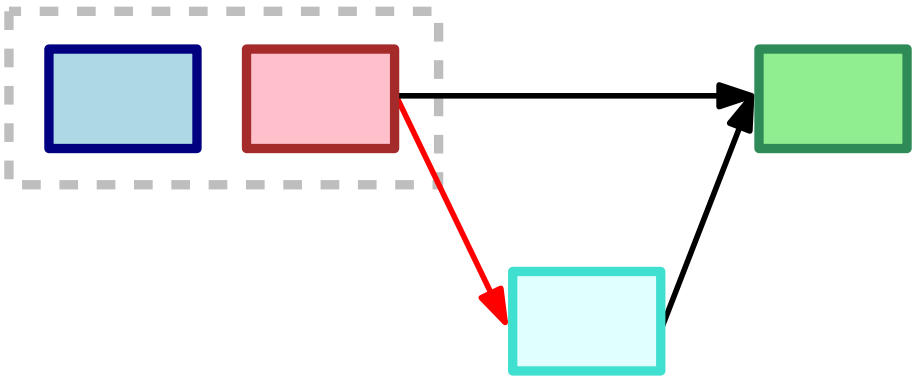
IN3405 BACHELORPROJECT

# Visualisatie en ondersteuning bij planning en scheduling



TU Delft

NedTrain

IN3405 BACHELORPROJECT

# Visualisatie en ondersteuning bij planning en scheduling

**Auteurs**
Erik Ammerlaan
Jan Elffers
Erwin Walraven
Wilco Wisse

**Commissie**

| | |
|---|---|
| prof. dr. Cees Witteveen | Technische Universiteit Delft |
| ir. Michel Wilson | Technische Universiteit Delft |
| ir. Bob Huisman | NedTrain |
| dr. phil. Hans-Gerhard Gross | Technische Universiteit Delft |

**Technische Universiteit Delft**
Faculteit EWI, Delft

**NedTrain**
Fleet Services, Utrecht



4 juli 2012

# Voorwoord

Het verslag dat voor u ligt is het resultaat van het bachelorproject van onze studie Technische Informatica aan de Technische Universiteit Delft. Bij dit project is het de bedoeling om de aangeleerde concepten en ontwikkelmethoden toe te passen in een realistisch project. Hierbij dient het volledige proces van softwareontwikkeling te worden doorlopen. Tevens is het de afsluiting van onze driejarige bachelor.

Gedurende tien weken hebben wij ons beziggehouden met het ontwikkelen van een planningsapplicatie voor NedTrain en de universiteit. Voor NedTrain dient de applicatie als *proof of concept*, om te laten zien wat voor mogelijkheden software allemaal heeft om het plannen van treinonderhoud te verbeteren. De universiteit zal het resultaat gebruiken voor onderzoek naar nieuwe en verbeterde algoritmen.

We willen een aantal mensen bedanken voor de begeleiding tijdens het project. Michel Wilson voor de wekelijkse begeleiding, nuttige suggesties en het meedenken over de applicatie. Bob Huisman voor de suggesties en ideeën vanuit het perspectief van NedTrain. Cees Witteveen voor de feedback bij de tussentijdse voortgangsbijeenkomst en ideeën voor nieuwe functionaliteiten.

Delft, juni 2012
*Erik Ammerlaan*
*Jan Elffers*
*Erwin Walraven*
*Wilco Wisse*

# Samenvatting

Dit bachelorproject wordt uitgevoerd voor NedTrain en de Technische Universiteit Delft. Zij werken samen voor onderzoek naar efficiëntere methoden om treinonderhoud te plannen. Er is inmiddels een algoritme beschikbaar dat gebruikt kan worden om instanties van het schedulingsprobleem van NedTrain op te lossen. Tevens is er een applicatie die gebruikt kan worden om oplossingen visueel weer te geven. Het doel van dit project is het uitbreiden en verbeteren van het bestaande programma. De applicatie zal worden gebruikt bij onderzoek naar nieuwe en verbeterde algoritmen, voor oriëntatie op de mogelijkheden van het gebruik van planningssoftware en bij een summerschool.

