

# Programmeerproject 2: Voorstudie Modeltrein Software Toepassing \*

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

2/3ba Computerwetenschappen

Academiejaar 2024-2025

---

\*<https://github.com/Jonas-Bruell/besturingssysteem-modeltreinen>

# Contents

<b>1</b>	<b>Introductie</b>	<b>1</b>
<b>2</b>	<b>Globaal Afhankelijkheidsdiagram</b>	<b>2</b>
<b>3</b>	<b>Railway</b>	<b>3</b>
3.1	Afhankelijkheidsdiagram . . . . .	3
3.2	Main . . . . .	3
3.3	Crossing . . . . .	3
3.4	Detection-block . . . . .	4
3.5	Light . . . . .	4
3.6	Segment . . . . .	4
3.7	Switch . . . . .	5
3.8	Switch-3way . . . . .	6
3.9	Switch-cross . . . . .	6
3.10	Train . . . . .	6
<b>4</b>	<b>Provider</b>	<b>7</b>
4.1	Afhankelijkheidsdiagram . . . . .	7
4.2	Main . . . . .	7
4.3	GUI . . . . .	7
4.3.1	Startup . . . . .	7
4.3.2	Command&Control . . . . .	7
4.4	Logic . . . . .	8
4.4.1	App . . . . .	8
4.4.2	Startup . . . . .	8
4.4.3	TCP . . . . .	8
4.4.4	Route-calculator . . . . .	8
4.4.5	Concurrency . . . . .	9
4.4.6	Timetables . . . . .	9
<b>5</b>	<b>Infrabel</b>	<b>9</b>
5.1	Afhankelijkheidsdiagram . . . . .	9
5.2	Main . . . . .	9
5.3	GUI . . . . .	9
5.3.1	Startup . . . . .	9
5.3.2	Admin-debug . . . . .	11
5.4	Logic . . . . .	11
5.4.1	App . . . . .	11
5.4.2	Startup . . . . .	11
5.4.3	TCP . . . . .	11
5.4.4	UDP . . . . .	12
5.4.5	Collision-detector . . . . .	12
5.4.6	Signalisator . . . . .	12
5.4.7	Concurrency . . . . .	12

# 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 Globaal Afhankelijkheidsdiagram

### 3 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.

#### 3.1 Afhankelijkheidsdiagram

#### 3.2 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	$(\emptyset \rightarrow pair)$	Geeft lijst van overwegen
get-crossing-state	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven overweg
set-crossing-state!	$(symbol symbol \rightarrow \emptyset)$	Update gegeven overweg naar gegeven status
get-detection-block-ids	$(\emptyset \rightarrow pair)$	Geeft lijst van detectie blokken
get-detection-block-state	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven detectieblok
reserve-detection-block!	$(symbol \rightarrow \emptyset)$	Reserveert gegeven detectieblok
get-light-ids	$(\emptyset \rightarrow pair)$	Geeft lijst van lichten
get-light-signal	$(symbol \rightarrow symbol)$	Geeft signaal weer van gegeven licht
set-light-signal!	$(symbol symbol \rightarrow \emptyset)$	Update gegeven licht met gegeven signaal
get-segment-ids	$(\emptyset \rightarrow pair)$	Geeft lijst van segmenten
get-segment-state	$(symbol \rightarrow symbol)$	Geeft status weer van gegeven segment
reserve-segment!	$(symbol \rightarrow \emptyset)$	Reserveert gegeven spoorwegsegment
get-switch-ids	$(\emptyset \rightarrow pair)$	Geeft lijst van wissels
get-switch-position	$(symbol \rightarrow symbol)$	Geeft positie weer van gegeven wissel
set-switch-position!	$(symbol symbol \rightarrow \emptyset)$	Update gegeven wissel met gegeven positie
reserve-switch!	$(symbol \rightarrow \emptyset)$	Reserveert gegeven wissel
get-train-ids	$(\emptyset \rightarrow pair)$	Geeft lijst van treinen
get-train-speed	$(symbol \rightarrow symbol)$	Geeft snelheid weer van gegeven trein
set-train-speed!	$(symbol symbol \rightarrow \emptyset)$	Update gegeven trein met gegeven snelheid

Table 1: Operaties van de Railway/Main klasse

#### 3.3 Crossing

De Crossing klasse is de atomaire abstractie van een spoorwegoverweg. De constructor neemt als argumenten het id van de overweg, een ‘connection’ (communicatie naar ondergelegen laag) en een lijst met spoorwegsegmenten waarover deze overweg brugt. Het openen en sluiten van deze overweg neemt wat tijd in beslag, waartussen de overweg als status ‘pending’ heeft.

