# Programmeerproject 2: Besturingssysteem Modeltreinen
# Specificatie voor Ontwikkelaars *

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

Academiejaar 2024-2025
Vrije Universiteit Brussel

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

# 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 Globaal afhankelijkheidsdiagram

Het besturingssysteem bestaat uit twee grote componenten: "infrabel" en "provider".
De "infrabel" component wordt via INFRABEL tot een object gemaakt en is de centrale server deze de modelspoorlijn aanstuurt. De "provider" component is een template die via "NMBS", "DB", ... verschillende objecten wordt gemaakt, en deze de clients vormen, deze via een TCP verbinding met INFRABEL communiceren.
Zowel "infrabel" als "provider" hebben een gemeenschappelijke abstractielaag, "railway", deze zorgt voor consistentie tussen beide componenten, zonder aan elkaar verbonden te zijn. Beide componenten hebben hun eigen, afzonderlijke "railway".

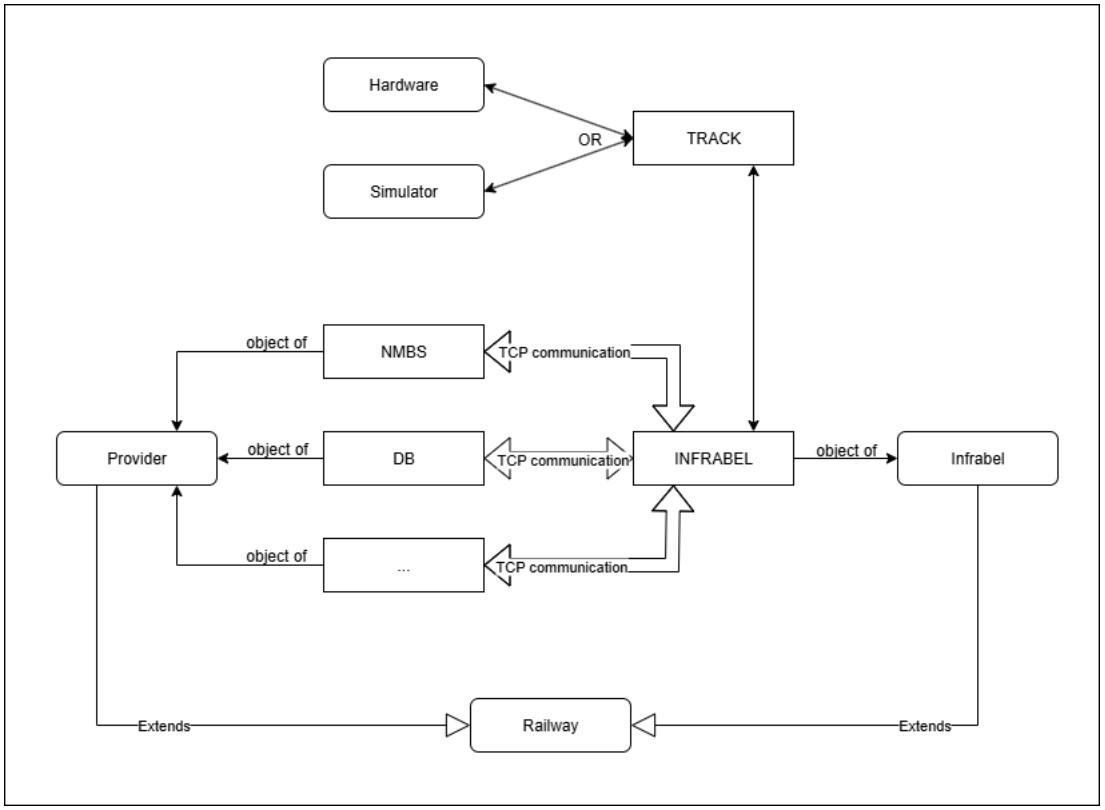



Figure 1: Globaal, versimpeld afhankelijkheidsdiagram

# 3 Track

De track-module is de abstractielaag over de interfaces van de simulator en de hardware. Deze module bevat de abstracties voor de verschillende hardware-componenten die de modelspoorweg bezit. Deze abstractielaag maakt dit project onafhankelijk van de externe libraries.

