Programmeerproject 2: Specificatie “Railway system” Fase III

Rayane Kouidane

2de Bachelor Computerwetenschappen

Rayane.Kouidane@vub.be

N°0587073

Academiejaar 2022-2023

Inhoudsopgave

1. Introductie 2
2. Algemene beschrijving 2
3. Software-architectuur 2
4. ADT's 3
   - 4.1 Train ADT 3
   - 4.2 Detection-block ADT 3
   - 4.3 Switch ADT 3
   - 4.4 Gui ADT 4
   - 4.5 Railway ADT 4
   - 4.6 NMBS ADT 5
   - 4.7 Infrabel 5
   - 4.8 Andere 6
5. TCP 7
6. Gebruikte bronnen 7

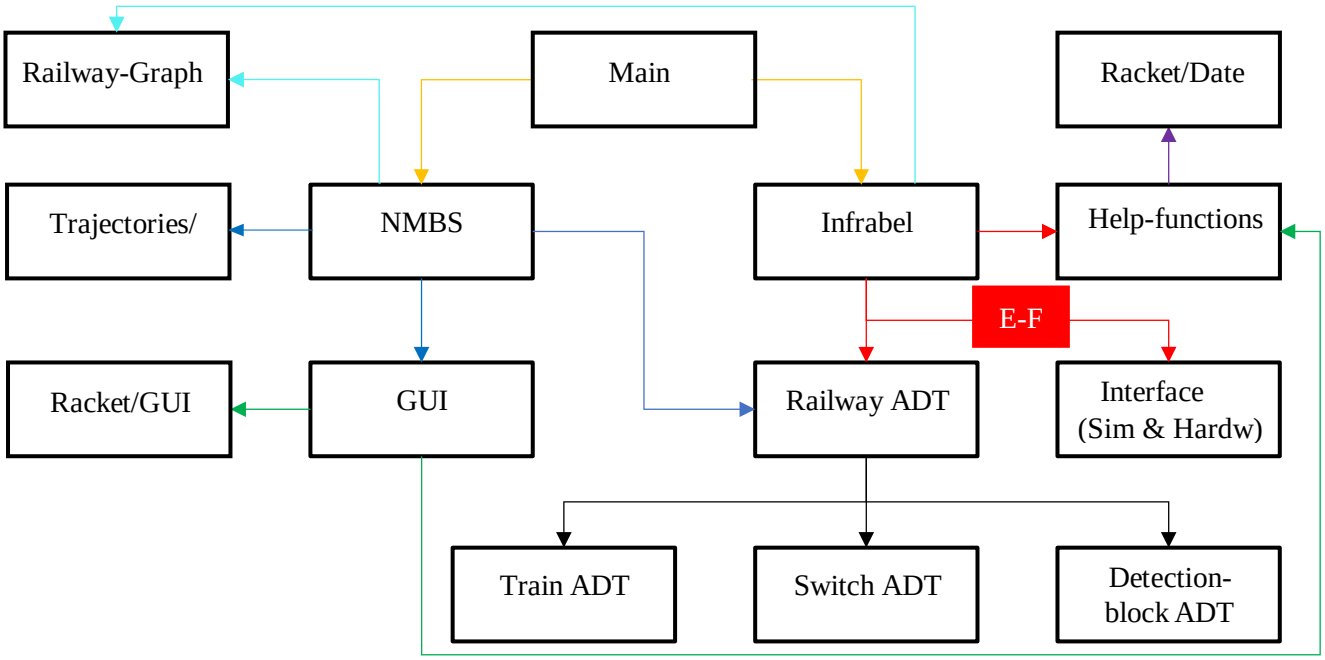
1. Introductie

Dit bestand bevat de specificatie van de derde fase van het project. Eerst zal ik een algemene beschrijving geven van wat de vereisten zijn voor de derde fase. Daarna de software-architectuur beschrijven met een afhankelijkheidsdiagram. Als laatst zal ik de verschillende componenten bespreken.

