VRIJE
UNIVERSITEIT
BRUSSEL

PROGRAMMEERPROJECT 2

Documentatie fase 3

Gérard Lichtert
gerard.Lichtert@vub.be
0557513

14 mei 2021

Sciences and Bio-Engineering Sciences

Inhoudsopgave

1 Inleiding

Dit document bevat het verslag van fase 3 van het vak "Programmeerproject 2". Het behandelt de eerste functionele vereisten en de 3 uigebreide vereisten. Vervolgens zal het over de gebruikte datastructuren gaan en het afhankelijkheids diagram. Deze datastructuren zullen er voor zorgen dat alle functionaliteiten werken en vanuit de GUI aangeroepen kunnen worden. Verder staat in dit verslag een beschrijving van de API tussen de infrabel- en NMBS-component, de planning en het logboek.

2 Functionele vereisten

In dit verslag worden de functionele vereisten per fase besproken. In de eerste fase werd de GUI gemaakt alsook de command and control. De code van de command en control moeten we zelf niet schrijven maar de code die de hardware aanstuurt wel. De code moet locomotieven kunnen laten starten en stoppen, hun snelheid en rijrichting aflezen en veranderen. Verder moeten we ook de de stand van de wissels kunnen uitlezen en verzetten. Via de detectieblokkken detecteren we waar een trein zich bevindt.
De GUI laat de eindgebruiker de toestand van het spoornetwerk en de locomotieven zien. Het laat ook interactie toe met de wissels en locomotieven.
In de tweede fase was het de bedoeling dat we de eerste twee uitgebreide vereisten implementeren. Voor dit project werd gekozen voor botsingpreventie aan de hand van een reservatie en bezettingsysteem. Dit zal ervoor zorgen dat een trein zijn snelheid op nul gezet wordt als het pad dat die gaat afleggen al gereserveerd of bezet is. Aangezien de stand van de wissels op eender welk moment gewijzigd kan worden, worden er veel spoorsegmenten gereserveerd tot de aanliggende detectieblokken. Dit is wel met de voorwaarde dat ze op het pad zijn van de richting van de trein. Een andere uitgebreide vereiste waar voor gekozen werd is het automatisch trajectbeheer. De manier waarop dit geimplementeerd is dat de eindbestemming ingevoerd wordt en het pad berekend word. Dit wordt slechts gedaan als de trein detecteerbaar is. Anders zal het pad nog eens herberekend worden wanneer die detecteerbaar is. Er wordt bij het trajectbeheer ook rekening gehouden met het reservatiesysteem om botsingen te voorkomen.
In de 3e fase is het de bedoeling dat we de laatste uitgebreide vereiste implementeren. Bij dit project is dit de Raspberry Pi vereiste. Dit houdt in dat het mogelijk moet zijn om infrabel te runnen op de Raspberry Pi en NMBS op de computer. Dit wordt behaald door het externe IP adres (of lokaal) mee te geven bij het aanmaken van het NMBS object.

3 ADT's

3.1 Track ADT

Het track ADT houdt bij welke spoorsegmenten verbonden zijn. Het houdt ook bij of het spoorsegment al dan niet gereserveerd is. De reservatiestand kan ook aangepast worden.

Naam	Signatuur	Beschrijving
new	(symbol, list → track%)	Maakt een track object aan. Verwacht de ID van het spoorsegment en een lijst met de ID's van de verbonden spoorsegmenten.
get-track-id	(/ → symbol)	Geeft de ID van het spoorsegment terug.
track-links	(/ → vector)	Geeft de vector terug met de ID's van de verbonden spoorsegmenten.
set-links!	(list → /)	Verandert de verbonden spoorsegmenten.
links-map	((symbol → any) → vector)	Voert een procedure uit op de ID's van de verbonden spoorsegmenten en geeft de opgespannen vector terug.
reserved?	(/ → symbol ∪ false)	Geeft de ID van de locomotief die het spoorsegment gereserveerd heeft of false.
reserve!	(symbol → /)	Verandert de reservatiestatus naar het meegegeven ID.
cancel-reservation!	(/ → /)	Zet het reservatiestatus op false
nr-of-links	(/ → integer)	Geeft het aantal verbonden spoorsegmenten terug
track?	(/ → boolean)	Test of iets tot de track classe behoort.

