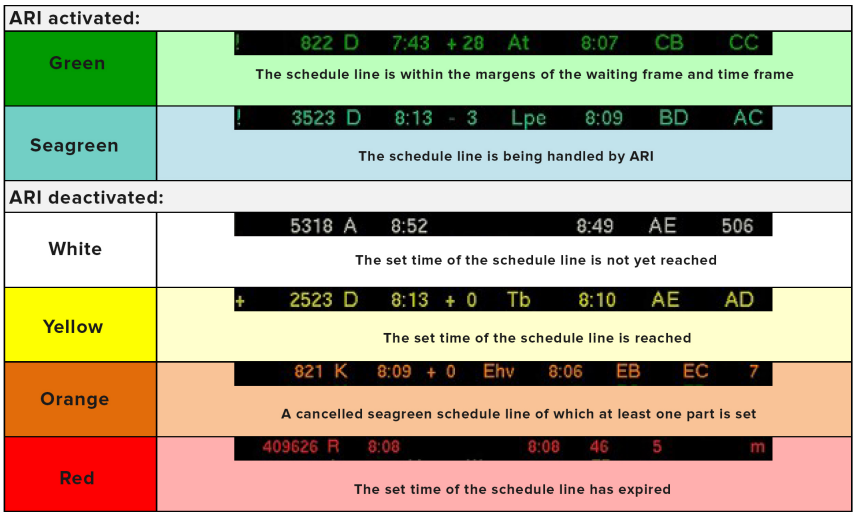

Figure 8: Schedule line color coding overview [46]

2.1.4 *Train Handling Document (TAD)*

In some deviant train related scenarios where `ARI` stops working handling agreements have been made for specific scenarios. These handling agreements are written down in Train Handling Documents (referred to as, `TAD(s), Dutch: Trein Afhandelings Document`), see figure 9 for an example that applies to the `TRDL` that is responsible of the Zuidtak operating area in Amsterdam.

A `TAD` consists of three distinct components: Waiting times passenger trains (referred to as, `WRT`), handling strategy by train or train series and if/then scenarios.

`WRT`:

- The `WRT` indicates for each railway junction how long trains can wait for a connection when arriving trains are delayed.
- The `WRT` does not apply if two trains are delayed that normally takes over from each other.
- The `WRT` determines the final connections of passenger trains at the end of a traffic day.

Handling strategy by train or train series:

- Handling strategies indicates for each station what needs to be done when there is an increasing delay.
- It prevents the emergence of a domino effect in case of delays.
- It ensures that there is no repression among transporters.

If/then scenarios:

- If/then scenarios are intended for delays that arise in the train schedule within the operating area of the TRDL.
- These scenarios become effective when a train exceeds a predetermined delay margin.
- A TRDL is authorized to adjust the train schedule according to the if/then scenarios within its operating area.

If the execution of the TAD results in an order change, a TRDL must give a standard message. If the TAD does not offer a solution a TRDL must consult with the DVL [6, 41]. In figure 9 are lines highlighted with specific colors to explain a TAD in more detail:

- **Blue**: A goods train has right to keep its own path on the railway network when the delay is 4 minutes or less.
- **Yellow**: Concerns the first and last trains of the day. If the specific train numbers in the left columns have no more then 10 minutes delay: the specific trains in the right columns have to wait for them, because they have a connection with each other. For some trains the connection applies only for certain days of the week.
- **Green**: Concerns train series that have a connection with other train series at a specific PPLG, in this case WP (Weesp) with trains heading to Asd/Shl (Amsterdam/Schiphol). In this case the heading indicates trains series with even numbers. The 4300 series, heading to Shl, with an Arrival hh:02 or hh:32 at WP, has a delay between WRT (meaning 0) and up to and including 4 minutes, when this delay is observed at the decision point Alm (Almere), then the 15800 train serie has to wait for the 4300 series.
- **Orange**: Concerns train series that have to change order with each to prevent the emergence of a domino effect, at a specific PPLG, in this case WP (Weesp) with trains heading to Asd/Shl (Amsterdam/Schiphol). In this case the heading indicates trains series with even numbers. The 1500 series, heading to Asd, with an Arrival hh:19 or hh:49 at WP, has a delay of 6 minutes or more, when this delay is observed at the decision point Ndb (Naarden-Bussum), then the 1500 train serie will change order with 14600 series.

There are three different tables of agreements in the TAD example in figure 9. The first table (containing yellow highlights) represents how to handle all the first and last trains of the day given a train series. The second table (containing green and orange highlights) represents how to handle train series at PPLG Wp that are going towards PPLGs: Asd/Shl. The third table represents how te handle trains series at PPLG Wp that are going towards PPLGs: Alm/Hvs.

Treindienstleider: Zuidtak **Ingangsdatum 10-12-2017**

ProRail

Goederentreinen t/m 4 minuten vertraging behouden het recht op het eigen pad
De laatste treinen van de dag wachten maximaal 10 minuten

Eerste trein	Vertrektijd	Wachtende trein
4308		15808(max. 10 min)
15808		4308(max. 10 min)
Laatste trein	Vertrektijd	Wachtende trein
4389		15889 ma t/m vr
15889		4389
15895		4395
4395		15895
15886		4386

Wp richting Asd/Shl

			vertraging		beslis	
Trein	naar	a.	van	t/m	punt	volgorde/afhandeling
4300	Shl	-.02/-.32	Wrt	4	Alm	15800 wacht
5700	Shl	-.17/-.47	Wrt	7	Ndb	14600 wacht
14600	Asd	-.20/-.50	Wrt	7	Alm	5700 wacht
1500	Asd	-.19/-.49	6	>	Ndb	14600 - 1500
1800	Shl	-.08	6	>	Alm	4300 - 1800
700	Shl	-.38	6	>	Alm	4300 - 700

Wp richting Alm/Hvs

			vertraging		beslis	
Trein	naar	a.	van	t/m	punt	volgorde/afhandeling
14600	Alm	-.09/-.39	Wrt	5	Asdm	5700 wacht
15800	Hvs	-.27/-.57	Wrt	5	Asdm	4300 wacht
(1)1600	Amf	-.27/-.57	5	>	Dvd	15800 - (1)1600
140/240	Amf	-.11	5	>	Asdm	5700 - 140/240
1500	Amf	-.11/-.41	5	>	Asdm	5700 - 1500

Figure 9: Example of a TAD for the TRDL Zuidtak (Amsterdam)

2.2 SIMULATION

A simulation can be used to mimic a real world environment and can be seen as a particular type of an executable model. A simulation is an execution of a model that can be created out of a complex system (a system that consist of multiple heterogeneous, interacting components) [34]. Simulating a railway network with traffic control is an example of a complex system.

Most of the time there are multiple interacting entities involved in simulations. For instance the rolling stock, in the rail transport industry referred to as any vehicle that moves on a railway. There can be a monolithic simulation model incorporating all rolling stocks and their interactions with one other and their environment. The alternative is to have individual models of each railway vehicle and the environment, and these models interact with one other via a well-defined and agreed-upon interface. The alternative can be preferred for the following reasons: the simulation load can be shared by multiple processors and these processors can be distributed over a network. The latter leads to the so called distributed simulation which is of interest [38].

2.2.1 *High Level Architecture (HLA)*

A High Level Architecture (referred to as HLA) is a standard architectural framework for distributed simulation. This architecture facilitates the reuse, interoperability and composability of simulations. The reuse is enabled by the interoperability and composability. The interoperability is the capability of a simulation to exchange information in a useful and meaningful way. The composability enables the selection and assembly of components in any combination to achieve the objective of the systems. This HLA framework essentially enables component-based simulation development [38, 44].

The basic components of a HLA consists of the following set of specifications [38]:"

- *HLA Framework and Rules*: Specifies the elements of systems design and introduces "a set of ten rules that together ensure the proper interaction of federates in a federation and define the responsibilities of federates and federations" [13]
- *Interface Specification*: The HLA Interface Specification defines "the standard services of and interfaces to the HLA runtime infrastructure. These services are used by the interacting simulations to achieve a coordinated exchange of information when they participate in a distributed federation" [12]

- *Object Model Template* (referred to as OMT): presents the mechanism to specify the data model – the information produced and consumed by the elements of the distributed simulation. The OMT describes "the format and syntax (but not content) of HLA object models" [14]"

FEDERATES Applications that implements or conforms to the HLA are called federates. These support and use the interfaces that are specified in the HLA Federate Interface Specification. These specifications ensures that federates can participate in a distributed simulation execution. A federate becomes a so called Joined Federate when it participates in a federation execution. During an execution a federate may join multiple times. New joined federates are created each time when a federate joins multiple executions. How a federate should be structured is not specified by the HLA, because it is only interested in the interface of a federate. A federate can be all kinds of applications, for example: simulations of systems, monitoring applications, gateways or live entities. Technically is a federate a single connection to the Runtime Infrastructure (referred to as, RTI), where it can be a single process or contain multiple processes running on several computers. In the process a federate can produce/consume data or both. Designing reusable set of simulation features as a federate is considered best practices. A federate can for instance represent a rolling stock platform that simulates any type of train or an TRDL workplace model as a scenario training simulator. Also, legacy simulation application can be wrapped as a federate and therefore enabled to participate in a federation execution (runtime instantiation, which is an actual simulation execution).