## 3.1 afhankelijkheidsdiagram

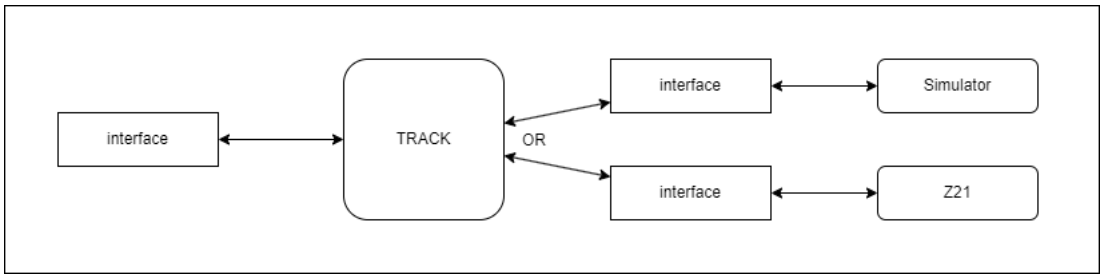



Figure 2: Afhankelijkheidsdiagram van de track-module

## 3.2 interface.rkt

De `Track` klasse is een interface die kan worden opgeroepen vanuit andere modules van het besturingssysteem. Deze klasse bevat alle nodige operaties om de hardware-componenten van de spoorweg aan te sturen.

| **Naam + signatuur** | **Uitleg** |
|---|---|
| `make-object` : $(symbol \cup \varnothing,\ symbol \cup \varnothing \rightarrow Track)$ | Constructor |
| `get-architecture` : $(\varnothing \rightarrow symbol)$ | Geeft de architectuur van de track weer. |
| `get-version` : $(\varnothing \rightarrow symbol)$ | Geeft de versie van de track weer. |
| `get-versions` : $(\varnothing \rightarrow pair)$ | Geeft verschillende versies van track weer. |
| `config!` : $(symbol,\ symbol\ \rightarrow \varnothing)$ | Configureert track met architecture & versie. |
| `start` : $(\varnothing \rightarrow \varnothing)$ | Start de Track module. |
| `stop` : $(\varnothing \rightarrow \varnothing)$ | Start de Track module. |
| `add-loco` : $(symbol,\ symbol,\ symbol\ \rightarrow \varnothing)$ | Plaatst een locomotief op de Track. |
| `get-loco-speed` : $(symbol\ \rightarrow number)$ | Geeft de snelheid van de gegeven locomotief. |
| `set-loco-speed!` : $(symbol,\ number\ \rightarrow \varnothing)$ | Zet de snelheid naar de opgegeven snelheid. |
| `get-detection-block-ids` : $(\varnothing \rightarrow pair)$ | Geeft lijst van detection-block ids terug. |
| `get-occupied-detection-blocks` : $(\varnothing \rightarrow pair)$ | Geeft lijst van bezette detection-blocks. |
| `get-switch-ids` : $(\varnothing \rightarrow pair)$ | Geeft lijst van wissels terug. |
| `get-switch-position` : $(symbol\ \rightarrow number)$ | Geeft positie van gegeven wissel. |
| `set-switch-position!` : $(symbol,\ number\ \rightarrow \varnothing)$ | Zet positie van opgegeven wissel. |
| `set-crossing-position!` : $(symbol,\ symbol\ \rightarrow \varnothing)$ | Zet de positie van de opgegeven overweg. |
| `set-sign-code!` : $(symbol,\ symbol\ \rightarrow \varnothing)$ | Zet het signaal van het opgegeven licht. |

Table 1: Operaties van de Track klasse

# 4 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 railway-klasse.

De railway-module is niet afhankelijk van de track-module, maar abstraheert deze door zijn oproepen te doen naar een "connection". Deze connection kan de track-interface zijn, maar bijvoorbeeld ook een client voor een provider, om op deze manier messages naar infrabel te sturen.