Tabel 1: Signaturen van track%

3.1.1 Toelichting

Het track ADT is eigenlijk een superclasse van de volgende ADT's. Dit is zodat de volgende ADT's de methoden erven van het track ADT. New maakt een nieuwe track ADT aan. Dit kan ook vervangen worden door (make-object track% < argumenten >). $get-track-id$ dient om de ID op te vragen. $track-links$ dient om de vector met verbonden spoorsegmenten op te vragen. $set-links!$ wijzigt de verbonden spoorsegmenten. $links-map$ voert een procedure uit op de vector-elementen. $reserved?$ geeft of de ID van het locomotief terug die het spoorsegment gereserveerd heeft of false. $cancel-reservation!$ zet de reservatiestand naar false. $nr-of-links$ geeft het aantal verbindingen terug. Classen zoals switch en detectieblokken zullen de methoden erven van deze classe.

3.2 Switch ADT

Het switch ADT erft de methoden van het Track ADT. Het zal dus bovenop de onderstaande methoden ook de methoden van het Track ADT bevatten. Het switch ADT houdt de stand van een wissel bij, welk spoorsegment verbonden is met de huidige stand en het staat toe om de

stand te veranderen.

Naam	Signatuur	Beschrijving
new	(symbol, list → switch%)	Maakt een nieuwe switch object aan
get-status	(/ → integer)	Geeft de stand van de wissel terug.
set-status!	(integer → integer)	Veranert de stand van de wissel.
get-merge-track	(/ → symbol)	Geeft de ID van het spoorsegment die niet met de stand van de wissels verandert.
get-linked-track	(/ → symbol)	Geeft de ID van het spoorsegment die verbonden is met de stand van de wissel.
switch?	(any → boolean)	Test of iets tot de switch classe behoort.

Tabel 2: Signaturen van switch%

3.2.1 Toelichting

Het switch ADT is een subclasse van het track ADT. Het kan dus dezelfde methoden gebruik als het track ADT. De extra methoden zijn *new* dat dezelfde argumenten verwacht als het track ADT (De ID van het spoorsegment en een lijst met de ID's van de verbonden spoorsegmenten.) Let op!: Zet de ID op de juiste index in de lijst. Als de stand van een wissel 1 is en het verbindt met het spoorsegment 1-5, dan zou de lijst '(U-1 1-5 ... etc...) moeten zijn. Dit is zodat de interne index van de stand van de wissels analoog is aan de stand van de wissels van de hardware. $get-status$ geeft de stand van de wissel terug als een nummer. $set-status!$ verandert de stand van de wissel naar het gegeven stand. Zorg er wel voor dat de index correct overeenkomt omdat het afleest van de vector die de verbonden spoorsegmenten bijhoudt. $get-merge-track$ geeft de ID van het spoorsegment die niet met de stand van de wissel verandert. $get-linked-track$ geeft de ID terug van de huidige verbonden spoorsegment aan de hand van de stand van de wissel.

3.3 Dblock ADT

Het detection block ADT is een subclasse van het track ADT. Het erft dus net zoals het switch ADT de methoden van het track ADT. Bovendien houdt het ook bij of dat er zich een locomotief op bevindt.

3.3.1 Toelichting

De methoden die bruikbaar zijn voor het detectieblok ADT zijn de methoden die het ADT erft van het track ADT. Dit is ook niet zonder de hierboven genoemde methoden die het arsenaal vervullen. *new* maakt een nieuwe detectieblok object aan. Het verwacht dezelfde argumenten zoals *new* van het track ADT. *occupied*? geeft de ID van de locomotief die het detectieblok bezet of false indien die niet bezet is. *occupy*! bezet het detectieblok met het gegeven ID van een locomotief. *vacant*! zal er voor zorgen dat het detectieblok terug vrij is.

Naam	Signatuur	Beschrijving
new	(symbol, list → dblock%)	Maakt een nieuw detectie block object aan.
occupied?	(/ → symbol ∪ false)	Geeft ID van de locomotief terug die momenteel de detectieblok bezet.
occupy!	(symbol → /)	Bezet de detectieblok met het gegeven ID.
vacant!	(/ → /)	Maakt de detectieblok vrij.
dblock?	(/ → boolean)	Test of ietst tot de detectieblok classe behoort.

