Programmeerproject 2:
Specificatie: Fase 3

Maxim Brabants

0576581

Maxim.Lino.Brabants@vub.be

Academiejaar: 2021-2022

Inhoud

1 Algemene omschrijving

Voor het tweede programmeerproject binnen het academiejaar 2021-2022 wordt er van ons verwacht
dat we een volledig controlesysteem gaan uitbouwen dat zal instaan voor de aansturing van een modelspoor inclusief de treinen die daarover zullen kunnen rijden. Dit alles zal volledig programmatorisch moeten gebeuren met een opsplitsing in enkele onderdelen. Eerst een kort overzicht van welke onderdelen precies aanwezig zijn en welke aspecten per onderdeel van belang zijn :

- Allereerst moet er iets aanwezig zijn dat rechtstreeks met de spoorelementen kan communiceren zodat er op bepaalde momenten een verandering in de spoorsituatie kan plaatsvinden. Hiervoor zal het Command Station (Z21) verantwoordelijk zijn. Deze zal a.d.h.v. een bepaald protocol met de hardware communiceren (deze software is reeds voorzien). Dit station gaan we kunnen aansturen door er van buitenaf commando's naar te sturen.

- Vervolgens hebben we nog een permanent draaiend component dat communiceert met de modelbouw hardware. Die software moet voortdurend actief blijven omdat deze juist de signalen stuurt naar het Command Station. Dit component laten we draaien op onze computer. **(Infrabel)**

- Als allerlaatst hebben we nog een component met een grafische interface die eigenlijk los staat van de logica voor het aansturen van de sporen, maar wel instaat voor het berekenen van bijvoorbeeld trajecten. **(NMBS)**

Vervolgens hebben we nog drie extra vereisten geïmplementeerd bovenop de basisvereisten:

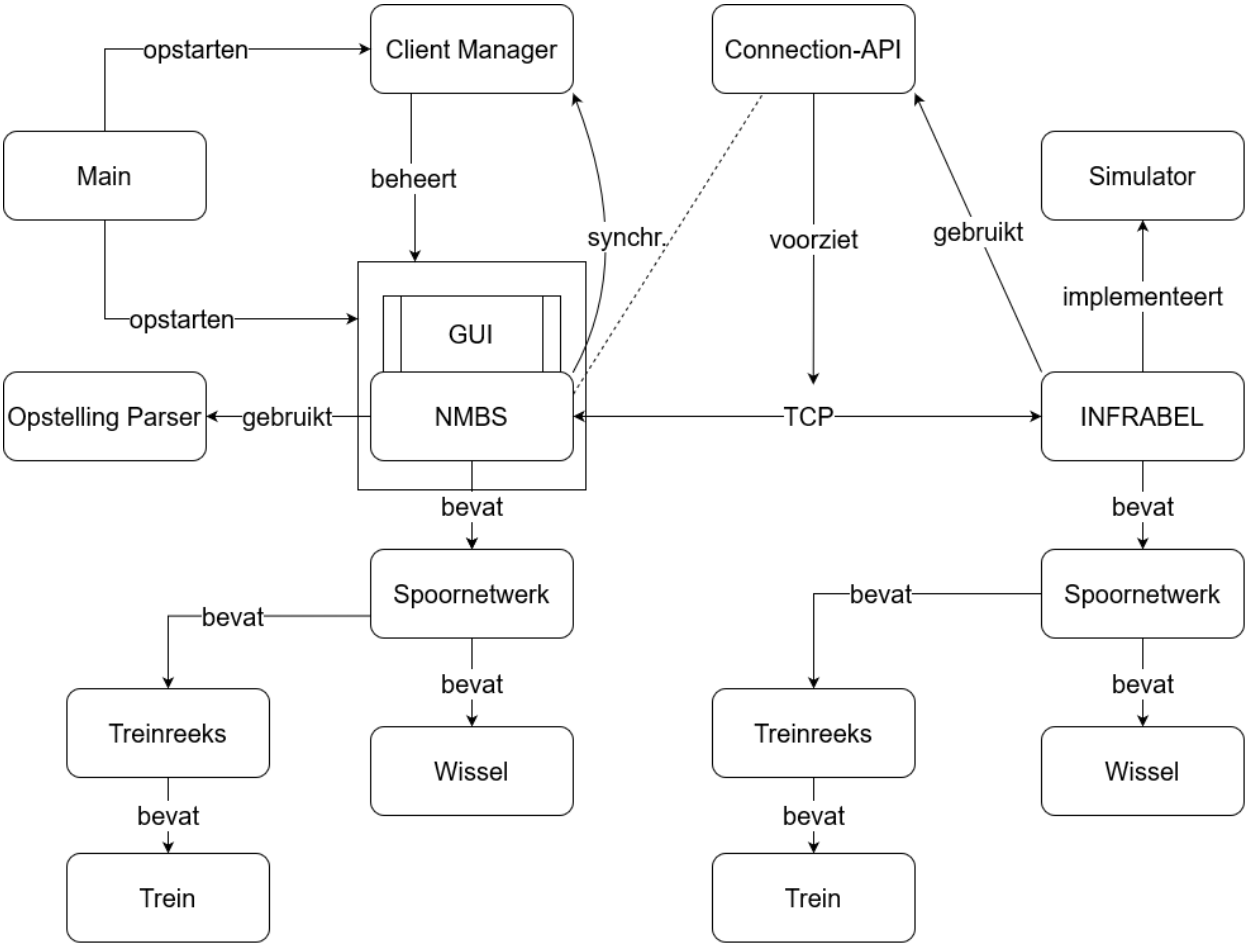
- Functionaliteit die automatische trajectberekening toelaat. Dit maakt het mogelijk om een locomotief vanaf een bepaald detectieblok te laten starten en automatisch te laten rijden tot aan een ander detectieblok.