FEDERATION The set of federates that have a common specification of data communication formulated in Federation Object Model (referred to as, FOM) is called a federation. When communication requirements of federates are specified in their Simulation Object Models (referred to as, SOMs), they are composed and interoperate over the RTI throughout federation execution.

RUNTIME INFRASTRUCTURE A HLA require an infrastructures to enable inter-federate communication. The RTI is the underlying managing software infrastructure, where federates interact with to participate in the distributed simulation and exchange data. It supports the rules provided by the Interface Specification of the HLA. Figure 10 provides an example overview of federations and federates interacting with the RTI.

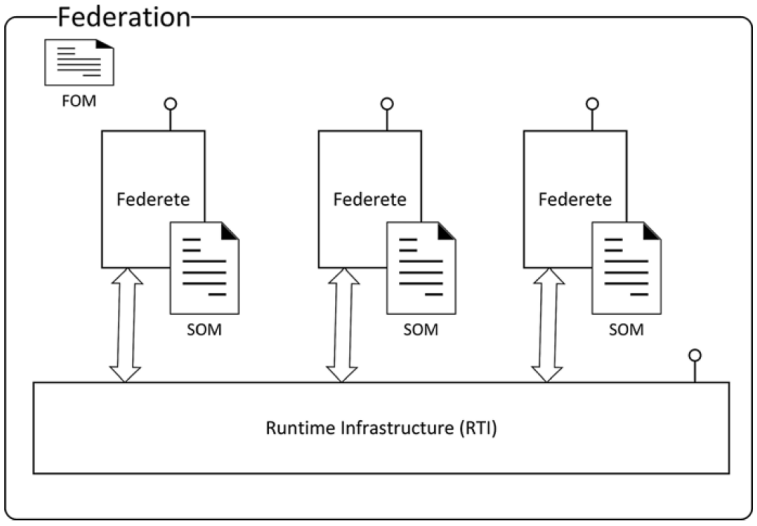

Figure 10: Federation and federates

2.2.2 *Simulation at ProRail: Trinity*

The distributed simulation at ProRail is called Trinity and is based on the HLA framework. Figure 11 gives a overview of Trinity. The Trinity simulator (Figure 11) consists of the following components:

- **RTI:** Runtime infrastructure, handling all messages between the simulation
- **Trinity FOM:** Federation Object Model, handling the objects
- **BAP:** Planning of NS, live system for personnel, train operators, conductors
- **Bapper:** Translating the BAP information for the RTI
- **Morpheus:** Train driver simulator
- **Friso:** Flexible Rail Infra Simulation Environment (see 2.2.3)
- **Friso Mapper DLL:** Translating the Friso messages for the RTI
- **PRLGame:** Human in the loop TRDLs
- **PRL Mapper:** Translating the PRLGame information for the RTI
- **BAM:** Planning of NS, live system for rolling stock
- **BOP:** Personnel simulation, Human in the loop operator for monitoring and controlling
- **Plan federate:** Containing and managing the master schedule
- **VOS werkplek:** Human in the loop DVLs
- **VOS:** VOS server for simulation
- **VOS federate:** Mimic live VOS system to the RTI

- **SBG++:** Railway occupation graph, visual representation of the schedule
- **Arp federate:** Enables the simulation of SBG++
- **Bootstrap federate:** Game leader, controlling scenarios, managing the simulation
- **ARI federate:** The master `ARI`

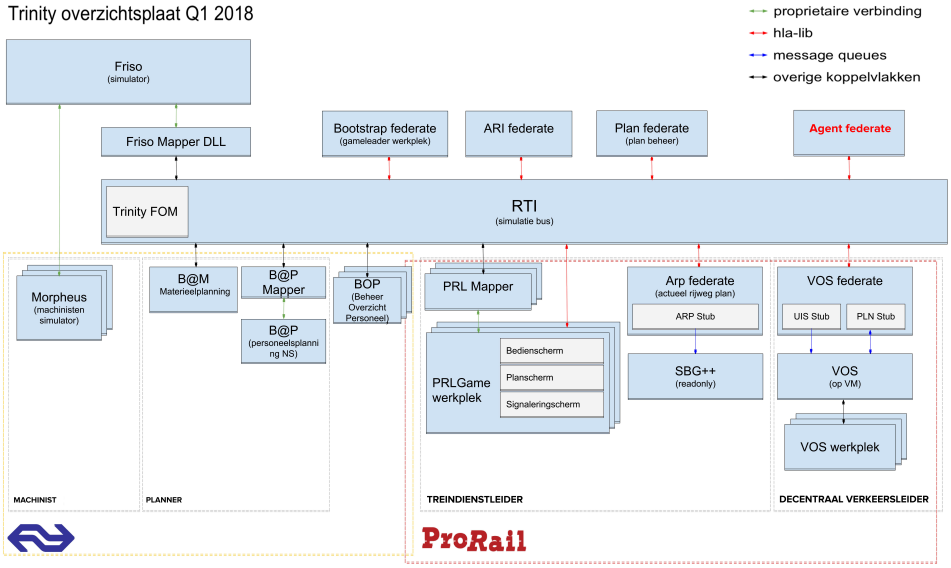

Figure 11: Trinity Overview Sheet Q1 2018

2.2.3 *FRISO*

The abbreviation FRISO stands for Flexible Rail Infra Simulation Environment (Dutch: *Flexibele Rail Infra Simulatie Omgeving*) and is one of Prorail own set of simulators. FRISO is a discrete event based microscopic simulator where train traffic within a railway network can be simulated at a detail level. FRISO can be used to research the following:

- Locally analyzing, comparing and improving the stability/robustness and efficiency of the schedule for a specific area.
- Comparing different control variants (`FCFS`, Fixed Order or Plan is leading) in a specific area.
- Tracking down and quantifying bottlenecks in the infrastructure for a specific area.
- Analyzing cause and effect relations for delays.
- Calculating the driving times of the trains.

- Quantifying effects of changes to the schedule.

In addition, FRISO has a mutation functionality that makes it possible to research changes to the infrastructure and the possibility to link it to an external traffic management system. FRISO can also be deployed as a simulator in a Railway Gaming Suite, where human-in-the-loop simulators such as PRL Game and Morpheus. Because of this, FRISO can also be used for training purposes or for testing the influence of new systems on the behavior of people who have to work with it [19, 35, 36].

2.2.4 *Agents Based Traffic Simulation*

Agents based simulations have been applied more often within the domain of traffic simulation, an overview is given by [7] and [18]. This agent based approach has been applied in various simulated traffic systems for instance: cars [21], airplanes [39] and trains [30]. One of the reasons agent based approaches are often applied is because they offer powerful techniques that are used for the development of large-scaled distributed systems to deal with uncertainty in a dynamic environment [7]. This is why an agent based approach is perfectly suited for train traffic control which takes place within a such an environment.

2.3 AGENTS AND MULTI-AGENT SYSTEMS

Multi-agent systems (referred to as, MAS) are a new software engineering (computational) paradigm and is considered innovative in computer science. The goal of such a system is to provide high-level abstractions to model and develop complex systems and with that improving solutions for industry problems related to interacting autonomous systems. The term agents is widely used within the literature without an exact definition that is adopted by everyone [11]. The next two quotes will provide the definitions I used for MAS and agents:

> "*A multi-agent system is one that consists of a number of agents, which interact with one-another.*"

> "*An agent is a computer system that is capable of independent action on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told). The main point about agents is they are autonomous: capable of acting independently, exhibiting control over their internal state. Thus: an agent is a computer system capable of autonomous action in some environment in order to meet its design objectives.*"

— Michael Wooldridge [42]

The agents oriented software engineering community of MAS aims to provide concepts and abstractions to conceptualize, model, implement, test and analyze interacting autonomous agent based systems. MAS are basically the development of a set of autonomous agents, how they are organized and how they interaction with their environment. These agents are required to make proactive decision themselves to achieve their goals or in a reactive manner by responding to incoming events of their (shared) environment. In MAS, the organized agents coordinate the agent's behavior in order to achieve the overall goals of the system. This is behavior can be endogenously or exogenously driven. Endogenously means the agents follow specification rules or protocols and exogenously means external software artifacts causes the behavior. [10, 42]

2.3.1 *Agent Based Modeling*

A commonly used approach for agent based modeling is the belief-desire-intention (referred to as, BDI) paradigm [17, 31, 32]. This approach formalizes the reasoning of a rational agent, which is an agent that always chooses the action that achieves the optimal expected outcome towards reaching its goals among all other feasible actions. An agents reasoning is based on its beliefs that are internal representations of information about its environment. The desires of an agent are the states its wants to achieve. Intentions are an agent's commitment to actions that are parts of a plan. An agent uses these concepts of belief, desire and intention to execute actions based on plans to achieve its goals. Based on the BDI paradigm several methodologies have been proposed and developed, like Tropos [5], Prometheus [20] and others [17, 43]. The underlying goal of all the authors is to provide a framework to conceptualize, model, implement, test and analyze complex (multi-)agent systems.