## 4.1 afhankelijkheidsdiagram

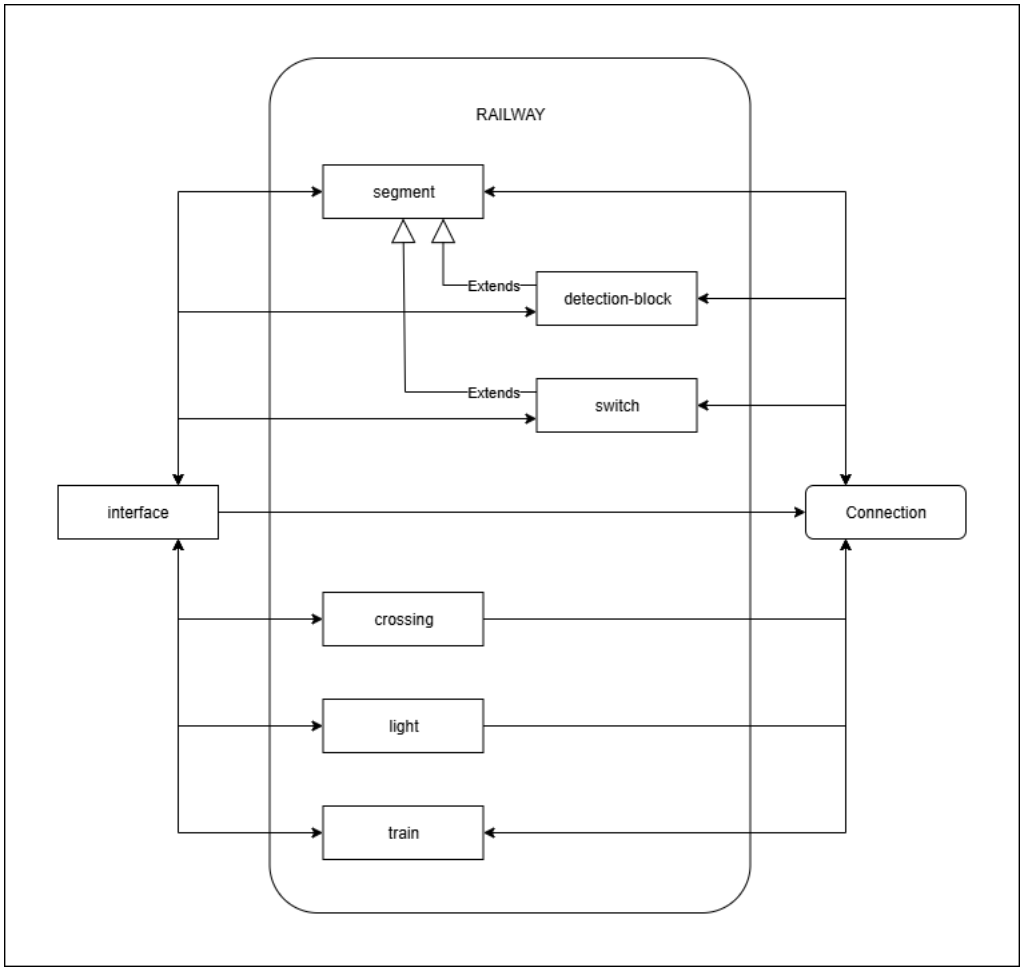



Figure 3: Afhankelijkheidsdiagram van de railway-module

## 4.2 interface.rkt

De `Railway` 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 interface hiervoor ook de nodige getters en setters.

