Programmeerproject 2:
Voorstudie Modeltrein Software Toepassing

Jonas Brüll, 0587194
(Jonas.Simon.E.Brull@vub.be)

2/3ba Computerwetenschappen

Academiejaar 2024-2025

Contents

1 Introductie

Dit document beschrijft het ontwerp van de implementatie van een Modeltrein Software Toepassing in de programmeertaal Racket. Het is een deel van het opleidingsonderdeel “Programmeerproject 2”.

Het onderwerp van het project is de ontwikkeling van een controlesysteem voor een modelspoor. De modeltreinen kunnen volledig digitaal aangestuurd worden door een extern programma aangestuurd met digitale commando's. Digitale locomotieven gebruiken het elektrisch circuit op de sporen als een datacommunicatiebus. Niet alleen de snelheid van de locomotieven wordt op deze manier digitaal geregeld, maar ook andere functionaliteiten zoals het veranderen van wissels

Tijdens deze opdracht wordt een controlesysteem ontwikkeld om modeltreinen aan te sturen. Het is de bedoeling dat tegen het einde van het academiejaar een systeem ontwikkeld werd dat gebruikers toelaat vanaf hun computer treinen over een modelspoor te laten rijden.

Het project bestaat uit drie grote componenten:

- Een Command Station (Z21) dat de verschillende elementen van het modelspoor aanstuurt. Dit Command Station communiceert met de hardware via het Digital Command Control (DCC) protocol. De software hiervoor zit reeds ingebouwd in het Command Station. Het Command Station is aan te sturen door er via het netwerk commando's naar te sturen door middel van de Z21-bibliotheek.
- Infrabel is de component die instaat voor de communicatie tussen de software en de modelbouwhardware (detectie van treinen en aansturing van treinen en wissels) en het doorlopend beheer van de infrastructuur (bv. automatisch remsysteem). Dit is deel van de infrastructuur en moet permanent draaien. Infrabel kan draaien op een computer of op een Raspberry Pi (een kleine gelimiteerde computer).
- NMBS en de grafische interface (GUI). NMBS staat in voor de functionaliteit die geen logisch onderdeel is van de infrastructuur maar er bovenop bouwt, zoals een grafische interface, het uitstippelen van trajecten of het opstellen van tijdstabellen. NMBS draait op je eigen computer en communiceert met Infrabel. Naar het einde van het project toe zal die communicatie via een netwerkverbinding (TCP) dienen te gebeuren.

Zowel de Infrabel module, als de Provider module zullen steunen op een railway module, deze de fysieke spoorwegopstelling (of diens simulator) modeleert en deze steeds dezelfde staat zal hebben.

2 Railway

De railway-module houdt ten allen tijden de staat van de fysieke modelspoorweg bij. Deze module bestaat uit verschillende atomaire abstracties van de verschillende hardware-componenten, die de modelspoorweg bezit, deze allemaal aangestuurd worden vanuit de main-klasse.

2.1 Main

De `Main` klasse is het startpunt van railway-module van de software en functioneert tevens ook als een interface. Om een digitaal equivalent van de staat van de modelspoorweg bij te houden, bezit deze klasse hiervoor ook de nodige getters en setters. Deze klasse berekent zelf welke spoorwegsegmenten bezet zijn en welke vrij, aan de hand van de locatie van de treinen. Hierdoor is het niet mogelijk om de status van spoorwegsegmenten, detectieblokken en wissels aan te passen. Om botsingen te voorkomen, is het well mogelijk deze te reserveren.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(pair \rightarrow Main)$	Constructor
`get-ajacent`	$(symbol \rightarrow pair)$	Geeft lijst van alle 'buren' van een segment
`get-crossing-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van overwegen
`get-crossing-state`	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven overweg
`set-crossing-state!`	$(symbol\ symbol \rightarrow \varnothing)$	Update gegeven overweg naar gegeven status
`get-detection-block-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van detectie blokken
`get-detection-block-state`	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven detectieblok
`reserve-detection-block!`	$(symbol \rightarrow \varnothing)$	Reserveert gegeven detectieblok
`get-light-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van lichten
`get-light-signal`	$(symbol \rightarrow symbol)$	Geeft signaal weer van gegeven licht
`set-light-signal!`	$(symbol\ symbol \rightarrow \varnothing)$	Update gegeven licht met gegeven signaal
`get-segment-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van segmenten
`get-segment-state`	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven segment
`reserve-segment!`	$(symbol \rightarrow \varnothing)$	Reserveert gegeven spoorwegsegment
`get-switch-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van wissels
`get-switch-position`	$(symbol \rightarrow symbol)$	Geeft postitie weer van gegeven wissel
`set-switch-position!`	$(symbol\ symbol \rightarrow \varnothing)$	Update gegeven wissel met gegeven positie
`reserve-switch!`	$(symbol \rightarrow \varnothing)$	Reserveert gegeven wissel
`get-train-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van treinen
`get-train-speed`	$(symbol \rightarrow symbol)$	Geeft snelheid weer van gegeven trein
`set-train-speed!`	$(symbol\ symbol \rightarrow \varnothing)$	Update gegeven trein met gegeven snelheid