2.3.2 *Agent Based Methodology: Prometheus*

As mentioned in the previous subsection there are several methodologies developed based on the BDI paradigm. The Prometheus [20] methodology will provide the guidelines in the development of this MAS. It is the most mature software development methodology for agent-based systems that is widely accepted and intended for industrial software developers and undergraduate students. Although Prometheus is detailed, it is intended as a set of guidelines and should be interpreted in conjunction with the common sense of the user(s). I will therefore use my own knowledge, gathered insights, common sense reasoning and the essence of Prometheus to develop a MAS of

train traffic controllers. The essence of Prometheus consists of three phases: "

1. ***System Specification****: where the system is specified using goals and scenarios; the system's interface to its environment is described in terms of actions, percepts and external data; and functionalities are defined.*
2. ***Architectural Design****: where agent types are identified; the system's overall structure is captured in a system overview diagram; and scenarios are developed into interaction protocols.*
3. ***Detailed Design****: where the details of each agent's internals are developed and defined in terms of capabilities, data, events and plans; process diagrams are used as a stepping stone between interaction protocols and plans.* " [20]

2.4 OO2APL

Various agent programming languages (referred to as, APLs) and frameworks have been proposed in the last few decades to support the development of autonomous agents and multi-agent systems [4]. These APLs can be divided into three different categories:

- *Declarative*: Strict focus on logic and being formal (for instance: DALI [8]).
- *Imperative*: Agent-oriented languages that are built upon and are in common with imperative languages such as JAVA (for instance: JACK [15]).
- *Hybrid*: This category is defined by combining the possibility to use a declarative approach and imperative programming (for instance 2APL [9])

My area of interest is with the hybrid approach and especially 2APL [9], which is a BDI agent-oriented APL. It implements agent concepts and abstraction by providing programming constructs. 2APL uses a declarative approach for the representation and reasoning of an agent's beliefs and goals and imperative approach for the agents plans and interface to their environment. These common constructs in agent programming literature, with emphasis on 2APL, are what OO2APL [10] is attempting to capture as object oriented design patterns and constructs. What is implemented in object-oriented technology are abstractions and some characteristic concepts of autonomous agents and multi-agent systems. It is achieved by initiating a Java library of object-oriented design patterns and constructs to capturing these characteristic and abstractions. I will use *OO2APL* to develop this multi-agent system.

> "We follow the argument that multi-agent programming technology can find its way to industry by providing a methodology that guides the development of autonomous agents and multi-agent systems in standard programming technology."
>
> — Mehdi Dastani and Bas Testerink [9]

Part II

METHODOLOGY

3 METHODOLOGY

In the previous chapter I have presented the various aspects of the Train Traffic Control at ProRail, Simulation, Agents and Multi-Agent Systems and 002APL that are needed in the development of this MAS. In this chapter, I will present the development process of the MAS of Train Traffic Control in three parts. Firstly, the Multi-Agent Specification part describes the system specifications process and follows the first phase of Prometheus [20]. Secondly, the Multi-Agent Design part describes the design process and follows the second and third phase of Prometheus [20]. Thirdly, the Multi-Agent Implementation part describes how this specified and designed multi-agent system is implemented in 002APL and connected to *Trinity* (ProRails distributed simulator).

3.1 MULTI-AGENT SPECIFICATION

The specification phase of Prometheus [20] will provide a rough idea of the system. In the next subsections I will give a rough description and define the requirements of the system in terms of:

- The goals of the system and use case scenarios
- Interface to the environment
- Functionalities

3.1.1 *Goals and Scenarios*

The goal of this multi-agent system, described on a high level, is to support TRDLs in their work that is not handled by the ARI. As mentioned in 2.1 safety has the highest priority, meaning the system has to achieve its goals without jeopardizing that priority. One of the TRDLs tasks that is not handled by ARI, is applying TADs when necessary as described in 2.1.4. This part of a TRDL's work will be supported by the system. From this higher goal and the determined work the system is supporting, I extracted the sub goals of the system: monitoring train traffic, determining if TAD is applicable, execute TADs, inform DVL of a particular TAD and listening to incoming TRDL messages. These sub goals of the system (see table 2) contribute to the higher level goal.

High-level goal
Support TRDLs in their work
that is not handled by the ARI
Sub goals
- Monitoring Train Traffic
- Determining if a TAD is applicable
- Execute TADs
- Inform DVL of particular TAD
- Listening to incoming TRDL messages

Table 1: The systems' high-level goal and sub goals

The system will support various scenarios when a TAD should be applied. I will give one example of a use case scenario responding to a particular event. The event is that train 1522 has to change order with another train to prevent the emergence of a domino effect at a specific PPLG, in this case WP (Weesp), with trains heading to Asd/Shl (Amsterdam/Schiphol). The 1522 train, heading to Asd, with an Arrival time of 12:19 at WP, has a delay of 8 minutes, observed at the decision point Ndb (Naarden-Bussum). According to the TAD the 1522 train has to change order with train 14626 who rides on schedule. The system will work as follows:

Step 0: GOAL: Monitoring Train Traffic.
Step 1: PERCEPT: Train 1522 has delay of 8 minutes observed at decision point Ndb.
Step 2: ACTION: Determining TAD is applicable.
Step 3: PERCEPT: Train 1522 is more then 6 minutes delayed and 14626 rides on schedule, so an order change has to be applied.
Step 4: ACTION: Order change plan is to be executed.
Step 5: ACTION: Environment is updated with the order change plan.

Table 2: The systems' high-level goal and sub goals

3.1.2 *Interface to the environment*

The environment in which the agent system is situated consists of the connection it has with the simulator. This connection ensures that the

agents are able to receive events from the distributed simulated environment. The agent system perceives directly from the environment, but its actions are indirectly transmitted to the environment.

3.1.3 *Functionalities*

The main functionality of the system is to determine if a TAD is applicable in the current state of the environment. The agents are therefore monitoring the environment, creating events for determining if a TAD is applicable. If so, the agents execute TADs. One of the actions can be to notify a DVL of this particular TAD. An agent does not want to miss notifications of other agents that is why it is listening to possible incoming messages so it gets triggered.

3.2 MULTI-AGENT DESIGN

The architectural design phase of Prometheus [20] will provide the Agent Types, the interactions between agents, and the overall system structure. The detailed design phase of Prometheus [20] provides the internals of agents that will also be described in the Agent Types subsection.

3.2.1 *Agent Types*

The multi-agent system has two kinds of agent types: a TRDL agent and a DVL agent.

3.2.1.1 *TRDL Agent*

The TRDL agent is modeled with the knowledge provided in subsection 2.1.2.

Name: TRDL Agent
Description: Monitors the Train Traffic, Determine if a TAD is applicable, Execute TAD, Message DVL when instructed by TAD
Lifetime: Instantiated when system is started. Demise when system is halted.
Goals: Monitoring Train Traffic, Determining if a TAD is applicable, Execute TADs, Inform DVL of particular TAD
Responds to: External monitoring events, Internal Execute TAD trigger

Table 3: Descriptor of a TRDL Agent

3.2.1.2 *DVL Agent*

The DVL agent is modeled with the knowledge provided in subsection 2.1.1.

Name: DVL Agent
Description: When a TRDL message is received, the corresponding TAD is executed
Lifetime: Instantiated when system is started. Demise when system is halted.
Goals: Listening to incoming TRDL messages, Execute TADs
Responds to: Incoming TRDL message

Table 4: Descriptor of a DVL Agent

3.2.2 *Agents Overview*

The agents overview in figure 12 provides insights in the internals of the two types of agents. It also shows the flow of information.

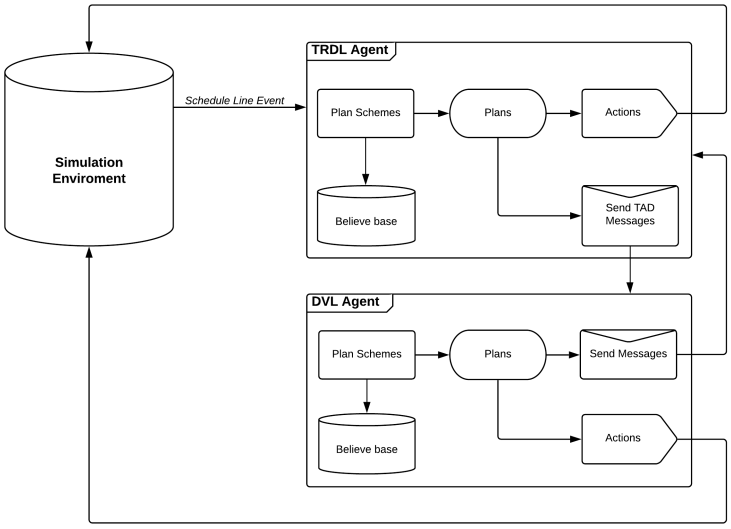

Figure 12: Agents Overview

3.2.3 *Interaction between Agents*

In the system are all the agents are connected with one another. This means that the agents are able to send messages to one another if nessasery. There is one meaningful interaction contributing to the higher goal and the sub goal: *Inform DVL of particular* `TAD`, see figure 13. This use case occurs when the `TAD` dictates that the `DVL` agent must be notified of the particular situation.