Tabel 3: Signaturen van dblock%

3.4 Locomotive ADT

Het locomotief ADT houdt de interne data van een locomotief bij zoals snelheid, richting, vorige locatie, huidige locatie, manuele modus, het pad en de eindbestemming. Dit komt natuurlijk ook met methoden om ze te veranderen. Het pad en eindbestemming wordt gebruikt voor het automatisch trajectbeheer.

Naam	Signatuur	Beschrijving
new	(symbol, symbol, symbol, symbol → locomotive%)	Maakt een nieuw lovomotief object aan.
reserve!	(list → /)	Slaat een lijst van ID's van spoorsegmenten op die de locomotief gereserveerd heeft.
clear-reservations!	(/ → /)	Verwijdert de lijst van reservaties.
made-reservations?	(/ → list ∪ false)	Geeft een lijst van ID's van de gereserveerde spoorsegmenten terug.
manual-override	(boolean → /)	Wijzigt de reservatieprotocol van de locomotief.
manual?	(/ → boolean)	Geeft het reservatieprotocol terug van de locomotief.
get-loco-id	(/ → symbol)	Geeft de ID van de locomotief terug.
locomotive?	(/ → boolean)	Test of iets tot de locomotief classe behoort.

Tabel 4: Signaturen van locomotive%

3.4.1 Toelichting

new maakt een nieuw locomotief object aan. Het neemt als eerste de ID van de locomotief, de richting van de locomotief, de huidige locatie en de vorige locatie. *reserve*! verwacht een lijst van ID's van spoorsegmenten die de locomotief gereserveerd heeft. $clear - reservations!$ verwijdert de lijst van ID's van de gereserveerde spoorsegmenten. De waarde hiervan verandert naar $false$. $made - reservations?$ geeft of de lijst terug van de van de ID's van de gereserveerde spoorsegmenten of $false$ indien de locomotief geen spoorsegmenten gereserveerd heeft. $manual - override$ zal dienen om de status van de manier van het reservatiesysteem te veranderen. Wanneer

de status *true* is zal het de bezetting van een detectieblok negeren, dit zal vooral dienen om naar dichtbijzijnde detectieblokken te kunnen navigeren zonder rekening te houden met locomotieven die te dicht in de buurt zijn. De enige argumenten die dus gebruikt kunnen worden hiervoor zijn dus ook de booleans. *manual?* geeft de status van de manuele modus terug. Als laatste $get - loco - id$ geeft de ID terug van de locomotief.

3.5 Railway ADT

Het railway ADT brengt de de spoor gerelateerde ADT's zoals het track ADT, switch ADT, het detectieblok ADT en het locomotief ADT samen. Zo houdt he railway ADT het volledig spoornetwerk samen. Het netwerk zelf wordt bewaard in een graaf. Het programma maakt hierdoor gebruik van de *graph* library. Buiten een graaf van het spoornetwerk zelf bewaart het ook een graaf met detectieblokken met bogen naar die gedefinieerd zijn op vlak van bereikbaarheid. Er is bijvoorbeeld dus een boog tussen detectieblok 1-4 en 1-1 omdat er een pad is van deze detectieblokken waarbij er geen andere detectieblok tussen zit en dat de trein niet van richting moet veranderen tussen deze detectieblokken (wel op de detectieblok zelf!). Op basis hiervan kunnen we dus trajecten berekenen. Hiervan en een speciale graaf die alleen de detectieblokken bevat. Dit is zodat we kunnen navigeren en paden kunnen berekenen aan de hand van paden tussen detectieblokken. De methoden van het Railway ADT zullen voornamelijk zoek-functionaliteiten aanbieden. Een voorbeeld hiervan is als we een locomotief object willen aanspreken dan zal het dit eerst opzoeken in de hashmap waar het opgeslagen staat. Hetzelfde geldt voor het aanspreken van een track ADT en zijn subclassen.

Naam	**Signatuur**	**Beschrijving**
new	(/ → railway%)	Maakt een nieuw railway object aan.
get-list-of-tracks	(/ → list)	Geeft een lijst van terug van de hashmap van alle spoorsegmenten.
get-track	(/ → track ∪ dblock ∪ switch ∪ false)	Geeft het gezochte spoor object terug.
make-track	(symbol, list → track)	Maakt een track object aan.
make-switch	(symbol, list → switch)	Maakt een wissel object aan.
make-dblock	(symbol, list → dblock)	Maakt een detectieblok object aan.
add-track!	(track ∪ dblock ∪ switch → track ∪ dblock ∪ switch)	Voegt het spoor object toe aan het spoornetwerk.
remove-track!	(symbol → /)	Verwijdert het gezochte object uit het spoornetwerk.
update-track!	(symbol (track ∪ dblock ∪ switch → any) → /)	Voert de meegegeven procedure uit op het gezochte object.
for-each-track	((track ∪ dblock ∪ switch → any) → any)	Voert een procedure uit op alle spoorsegmenten
for-each-link	((track ∪ dblock ∪ switch → any), symbol → vector)	Voert een procedure uit op de aanliggende spoorsegmenten van het gezochte spoorsegment.
get-railway-graph	(/ → graph)	Geeft de graaf met het spoornetwerk terug.