2. Algemene beschrijving

Voor de derde fase van het project moesten we de laatste twee basisvereisten toevoegen aan het project en een derde extra vereiste kiezen en deze implementeren. Infrabel en NMBS moeten vanaf nu in aparte processen draaien en communiceren via TCP. Daarnaast moet ons project op de simulator en de hardware werken. Als derde vereiste heb ik gekozen voor *Automatische berekening*.

3. Software-architectuur



Figuur 1: Afhankelijkheidsdiagram Fase III

- *Main* is afhankelijk van *NMBS* en *Infrabel*, omdat we daarin het controlesysteem kunnen opstarten door *NMBS* en *Infrabel* aan te maken en starten.
- Infrabel is afhankelijk van het *Railway ADT*, *Interface* (zowel simulator als hardware), *Railway-Graph* en *Help-functions*. *Help-functions* is een bestand dat hulpprocedures bevat. Dit bestand is zelf afhankelijk van *Racket/Date*. Het gebruikt ook procedures uit *Railway-Graph*. Hiermee kan het weten welke trein op welk detectieblok is wanneer we werken op de hardware. Via procedures uit de *Interface* en *Railway ADT* kan *Infrabel* het spoornetwerk van verschillende onderdelen wijzigen. Deze zijn: de simulator/hardware opstelling en objecten. Daarom is het *Railway ADT* afhankelijk van het *Train*, *Switch* en *Detection-block ADT*.
- *NMBS* is afhankelijk van *Railway ADT*, *Railway-Graph* en de *GUI*. Wanneer de gebruiker een automatisch traject wil afleggen dan zal *NMBS* het kortste pad berekenen en deze sturen naar *Infrabel*. De *GUI* gebruikt hulpprocedures uit *Help-functions* en ook *Racket/GUI*. Hiermee kan het knoppen, sliders… tekenen.
- Wanneer een gebruiker een traject uit een tijdstabel wil starten, dan zal *NMBS* het juiste traject moeten opzoeken in het bestand *Trajectories* en deze vertalen naar de juiste instructies. Wanneer het traject vertaald is, wordt deze gestuurd naar *Infrabel* via TCP.
- Voor het *Train ADT*, *Switch ADT*, *Detection-block ADT*, *Railway ADT*, tijdstabellen uitlezen, automatische trajectberekening, graf en dynamische snelheid zijn er unit-tests. Voor de andere zijn er geen tests omdat het moeilijk is om te controleren of iets wel of niet werkt via *RackUnit*. Je kan dit enkel controleren door het geheel te starten. Meer informatie over het *Execution Facade* in 4.8.

4. ADT's

4.1 Train ADT

Naam	Signatuur
make-train	symbol → Train
id	∅ → symbol
speed	∅ → number
speed!	number → ∅
direction	∅ → boolean
inverse-direction!	∅ → ∅
prev-dtb	∅ → symbol
prev-dtb!	symbol → ∅
curr-dtb	∅ → symbol
curr-dtb!	symbol → ∅
trajectory	∅ → symbol
trajectory?	∅ → boolean
trajectory!	∅ → ∅

Tabel 1: Operaties Train ADT

Het Train ADT bevat alle procedures die nodig zijn om informatie te vragen over de trein: de id van de trein (`id`), de snelheid (`speed`), de richting (`direction`), de vorige positie (`prev-dtb`), huidige positie (`curr-dtb`) en een boolean (`trajectory?`) die #t is wanneer een trein een traject volgt. Dus wanneer `trajectory` niet leeg is. Het bevat ook procedures om die informatie te wijzigen: `speed!`, `inverse-direction!, prev-dtb! en curr-dtb!` wijzigen de snelheid, richting, vorige positie en huidige positie van de trein. Met `trajectory!` kan je de waarde van `trajectory` veranderen.