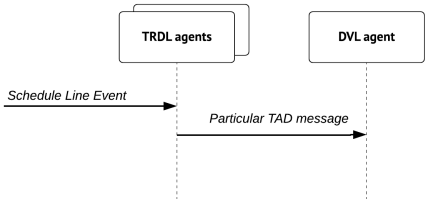

Figure 13: Interaction between agents

3.2.4 *System Overview*

A simplified overview of the system is given in figure 14 with all the agent types and connections to the environment and one another.

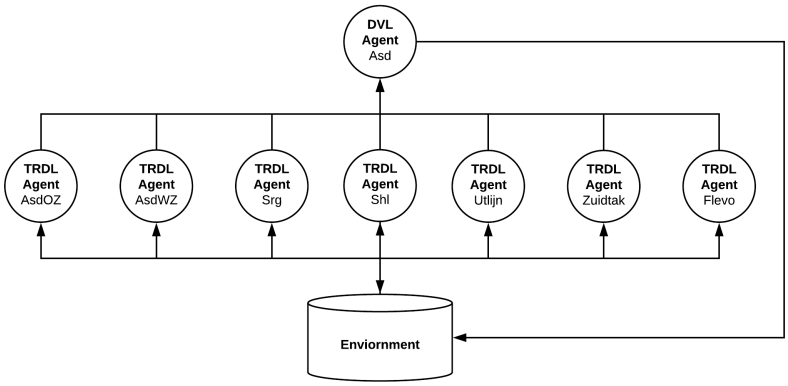

Figure 14: System Overview

3.3 MULTI-AGENT IMPLEMENTATION

In the previous sections Multi-Agent Specification and Multi-Agent Design, all the modeling is provided to actually implement this multi-agent system. The implementation of this multi-agent system will be

described in the following three subsections: The implementation in 002APL, the Connection with the Simulator, and the Models for Testing the implemented multi-agent system that is connected to ProRails distributed simulator Trinity.

3.3.1 *002APL Implementation*

In this subsection I will give a description of the 002APL implementation of the multi-agent system as modeled in the previous two sections 3.1 and 3.2. The 002APL JAVA library I used can be found on Github and is created by Bas Testerink in collaboration with Mehdi Dastani (see figure 15).

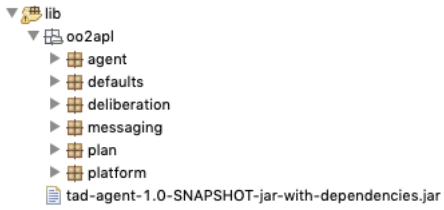

Figure 15: Java library packages of the multi-agent system

In the system all the agents are connected with one another. When the agent platform is created, the agents are getting instantiated with an adress book, which is stored in the context of the agents. The implementation of the agents is dedicated to the Train Traffic Control of the Amsterdam Area.

Figure 16: Java packages of the TRDL agents and DVL agent

Every agent in the system has four kinds of plan schemes, as shown in figure 17. Through this, the agents can be triggered by all sorts of events: external events, goal driven events, the agents' own internal driven events, or message events.

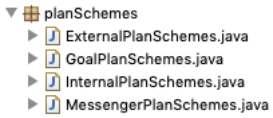

Figure 17: Java plan schemes package of the TRDL agents and DVL agent

All the TRDL agents in the system have a set of plans, as shown in figure 18. These plans are responsible for the generation of all the actions an agent can execute.

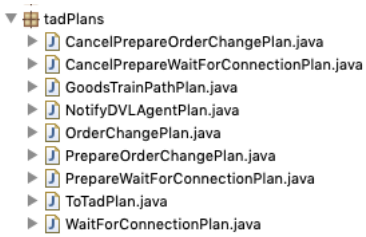

Figure 18: Java plans of the TRDL agents

3.3.1.1 *TAD Templates*

Every assigned area of a TRDL has its own TAD that applies to it. I decided to create TAD-templates for every TRDL assigned area, as can be seen in figure 19. These templates capture all the agreements of the associated TAD. The templates can be seen as the plan schemes in the multi-agent system, since every agreement of the TAD entails its own plan which entails a specific action.

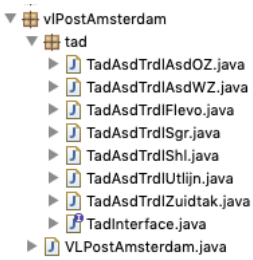

Figure 19: TADs of post Amsterdam

Every schedule line of every PPLG in an assigned area of an agent has to be processed by the agent before it is able to determine if a TAD is applicable. Before a schedule line is processed by the agent, three kinds of pre-processes are applied. Every agent has access to the util package, see figure 20, that contains the pre-processing functions:

1. **Train number conversion:** since TAD documents only specify how te handle train series.
2. **Time conversion:** the time has to be rounded to minutes to determine the delay.
3. **Train number check:** to determine if a train number is an even or odd train series.

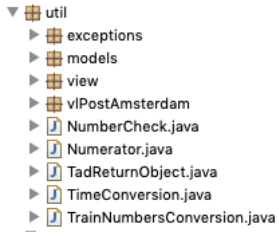

Figure 20: Java util package of the multi-agent system

When the pre-processing is done, the agent can determine if the schedule line in question is applicable for a TAD. Figures 21 - 23 show pieces of code that represent three kinds of handling agreements from a TAD-template:

- How to handle the last trains of the day
- How to handle an order change
- How to handle connections

Figure 21 shows an example of code that handles the last trains of the day. Here line 2: check the train number of a given schedule line. Line 3 - 7: checks if its a Monday - Friday. Line 8: checks if the delay is between 1 and 10 minutes. Line 9 - 13: action is set to "Wait for Connection", trainsWait are cleared, train 15889 is added and a TadReturnObject is created with these parameters and returned.

```
1  // 4389 15889 ma t/m vr (Max. 10 min)
2  if(trainNumber == 4389 && (
3          dayOfTheWeek.equals(DayOfWeek.MONDAY)    ||
4          dayOfTheWeek.equals(DayOfWeek.TUESDAY)   ||
5          dayOfTheWeek.equals(DayOfWeek.WEDNESDAY) ||
6          dayOfTheWeek.equals(DayOfWeek.THURSDAY)  ||
7          dayOfTheWeek.equals(DayOfWeek.FRIDAY)   )){
8      if(delayInMinutes > 1 && delayInMinutes <= 10){
9          action = "Wait for Connection";
10         trainsWait.clear();
11         trainsWait.add(15889);
12         TadReturnObject tadReturnObject = new TadReturnObject(...);
13         return tadReturnObject;
14     }
15 }
```

Figure 21: One TAD last train code example

Figure 22 shows an example of code that handles an order change. Here line 2 - 4: check the train number of a given schedule line to be of 700 series or 300700 series or 310700 series. Line 5 - 8: check if its an even number, if the activity equals "D" and if the schedule time of the train is HH:22 and if the delay is greater or equal to 8 minutes. Line 9 - 13: If that is the case at decision point "Lls", action is set to "Order Change", orderChange is set to "14600e - 700e" and a TadReturnObject is created with these parameters and returned. Line 14 - 18: If that is the case before decision point "Lls", action is set to "Prepare Order Change", orderChange is set to "14600e - 700e" and a TadReturnObject is created with these parameters and returned. Line 21 - 24: If that is not the case "Lls", action is set to "Check to cancel Prepare Order Change" and a TadReturnObject is created with these parameters and returned.

```
1  // 700e Shl d. -.22 8 > Lis 14600e - 700e
2  if( trainSerieNumber == 7       ||
3          trainSerieNumber == 3007 ||
4          trainSerieNumber == 3107){
5              if(checkNumber.NumberIsEven(trainNumber)){
6                  if(activity.equals("D")){
7                      if(scheduleTimeMinutes == 22){
8                          if(delayInMinutes >= 8){
9                              if(desicionPoint.equals("Lls")){
10                                 action      = "Order Change";
11                                 orderChange = "14600e - 700e";
12                                 TadReturnObject tadReturnObject = new TadReturnObject(...);
13                                 return tadReturnObject;
14                             }else{
15                                 action      = "Prepare Order Change";
16                                 orderChange = "14600e - 700e";
17                                 TadReturnObject tadReturnObject = new TadReturnObject(...);
18                                 return tadReturnObject;
19                             }
20
21                         }else if(desicionPoint.equals("Lls")){
22                             action = "Check to cancel Prepare Order Change";
23                             TadReturnObject tadReturnObject = new TadReturnObject(...);
24                             return tadReturnObject;
25                         }
26                     }
27                 }
28             }
29 }
```