Het schedulingsprobleem dat opgelost wordt, bestaat uit een verzameling projecten waarin onderhoud gepleegd moet worden. Zo'n project representeert een trein en heeft een beperkte tijdsduur. Binnen een project zijn er verschillende taken die uitgevoerd moeten worden, waarbij eventueel gebruik wordt gemaakt van resources met eindige capaciteit. Het schedulingsprobleem is een NP-volledig probleem. Bij het oplossen wordt een *greedy* algoritme gebruikt, die met behulp van *heuristieken* bepaalt welke actie ondernomen moet worden. Er kan een voorrangsrelatie worden toegevoegd tussen twee taken of er kunnen taken worden gegroepeerd. Voor het oplossen gebruikt het programma een externe solver.

De ontwikkelde applicatie bevat, vergeleken met de bestaande applicatie, veel nieuwe functionaliteiten. Belangrijke functionaliteiten zijn het tonen van voorrangsrelaties, ondersteuning voor groepen, het simuleren van het oplosproces en het vergelijken van oplossingen. In de solver zijn aanpassingen gedaan om samen te kunnen werken met de applicatie en er zijn verbeteringen doorgevoerd in de code.

De applicatie bestaat uit verscheidene packages, namelijk: `model`, `data`, `controller`, `widgets`, `solve` en `util`. Het model representeert een instantie van het schedulingsprobleem. De widgets zorgen voor de Graphical User Interface van het programma. De controller staat tussen de widgets en het model in en gebruikt dataklassen voor de toegang tot het bestandssysteem en de database. De package `util` bevat hulpmethoden die gebruikt kunnen worden in het model of in de widgets.

Bij de ontwikkeling is gebruikgemaakt van principes uit de *scrum* ontwikkelmethodiek. Hierbij had iedere sprint een tijdsduur van ruim een week. Dagelijks werd er een *sprint meeting* gehouden. Gezien de opzet van het project was het niet mogelijk om alle facetten van *scrum* toe te passen. Voor de testsuite werd gebruikgemaakt van het Google Test framework. De tests werden, samen met andere controles, automatisch uitgevoerd door Jenkins, een platform voor *continuous integration*. Bugs en *scrum* backlogs werden bijgehouden met behulp van Mantis. Voor versiebeheer werden Subversion en Mercurial gebruikt.

De Software Improvement Group (SIG) heeft de code tussentijds beoordeeld. Hierbij werd een score van bijna vier sterren toegekend. De adviezen van SIG zijn, voor zover mogelijk, verwerkt in het programma. Het verlagen van de Component Independence was te ingrijpend om binnen de beschikbare tijd te realiseren.

We zijn tevreden met het uiteindelijke product en het verloop van het project. Wij zijn dan ook van mening dat de gestelde doelen zijn behaald. Met het oog op de toekomst zijn er enkele aanbevelingen gedaan voor zowel de solver als de applicatie.

# Woordenlijst

**Activiteit**: Zie *Taak*.

**Backlog**: In het product backlog worden alle functionaliteiten bijgehouden die tijdens het project ontwikkeld zouden moeten worden. In het sprint backlog staan alle functionaliteiten die in de sprint moeten worden toegevoegd. De functionaliteiten in het backlog worden ook wel doelen genoemd.

**Business-entiteit**: Een specifieke klasse die een onderdeel van het probleemdomein representeert. Het kan hierbij gaan om een document, een fysiek object of een concept dat mensen gebruiken bij hun werk.

**Earliest Starting Time**: De vroegst mogelijke tijd waarop een taak kan plaatsvinden in een geldig schedule.

**Flexibiliteit**: De mogelijkheid om onverwachte omstandigheden te absorberen.

**Greedy**: Een manier waarop een algoritme beslissingen neemt. Hierbij wordt in elke stap de keuze gemaakt die op dat moment het beste is, in de hoop een optimale oplossing te vinden.