Table 1: Operaties van de Railway/Main klasse

2.2 Crossing

De `Crossing` klasse is de atomaire abstractie van een spoorwegoverweg.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(symbol \rightarrow Crossing)$	Constructor
`get-state`	$(\varnothing \rightarrow symbol)$	Geeft de status van de overweg
`set-state!`	$(symbol \rightarrow \varnothing)$	Verandert de status van de overweg, waardoor deze opent/sluit

Table 2: Operaties van de Railway/Crossing klasse

2.3 Detection-block

De `Detection-block` klasse is de atomaire abstractie van een detectieblok. Deze is een uitbreiding van de `Segment` klasse.

Naam	Signatuur	Uitleg
`make-object`	$(symbol \rightarrow Detection\text{-}block)$	Constructor
`get-state`	$(\varnothing \rightarrow symbol)$	Geeft de status van het detection block
`set-state!`	$(symbol \rightarrow \varnothing)$	Update de status van het detection block

Table 3: Operaties van de Railway/Detection-block klasse

2.4 Light

De `Light` klasse is de atomaire abstractie van een licht.

Naam	Signatuur	Uitleg
`make-object`	$(symbol \rightarrow Light)$	Constructor
`get-signal`	$(\varnothing \rightarrow symbol)$	Geeft het signaal van het Light
`set-signal!`	$(symbol \rightarrow \varnothing)$	Update het signaal van het Light

Table 4: Operaties van de Railway/Light klasse

2.5 Segment

De `Segment` klasse is de atomaire abstractie van een spoorwegsegment.

Naam	Signatuur	Uitleg
`make-object`	$(symbol \rightarrow Segment)$	Constructor
`get-state`	$(\varnothing \rightarrow symbol)$	Geeft de status van het segment
`set-state!`	$(symbol \rightarrow \varnothing)$	Update de status van het segment

Table 5: Operaties van de Railway/Segment klasse

2.6 Switch

De `Switch` klasse is de atomaire abstractie van een spoorwegwissel.

Naam	Signatuur	Uitleg
`make-object`	$(symbol \rightarrow Switch)$	Constructor
`get-position`	$(\varnothing \rightarrow symbol)$	Geeft de status van de wissel
`set-position!`	$(symbol \rightarrow \varnothing)$	Update de status van de wissel

Table 6: Operaties van de Railway/Switch klasse

2.7 Switch-3way

De `Switch-3way` klasse is de atomaire abstractie van een spoorwegwissel deze van 3 sporen laat samenkomen tot 1 spoor. Deze klasse is een uitbreiding is van de `Switch` klasse.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(symbol \rightarrow Switch\text{-}3way)$	Constructor
`get-position`	$(\varnothing \rightarrow symbol)$	Geeft de status van de wissel
`set-position!`	$(symbol \rightarrow \varnothing)$	Update de status van de wissel

Table 7: Operaties van de Railway/Switch klasse

2.8 Switch-cross

De `Switch-cross` klasse is de atomaire abstractie van een spoorwegwissel deze de kruising maakt tussen 2 keer 2 sporen. Deze klasse is een uitbreiding van de `Switch` klasse.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(symbol \rightarrow Switch\text{-}cross)$	Constructor
`get-position`	$(\varnothing \rightarrow symbol)$	Geeft de status van de wissel
`set-position-1!`	$(symbol \rightarrow \varnothing)$	Update de status van het eerste paar in de wissel
`set-position-2!`	$(symbol \rightarrow \varnothing)$	Update de status van het tweede paar in de wissel