Figure 22: One TAD order change code example

Figure 23 shows an example of code that handles connections. Here line 2 - 3: check the train number of a given schedule line to be of 146 series or 314600 series. Line 4 - 5: check if its an even number, and if the schedule time of the train is HH:09 or HH:39, and if the delay is greater then 1 and less or equal to 5 minutes. Line 7 - 12: If that is the case at decision point "Asdm", action is set to "Wait for Connection", extraInfo is set to "57000", trainsWait add 5700 and a TadReturnObject is created with these parameters and returned. Line 14 - 18: If that is the case before decision point "Asdm", action is set to "Prepaire Wait for Connection", extraInfo is set to "57000", trainsWait add 5700 and a TadReturnObject is created with these parameters and returned. Line 20 - 23: If that is not the case "Asdm", action is set to "Check to cancel Wait for Connection" and a TadReturnObject is created with these parameters and returned.

```
// 14600o Alm -.09/-.39 Wrt 5 Asdm 5700o wacht
if(trainSerieNumber == 146   ||
   trainSerieNumber == 3146 ){
    if(checkNumber.NumberIsEven(trainNumber) == false){
            if(scheduleTimeMinutes == 9 || scheduleTimeMinutes == 39){
                if(delayInMinutes > 1 && delayInMinutes <= 5){
                    if(desicionPoint.equals("Asdm")){
                        action    = "Wait for Connection";
                        extraInfo = "5700o";
                        trainsWait.add(5700);
                        TadReturnObject tadReturnObject = new TadReturnObject(...);
                        return tadReturnObject;
                    }else{
                        action    = "Prepare Wait for Connection";
                        extraInfo = "5700o";
                        trainsWait.add(5700);
                        TadReturnObject tadReturnObject = new TadReturnObject(...);
                        return tadReturnObject;
                    }
                }else if(desicionPoint.equals("Asdm")){
                    action = "Check to cancel Wait for Connection";
                    TadReturnObject tadReturnObject = new TadReturnObject(...);
                    return tadReturnObject;
                }
            }
    }
}
```

Figure 23: One TAD connection code example

Figure 24 shows the content of a TAD return Object.

```
public class TadReturnObject {

    String        pplg;
    int           trainNumber;
    String        activity;
    String        from;
    String        to;
    LocalDateTime scheduleTime;
    LocalDateTime setTime;
    int           delay;
    String        desicionPoint;
    int           delayInMinutes;
    String        action;
    String        orderChange;
    String        extraInfo;
    String        message;
    List<Integer> trainsWait = new ArrayList<Integer>();
```

Figure 24: TADreturnObject

3.3.2 *Connection with the Simulator*

The connection to simulator Trinity at ProRail, as mentioned in subsection 2.2.2, is obtained via a custom made library developed by Arie van de Berg (developer at ProRail). The library connects to CERTI which is the open source HLA RTI used by the Trinity simulator. The library is added to the multi-agent system via an import and is one of the first things to be executed before the agent platform is created. The HLA connection creates a Plan object showed by figure 25. This

Plan object of plans represents the simulation environment and is given to the agents when they are instantiated. From this point the agents know how to connect to the simulation environment and are able to get the real time train schedules. The amount of train schedules the agents can get is limited to the top 20 schedule lines of a given PPLG. This amount turned out to be sufficient for the purpose of this system, but can always be adjusted.

```
////////////////////////////////////////////////////////////////
// HLA
////////////////////////////////////////////////////////////////

HLAConnection hlaConnect = new DefaultHLAConnection(new String []{ }, 20);

Plan plans = hlaConnect.getPlan();
```

Figure 25: HLA connection code

3.3.3 *Models for Testing*

In order to verify if this multi-agent system within the simulation environment is doing what it is supposed to do, a disruption model has been created: this is a model of disruptions on the railway from certain points of time. These disruptions can very from, for example, trains that have entrance disruptions or delays, to broken signs or switches on the tracks. These models make it possible to simulate the specific scenarios where a TAD has to be applied and therefore test the performance of the system. These models are created in the scenario management of the simulator and are able to simulate almost every scenario.

I have created and tested multiple disruption models to verify this multi-agent system. The disruption model that I used for demonstrating the system is meant to show two types of TADs: a train that has to wait for a connection with another train and an order change of two trains. Order changes and connections are the two most occurring situations within train traffic control where TADs are applied and these are represented by this disruption scenario.

The exact model used consists of the two disruptions:

1. Train 15825 with an entrance disruption +3 minutes at PPLG ASD.
2. Train 722 is given an delay of +8 minutes from PPLG Stb.

3.3.4 *Deliberation Cycle: Sense Reason Act*

In this section I will describe the deliberation cycle of the multi-agent system. When the agent program is started, two things are done as a prerequisite:

1. A connection with the HLA simulatior is established
2. Agents output window is created

Then the agents platform is created by the following steps:

1. Agents are instantiated with (Type, PPLG book, HLA connection)
2. Agents' IDs are put in adressbook and given to the agents
3. Agents get an external TAD goal trigger

From this point the agent becomes TAD Goal driven and the deliberation cycle of sense reason act is initiated:

1. Tad goal trigger toTad plan
2. toTad Plan scans 20 schedule lines for every PPLG where the agent is responsible for and sends an internaltrigger for every schedule line to check for TAD
3. These schedule line are checked for TAD, if that is the case: the right plan is selected
4. The plan is an execute once plan and also creates the agents output that is displayed in the agents notification window.

Part III

RESULTS

4

RESULTS

In the previous chapter I have presented the methodology of my study. In this chapter I will present the results of the developed MAS of Train Traffic Control as described in the methodology in three parts: Firstly, the Multi-Agent System of Train Traffic Control part describes the outcome of the implemented MAS. Secondly, the Systems Setup part describes the setup in which the MAS was implemented. Thirdly, the Multi-Agent System and Simulation Run part provides an detailed overview of the simulation run.

4.1 MULTI-AGENT SYSTEM OF TRAIN TRAFFIC CONTROL

The main result of this study is a proof of concept (POC) of a successfully modeled, implemented and working multi-agent system of Train Traffic Control. The developed system is connected to a distributed simulation environment and capable of determining if a TAD is to be applied for the current state of this environment. The POC as described in the implementation section 3.3, can be seen as a foundation to eventually replace the need of TRDLs as human-in-the-loop simulators. This multi-agent system is focused on the Traffic Control post Amsterdam and area, since it can easily be implemented for all posts, it can work on a national scale.

4.2 SYSTEMS SETUP

The section will describe the system setup that is used for the demonstration of the multi-agent system at ProRail. The actual detailed simulation run of the demonstration is provided in section 4.3. All the simulations are performed on Windows 10 machines.

4.2.1 *Multi-Agent System as Executable JAR*

Due to connection issues from the IDE of my laptop to the CERTI (HLA RTI) of the simulator, the decision was made te run the program from an executable JAR. An executable JAR is an executable Java program, along with any libraries the program uses, that is packaged in a JAR file.

The multi-agent system is executed as follows: from the location of the JAR file: right click on the mouse and select: open PowerShell-window. In the Windows Powershell Type: *java -jar TadAgentDemo.jar* and press enter to execute the program. Then type: *'tadGoal'* in the

powershell and press enter to start the multi-agent system. A Java Window will open, where all the notifications of the TRDL agents are outputted.

4.2.2 *Simulator*

In order for the multi-agent system to work, the right simulation settings have to be selected, because the MAS is created for Traffic Control post Amsterdam. The simulation consists of FRISO and PRLgame. In FRISO the simulation model is selected (Amsterdam), then the disruption model is selected from the scenario management. If the model and scenario are all loaded and PRLgame is started the simulation can start.

4.3 MULTI-AGENT SYSTEM AND SIMULATION RUN

The results of the simulation run using the disruptive model as described in section 3.3.3 are presented below in form of screenshots of the corresponding simulation and the agents' output. Important note: in order to read and interpret the screenshots zooming in is required. The screenshot images are too big to display them in any other way. I will first explain the different screens that are shown in the screenshots below from left to right see figures 26- 31:

- Figure 26: Simulation manager
- Figure 27: Schedule lines per PPLG
- Figure 28: Graphical representation of the trains
- Figure 29: FRISO's grafical representation of the trains
- Figure 30: Windows Powershell
- Figure 31: Agent Notification window