- Een vereiste die toelaat om andere spooropstellingen in te laten lezen vanuit een tekstbestand. Hiervoor wordt dan een bepaalde format afgesproken die definieert hoe zulk tekstbestand eruit moet zien.

- De gebruiker kan ook meerdere clients toevoegen. D.w.z. een extra NMBS-component met bijhorende grafische gebruikersinterface waarmee elke client de spoorsituatie realtime kan meevolgen in zijn eigen venster. Hierbij moeten alle vensters weliswaar gesynchroniseerd blijven.

2 Software-architectuur

Op de volgende bladzijde volgt een overzicht van de gehele software-architectuur die bij deze toepassing hoort. Hierop is waar te nemen welke componenten er aanwezig zijn en hoe ze aan elkaar gerelateerd zijn. Onder het diagram volgt nog een beschrijving die kort de relaties bespreekt ter verduidelijking.



2.1 Beschrijving relaties

Main - Client Manager - (NMBS + GUI)

De toepassing vertrekt altijd vanuit het Main-bestand zoals te zien is op het diagram. Dit bestand start twee onderdelen op die het verdere verloop van het programma zullen bepalen. Eerst wordt er namelijk een client manager aangemaakt die doorheen de toepassing alle actieve clients gaat bijhouden en ervoor zorgt dat deze gesynchroniseerd blijven in het geval dat er veranderingen optreden in de GUI van een bepaalde client. We zien een client altijd als een NMBS-component gepaard met een grafische gebruikersinterface (GUI). Daarnaast wordt er ook een NMBS-component aangemaakt (samen met een GUI) en hierbij geven we de client manager mee aan NMBS zodat deze op elk moment met elkaar kunnen communiceren. Het diagram hierboven stelt de situatie voor waarbij er maar één client actief is.

Opstelling Parser - (NMBS + GUI)

De NMBS-component bevat dus al de grafische user interface bij aanmaak zoals zichtbaar hierboven. De gebruiker kan vanuit de GUI een tekstbestand ingeven van waaruit de nieuwe opstelling wordt gelezen. Hierbij wordt de naam van het bestand eerst doorgegeven aan NMBS. Deze gaat op zijn beurt de naam doorgeven aan de ‘Opstelling Parser’ die op zijn beurt een input stream gaat openen met het bestand, de gedefinieerde opstelling gaat uitlezen en terug gaat geven aan NMBS zodat NMBS deze kan doorsturen naar INFRABEL.