Naam	Signatuur	Uitleg
<code>make-object</code>	$(symbol\ Object\ list \rightarrow Crossing)$	Constructor
<code>get-id</code>	$(\emptyset \rightarrow symbol)$	Geeft het <b>id</b> van de overweg.
<code>get-state</code>	$(\emptyset \rightarrow symbol)$	Geeft de <b>status</b> van de overweg.
<code>get-segments</code>	$(\emptyset \rightarrow list)$	Geeft de segmenten waarover de overweg brugt.
<code>set-state!</code>	$(symbol \rightarrow \emptyset)$	Open of sluit de overweg adhv een status.

Table 2: Operaties van de Railway/Crossing klasse

### 3.4 Detection-block

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

Naam	Signatuur	Uitleg
<code>make-object</code>	$(symbol \rightarrow Detection-block)$	Constructor
<code>get-state</code>	$(\emptyset \rightarrow symbol)$	Geeft de status van het detection block
<code>set-state!</code>	$(symbol \rightarrow \emptyset)$	Update de status van het detection block

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

### 3.5 Light

De **Light** klasse is de atomaire abstractie van een licht. De constructor neemt als argumenten het **id** van het licht, een ‘**connection**’ (communicatie naar ondergelegen laag) en een spoorwegsegment waarover dit licht gaat. Het licht kan de volgende statussen hebben: **Hp0**, **Hp1**, **Hp0+Sh0**, **Ks1+Zs3**, **Ks2**, **Ks2+Zs3**, **Sh1**, **Ks1+Zs3+Zs3v**.

Naam	Signatuur	Uitleg
<code>make-object</code>	$(symbol\ Object\ symbol \rightarrow Light)$	Constructor
<code>get-id</code>	$(\emptyset \rightarrow symbol)$	Geeft het <b>id</b> van het licht.
<code>get-signal</code>	$(\emptyset \rightarrow symbol)$	Geeft het <b>signal</b> van het licht.
<code>get-segment</code>	$(\emptyset \rightarrow symbol)$	Geeft het segment waarover het licht gaat.
<code>set-signal!</code>	$(symbol \rightarrow \emptyset)$	Update het signaal van het licht.

Table 4: Operaties van de Railway/Light klasse

### 3.6 Segment

De **Segment** klasse is de atomaire abstractie van een spoorwegsegment. De constructor neemt als argumenten het **id** van het spoorwegsegment, een ‘**connection**’ (communicatie naar ondergelegen laag) en de twee andere spoorwegelementen waartussen dit spoorwegsegment zich bevindt. De status van het spoorwegsegment kan ofwel **free**, ofwel **reserved** zijn.

Naam	Signatuur	Uitleg
<code>make-object</code>	$(symbol\ Object\ symbol\ symbol \rightarrow Segment)$	Constructor
<code>get-id</code>	$(\emptyset \rightarrow symbol)$	Geeft het <b>id</b> van de overweg.
<code>get-state</code>	$(\emptyset \rightarrow symbol)$	Geeft de <b>status</b> van de overweg.
<code>get-next</code>	$(\emptyset \rightarrow list)$	Geeft het volgende spoorwegelement weer.
<code>get-prev</code>	$(\emptyset \rightarrow list)$	Geeft het vorige spoorwegelement weer.
<code>set-state!</code>	$(symbol \rightarrow boolean)$	Update de <b>status</b> van het segment. False wanneer ( <code>set-state!</code> <code>'reserved</code> ) indien <b>Segment</b> reeds gereserveerd.

Table 5: Operaties van de Railway/Segment klasse

### 3.7 Switch

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

Naam	Signatuur	Uitleg
<code>make-object</code>	$(symbol \rightarrow Switch)$	Constructor
<code>get-position</code>	$(\emptyset \rightarrow symbol)$	Geeft de status van de wissel
<code>set-position!</code>	$(symbol \rightarrow \emptyset)$	Update de status van de wissel

Table 6: Operaties van de Railway/Switch klasse

### 3.8 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
<b>make-object</b>	$(symbol \rightarrow Switch-3way)$	Constructor
<b>get-position</b>	$(\emptyset \rightarrow symbol)$	Geeft de status van de wissel
<b>set-position!</b>	$(symbol \rightarrow \emptyset)$	Update de status van de wissel

Table 7: Operaties van de Railway/Switch klasse

### 3.9 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
<b>make-object</b>	$(symbol \rightarrow Switch-cross)$	Constructor
<b>get-position</b>	$(\emptyset \rightarrow symbol)$	Geeft de status van de wissel
<b>set-position-1!</b>	$(symbol \rightarrow \emptyset)$	Update de status van het eerste paar in de wissel
<b>set-position-2!</b>	$(symbol \rightarrow \emptyset)$	Update de status van het tweede paar in de wissel