| **Naam + signatuur** | **Uitleg** |
|---|---|
| `make-object` : (*Connection* $\rightarrow$ *Railway*) | Constructor |
| `start-control-panel` : ($\varnothing \rightarrow \varnothing$) | Start het controlepaneel van de railway. |
| `get-segment-ids` : ($\varnothing \rightarrow$ *pair*) | Geeft lijst van segmenten. |
| `get-segment-next` : (*symbol* $\rightarrow$ *symbol*) | Geeft volgende (wijzerzin) segment terug. |
| `get-segment-prev` : (*symbol* $\rightarrow$ *symbol*) | Geeft vorige (wijzerzin) segement terug. |
| `get-segment-state` : (*symbol* $\rightarrow$ *symbol*) | Geeft status weer van gegeven segment. |
| `set-segment-state!` : (*symbol*, *symbol* $\rightarrow \varnothing$) | Zet status van het opgegeven segment. |
| `get-detection-block-ids` : ($\varnothing \rightarrow$ *pair*) | Geeft lijst van detectieblokken. |
| `get-detection-block-next` : (*symbol* $\rightarrow$ *symbol*) | Geeft volgende (wijzerzin) detectieblok. |
| `get-detection-block-prev` : (*symbol* $\rightarrow$ *symbol*) | Geeft vorige (wijzerzin) detectieblok. |
| `get-detection-block-state` : (*symbol* $\rightarrow$ *symbol*) | Geeft status weer van gegeven detectieblok. |
| `set-detection-block-state!` : (*symbol* $\rightarrow \varnothing$) | Zet status van gegeven detectieblok. |
| `get-switch-ids` : ($\varnothing \rightarrow$ *pair*) | Geeft lijst van wissels terug. |
| `get-switch-next` : (*symbol* $\rightarrow$ *pair*) | Geeft lijst volgende (2-segmenten kant) segmenten. |
| `get-switch-next-left` : (*symbol* $\rightarrow$ *symbol*) | Geeft linker volgende (2-segmenten kant) segment. |
| `get-switch-next-right` : (*symbol* $\rightarrow$ *symbol*) | Geeft rechter volgende (2-segmenten kant) segment. |
| `get-switch-prev` : (*symbol* $\rightarrow$ *symbol*) | Geeft vorige (1-segment kant) segment. |
| `get-switch-state` : *symbol* $\rightarrow$ *symbol*) | Geeft status weer van gegeven wissel. |
| `set-switch-state!` : (*symbol*, *symbol* $\rightarrow \varnothing$) | Zet status van het opgegeven wissel. |
| `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. |
| `get-crossing-ids` : ($\varnothing \rightarrow$ *pair*) | Geeft lijst van overwegen. |
| `get-crossing-segments`: (*symbol* $\rightarrow$ *pair*) | Geeft lijst met segmenten tussen de overweg. |
| `get-crossing-position` : (*symbol* $\rightarrow$ *symbol*) | Geeft positie weer van slagbomen gegeven overweg. |
| `set-crossing-position!` : (*symbol symbol* $\rightarrow \varnothing$) | Update gegeven overweg naar gegeven positie |
| `get-light-ids` : ($\varnothing \rightarrow$ *pair*) | Geeft lijst van lichten |
| `get-light-segment` : (*symbol* $\rightarrow$ *symbol*) | Geeft segment terug waaraan licht grenst. |
| `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-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 2: Operaties van de Railway klasse

### 4.3 crossing.rkt

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 van 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** |
|---|---|
| `make-object` : (*symbol Object list* $\rightarrow$ *Crossing*) | Constructor |
| `get-id` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `id` van de overweg. |
| `get-state` : ($\varnothing \rightarrow$ *symbol*) | Geeft de `status` van de overweg. |
| `get-segments` : ($\varnothing \rightarrow$ *list*) | Geeft de segmenten waarover de overweg brugt. |
| `set-state!` : (*symbol* $\rightarrow \varnothing$) | Open of sluit de overweg adhv een status. |

Table 3: Operaties van de Railway/Crossing klasse

### 4.4 detection-block.rkt

De `Detection-block` klasse is de atomaire abstractie van een detectieblok. Deze is een uitbreiding van de `Segment` klasse. De constructor neemt als argumenten het `id` van het detectieblok, een '`connection`' (communicatie naar ondergelegen laag) en de twee andere spoorwegelementen waartussen dit detectieblok zich bevindt. De status van het detectieblok kan ofwel `free`, ofwel `reserved`, ofwel `occupied` zijn.