Tabel 5: Signaturen van railway%

Naam	Signatuur	Beschrijving
get-dblock-graph	(/ → graph)	Geeft de graaf met alleen detectieblokken terug.
make-locomotive	(symbol, symbol, symbol, symbol → locomotive)	Maakt een locomotief object aan aan de hand van de gegeven ID, richting, huidige detectieblok en vorige detectieblok.
add-locomotive!	(locomotive → /)	Voegt een locomotief object toe aan het spoornetwerk.
remove-locomotive!	(symbol → /)	Verwijdert het gezochte locomotief object van het spoornetwerk.
update-locomotive!	(locomotive → /)	Verandert gegevens van een locomotief door het te vervangen met een nieuwe.
get-locomotive	(symbol → locomotive ∪ false)	Geeft het gezochte locomotief object terug of false indien deze niet bestaat.
for-each-loco	((locomotive → any) → any)	Voert een procedure uit op alle locomotief objecten.
railway?	(any → boolean)	Geeft terug of iets tot de railway classe behoort.

Tabel 6: Vervolg van de signaturen van railway%

3.5.1 Toelichting

Niet alle methoden zullen besproken worden omdat sommige methoden zoals $make - switch$, $make - dblock$, $make - track$ en $make - locomotive$ Analoog zijn aan de methode new van het respectievelijke object dat het aanmaakt. De $add - track!$ en $add - locomotive!$ voegen beiden respectievelijk hun spoor objecten of locomotief objecten toe aan het spoornetwerk. $remove - track!$ en $remove - locomotive!$ verwijderen gezochte objecten van het spoornetwerk terwijl $update - track!$ en $update - locomotive!$ de interne opslag van de objecten aanpassen. Let wel op dat $update - track!$ een procedure vraagt bij de oproep en $update - locomotive$ een nieuw locomotief object vraagt die het oude vervangt. $get - track$ en $get - locomotive$ geven respectievelijk het gezochte spoor object terug of het locomotief object. Wanneer het gezochte object niet bestaat zal het false teruggeven. Dan hebben we nog $get - list - of - tracks$ dat een hash-¿list teruggeeft van het spoornetwerk (alleen de spoor objecten, dus geen locomotieven). $for - each - track$ mapt een procedure uit op alle spoor objecten. $for - each - link$ voert dan een procedure uit op alle aanliggende spoorobjecten van het gezochte spoorsegment. $get - railway - graph$ geeft de graaf terug van het spoornetwerk. Dit is belangrijk zodat clients kunnen 'syncen' met Infrabel. Bovendien helpt $get - dblock - graph$ hier ook mee want dit geeft dan de graaf terug met alleen maar detectieblokken voor het trajectbeheer. Ten laatste hebben we nog $for - each - loco$ die ene procedure uitvoert op elk bestaande locomotief object.

3.6 Infrabel ADT

Het infrabel ADT is de server van het programma. Dit houdt de interne werking van het gehele spoornetwerk inclusief reservatiesysteem en een deel van de trajectbeheer. Het voert dus de automatische processen uit die telkens als een locomotief van plek verandert uit alsook de staat van één van de functionaliteiten aangepast wordt. Dit geldt ook voor de spoor objecten. Het infrabel ADT biedt ook communicatie aan voor clients (NMBS) die op hun beurt ook berekeningen uitvoeren en vervolgens methoden oproepen binnen infrabel.