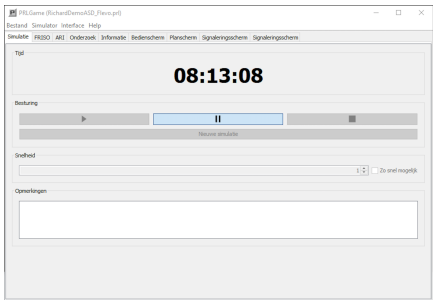

Figure 26: Simulation manager

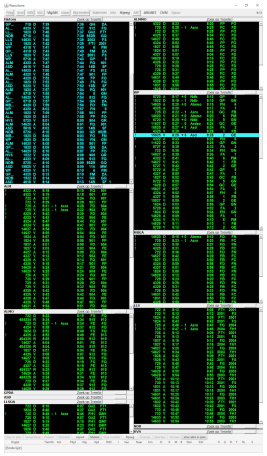

Figure 27: Schedule lines per PPLG

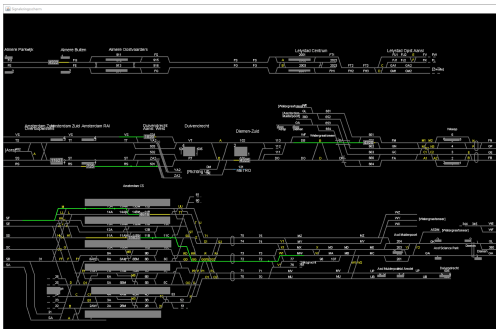

Figure 28: Schedule lines per PPLG

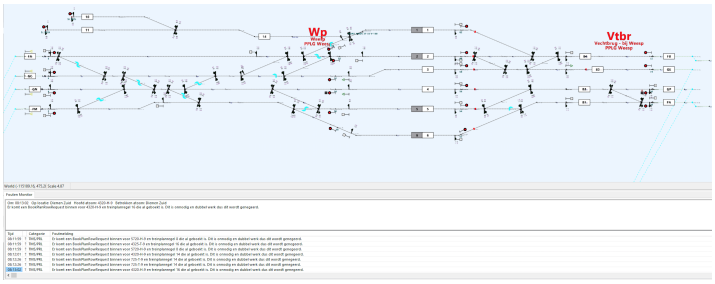

Figure 29: FRISO's grafical representation of the trains

Figure 30: Windows Powershell

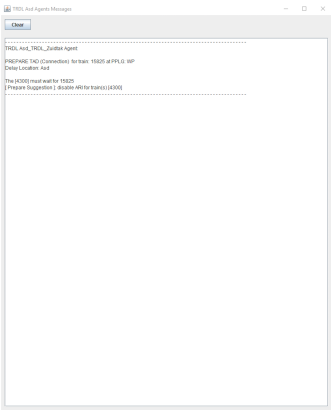

Figure 31: Agent Notification window

De first screenshot 32 below shows that the highlighted line of the schedule line per PPLG event triggered the MAS to come into action and produced an output on the agent notification window. The agents message is Prepare connection TAD for train 15825. Prepare connection means the TRDL should make preparations for a possible wait for connection for the given train.

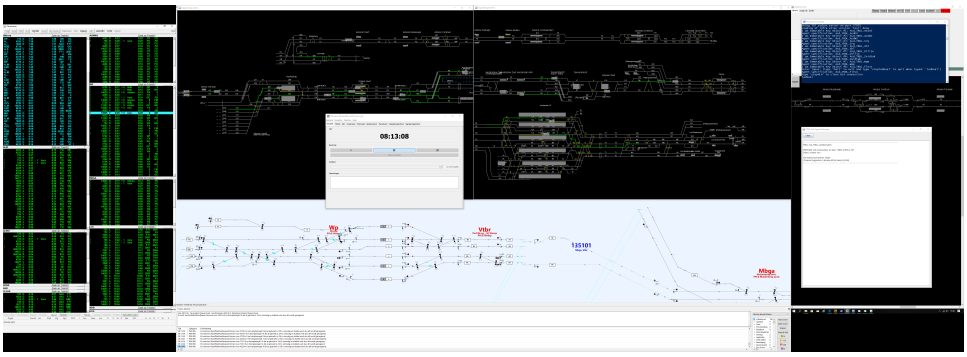

Figure 32: Screenshot 1: Simulation Run Time 08:13:08

De second screenshot 33 below shows that the highlighted line of the schedule line per PPLG event triggered the MAS to come into action and produced an output on the agent notification window. The agents message is Prepare order change TAD for train 722. Prepare order change means the TRDL should make preparations for a possible order change for the given train.

Figure 33: Screenshot 2: Simulation Run Time 08:15:05

De third screenshot 34 below shows that the highlighted line of the schedule line per PPLG event triggered the MAS to come into action and produced an output on the agent notification window. The agents message is Definite connection TAD for train 15825, since the delay is within the threshold of the TAD at decision point. Definite connection TAD means the TRDL should let the given train wait for its connection.

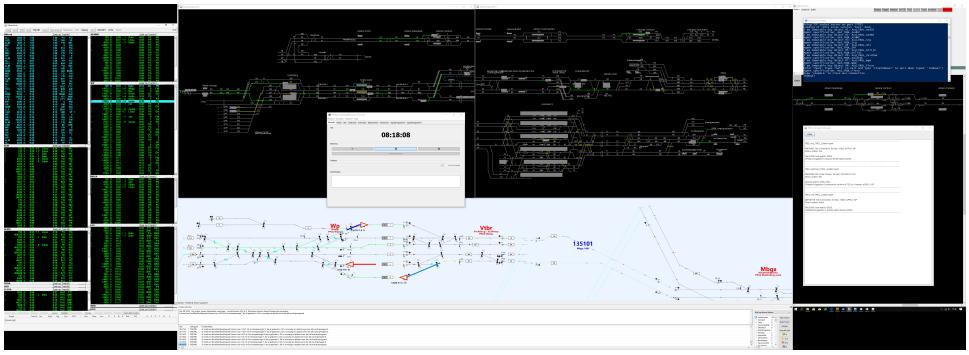

Figure 34: Screenshot 3: Simulation Run Time 08:18:08

Screenshots 3 until 8 shown in figures 34 - 39, perfectly demonstrate what is called the tail lights effect. This is what happens when the wait for connection TAD is not executed. Train 15825 arrives at platform 2 which has a connection with train 4325 that is already at platform 1: as can be seen directly below the simulator manager.