4.2 Detection-block ADT

Naam	Signatuur
make-detection-block	symbol → Detection-block
id	∅ → symbol
last-train	∅ → symbol
last-train!	symbol → ∅

Tabel 2: Operaties Detection-block ADT

Het Detection-block ADT representeert een detectie-block. Deze bevat de id van de detectieblok (`id`) en de laatste trein die op de detectieblok was (`last-train`). Met `last-train!` kan je de laatste trein wijzigen.

4.3 Switch ADT

Naam	Signatuur
make-switch	symbol → Switch
id	∅ → symbol
state	∅ → number
state!	number → ∅

Tabel 3: Operaties Switch ADT

Het Switch ADT bevat de procedures om informatie te vragen over een wissel en de staat ervan te wijzigen. De procedure `id` en `state` geven respectievelijk de id van de switch en de staat terug. `state!` wijzigt de positie van de wissel.

4.4 Gui ADT

Naam	Signatuur
make-gui	Ø → Gui
start	(Ø → Ø) → Ø
exit	(Ø → Ø) → Ø
draw-train	(symbol → Ø) → Ø
draw-switches	(Ø → Ø) → Ø
draw-dt-blocks	(Ø → Ø) → Ø
draw-time-tbl	(Ø → Ø) → Ø
draw-trajectory	(symbol → Ø) (Ø → Ø) → Ø

Tabel 4: Operaties GUI ADT

Het Gui ADT is het ADT die zal zorgen voor het tekenen van de userinterface. De procedure `start` zorgt ervoor dat wanneer we op een zekere knop drukken, bepaalde evenementen plaatsvinden zoals het openen van een nieuwe frame. De procedure exit zal het frame sluiten, alle threads, die nodig zijn voor het updaten van de berichten, stoppen en andere functies uitvoeren. Daarnaast zorgt de Gui voor het tekenen van de commands voor de treinen, wissels, detectieblokken, om trajecten uit een bestand te kunnen uitvoeren en voor de automatische trajectberekeningen. Deze procedures nemen een callback als argument waarin alle procedures zitten die we nodig hebben wanneer we bijvoorbeeld op een zekere knop drukken.

4.5 Railway ADT

Naam	Signatuur
make-railway	Ø → Railway
trains	Ø → pair
detection-blocks	Ø → pair
switches	Ø → pair
reset!	Ø → Ø
add-train!	symbol symbol symbol → Ø
train-speed	symbol → number
train-speed!	symbol number → Ø
train-prev-dtb	symbol → symbol
train-prev-dtb!	symbol symbol → Ø
train-curr-dtb	symbol → symbol
train-curr-dtb!	symbol symbol → Ø
train-direction	symbol → boolean
train-inverse-direction!	symbol → Ø
train-trajectory	symbol → pair
train-trajectory?	symbol → boolean
train-trajectory!	symbol pair → Ø
make-switches!	pair → Ø
switch-position	symbol → number
switch-position!	symbol number → Ø
make-detection-blocks!	pair → Ø
dtb-last-train	symbol → symbol
dtb-last-train!	symbol symbol → Ø

Tabel 5: Operaties Railway ADT

Het Railway ADT stelt het spoornetwerk voor. Het houdt de volgende zaken bij: de aangemaakte treinen, de wissels en de detectieblokken. Deze kunnen we terugkrijgen door respectievelijk `trains`, `detection-blocks` en `switches` op te roepen. Om deze lijsten weer leeg te maken kunnen we de procedure `reset!` gebruiken. `add-train!` zal een nieuw train-object toevoegen aan de lijst van treinen. Om de snelheid van een trein te kennen, gebruiken we de procedure `train-speed` en om deze te wijzigen gebruiken we `train-speed!`. Dankzij `train-prev-dtb!` en `train-curr-dtb!` kunnen we de waarde van de vorige positie (`train-prev-dtb`) en de laatste positie (`train-`