Naam	Signatuur	Beschrijving
new	(boolean → infrabel%)	Maakt een nieuw infrabel object aan.
add-loco!	(symbol, symbol, symbol → /)	Maakt een nieuw locomotief object aan en slaat deze intern op.
remove-loco!	(symbol → /)	Verwijdert het gezochte locmotief object van de interne opslag.
set-loco-speed!	(symbol, integer → /)	Verandert de snelheid van een locomtief.
get-loco-speed	(symbol → integet)	Vraagt de snelheid van een locomotief op.
set-loco-destination!	(symbol, symbol, list → /)	Defineerd een eindbestemming voor een locomotief met gegeven pad.
set-loco-direction!	(symbol, symbol →)	Verandert de richting van een locomotief.
get-list-of-dblocks	(/ → list)	Geeft een lijst terug van de ID's van de detectieblokken in het spoornetwerk.
get-list-of-switches	(/ → list)	Geeft een lijst terug van de ID's van de wissels in het spoornetwerk.
get-list-of-locos	(/ → list)	Geeft een lijst van de ID's van de locomotieven die zich op het spoornetwerk bevinden.
set-switch-position!	(symbol, integer → /)	Verandert de stand van een wissel.
set-manual-mode!	(symbol, boolean → /)	Verandert de status van het reservatieprotocol van een locomotief.
fetch-railway	(/ → list)	Geeft een lijst terug met alle bogen in het spoornetwerk.
fetch-reservation-graph	(/ → list)	Geeft een lijst terug met alle bogen in de graaf met allen detectieblokken.
infrabel?	(any → boolean)	Test of iets tot de infrabel classe hoort.

Tabel 7: Signaturen van infrabel%

3.6.1 Toelichting

new maakt een nieuw infrabel object aan. Het verwacht een boolean om de initatie ervan op de simulator of op de hardware te doen. $add - loco$ verwacht een ID, de ID van het huidige detectieblok en de ID van het vorige detectieblok om een locomotief object aan te maken en op te slagen. $remove - loco$ verwacht alleen een ID om het locomotief object op te zoeken en te verwijderen. $set - loco - speed!$ verandert de snelheid van het gezochte locomotief naar de meegegeven waarde. $get - loco - speed$ zal deze waarde dan teruggeven. $set - loco - destination!$ zet de eindbestemming van de locomotief alsook het pad. Zo kan het automatische trajectbeheer algoritme het pad verder uitvoeren. $set-loco-direction!$ verandert de richting van een locomotief naar de gegeven richting. $get - list - of - dblocks$, $get - list - of - switche$ en $get - list - of - locos$ geven een lijst terug met de ID's van de respectievelijke objecten. $set - switch - position!$ verandert de stand van het gezochte wissel naar de meegegeven stand (meestal 1 of 2). $set - manual - mode!$ verandert het reservatieprotocol van de gezochte locomotief. Zo kan het nog steeds reservaties uitvoeren wanneer er een andere locomotief te dicht bij is (let op voor botsingen!). Dit is voornamelijk bedoelt voor de binnenkant van de $setup - hardware$ modus.

$fetch-railway$ geeft een lijst terug met alle bogen in het spoornetwerk en $fetch-reservation-graph$ zal hetzelfde doen maar voor de graaf met alleen detectieblokken.

3.7 GUI ADT

Het GUI ADT biedt functionaliteiten aan voor het interageren met de user interface die NMBS aanbiedt. Dit is om te kunnen interageren met het spoornetwerk. Het bevat vooral teken methoden om de UI te bewerken en informatie bij te werken. Een aantal voorbeelden hiervan zijn bevoorbeeld de berichten in het logboek die aangepast worden telkens dat er een log toegevoegd wordt. Het biedt ook interactie toe met NMBS die op zijn beurt interactie aanbiedt met Infrabel.

Naam	Signatuur	Beschrijving
new	(integer, integer, integer → gui)	Maakt een gui object aan.
add-log!	(string → /)	Voegt een log entry toe aan het logboek.
set-loco-speed!	(symbol, integer → /)	Verandert de snelheid op de UI.
set-dblock-occupied!	(symbol, symbol → /)	Verandert de bezetheid van een detectieblok naar bezet op de UI.
free!	(symbol → /)	Verandert de bezetheid van een detectieblok naar vrij op de UI.
reserve-dblock!	(symbol, symbol → /)	Verandert de reservatiestatus van een detectieblok naar het meegegeven ID van een locomotief op de UI.
reserve-switch!	(symbol, symbol → /)	Verandert de reservatiestatus van een wissel naar het meegegeven ID van een locomotief op de UI.
free-dblock!	(symbol → /)	Verandert de reservatiestatus van een detectieblok naar "Vacant".
free-switch!	(symbol → /)	Verandert de reservatiestatus van een wissel naar "Vacant".
set-switch-position!	(symbol, integer → /)	Verandert de stand van een wissel op de UI.
halt!	(symbol → /)	Zet de snelheid van een locomotief op 0 op de UI.
set-to-not-set	(symbol → /)	Verandert het bericht in de route tabblad wanneer een bestemming bereikt is naar "destination reached".
set-route!	(symbol, list ∪ string)	Geeft de berekende route weer op de UI.
add-locomotive!	(symbol, integer, (integer → /), symbol, (symbol → /), boolean, (boolean → /), symbol, symbol ∪ false, (symbol ∪ false) → /)	Voegt een locomotief toe aan de UI en biedt interactie toe aan de hand van sliders en knoppen.
remove-locomotive!	(symbol → /)	Verwijdert een locomotief van de UI.