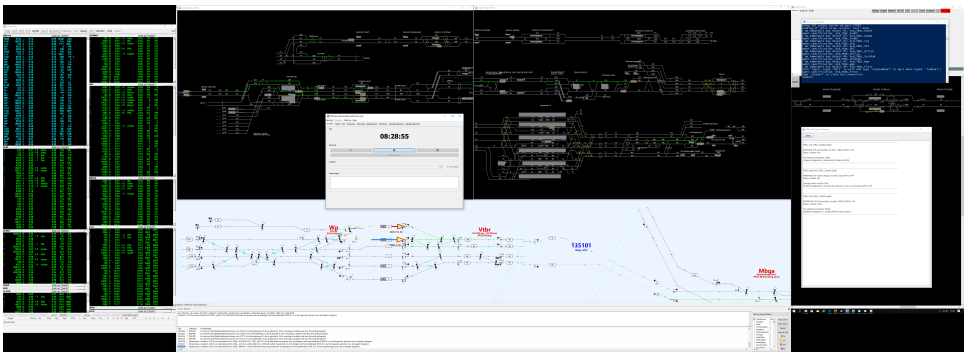

Figure 35: Screenshot 4: Simulation Run Time 08:28:55

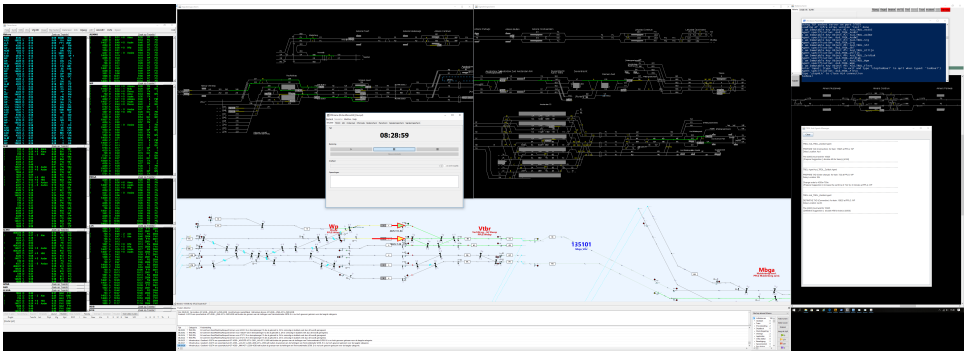

Figure 36: Screenshot 5: Simulation Run Time 08:28:59

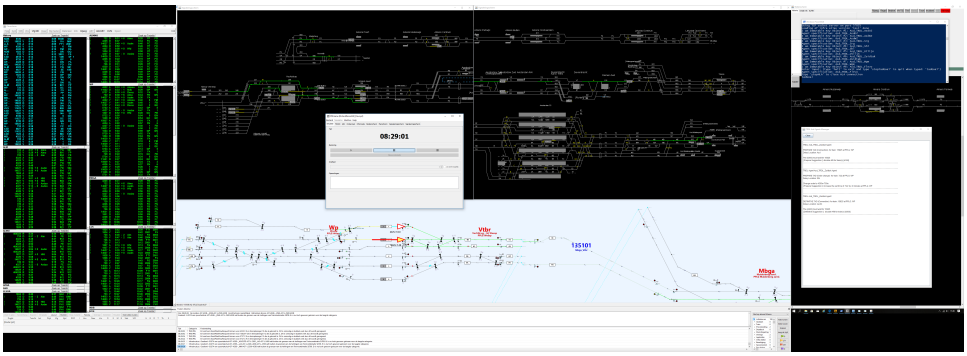

Figure 37: Screenshot 6: Simulation Run Time 08:29:01

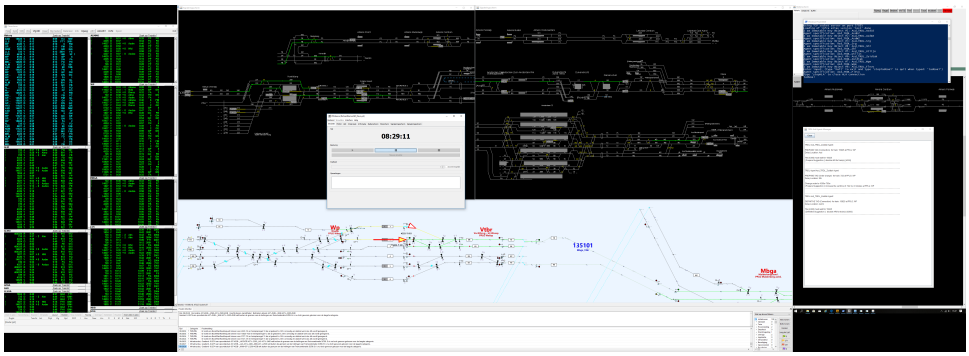

Figure 38: Screenshot 7: Simulation Run Time 08:29:11

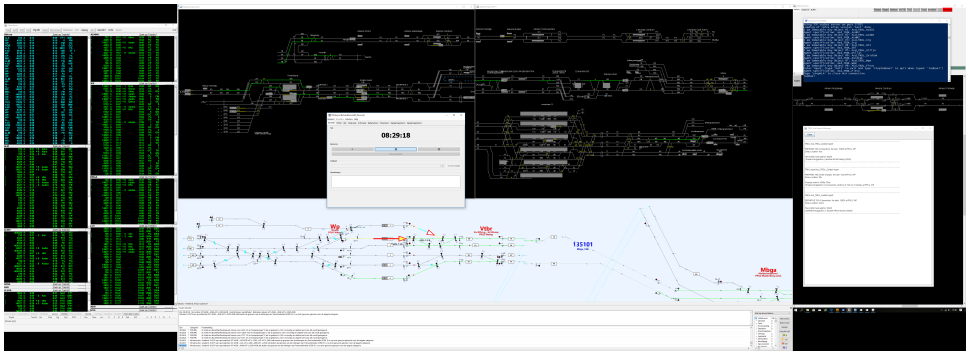

Figure 39: Screenshot 8: Simulation Run Time 08:29:18

De ninth screenshot 40 below shows that the highlighted line of the schedule line per PPLG event did not trigger the MAS to come into action and produced an output on the agent notification window. The agents message is still active Prepare connection TAD for train 722.

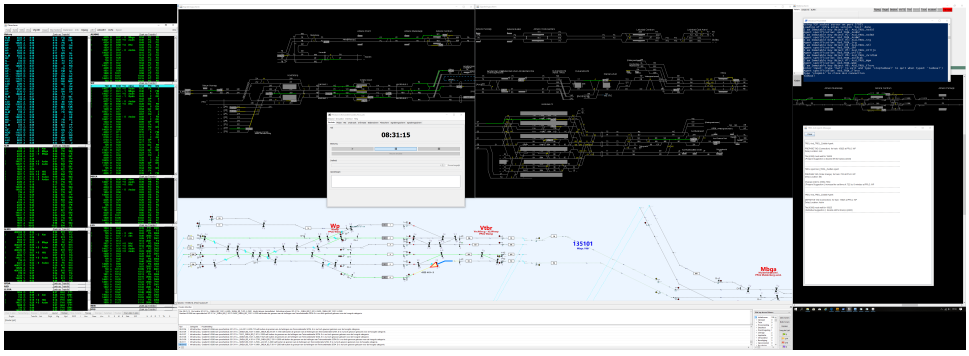

Figure 40: Screenshot 9: Simulation Run Time 08:31:15

De tenth screenshot 41 below shows that the highlighted line of the schedule line per PPLG event triggered the MAS to come into action

and produced an output on the agent notification window. The agents message is Cancel connection TAD for train 722, since the delay is lower than the threshold of the TAD at decision point. Cancel connection TAD means the TRDL should cancel the preparation for a possible connection for the given train.

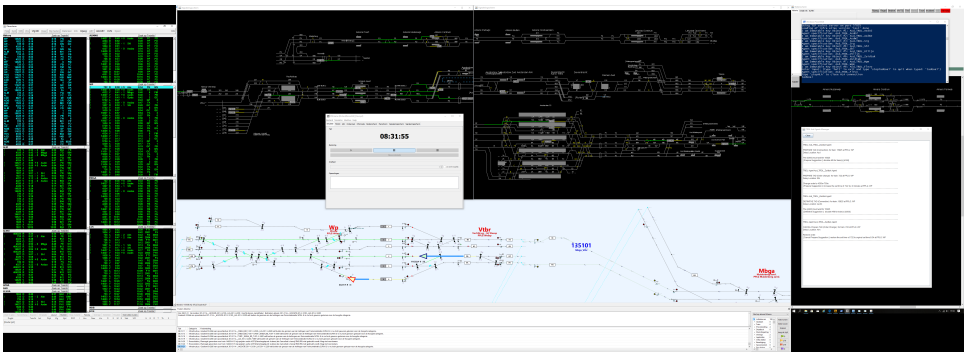

Figure 41: Screenshot 10: Simulation Run Time 08:31:55

Part IV

CONCLUSION AND DISCUSSION

5 CONCLUSION

In this chapter, I will discuss the most important findings and provide answers to the research question and related sub-questions. The research question was:

> How to specify, design, and implement a multi-agent system within a simulated environment that assists train traffic controllers (TRDLs) in their work that is not handled by ProRail's Automatic Railway Setting (ARI)?

Subquestions:

1. How do Train Traffic Controllers (TRDLs) work with ARI in everyday practice?
2. What are the limitations of ARI that TRDLs encounter in practice?
3. How can a multi-agent system be modeled to overcome these limitations?
4. What is an efficient and effective agent based design method for a multi-agent system?
5. How to implement a multi-agent system onto ProRail's distributed simulation environment?

All the traffic management work of Train Traffic Controllers is, in the optimal scenario, done automatically by the control systems such as ARI. The limitations of the ARI arises in the situations where trains deviate too much from the schedule. This multi-agent system solves the limitation of applying handling procedures called TADs on particular delay scenarios.

The Prometheus methodology turned out to be a very efficient and effective method of modeling agent systems, because it is intended to be interpreted in conjunction with, in this case my own, common sense. I managed to model a multi-agent system of Train Traffic Control using the Prometheus methodology. The modeled multi-agent system was successfully implemented in a JAVA program using the OO2APL library. Via a HLA library the MAS was able to connect as an federate to ProRails HLA simulator Trininty. Disruption models make it possible to simulate the specific scenarios where a TAD has to be applied and therefore test the performance of the system. This MAS is quite generic, because, for instance, a change of context makes the system applicable for every train traffic control area within ProRail. Therefore the system can scale quite easily, but it is also simple to add more capabilities to the agents.

I was able to successfully demonstrate this working multi-agent system within a distributed simulation environment that notifies when a TAD should be applied to ProRail. The system not only does it at the decision point, but it gives the TRDLs a heads up to possibly prepare a TAD.

6 DISCUSSION

6.1 INTERPRETATION OF RESULTS

This study led to an actual working proof of concept. Specifying the MAS was done by extensive literature study and talking with lots of different employees at ProRail. When a good understanding of TRDLs was established we decided to create a MAS that could carry out TADs. From that point the designing phase started and with the Prometheus methodology the systems' specification and design was created. Other methodologies where considered, but Prometheus was chosen as the best fit. This led to the implementation phase of the research project, which created a fundamental multi-agent system of DVL and TRDL agents.

This study provides new and different insights into the possibilities of implementing agent based technology within the train traffic control domain. This is how multi-agent programming technology can find its way into the industry by providing a proof of concept that hopefully guides the development of autonomous agents and multi-agent systems in standard programming technology.

6.2 LIMITATIONS

The biggest limitation of this research project was the time in which the MAS had to be developed. I initially got six month to specify, design, and implement a multi-agent system. The process takes a lot of time.

One of the limitations of the agent system due to time is the lack of active communication. The MAS is implemented in a way that communication/messages between agents are possible, but for TAD this was not initially necessary. Also the OO2APL library has no initial reference to other agents. This means you need to create an address book that is given to the agents in order to communicate to one another. Another limitation of the MAS the agents' outputs. Also due to lack of time the agent's output is purely textual. The system as works an advice system for TRDLs that does simulation runs for training purposes.

The main limitation for connecting, testing and verifying the MAS was the speed of the simulations. It took a lot of time booting up the simulator and get it running. In some occasions it would crash or your initial disruption model was not testing the right premise. Then a whole new simulation run had to be initiated. The connection of

the MAS to the simulator was also a difficult process. I was dependent of programmers at ProRail to write the HLA connection library. I never got to connect directly from my laptop to the simulator. Only with an executable jar of the MAS program running it on the simulator computer itself. The main disadvantage was that it took a lot of extra time when adjustments to the MAS had to be made, since with every change a new jar had to be made and transferred to the simulator computer.