Table 8: Operaties van de Railway/Switch klasse

2.9 Train

De `Train` klasse is de atomaire abstractie van een trein.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(\varnothing \rightarrow Train)$	Constructor
`get-location`	$(\varnothing \rightarrow symbol)$	Geeft terug op welk segment/detectie-blok/wissel de trein is
`get-next-location`	$(\varnothing \rightarrow symbol)$	Geeft volgende segment/detectie-blok/wissel van de trein
`get-train-speed`	$(\varnothing \rightarrow symbol)$	Geeft de snelheid van de trein
`set-train-speed!`	$(symbol \rightarrow \varnothing)$	Update de snelheid van de trein

Table 9: Operaties van de Railway/Train klasse

3 Provider

De Provider-module is een abstractie van de software die gebruikt kan worden door clienten (bijvoorbeeld NMBS). Deze module is verantwoordelijk voor het beheren van routes op het treinnetwerk, moet ervoor zorgen dat klanten correcte timetables hebben en het moet mogelijk zijn om na een crash de huidige staat van de fysieke spoorweg terug te krijgen.

3.1 Main

De `Main` klasse is het startpunt van de module en tevens ook de interface naar andere modules toe. De module bestaat uit twee onafhankelijke delen, een Grafische User Interface (GUI) en de interne logica (Logic), waar de main-klasse de interface speelt tussen de twee.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(\varnothing \rightarrow \mathit{Provider})$	Constructor
`get-x`	$(\mathit{symbol} \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow \mathit{symbol})$	Placeholder voor setters voor updaten van informatie

Table 10: Operaties van de Provider/Main klasse

3.2 GUI

3.2.1 Startup

De `Startup` klasse is de grafische representatie van het opstart menu, waar onder andere de configuratie voor de TCP verbinding zal kunnen gebeuren.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(\varnothing \rightarrow \mathit{Startup})$	Constructor

Table 11: Operaties van de Provider/GUI/Startup klasse

3.2.2 Command&Control

De `Command&Control` klasse is de grafische user interface deze module, de gebruikers van de Provider zullen hier de volledige applicatie meer kunnen besturen.

Naam	**Signatuur**	**Uitleg**
`make-object`	$(\varnothing \rightarrow \mathit{C\&C})$	Constructor
`update!`	$(\varnothing \rightarrow \varnothing)$	Update de GUI naar de nieuwe staat van het systeem

Table 12: Operaties van de Provider/GUI/Command&Control klasse

3.3 Logic

3.3.1 App

De `App` klasse staat in voor de interne logica van het provider-systeem. Verschillende functies worden gedelegeerd naar de verschillende klasses waar deze op steunt.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow App)$	Constructor
`get-x`	$(symbol \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

Table 13: Operaties van de Provider/Logic/App klasse

3.3.2 Startup

De `Startup` klasse is verantwoordelijk voor alle logica die gepaart gaat met het opstarten van de applicatie

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Startup)$	Constructor
`get-x`	$(symbol \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

Table 14: Operaties van de Provider/Logic/Startup klasse

3.3.3 TCP

De `TCP` klasse is verantwoordelijk voor de communicatie met de Infrabel module.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow TCP)$	Constructor
`send-message`	$(string \rightarrow \varnothing)$	Verstuur een bericht naar de Infrabel module

Table 15: Operaties van de Provider/Logic/TCP klasse

3.3.4 Route-calculator

De `Route-calculator` klasse is verantwoordelijk voor het berekenen van de korste route tussen twee segmenten.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Route\text{-}calculator)$	Constructor
`calculate-shortest-between`	$(symbol\ symbol \rightarrow pair)$	Berekent kortste route

Table 16: Operaties van de Provider/Logic/Route-calculator klasse

3.3.5 Concurrency

De `Concurrency` klasse is verantwoordelijk voor het opstaan en inladen van de software.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Concurrency)$	Constructor
`save`	$(string \rightarrow \varnothing)$	Slaat huidige staat van de software op.
`load!`	$(string \rightarrow \varnothing)$	Laadt de staat van de software in van een bestand.