Tabel 8: Signaturen van railway%

draw-dblock-info!	(symbol, string, string → /)	Tekent & initaliseert informatie over een detectieblok op het scherm.
draw-switches-info	(symbol, integer, symbol, (integer → /) → /)	Tekent informatie van de wissels en biedt interactie toe met de stand van de wissels op de UI.
init-tab-3	((symbol, symbol, symbol → /), (symbol → /)→ /)	Initialiseert de 3e tabblad om treinen toe te voegen en te verwijderen met de meegegeven procedures.
init-tab-1	((/ → /) → /)	Initialiseert de eerste tabblad om het programma af te kunnen sluitne.
init	(/ → /)	Initialiseert het skelet van de UI.
start-gui	(list → /)	Start de UI op en staat toe dat ermee geinterageerd kan worden.
exit	(/ → /)	Doet de UI dicht.

Tabel 9: vervolg van de signaturen van infrabel%

3.7.1 Toelichting

new maakt een nieuw gui object aan met de gegeven lengte en breedte alsook spatie tussen elementen in de UI. $add - log!$ voegt een bericht toe aan het logboek dat de UI bijhoudt. $set - loco - speed!$ verandert de weergegeven snelheid van een gezochte locomotief met de gegeven snelheid. $set-dblock-occupied!$ vraagt de ID van een detectieblok en de ID van een locomotief om bezetting van de detectieblok te veranderen op de UI. $free!$ zet de bezetting van de meegegeven detectieblok ID naar "Vacant". $reserve - dblock!$ verwacht de ID van een detectieblok en de ID van een locomotief om de reservatiestatus van de detectieblok op de UI te veranderen naar de ID van de locomotief. $reserve - switch!$ doet op dezelfde manier hetzelfde voor de wissels. $free-dblock!$ en $free-switch!$ werken op dezelfde manier als $free!$ maar voor de reservatiestatus in plaats van de bezetting. Het zet respectievelijk de reservatiestatus naar "Vacantöp de UI. $set - switch - position!$ voor de gegeven wissel ID en nieuwe wisselstand, verandert de huidige stand van de wissel naar de nieuwe wisselstand. *halt* zet de snelheid van de gezochte locomotief naar 0. Wanneer een locomotief zijn eindbestemming bereikt heeft moet $set - to - not - set$ gebruikt worden. Het enige nodige argument is de ID van de locomotief die de eindbestemming bereikt heeft. $set - route!$ zal analoog aan de andere methoden de route die op de UI staat van de gegeven locomotief ID. Nu komen de iets complexere methoden. $add - locomotive!$ voegt een locomotief toe aan de UI. Het verwacht als eerste de ID van een locomotief dan de snelheid van de locomotief, een procedure om de snelheid te veranderen, de richting, een procedure om de richting te veranderen van de locomotief, de status van de manuele modus, de procedure om de manuele status te veranderen, de locatie (detectieblok), de eindbestemming en een procedure om de eindbestemming te wijzigen. Dit is allemaal zodat de sliders en knoppen die dienen om met de locomotief te interageren de juiste methoden oproepen. Om een locomotief van de UI te verwijderen is het veel simpeler: We roepen $remove-locomotive!$ op met de ID van de locomotief die we willen verwijderen. $draw - dblock - info!$ verwacht de ID van een detectieblok, een string met wat de bezetting is en een string met wat de reservatiestatus is. $draw - switches - info$ doet hetzelfde maar neemt in plaats van 2 strings de wisselstand als een integer, de reservatiestatus als een symbool en een procedure om de stand van de wissels te veranderen. $init - tab - 3$ verwacht een procedure om een locomotief toe te voegen waarvan het alle argumenten symbolen zijn. Het eerste is de ID van de locomotief dan de ID van de vorige locatie (detectieblok) en als