(NMBS + GUI) - Spoornetwerk

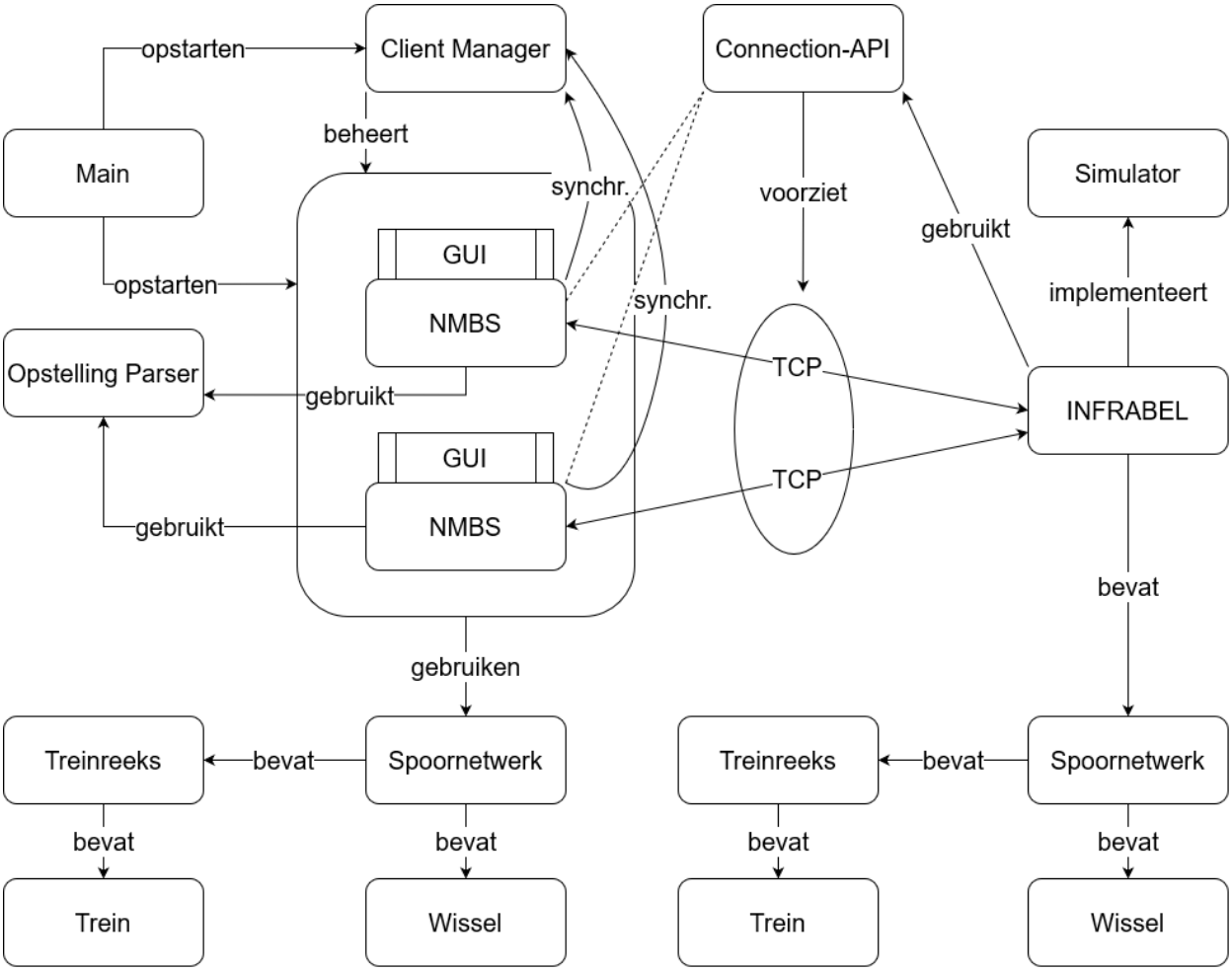
NMBS houdt een eigen voorstelling bij van het spoornetwerk. Alles wat aan de INFRABEL-kant gebeurt, wordt ook toegepast op de versie van het spoornetwerk dat bij NMBS hoort zodat deze op elk moment gesynchroniseerd blijven. Mocht de verbinding tussen NMBS en INFRABEL wegvallen, dan behouden ze beide nog steeds een up-to-date versie van het spoornetwerk. Het spoornetwerk in dit geval houdt de identificatienummers bij van zowel de wissels als de detectieblokken. Ook een reeks van treinen zit hierin vervat met alle treinen die op het spoor aanwezig zijn. Verder worden ook de wissels bijgehouden die horen bij de spooropstelling.

INFRABEL - Simulator - Spoornetwerk

INFRABEL is de component die conceptueel gezien het dichtst bij het spoor ligt en dus rechtstreeks communiceert met de simulator om zo een verandering op het spoor teweeg te brengen. Als we onze toepassing zouden willen laten draaien op de fysieke hardware-opstelling dan kan dat eenvoudig door in INFRABEL de oproepen van de functies afkomstig van de simulator te vervangen door de Z21-bibliotheek. Merk dus op dat INFRABEL net zoals NMBS nauwkeurig de spoorsituatie moet bijhouden (rijdende treinen, wissels, status detectieblokken, …). Het verschil met NMBS is dat INFRABEL effectief de verandering aan het spoor zal doorvoeren.