6.3 FUTURE RESEARCH

The present study provides a proof of concept and a foundation to build upon. ProRail was very pleased with the results and are exploring new possible applications of agent based technology. This multi-agent system can be further extended, since it is quite generic, for instance only a change of context makes the system applicable for every train traffic control area within ProRail. Capabilities can be simply added to the agents. This makes the system scale quite easily.

Different implementations and configurations of collaboration of TRDL agents can be explored. It will be interesting to research how this effects train traffic control.

An interesting follow-up study would be on how to implementing a multi-agent system. I had a very interesting and also philosophical discussion with my University supervisor of making the environment of the system smarter. In the current literature there is not much information on this topic.

In my own research time Wilco Tielman from ProRail and I also tried to use machine learning on actual train traffic data to predict when an order change has to be applied. We managed to create an WEKA order model, but due to limited time and the scope of my research project, we could not explore this interesting idea, see figure 42.

Figure 42: Java Weka folder of the multi-agents system

6.4 CONCLUDING REMARKS

The present study shows that a multi-agent system can be successfully specified, designed and implemented to support TRDLs in their work within a distributed simulated environment. This study is meaningful for ProRail because it demonstrates the power of agent based

technology for Train Traffic Control and the many possibilities beyond that.

BIBLIOGRAPHY

[1] I. Vaes B. Kupers. *Software Requirements Specification (SRS) Automatische Rijweginstelling (ARI)*. Version 12.0. ProRail ICT Logistiek Be- en Bijsturing, 2016.

[2] Ana LC Bazzan. "Opportunities for multiagent systems and multiagent reinforcement learning in traffic control." In: *Autonomous Agents and Multi-Agent Systems* 18.3 (2009), p. 342.

[3] Eric Bonabeau. "Agent-based modeling: Methods and techniques for simulating human systems." In: *Proceedings of the National Academy of Sciences* 99.suppl 3 (2002), pp. 7280–7287.

[4] Rafael H Bordini, Lars Braubach, Mehdi Dastani, A El F Seghrouchni, Jorge J Gomez-Sanz, Joao Leite, Gregory O'Hare, Alexander Pokahr, and Alessandro Ricci. "A survey of programming languages and platforms for multi-agent systems." In: *Informatica* 30.1 (2006).

[5] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. "Tropos: An agent-oriented software development methodology." In: *Autonomous Agents and Multi-Agent Systems* 8.3 (2004), pp. 203–236.

[6] ProRail Verkeersleiding Bedrijfsbureau Centraal. *Werkwijze treindienstleider Mei 2018*. Mei 2018. Versie Mei 2018. ProRail, 2018.

[7] Bo Chen and Harry H Cheng. "A review of the applications of agent technology in traffic and transportation systems." In: *IEEE Transactions on Intelligent Transportation Systems* 11.2 (2010), pp. 485–497.

[8] Stefania Costantini and Arianna Tocchio. "A logic programming language for multi-agent systems." In: *European Workshop on Logics in Artificial Intelligence*. Springer. 2002, pp. 1–13.

[9] Mehdi Dastani and John-Jules Ch Meyer. "A practical agent programming language." In: *International Workshop on Programming Multi-Agent Systems*. Springer. 2007, pp. 107–123.

[10] Mehdi Dastani and Bas Testerink. "Design patterns for multi-agent programming." In: *International Journal of Agent-Oriented Software Engineering* 5.2-3 (2016), pp. 167–202.

[11] Stan Franklin and Art Graesser. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents." In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer. 1996, pp. 21–35.

[12] HLA Working Group et al. "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-Federate interface specification." In: *IEEE Standard* (2000), pp. 1516–2000.

[13] HLA Working Group et al. "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-framework and rules." In: *IEEE Standard* (2000), pp. 1516–2000.

[14] HLA Working Group et al. "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-object model template (OMT) specification." In: *IEEE Standard* (2000), pp. 1516–2000.

[15] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. "JACK intelligent agents-summary of an agent infrastructure." In: *5th International conference on autonomous agents*. 2001.

[16] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. "A roadmap of agent research and development." In: *Autonomous agents and multi-agent systems* 1.1 (1998), pp. 7–38.

[17] David Kinny, Michael Georgeff, and Anand Rao. "A methodology and modelling technique for systems of BDI agents." In: *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer. 1996, pp. 56–71.

[18] Martijn Mes, Matthieu Van Der Heijden, and Aart Van Harten. "Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems." In: *European journal of operational research* 181.1 (2007), pp. 59–75.

[19] AD Middelkoop and L Loeve. "Simulation of traffic management with FRISO." In: *WIT Transactions on the Built Environment* 88 (2006).

[20] Lin Padgham and Michael Winikoff. "Prometheus: A methodology for developing intelligent agents." In: *International Workshop on Agent-Oriented Software Engineering*. Springer. 2002, pp. 174–185.

[21] Praveen Paruchuri, Alok Reddy Pullalarevu, and Kamalakar Karlapalem. "Multi agent simulation of unorganized traffic." In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. ACM. 2002, pp. 176–183.

[22] ProRail. *Software Design Description (SDD) Automatische Rijweginstelling (ARI)*. Version 19.0. ProRail, 2017.

[23] ProRail. *Organisatie*. 2018. URL: https://www.prorail.nl/reizigers/wie-zijn-we/organisatie (visited on 01/08/2018).

[24] ProRail. *Over ProRail*. 2018. URL: https://www.prorail.nl/reizigers/over-prorail (visited on 01/08/2018).

[25] ProRail. *Over Verkeersleiding*. 2018. URL: https://prorailbv.sharepoint.com/sites/focus/info-over/org/operatie/VL (visited on 02/08/2018).

[26] ProRail. *Over Verkeersleidingsposten*. 2018. URL: http://www.jaarverslagprorail.nl/kerncijfers/ (visited on 01/08/2018).

[27] ProRail. *Over Verkeersleidingsposten*. 2018. URL: https://prorailbv.sharepoint.com/sites/focus/info-over/org/operatie/VL/Paginas/Verkeersleidingsposten.aspx (visited on 02/08/2018).

[28] ProRail. *ProRail in cijfers*. 2018. URL: https://www.prorail.nl/reizigers/over-prorail/wat-doet-prorail/prorail-in-cijfers (visited on 01/08/2018).

[29] ProRail. *Wat doet ProRail*. 2018. URL: https://www.prorail.nl/reizigers/wat-doet-prorail (visited on 01/08/2018).

[30] Hugo Proenca and Eugenio Oliveira. "MARCS multi-agent railway control system." In: *Ibero-American Conference on Artificial Intelligence*. Springer. 2004, pp. 12–21.

[31] Anand S Rao and Michael P Georgeff. "Modeling rational agents within a BDI-architecture." In: *KR* 91 (1991), pp. 473–484.

[32] Rosaldo JF Rossetti, Rafael H Bordini, Ana LC Bazzan, Sergio Bampi, Ronghui Liu, and Dirck Van Vliet. "Using BDI agents to improve driver modelling in a commuter scenario." In: *Transportation Research Part C: Emerging Technologies* 10.5-6 (2002), pp. 373–398.

[33] ProRail ICT Services. *Gebruiksvoorschrift: Voor de bediening van wissel- en seininrichtingen met behulp van Procesleiding Rijwegen*. 1st. Version 42.0 (1.0). ProRail, 2016.

[34] Robert Siegfried. *Modeling and simulation of complex systems: A framework for efficient agent-based modeling and simulation*. Springer, 2014.

[35] J. Steneker, M. van Schayk, B.D. Cunes, J.C. Goosen, and L.S. Koelewijn. *Conceptueelmodel FRISO*. Version 4.4.11 RGS. INCONTROL Simulation Solutions, 2018.

[36] J. Steneker, M. van Schayk, B.D. Cunes, J.C. Goosen, and L.S. Koelewijn. *Technisch ontwerp FRISO*. Version 4.4.11 RGS. INCONTROL Simulation Solutions, 2018.

[37] Herman Sulmann. *Railverkeersleiding van Sein tot Sein*. 1st. Koninklijke van de Garde, Zaltbommel, 2000.

[38] Okan Topçu, Umut Durak, Halit Oğuztüzün, and Levent Yilmaz. *Distributed simulation: A model driven engineering approach*. Springer, 2016.

[39] Kagan Tumer and Adrian Agogino. "Distributed agent-based air traffic flow management." In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM. 2007, p. 255.

[40] I. Vaes. "Beveiliging en beheersing anno 2006." In: (2006).

[41] ProRail Verkeersleiding Afdeling Veiligheid. *Handboek treindienstleider Mei 2018*. Mei 2018. Versie Mei 2018. ProRail, 2018.

[42] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[43] Michael Wooldridge, Nicholas R Jennings, and David Kinny. "A methodology for agent-oriented analysis and design." In: *Proceedings of the third annual conference on Autonomous Agents*. ACM. 1999, pp. 69–76.

[44] Bernard P Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.

[45] Movares ingenieurs en adviseurs. *Telefoonnummers van Verkeersleidinggebieden in Nederland*. Version 7.4a. 2017.

[46] VL Vakopleidingen | basisleertraject treindienstleider. *Systemen ARI en ABT*. Version 5.0. ProRail, 2016.