**Hard precedence**: Een voorrangsrelatie die door de gebruiker is toegevoegd aan de instantie.

**Heuristiek**: Vuistregel die wordt gebruikt bij het nemen van een beslissing door een algoritme.

**Jenkins CI**: Een *continuous integration* platform voor software ontwikkeling.

**LP-solver**: Een solver die optimaliseringsproblemen kan oplossen door gebruik te maken van *linear programming*.

**Latest Completion Time**: De laatst mogelijke eindtijd van een taak in een geldig schedule. Deze kan worden afgeleid uit de *Latest Starting Time*.

**MVC**: Zie *Model-view-controller model*.

**Mantis**: Bugtracker voor het bijhouden van bugs en het monitoren van sprint backlogs.

**Mercurial**: Een gedistribueerd versiebeheersysteem.

**Model-view-controller model**: Een model waarbij verantwoordelijheden binnen software worden opgedeeld in een datamodel (model), datapresentatie (view) en de logica (controller).

**Observer pattern**: Ontwerppatroon dat wordt gebruikt bij de ontwikkeling van software. Hierbij worden objecten binnen het programma op de hoogte gesteld van toestandsveranderingen bij andere objecten.

**Plannen**: Het identificeren van taken die uitgevoerd moeten worden om een project

te voltooien. Hierbij wordt ook gekeken welke resources er nodig zijn om de taken uit te voeren en welke voorrangsrelaties er tussen taken moeten gelden.

**Planner**: Een persoon die taken identificeert en bepaalt welke resources er nodig zijn. De planner bepaalt uiteindelijk ook de definitieve starttijden van taken en wijst resources toe. Zie ook *Plannen*.

**Precedence**: Zie *Voorrangsrelatie*.

**Project**: Een verzameling van taken die uitgevoerd moeten worden tussen de *release date* en *due date*. Hierbij wordt gebruikgemaakt van de beschikbare resources en worden de onderlinge voorrangsrelaties van taken in acht genomen. Bij NedTrain representeert een project het onderhoud dat aan een trein moet worden uitgevoerd.

**Resource**: Een middel dat nodig is om een taak uit te voeren, bijvoorbeeld een spoor in de werkplaats.

**Resource-conflict**: Een gezamenlijke vraag van taken op een resource, die groter is dan de beschikbare capaciteit op dat moment.

**Resource-profiel**: Het gebruik van een resource over de tijd.

**Scheduling**: Het toekennen van start- en eindtijden aan taken.

**Scrum**: Een methodiek om software te ontwikkelen.

**Soft precedence**: Een voorrangsrelatie die tijdens het oplossen is toegevoegd om een resource-conflict te verhelpen.

**Solver**: Command-line schedulingsprogramma dat als invoer een probleeminstantie krijgt en een schedulingsalgoritme draait om deze op te lossen.

**Sprint**: Een periode van één tot twee weken waarin werkende software wordt ontwikkeld met nieuwe functionaliteiten.

**Subversion**: Een niet gedistribueerd versiebeheersysteem.

**Taak**: Een ondeelbare eenheid werk, die mogelijk één of meer resources nodig heeft en een vaste tijdsduur. Tussen een tweetal taken kan een voorrangsrelatie gelden. Onderdeel van een project.

**Trein**: Een fysiek voertuig dat op het spoor rijdt in reizigersdienst. Indien gebruikt in *scheduling*context, zie *Project*.

**Voorrangsrelatie**: Een relatie tussen twee taken die aangeeft dat de ene taak moet plaatsvinden nadat de andere taak is voltooid.

# Inhoudsopgave

# 1 Inleiding

NedTrain houdt zich bezig met het onderhoud aan treinstellen. Bij het plannen moet worden bepaald wanneer het onderhoud aan de treinen wordt uitgevoerd. In samenwerking met NedTrain doet de Technische Universiteit Delft onderzoek naar het plannen en schedulen hiervan. In het verleden is er een applicatie ontwikkeld die het plannen vereenvoudigt en de ontwikkelde algoritmen gebruikt om een gemaakte planning te visualiseren.