laatste de ID van de huidige locatie (detectiblok). Ten tweede verwacht het een procedure om locomotieven te verwijderen. Hiervan moet het eerste argument de ID zijn van de locomotief. $init - tab - 1$ Initialiseert de eerste tabblad. De meegegeven procedure moet een procedure zijn om de applicatie te sluiten. *init* maakt het frame aan en start de UI. $start - gui$ start de UI op met de startmethoden van het programma. Dit zal dan meegegeven worden in een lijst van strings met de mogelijke startup opties. *exit* sluit de UI af.

3.8 NMBS ADT

Het NMBS ADT is de client die zich moet verbinden met de server (Infrabel). Het het berekend het pad bij automatische trajectbeheer en behoudt de informatie over het spoornetwerk up to date. Dit doet het NMBS ADT doormidden van TCP communicatie met Infrabel waar het overgrote deel van de data opgevraagd wordt.

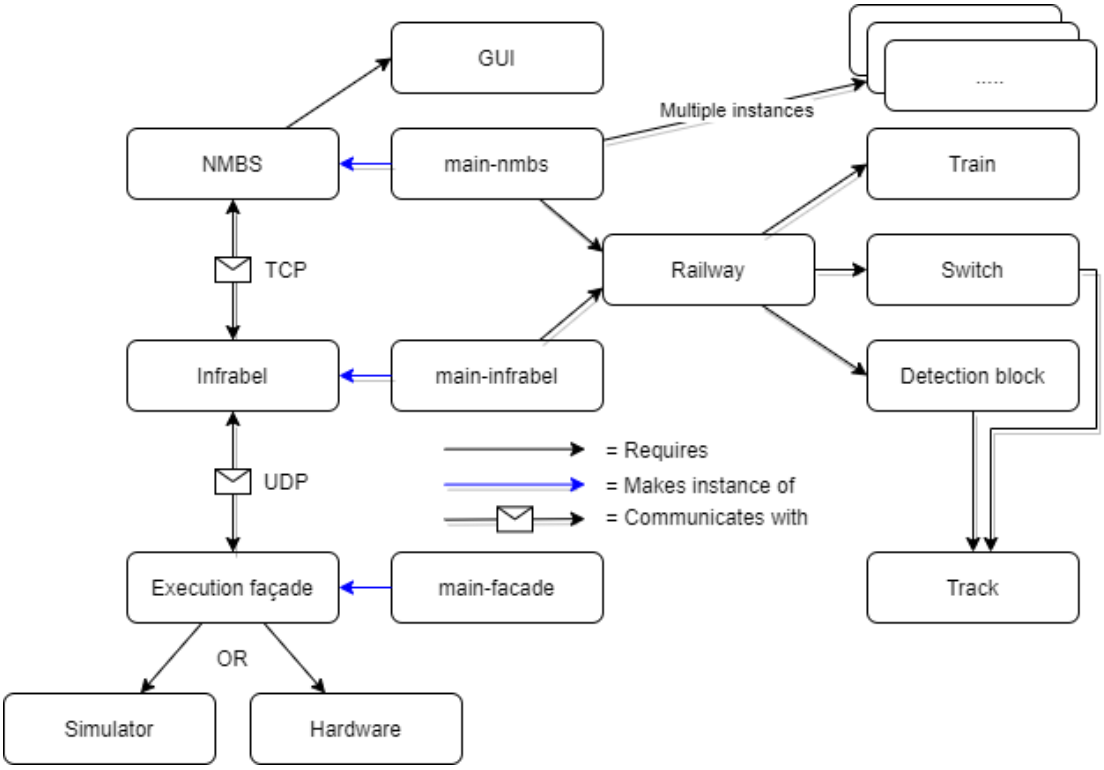
Naam	**Signatuur**	**Beschrijving**
new	(string $\rightarrow$ nmbs)	Maakt een NMBS object aan.
start-nmbs	(/ $\rightarrow$ /)	Initialiseert het nmbs object en start de UI om ermee te kunnen interageren.