| **Naam + signatuur** | **Uitleg** |
|---|---|
| `make-object` (*symbol Object symbol symbol* $\rightarrow$ *Detection-block*) | Constructor |
| `get-id` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `id` van het detectieblok. |
| `get-state` : ($\varnothing \rightarrow$ *symbol*) | Geeft de `status` van het detectieblok. |
| `get-next` : ($\varnothing \rightarrow$ *symbol*) | Geeft het volgende spoorwegelement weer. |
| `get-prev` : ($\varnothing \rightarrow$ *symbol*) | Geeft het vorige spoorwegelement weer. |
| `set-state!` : (*symbol* $\rightarrow$ *boolean*) | Update de `status` van het detectieblok.<br>• False wanneer `(set-state! 'reserved)` indien `Detection-block` gereserveerd.<br>• False wanneer `(set-state! 'reserved)` indien `Detection-block` occupied.<br>• False wanneer `(set-state! 'occupied)` indien `Detection-block` occupied. |

Table 4: Operaties van de Railway/Detection-block klasse

### 4.5 light.rkt

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** |
|---|---|
| `make-object` : (*symbol Object symbol* $\rightarrow$ *Light*) | Constructor |
| `get-id` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `id` van het licht. |
| `get-signal` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `signal` van het licht. |
| `get-segment` : ($\varnothing \rightarrow$ *symbol*) | Geeft het segment waarover het licht gaat. |
| `set-signal!` : (*symbol* $\rightarrow \varnothing$) | Update het signaal van het licht. |

Table 5: Operaties van de Railway/Light klasse

### 4.6 segment.rkt

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 |
|---|---|
| `make-object` : (*symbol Object symbol symbol* $\rightarrow$ *Segment*) | Constructor |
| `get-id` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `id` van het segment. |
| `get-state` : ($\varnothing \rightarrow$ *symbol*) | Geeft de `status` van het segment. |
| `get-next` : ($\varnothing \rightarrow$ *symbol*) | Geeft het volgende spoorwegelement weer. |
| `get-prev` : ($\varnothing \rightarrow$ *symbol*) | Geeft het vorige spoorwegelement weer. |
| `set-state!` : (*symbol* $\rightarrow$ *boolean*) | Update de `status` van het segment. False wanneer `(set-state! 'reserved)` indien `Segment` reeds gereserveerd. |

Table 6: Operaties van de Railway/Segment klasse

### 4.7 switch.rkt

De `Switch` klasse is de atomaire abstractie van een spoorwegwissel. Deze is een uitbreiding van de `Segment` klasse. De constructor neemt als argumenten het `id` van de wissel, een '`connection`' (communicatie naar ondergelegen laag) en de andere spoorwegelementen waartussen deze wissel zich bevindt, de ingaande als `symbol` en de uitgaande als een `cons`-cel van `symbol`s. De positie van de wissel kan ofwel `left`, ofwel `right`, zijn.

| Naam+ signatuur | Uitleg |
|---|---|
| `make-object` : (*symbol Object symbol pair* $\rightarrow$ *Switch*) | Constructor |
| `get-id` : ($\varnothing \rightarrow$ *symbol*) | Geeft het `id` van de wissel. |
| `get-state` : ($\varnothing \rightarrow$ *symbol*) | Geeft de `status` van de wissel. |
| `get-position` : ($\varnothing \rightarrow$ *symbol*) | Geeft de `position` van de wissel |
| `get-next-left` : ($\varnothing \rightarrow$ *symbol*) | Geeft het volgende spoorwegelement weer. |
| `get-next-right` : ($\varnothing \rightarrow$ *symbol*) | Geeft het volgende spoorwegelement weer. |
| `get-prev` : ($\varnothing \rightarrow$ *symbol*) | Geeft het vorige spoorwegelement weer. |
| `set-state!` : (*symbol* $\rightarrow \varnothing$) | Update de `status` van de wissel. |
| `set-position!` : (*symbol* $\rightarrow \varnothing$) | Update de status van de wissel |