Table 17: Operaties van de Provider/Logic/Concurrency klasse

3.3.6 Timetables

De `Timetables` klasse is verantwoordelijk voor het genereren van tijdstabellen.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Timetables)$	Constructor
`generate-timetable`	$(symbol \rightarrow pair)$	Genereert timetables voor trein

Table 18: Operaties van de Provider/Logic/Timetables klasse

4 Infrabel

De Infrabel-module is verantwoordelijk voor het beheren van het treinnetwerk, moet ervoor zorgen dat het volledige netwerk aangestuurt kan worden en dat dit veilig gebeurt.

4.1 Main

De `Main` klasse is het startpunt van de module en tevens ook de interface naar andere modules toe. De module bestaat uit twee onafhankelijke delen, een Grafische User Interface (GUI) en de interne logica (Logic), waar de main-klasse de interface speelt tussen de twee.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Main)$	Constructor
`get-x`	$(symbol \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

Table 19: Operaties van de Infrabel/Main klasse

4.2 GUI

4.2.1 Startup

De `Startup` klasse is de grafische representatie van het opstart menu, waar onder andere de configuratie voor de TCP verbinding zal kunnen gebeuren.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Startup)$	Constructor

Table 20: Operaties van de Infrabel/GUI/Startup klasse

4.2.2 Admin-debug

De `Admin-debug` klasse is de grafische user interface deze module, de programmeurs van Infrabel zullen hier de volledige applicatie meer kunnen besturen.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow \textit{Admin-debug})$	Constructor
`update!`	$(\varnothing \rightarrow \varnothing)$	Update de GUI naar de nieuwe staat van het systeem

Table 21: Operaties van de Provider/GUI/Admin-debug klasse

4.3 Logic

4.3.1 App

De `App` klasse is verantwoordelijk voor de interne logica van de Infrabel module. Het besteet verschillende taken uit aan de klasses waarop deze steunt.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow \textit{App})$	Constructor
`get-x`	$(\textit{symbol} \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow \textit{symbol})$	Placeholder voor setters voor updaten van informatie

Table 22: Operaties van de Provider/Logic/App klasse

4.3.2 Startup

De `Startup` klasse is verantwoordelijk voor alle logica die gepaart gaat met het opstarten van de applicatie

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow \textit{Startup})$	Constructor
`get-x`	$(\textit{symbol} \rightarrow \varnothing)$	Placeholder voor getters voor uitwisseling van informatie
`set-x!`	$(\varnothing \rightarrow \textit{symbol})$	Placeholder voor setters voor updaten van informatie

Table 23: Operaties van de Provider/Logic/App klasse

4.3.3 TCP

De `TCP` klasse is verantwoordelijk voor de communicatie met de Provider module.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow \textit{TCP})$	Constructor
`send-message`	$(\textit{string} \rightarrow \varnothing)$	Verstuur een bericht naar de Provider module

Table 24: Operaties van de Provider/Logic/TCP klasse

4.3.4 UDP

De `UDP` klasse is een abstractie bovenop ofwel de Z21, ofwel de simulator. Aangezien het mogelijk moet blijven om op beiden te steunen, wordt de abstractie gemaakt.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow UDP)$	Constructor
`start`	$(\varnothing \rightarrow \varnothing)$	Start simulator / hardware.
`stop`	$(\varnothing \rightarrow \varnothing)$	Stopt simulatie / hardware.
`add-loco`	$(symbol\ symbol\ symbol\ \rightarrow \varnothing)$	Voegt locomotief toe.
`get-loco-speed`	$(symbol\ \rightarrow number)$	Geeft snelheid locomotief.
`set-loco-speed!`	$(symbol\ number\ \rightarrow \varnothing)$	Update snelheid locomotief.
`get-detection-block-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van detectieblokken.
`get-occupied-detection-blocks`	$(\varnothing \rightarrow pair)$	Geeft lijst bezette detectieblokken.
`get-switch-ids`	$(\varnothing \rightarrow pair)$	Geeft lijst van wissels.
`get-switch-position`	$(symbol\ \rightarrow number)$	Geeft positie van opgegeven wissel.
`set-switch-position!`	$(symbol\ number\ \rightarrow \varnothing)$	Update positie van opgegeven wissel.
`open-crossing!`	$(symbol\ \rightarrow \varnothing)$	Opent opgegeven overweg.
`close-crossing!`	$(symbol\ \rightarrow \varnothing)$	Sluit opgegeven overweg.
`set-sign-code!`	$(symbol\ symbol\ \rightarrow \varnothing)$	Updatet opgegeven licht.