Tabel 10: vervolg van de signaturen van nmbs%

3.8.1 Toelichting

new maakt een nieuw nmbs object aan om er mee te interageren. Het probeert ook direct een verbinding te maken met het meegegeven IP adres. Dit moet dus het IP adres zijn van de server of infrabel. $start - nmbs$ start het object op en bij gevolg ook de UI. De gebruiker zal dan gevraagd worden op welke modus men de simulator of hardware wilt starten. Hierna kan men verder interageren met de UI en het spoornetwerk.

4 Afhankelijkheids diagram



Figuur 1: Afhankelijkheids diagram

5 Beschrijving van API tussen infrabel- en NMBS-component

NMBS naar Infrabel	Infrabel terug naar NMBS
Vraagt informatie over de spoor objecten en locomotief objecten	Geeft lijsten van informatie over de spoor objeten en locomotief objecten terug.
Vraagt om de verbinding te sluiten	/
Vraagt om de stand van een wissel te veranderen	Stuurt de nieuwe stand van de wissel terug
Vraagt om de eindbestemming te zetten en het automatisch trajectbeheer in gang te steken	Vraagt een herberekening indien dit aangevraagd is terwijl de locomotief niet detecteerbaar is.
Vraagt om een locomotief to te voegen aan het spoornetwerk.	Geeft de gegevens van de toegevoegde locomotief terug om op de UI te kunnen zetten.
Vraagt om de richting van een locomotief te veranderen	Geeft de nieuwe richting terug.

Vraagt om de snelheid van een locomotief te wijzigen.	Geeft alleen de nieuwe snelheid terug indien deze nul is.
Vraagt om de het reservatieprotocol te wijzigen van een locomotief	/
Vraagt om een locomotief van het spoornetwerk te verwijderen	/
Vraagt om de simulator of hardware op een gegeven manier te starten	/

Tabel 11: Van NMBS naar Infrabel en terug

Infrabel naar NMBS	**NMBS terug naar Infrabel**
Stuurt logboek evenementen (verandering snelheid, richting etc.)	/
Stuurt berekende routes	/
Stuurt een lijst van de bezette spoorsegmenten	/
Stuurt een lijst van de gereserveerde spoorsegmenten	/

6 Planning

Week	**Gepland**
5	18/10 Indienen voorstudie.
6	Stuurt berekende routes & Feedback afwachten, ADT detectieblok en switch implementeren.
7	ADT locomotive & track implementeren.
8	ADT railway & beginnen aan ADT infrabel.
9	ADT infrabel & NMBS component maken.
10	Laatste aanpassingen NMBS.
11	18/10 Indienen fase 1: code en documentatie.
12	Feedback afwachten.
13 & 14	Implementeren van grafen voor het automatisch trajectbeheer.
15 & 16	Automatisch trajectbeheer implementeren.
17	Mogelijk maken dat GUI het traject kan tonen aan de hand van het trajectbeheer.
18 & 19	Botsingen voorkomen implementeren.
20	GUI aanpassingen voor mogelijk extra componenten.
21	Feedback vragen.
22 + 23 + 24 + 25	Aanpassingen doorvoeren aan de hand van de feedback.
26	14/03 Indienen fase 2: code en documentatie.
27	Feedback afwachten.

28 + 29 + 30 + 31	RPI.
32	GUI aanpassen indien nodig.
33 + 34 + 35	Hardware film maken/aanpassingen doorvoeren indien nodig.
36	23/05 Indienen fase 3: code en documentatie.

Tabel 12: Planning

7 Logboek

Week	Gepland
5	18/10 Ingediend voorstudie.
7	ADT locomotive & track implementeren.
8	ADT railway & beginnen aan ADT infrabel.
9	ADT infrabel & NMBS component maken.
11	Code en documentatie ingeleverd (fase 1).
24	Aanpassingen detectieblok ADT, Track ADT, switch ADT.
25	Aanpassingen Locomotive ADT, Infrabel ADT.
26	Aanpassingen NMBS ADT, Implementeren reservatiesysteem en deel van TCP. Code en documentatie ingeleverd (fase 2).
29	Powerpoint gemaakt en projectverdediging.
30	Aanpassingen reservatiesysteem.
31 & 32	Aanpassingen reservatiesysteem, GUI en trajectbeheer
33	Aanpassingen automatisch trajectbeheer en testen met RPI.
34	Verslagen geschreven.
36	code en documentatie ingeleverd (fase 3).

Tabel 13: Logboek