Table 7: Operaties van de Railway/Switch klasse

### 4.8 train.rkt

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

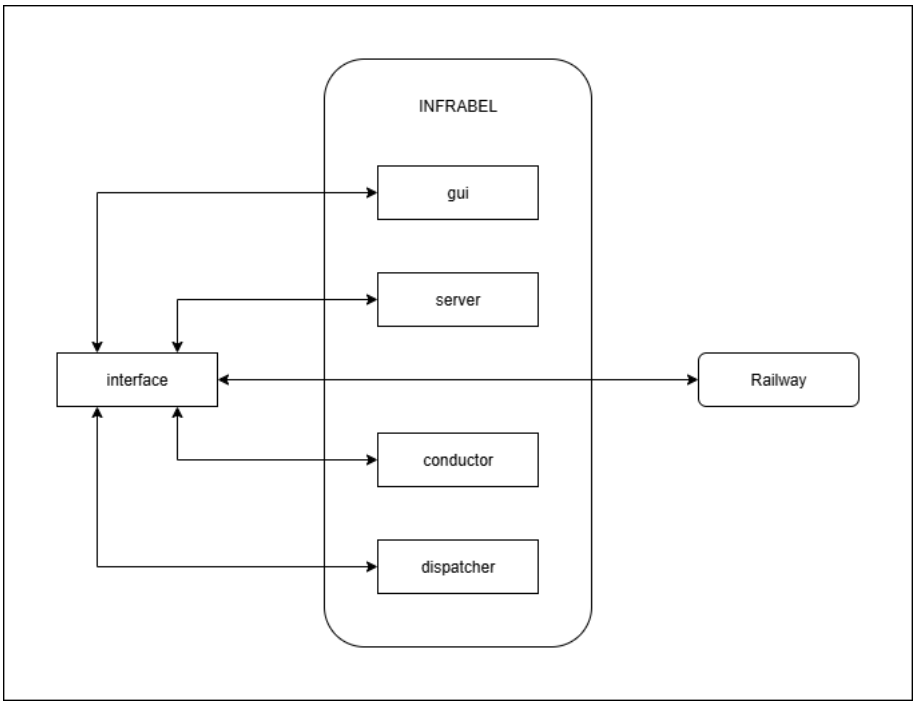
| Naam + signatuur | Uitleg |
|---|---|
| `make-object` : ($\varnothing \rightarrow$ *Train*) | Constructor |
| `get-location` : ($\varnothing \rightarrow$ *symbol*) | Geeft terug op welk segment/detectie-blok/wissel de trein is |
| `get-next-location` : ($\varnothing \rightarrow$ *symbol*) | Geeft volgende 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 8: Operaties van de Railway/Train klasse

# 5 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.

## 5.1 afhankelijkheidsdiagram




## 5.2 interface.rkt

De `Infrabel` klasse is het startpunt van de module en tevens ook de interface naar andere modules toe. Deze interface wordt opgeroepen vanuit een afzonderlijke file `INFRABEL.rkt`, deze voor de eindgebruiker gemakkelijk terug te vinden is.

| Naam | Signatuur | Uitleg |
|---|---|---|
| `make-object` | $(\varnothing \rightarrow \mathit{Infrabel})$ | 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 9: Operaties van de Infrabel/Main klasse