`curr-dtb)` wijzigen. Om te weten in welke richting een trein aan het rijden is, roepen we `train-direction` op. Om de richting te veranderen in de tegengestelde richting gebruik je `train-inverse-direction!`. We kunnen ook weten of een trein een traject aan het uitvoeren is door `train-trajectory?` op te roepen. Als je het pad van een trein wil terugkrijgen dan gebruik je `train-trajectory`. Om het pad te wijzigen gebruiken we `train-trajectory!`. Met `make-switches!` en `make-detection-blocks!` maken we de wissels en detectieblokken aan en zetten we die in een lijst. `switch-position` en `switch-position!` zorgen ervoor dat we de positie van een wissel kunnen lezen en deze veranderen. Met `dtb-last-train` en `dtb-last-train!` kan je weten welke trein het laatst op een bepaalde detectieblok was en deze updaten als het verandert.

4.6 NMBS ADT

Naam	Signatuur
make-nmbs	`∅ → Nmbs`
start	`∅ → ∅`
stop	`∅ → ∅`
add-train!	`symbol symbol symbol → ∅`
speed!	`symbol number → ∅`
train-curr-dtb!	`symbol symbol → ∅`
dtb-last-train!	`symbol symbol → ∅`
train-inverse-direction!	`symbol → ∅`
train-trajectory!	`symbol pair → ∅`
calculate-trajectory	`symbol symbol →` pair
make-detection-blocks	`pair → ∅`
make-switches	`pair → ∅`
switch-position!	`symbol number → ∅`

Tabel 6: Operaties NMBS ADT

Het NMBS ADT zorgt ervoor dat wanneer de gebruiker iets wijzigt in de GUI, dit ook werkelijk gebeurt. Dit is mogelijk door gebruik te maken van callbacks, die opgeroepen worden in de GUI. De callbacks bevatten de procedures om bijvoorbeeld de snelheid van de trein te veranderen, de posities van de wissels te wijzigen…. De procedure `start` zal de GUI opstarten en de nodige procedures uitvoeren. Bijvoorbeeld: wanneer de gebruiker op "simulator" drukt, dan wordt de simulator geopend. Met de procedure `stop`, kunnen we (door op de knop "Exit" te drukken) de GUI sluiten. De simulator wordt ook gestopt. Daarnaast heeft NMBS ook een eigen representatie van de railway, hiermee kan het bijvoorbeeld de positie van treinen in de GUI tonen. `calculate-trajectory` zal het kortste pad berekenen tussen de laatste positie van een trein en een gegeven bestemming. Alle andere functies werken zoals in '4.5 Railway ADT'.

4.7 Infrabel

Naam	Signatuur
make-infrabel	`∅ → Infrabel`
start	`symbol pair → ∅`
stop	`∅ → ∅`
switch-option!	`boolean → ∅`
speed!	`symbol number → ∅`
train-inverse-direction!	`symbol → ∅`
train-inverse-speed!	`symbol → ∅`
train-trajectory!	`symbol pair → ∅`
process-trajectories!	`pair boolean → ∅`
calculate-trajectory	`pair → ∅`
switch-position!	`symbol number → ∅`