NMBS - Connection-API - INFRABEL

Op dit moment kunnen we onze toepassing eigenlijk in twee onderdelen splitsen. Een NMBS-component samen met een GUI en INFRABEL, aangezien INFRABEL onafhankelijk moet kunnen draaien weliswaar op onze computer of op een Raspberry Pi, maar in dit diagram is het gemodelleerd op een manier dat het op onze eigen computer draait. Om deze twee Racketprogramma's toch met elkaar te kunnen laten communiceren, hebben we een TCP-verbinding voorzien tussen de twee. Dit wordt mogelijk gemaakt door twee aparte processen te voorzien (één in elke component) die elk op hun eigen de input poort uitlezen. Om TCP-berichten te versturen van de ene component naar de andere is er een API voorzien. Deze definieert alle mogelijke berichten die verzonden kunnen worden. Bij elk bericht wordt ervanuit gegaan dat de output poort wordt meegegeven zodat de API weet voor welke poort het bericht bestemd is. Een belangrijke voorwaarde is dat beide componenten deze API importeren.



Hierboven is het diagram te zien voor de situatie waarbij er meerdere clients actief zijn (in dit geval twee). Er is dus een tweede NMBS component die zijn eigen grafische gebruikersinterface heeft. Elk moeten ze bij iedere verandering in de GUI dit doorgeven aan de client manager zodat deze op zijn beurt de verandering kan doorvoeren voor elke client die actief is. Ze gebruiken ook allemaal gezamenlijk de operaties van de API om zo afzonderlijk te kunnen communiceren met INFRABEL via TCP. In het geval dat er een nieuwe client wordt aangemaakt a.d.h.v. de gebruikersinterface, dan wordt alle nodige info meegegeven aan de client manager zodat de GUI van de nieuwe client gesynchroniseerd is met de andere clients.

3 Beschrijving componenten

3.1 Trein-ADT

Dit type maak ik aan zodat ik later in mijn programma op elk gewenst moment alle treinen die op het spoor aanwezig zijn ter beschikking heb in de vorm van verpakte objectjes waarbij elk objectje van dittype beschikt over zijn unieke identificatienummer, de huidige richting naar waar de trein rijdt, het segment waarin hij start en ten slotte de momentele snelheid die de trein aanneemt.

3.2 Wissel-ADT

Als we gaan kijken naar de wissels die zich op verschillende plekken op het spoor kunnen bevinden, dan moet het mogelijk kunnen zijn om elk van deze wissels individueel te verzetten en de huidige stand op te vragen. Hiervoor hebben we ook twee operaties voorzien die respectievelijk dit doen.

3.3 Treinreeks-ADT

Om alle instanties van het 'Trein'-type bij te kunnen houden doorheen ons programma, maken we een extra type aan die een oneindige reeks van treintypen kan bijhouden. Op die manier kunnen we ongeacht op welk moment een gewenste operatie uitvoeren op een bepaalde trein. Deze treinreeks houden we dan bij in ons spoornetwerk.

3.4 Spoornetwerk-ADT

Van dit type maken we slechts één instantie aan, aangezien we ons focussen op één spooropstelling tegelijk. Deze houdt alle informatie bij die eigen is aan een spooropstelling zoals alle wissels en detectieblokken die behoren tot een bepaalde spooropstelling alsook de treinen die erop rijden, maar initieel bij de aanmaak van een spoornetwerk rijden er logischerwijs nog geen treinen op. Deze kunnen later natuurlijk nog wel toegevoegd worden aan het spoor. We houden in dit type alle wissel –en detectiebloknummers bij.

3.5 INFRABEL

Dit type/ADT is feitelijk een beetje een extra abstractie bovenop het spoornetwerk. Het wordt gebruikt als tussencomponent om onderdelen van het spoornetwerk te manipuleren en/of op te vragen. In onze toepassing is het de component die rechtstreeks de functionaliteit van de simulator oproept en dit is ook de enige component die afhankelijk mag zijn van de simulator. Alle wijzigingen die gemaakt worden met betrekking tot het spoor moeten gecommuniceerd worden met INFRABEL want zoals eerder aangehaald, moet INFRABEL voortdurend kunnen blijven werken met de meest actuele versie van het spoornetwerk in het geval dat de verbinding tussen NMBS en INFRABEL zou wegvallen. We zien hier INFRABEL als de server, aangezien deze het spoornetwerk beheert en wijzigingen kan maken.