Table 8: Operaties van de Railway/Switch klasse

### 3.10 Train

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

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

Table 9: Operaties van de Railway/Train klasse



## 4 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.

### 4.1 Afhankelijkheidsdiagram

### 4.2 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
<code>make-object</code>	$(\emptyset \rightarrow Provider)$	Constructor
<code>get-x</code>	$(symbol \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<code>set-x!</code>	$(\emptyset \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

Table 10: Operaties van de Provider/Main klasse

### 4.3 GUI

#### 4.3.1 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
<code>make-object</code>	$(\emptyset \rightarrow Startup)$	Constructor

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

#### 4.3.2 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
<code>make-object</code>	$(\emptyset \rightarrow C\&C)$	Constructor
<code>update!</code>	$(\emptyset \rightarrow \emptyset)$	Update de GUI naar de nieuwe staat van het systeem

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

## 4.4 Logic

### 4.4.1 App

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow App)$	Constructor
<b>get-x</b>	$(symbol \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<b>set-x!</b>	$(\emptyset \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

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

### 4.4.2 Startup

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow Startup)$	Constructor
<b>get-x</b>	$(symbol \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<b>set-x!</b>	$(\emptyset \rightarrow symbol)$	Placeholder voor setters voor updaten van informatie

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

### 4.4.3 TCP

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow TCP)$	Constructor
<b>send-message</b>	$(string \rightarrow \emptyset)$	Verstuur een bericht naar de Infrabel module

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

### 4.4.4 Route-calculator

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow Route-calculator)$	Constructor
<b>calculate-shortest-between</b>	$(symbol\ symbol \rightarrow pair)$	Berekent kortste route

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

#### 4.4.5 Concurrency

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

Naam	Signatuur	Uitleg
<code>make-object</code>	$(\emptyset \rightarrow \textit{Concurrency})$	Constructor
<code>save</code>	$(\textit{string} \rightarrow \emptyset)$	Slaat huidige staat van de software op.
<code>load!</code>	$(\textit{string} \rightarrow \emptyset)$	Laadt de staat van de software in van een bestand.

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

#### 4.4.6 Timetables

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

Naam	Signatuur	Uitleg
<code>make-object</code>	$(\emptyset \rightarrow \textit{Timetables})$	Constructor
<code>generate-timetable</code>	$(\textit{symbol} \rightarrow \textit{pair})$	Genereert timetables voor trein

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

### 5 Infrabel

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

#### 5.1 Afhankelijkheidsdiagram

#### 5.2 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
<code>make-object</code>	$(\emptyset \rightarrow \textit{Main})$	Constructor
<code>get-x</code>	$(\textit{symbol} \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<code>set-x!</code>	$(\emptyset \rightarrow \textit{symbol})$	Placeholder voor setters voor updaten van informatie

Table 19: Operaties van de Infrabel/Main klasse

#### 5.3 GUI

##### 5.3.1 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
<code>make-object</code>	$(\emptyset \rightarrow Startup)$	Constructor

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

### 5.3.2 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
<b>make-object</b>	$(\emptyset \rightarrow \text{Admin-debug})$	Constructor
<b>update!</b>	$(\emptyset \rightarrow \emptyset)$	Update de GUI naar de nieuwe staat van het systeem

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

## 5.4 Logic

### 5.4.1 App

De **App** klasse is verantwoordelijk voor de interne logica van de Infrabel module. Het bestaat verschillende taken uit aan de klassen waarop deze steunt.

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow \text{App})$	Constructor
<b>get-x</b>	$(\text{symbol} \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<b>set-x!</b>	$(\emptyset \rightarrow \text{symbol})$	Placeholder voor setters voor updaten van informatie

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

### 5.4.2 Startup

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow \text{Startup})$	Constructor
<b>get-x</b>	$(\text{symbol} \rightarrow \emptyset)$	Placeholder voor getters voor uitwisseling van informatie
<b>set-x!</b>	$(\emptyset \rightarrow \text{symbol})$	Placeholder voor setters voor updaten van informatie

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

### 5.4.3 TCP

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

Naam	Signatuur	Uitleg
<b>make-object</b>	$(\emptyset \rightarrow \text{TCP})$	Constructor
<b>send-message</b>	$(\text{string} \rightarrow \emptyset)$	Verstuur een bericht naar de Provider module

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

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

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

#### 5.4.5 Collision-detector

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

Naam	Signatuur	Uitleg
make-object	$(\emptyset \rightarrow Collision-detector)$	Constructor
calculate-collision?	$(symbol\ symbol \rightarrow boolean)$	Returnt mogelijke botsingen

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

#### 5.4.6 Signalisator

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

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

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

#### 5.4.7 Concurrency

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

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

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