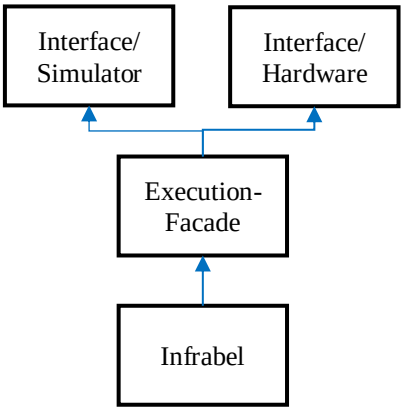
Tabel 7: Operaties Infrabel ADT

Infrabel is verantwoordelijk voor de communicatie met de simulator/hardware en het aansturen van de verschillende componenten. Het heeft ook een eigen representatie van het spoornetwerk. Wanneer we Infrabel willen starten dan roepen we `start` op. Deze procedures zal alle treinen, detectieblokken en wissels aanmaken en deze updaten in railway-infrabel en railway-nmbs. Het zal ook de interface (simulator of hardware) en update-loops starten. `stop` zal de nodige threads en infrabel stoppen. Infrabel bevat de procedures die zullen zorgen voor de dynamische snelheden, tijdstabellen uitlezen en automatische trajectberekeningen. `switch-option!` is een functie waarmee we kunnen kiezen tussen simulator en hardware. Het zal een boolean wijzigen waardoor de functies zullen werken op de simulator of hardware. `process-trajectories!` zal voor elk traject een thread starten waarin het pad uitgevoerd wordt. De andere procedures werken zoals in '4.5 Railway ADT'. Het volgende is de uitleg van hoe de drie gekozen vereisten werken:

- **Dynamische snelheden**
  Voor het berekenen van de optimale snelheid kijken we naar de lengte van het spoor, het aantal treinen en ook de snelheid van deze treinen. Op lange rechte sporen met een detectieblok zal de trein snel rijden. Voor de rest zal het steeds traag rijden omdat het zich op korte sporen of bochten bevindt. Wanneer er meerdere treinen op een spoor rijden, dan moeten ze trager bewegen. Maar als er bijvoorbeeld een trein een snelheid heeft van 30. Dan moeten de andere treinen zich aanpassen aan deze snelheid.
- **Tijdstabellen uitlezen uit een bestand**
  Om een trein een traject (uit een bestand) te laten volgen, wordt er een loop gebruikt die lijn per lijn het bestand zal lezen en omvormen naar instructies. Dit kunnen we terugvinden in de procedure `make-trajectory!`. Eerst leest het de id van de trein en daarna kan het verder gaan met de volgende instructies. Afhankelijk van wat er staat, zal het bepaalde taken uitvoeren. Bijvoorbeeld: *S-2-3/3* zal wissel S-2-3 op positie 3 zetten, *1-1/stop* zal de trein laten stoppen wanneer het op detectieblok 1-1 is, *+start* en *-start* dienen om de trein weer te starten (in dezelfde of tegengestelde richting) nadat het gestopt werd en *finish* betekent dat het traject gedaan is. Wanneer een trein een traject aan het uitvoeren is, kan de gebruiker de staat van de trein niet wijzigen. Ik koos voor deze stijl omdat de instructies duidelijk zijn voor de gebruiker en niet zo moeilijk zijn te vertalen voor de programmeur.
- **Automatische trajectberekeningen**
  Wanneer de gebruiker een automatisch berekend traject start, dan zal NMBS het traject berekenen. Het zal een graf gebruiken dat terug te vinden is in *railway-graph.rkt*. Het gebruikt een procedure genaamd 'shortest-path' uit de cursus 'Algoritmen & Datastructuren 2'. Om te weten op welke positie de wissels moeten staan, kijken we naar de labels van de bogen. Bijvoorbeeld: de label van de knoop met label 'S-20 naar de knoop '1-5 is 1. Dit betekent dat wanneer we van S-20 naar 1-5 willen gaan, dan moet S-20 op positie 1 staan. Wanneer de trein het pad niet meer volgt dan zal het, als het nodig is, van richting veranderen en het nieuwe pad berekenen. Dit is om bijvoorbeeld van een been van een wissel naar het ander te gaan. Om naar detectieblok 2-8 te gaan is het moeilijk omdat er geen detectieblok is om de richting van de trein te kunnen veranderen. Als oplossing kijk ik of de trein naar 2-8 gaat en als alle wissels op de juiste positie staan. Daarna stop ik de loop gedurende een bepaalde tijd en als laatst verander ik de richting van de trein.