### 5.2.1 startup.rkt

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

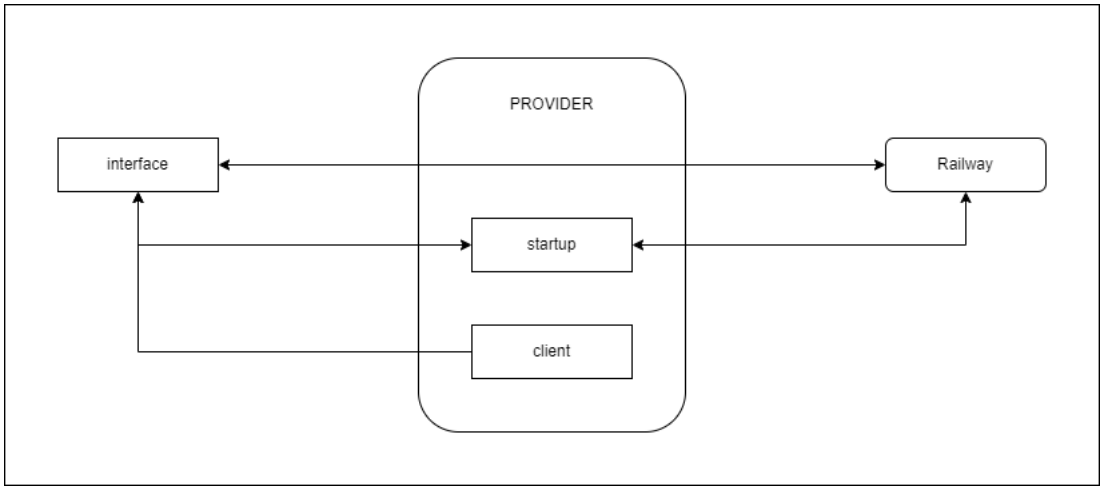
| Naam | Signatuur | Uitleg |
|---|---|---|
| `make-object` | $(\varnothing \rightarrow \mathit{Startup})$ | Constructor |

Table 10: Operaties van de Infrabel/GUI/Startup klasse

# 6 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.

## 6.1 afhankelijkheidsdiagram




## 6.2 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 |
|---|---|---|
| `make-object` | $(\varnothing \rightarrow Provider)$ | 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 11: Operaties van de Provider/Main klasse

### 6.2.1 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 12: Operaties van de Provider/GUI/Startup klasse

# 7 Verdere uitbreidingen en todo's

## 7.1 TCP communicatie tussen Infrabel en Providers

De communicatie tussen Infrabel en Providers staat nog niet op punt. In de huidige opstelling spreekt de Railway van Providers rechtstreeks met een verkeerde insteek met Infrabel, aangezien een TCP verbinding geen "return waarde" heeft, maar de methodes uit Railway deze communiceren met de Infrabel server deze wel verwachten. Hierdoor moet er oftewel een betere interface gemaakt worden hiertussen, deze dit kan opvangen, of moet Railway gerefactored worden.

Momenteel is de Railway module een heel uitgebreide module, die soms voor meer omwegen binnen Provider zorgt, dan hij nuttig is. Het zou kunnen zijn dat er te veel verantwoordelijkheden in deze geleelde module geraakt zijn en dat ik hier ofwel met thin interfaces moet werken, of stukken van deze verantwoordelijkheid wegnemen uit Railway.

## 7.2 Automatische Route Calculatie

De automatische route calculatie is een onderdeel dat thuis hoort in de Providers logica, deze is nog niet geïmplementeerd. (problemen met communicatie tussen Providers en Infrabel)

## 7.3 Testing van Infrabel en Provider componenten

De Railway component wordt grondig getest, maar de Infrabel en Provider modules worden nog niet voldoende getest.

## 7.4 Bijzondere wissels

De 3-wegs wissel en kruis-wissel zijn voorzien en bestaan al grotendeels, maar zijn niet volledig.

## 7.5 Betere GUI & gestandaardiseerde GUI elementen

De huidige GUI is nogal "rough around the edges", zoals de Engelsen dat mooi kunnen uitdrukken. Deze heeft nog veel werk nodig om "mooi" te zijn. Een eerste stap hier zou zijn om een grote refactoring te doen van de verschillende stukken en elementen van de GUI en deze op te splitsen in "GUI elements" die hergebruikt kunnen worden.

## 7.6 Infrabel op Raspberry Pi

Het project is zodanig gebouwd dat het ook op een Raspberry Pi kan draaien. Hier heb ik vorig jaar het nodige werk voor gedaan en dit heeft toen ook gewerkt. Tijdens het schrijven van het project dit jaar had ik echter geen toegang meer tot een werkende Raspberry Pi, waardoor ik niet kan testen of dit, na vele refactors, nog steeds werkt.

## 7.7 Meerdere Providers

Het project is zodanig gebouwd dat er meerdere Providers kunnen aansluiten. Echter ben ik hier niet volledig mee klaar geraakt en werkt dit niet volledig.