Table 25: Operaties van de Provider/Logic/UDP klasse

4.3.5 Collision-detector

De `Collision-detector` klasse is verantwoordelijk voor het voorkomen van botsingen.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Collision\text{-}detector)$	Constructor
`calculate-collision?`	$(symbol\ symbol\ \rightarrow boolen)$	Returnt mogelijke botsingen

Table 26: Operaties van de Provider/Logic/Collision-detector klasse

4.3.6 Signalisator

De `Signalisator` klasse is verantwoordelijk dat alle signalisatie rondom de spoorweg correct is.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Signalisator)$	Constructor
`update!`	$(string\ \rightarrow \varnothing)$	Update klasse met huidige staat spoorweg

Table 27: Operaties van de Provider/Logic/Signalisator klasse

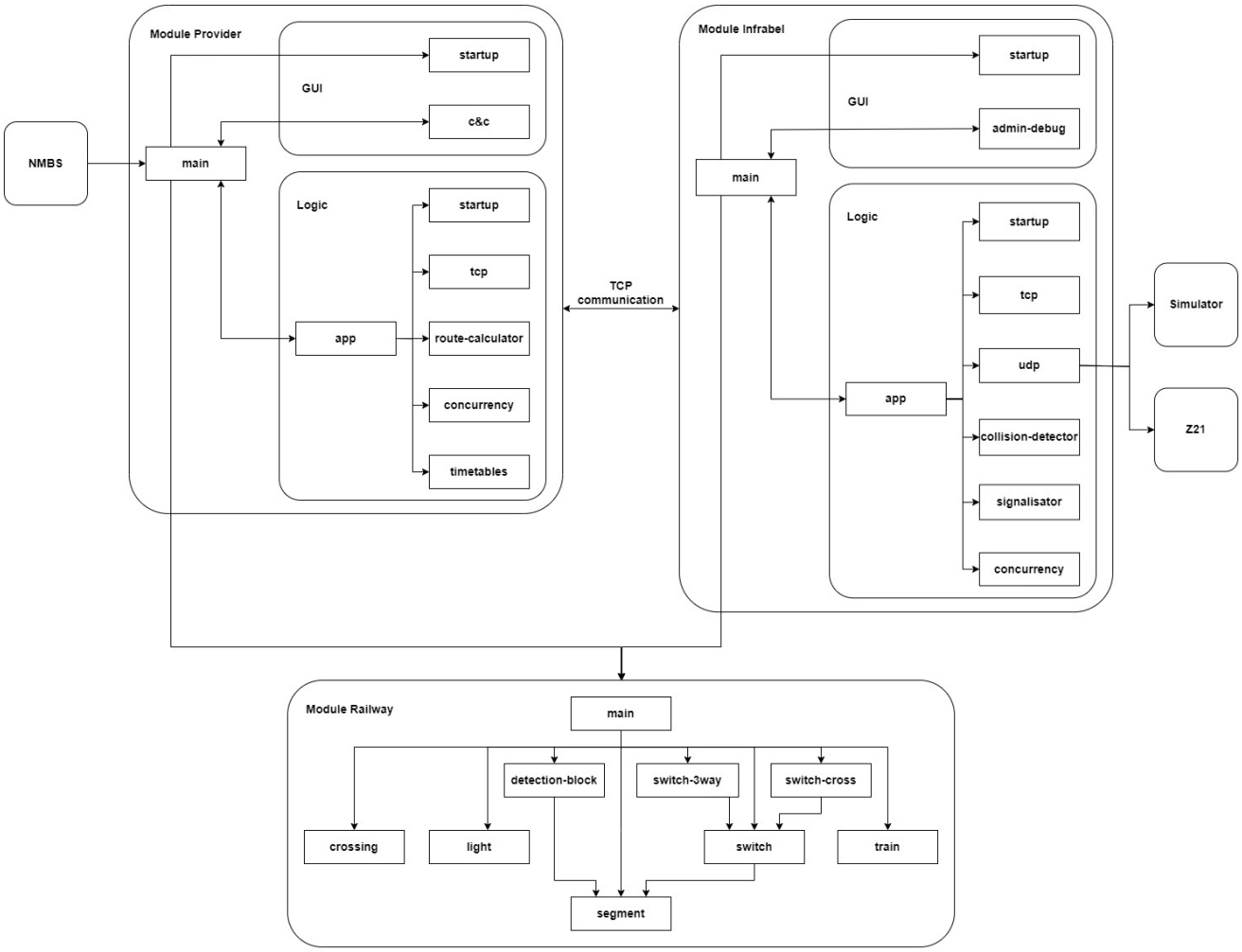
4.3.7 Concurrency

De `Concurrency` klasse is verantwoordelijk voor het opstaan en inladen van de software.

Naam	Signatuur	Uitleg
`make-object`	$(\varnothing \rightarrow Concurrency)$	Constructor
`save`	$(string\ \rightarrow \varnothing)$	Slaat huidige staat van de software op.
`load!`	$(string\ \rightarrow \varnothing)$	Laadt de staat van de software in van een bestand.

Table 28: Operaties van de Provider/Logic/Concurrency klasse

5 Afhankelijkheidsdiagram



6 Planning

Volgens het Agile-Scrum model, gezien zoals in Software Engineering uit de derde bachelor, ga ik het bouwen van het software systeem opsplitsen in een opeenvolging van 1 week durende sprints.

Week	**Actie**
week 5: 20/10	**Indienen voorstudie**
week 6	Sprint 1: Railway module bouwen + testen + documentatie
week 6: 25/10	*WPO: feedback voorstudie en voorbereiding fase 1*
week 7	Sprint 2: Infrabel logica bouwen + testen + documentatie