4.8 Andere

- *Execution-Facade.rkt* werkt als een abstractielaag tussen Infrabel en de interface van de simulator en hardware. Wanneer Infrabel een procedure gebruikt zoals `set-loco-speed!`, dan zal de Execution-Facade de juiste functie oproepen door te kijken naar de boolean sim?. Als sim? #t is dan zal het de procedure uit de simulator geven, anders deze van de hardware.

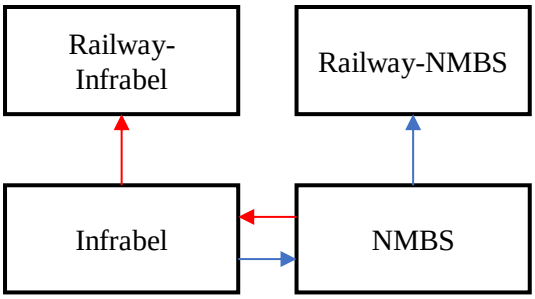


Figuur 2: Werking Execution Facade

- *Railway-graph.rkt* bevat de graf van de spooropstelling en ook procedures om het kortste pad en de buren gemakkelijk te kunnen vragen met de id in plaats van het cijfer van de node te moeten geven.
- Het bestand *helpfunctions.rkt* bevat een aantal hulpfuncties. Het bevat procedures om bijvoorbeeld een trein-id te genereren, om de tijd terug te krijgen, om te testen of een element een wissel is…
- *constants.rkt* bevat constanten die worden gebruikt door de verschillende componenten.
- het bestand *trajectories* bevat de txt-bestanden met trajecten die treinen kunnen uitvoeren.

5. TCP

In figuur 3 kunnen we een schematische weergave zien van de communicatie tussen NMBS en Infrabel. De blauwe pijl toont de communicatie vanuit Infrabel en de rode vanuit NMBS. Wanneer een bericht wordt gestuurd via TCP naar NMBS, dan wordt het eerst geëvalueerd. Daarna zal het ofwel het spoornetwerk updaten ofwel het resultaat terugsturen naar Infrabel. Wanneer een gebruiker het spoornetwerk wijzigt in de GUI dan moet NMBS berichten sturen naar Infrabel om het spoornetwerk van beide componenten synchroon te houden.



Figuur 3: Communicatie tussen NMBS en Infrabel

Infrabel → NMBS	NMBS → Infrabel
`add-train!`	`switch-option!`
`Speed!`	`speed!`
`train-inverse-direction!`	`train-inverse-direction!`
`train-trajectory!`	`train-inverse-speed!`
`dtb-last-train!`	`train-trajectory!`
`train-curr-dtb!`	`process-trajectories!`
`calculate-trajectory`	`calculate-trajectory`
`Switch-position!`	`switch-position!`
`make-detection-blocks!`	
`make-switches!`	
`trajectory!`	

Tabel 8: Berichten tussen NMBS en Infrabel

Meer informatie over deze procedures kan je terugvinden in '4.6 NMBS ADT' en '4.7 Infrabel'. Wanneer we met TCP werken hebben we meestal een client en een server. Voor deze toepassingen is Infrabel de server en NMBS de client. NMBS moet via de GUI controleren en bij elke veranderingen Infrabel hiervoor verwittigen. Infrabel moet de berichten opvangen en de spoornetwerken updaten.

6. Gebruikte bronnen

Onderwerp	Link
`GUI`	https://docs.racket-lang.org/gui/
`Threads`	https://docs.racket-lang.org/reference/threads.html https://docs.racket-lang.org/guide/concurrency.html https://gist.github.com/tfidfwastaken/dbcd525e03b7fb1561342a80f54371a0
`I/O`	https://docs.racket-lang.org/guide/i_o.html
`Date`	https://docs.racket-lang.org/reference/time.html
`Structs`	https://stackoverflow.com/questions/58994448/access-a-struct-field-if-the-struct-is-part-of-a-list

Tabel 9: Gebruikte bronnen