3.6 NMBS

Deze component draait op onze computer en deze omvat ook de grafische gebruikersinterface. Het bevat alle operaties die betrekking hebben tot het beheren van treinen, trajectberekening, etc.
Andere operaties die betrekking hebben op een verandering van de spoorsituatie of dergelijke voert NMBS ook uit op zijn eigen versie van het spoornetwerk maar deze dienen ook doorgestuurd te worden naar INFRABEL zodat deze de effectieve verandering kan doorvoeren op het spoor. We zien hier NMBS als een client in de server-client-architectuur, want als we meerdere clients zouden bijmaken dan zou het een serverarchitectuur zijn waarbij de server (INFRABEL) communiceert met meerdere clients.

3.7 Graphical User Interface

Voor de eindgebruiker hebben we een user interface gemaakt die ons toelaat onze functionaliteit te testen op enkele opstellingen in de simulator. Hier krijgen we de simulator met de gekozen spooropstelling te zien en verschijnt ons GUI-venster waarmee we operaties kunnen gaan uitvoeren met betrekking tot die opstelling. Het grafisch venster zal altijd aangepast zijn afhankelijk van de gekozen opstelling.

3.8 Opstelling parser

In de projectmap (op hetzelfde niveau als het GUI-bestand) is er ook nog een bestand 'opstelling-parser' te vinden. Om de overzichtelijkheid en duidelijkheid een beetje te bewaren, is dit bestand aangemaakt om alles te bundelen dat te maken heeft met het inlezen van een zelf gedefinieerde opstelling. Van zodra we van onze GUI de input voor de naam van het tekstbestand hebben opgevangen waarin de opstelling gedefinieerd staat, geven we deze naam door aan NMBS en deze zal de parser oproepen die het verdere werk zal afhandelen. (opstelling uitlezen en teruggeven aan NMBS)

In de parser staat namelijk een functie die een filestream opent met het meegegeven bestand dat overeenkomt met de opgegeven bestandsnaam. Daarna volgt de procedure om effectief het bestand regel per regel uit te lezen. Dit gebeurt volgens een aantal voorwaarden die beschreven staan in de handleiding. De componenten en verbindingen tussen deze worden uiteindelijk meegegeven aan een functie in de simulator die het finale werk zal afronden om effectief de opstelling grafisch weer te geven, Maar voordat dit gebeurt moet de opstelling eerst nog doorgestuurd worden naar INFRABEL.

3.9 Client Manager

Zoals al eerder besproken, is dit de component die alle actieve NMBS-componenten beheert. Bij elke wijziging die gebeurt op één specifieke GUI van een client, wordt toegepast op alle andere clients. Het kan in feite gezien worden als een synchronizer van alle clients. De manier waarop alle NMBS-instanties de client manager kunnen aanspreken is door telkens bij aanmaak van een nieuwe NMBS, de client manager mee te geven als parameter. Als er een nieuwe client wordt toegevoegd via de GUI, dan wordt altijd alle nodige info meegegeven aan de client manager om zo de nieuwe client te kunnen synchroniseren.

3.10 Connection-API

Voordien werd aangehaald dat in zowel de NMBS-component als de INFRABEL-component een proces draait dat voortdurend van de input port leest. Dit zorgt ervoor dat ze constant van elkaar TCP-traffic kunnen ontvangen. De manier waarop ze zelf TCP-berichten kunnen versturen is door deze API aan te spreken. Deze voorziet operaties die voor één van beide bedoeld zijn. Hierbij moet steeds de output port van de bestemmeling meegegeven worden zodat de API op elk moment weet naar welke port hij de traffic moet doorsturen. Een aantal voorbeelden van zulke berichten die verstuurd kunnen worden zijn: het opvragen van de switch-ids, het laten veranderen van een wisselstand, …

Dit zijn alle berichten bedoeld voor INFRABEL (NMBS -> INFRABEL):

- request-switch-ids
- request-detection-block-ids
- request-train-speed
- request-loco-detection-block
- send-train-message
- send-change-switch
- send-change-train-speed
- send-formation

Dit zijn alle berichten bedoeld voor NMBS (INFRABEL -> NMBS):

- send-switch-ids
- send-detection-block-ids
- send-draw-train
- send-draw-train-speed
- send-draw-loco-block