week 8	Sprint 3: Provider logica bouwen + testen + documentatie
week 9	Sprint 4: GUI provider bouwen + testen + documentatie
week 9: 12/11	*WPO: massaprogrammeren fase 1*
week 9: 17/11	EIGEN DEADLINE FASE 1
week 10	(week van Saint-V, voorbereidingen evenement)
week 11:	Troubleshooting + testing + documentatie
week 11: 01/12	**Deadline fase 1**
week 12	(week van Vrijzinnig Zangfeest, voorbereidingen evenement)
week 13:	Voorbereiding fase 2, inhalen eventuele tekorten
week 13: 13/12	*WPO: feedback fase 1 en voorbereiding fase 2*
week 14:	Voorbereiding examens
week 15-16	(Blok: wintervakantie)
week 17-20	*Examenperiode*
week 21	(lesvrije week)
week 22	Sprint 5: TCP splitsing Provider Infrabel+ testen + documentatie
week 23	Sprint 6: Implementatie botspreventie + testen + documentatie
week 23-24	*WPO: massaprogrammeren fase 2*
week 24	Sprint 7: Implementatie hardware + testen + documentatie
week 25	Sprint 8: Implementatie hardware + testen + documentatie
week 25: 09/03	EIGEN DEADLINE FASE 2
week 26	Troubleshooting + testing + documentatie
week 26: 16/03	**Deadline fase 2**
week 27-28	Voorbereiding fase 3, inhalen eventuele tekorten
week 28:	**Mondelinge verdediging fase 2**
week 29	Sprint 9: Automatische trajectberekening + testen + documentatie
week 30-31	(Blok: lentevakantie)
week 32	Sprint 10:
week 32-33:	*WPO: massaprogrammeren fase 3*
week 33	Sprint 11: Automatische trajectberekening + testen + documentatie
week 34	Sprint 12: Extra vereisten + testen + documentatie
week 35	Sprint 13: Extra vereisten + testen + documentatie
week 35: 18/05	EIGEN DEADLINE FASE 3
week 36	Troubleshooting + testing + documentatie
week 36: 25/05	**Deadline fase 3**
examenperiode	**Mondelinge verdediging fase 3**

Table 29: Planning Modeltrein Software Toepassing