In dit verslag wordt het verloop van het bachelorproject beschreven, waarbij een bestaande planningsapplicatie is uitgebreid en verbeterd. Hierbij wordt aandacht besteed aan de achtergrond van het project, de werkwijze en het resultaat. Hoofdstuk 2 beschrijft de achtergrond van het probleem en geeft een korte analyse van de eisen die worden gesteld aan een oplossing. Vervolgens wordt er in het derde hoofdstuk aandacht besteed aan het schedulingsprobleem dat centraal staat bij NedTrain, het onderzoek en in de ontwikkelde software. Hoofdstuk 4 en 5 geven een overzicht van de functionaliteiten van de opgeleverde applicatie en het ontwerp. In hoofdstuk 6 wordt een overzicht gegeven van het ontwikkelproces. Ten slotte bevat hoofdstuk 8 een conclusie en enkele aanbevelingen voor de toekomst.

# 2 Probleemstelling en analyse

Dit bachelorproject wordt uitgevoerd voor NedTrain. NedTrain is als dochteronderneming van NS Groep verantwoordelijk voor het onderhoud aan treinmaterieel. NedTrain werkt samen met de Technische Universiteit Delft om het onderhoud aan treinstellen zo efficiënt mogelijk te schedulen. Momenteel wordt dit voor een deel handmatig uitgevoerd.

In de Algoritmiekgroep, die onderdeel is van de afdeling Software and Computer Technology, wordt onder andere onderzoek gedaan naar schedulingsproblemen. Voor het oplossen van het schedulingsprobleem waar NedTrain mee te maken heeft, is er een algoritme geïmplementeerd door afstudeerder Ronald Evers [4]. Deze implementatie is gebaseerd op het werk van Cesta, Oddi en Smith [2, 3]. Later zijn er een aantal verbeteringen doorgevoerd door promovendus Michel Wilson. Tevens is er een eenvoudige planningsapplicatie, die de mogelijkheid biedt om onderhoud te plannen. De applicatie kan vervolgens een toekenning van starttijden aan taken laten bepalen door een extern programma. De opdracht bij dit bachelorproject omvat het verbeteren van het bestaande programma en het uitbreiden ervan met nieuwe functionaliteiten voor onderzoeksdoeleinden en voor NedTrain.

Een van de belangrijkste ontwikkeldoelen is het inzichtelijk maken aan de gebruiker welke keuzes worden gemaakt tijdens het vinden van een oplossing voor het schedulingsprobleem. Hierbij kan gedacht worden aan het tonen van de stappen die een algoritme zet om een oplossing te verkrijgen. De planner op het bedrijf moet bijvoorbeeld kunnen beargumenteren hoe hij tot een bepaald rooster komt; hij moet niet alleen beslissingen nemen omdat een programma het zegt. Als de planner inzicht krijgt in het oplosproces, zal het ook eenvoudiger zijn om tot een nieuwe oplossing te komen als er onverwachte wijzigingen optreden. Het toevoegen, verwijderen en tonen van relaties tussen taken speelt eveneens een belangrijke rol. Er zijn ook nieuwe ontwikkelingen op onderzoeksgebied, zoals het groeperen van (onderhouds)taken. Hiervoor dient het programma ondersteuning te bieden. Een compleet overzicht van requirements is te vinden in het bijgesloten Requirements Analysis Document.

De ontwikkelde applicatie zal voor verschillende doeleinden gebruikt worden. Ten eerste wil NedTrain een applicatie hebben om te oriënteren op de mogelijkheden van het gebruik van planningssoftware. Ten tweede zal de applicatie gebruikt worden voor onderzoek naar nieuwe en verbeterde algoritmen. Daarnaast heeft het ontwikkelde programma een educatief doeleinde. Het zal tijdens een summerschool worden gebruikt. Hierbij komen onder andere verschillende aspecten van planning en scheduling aan de orde. De applicatie zal bijdragen aan het tonen van verschillende aspecten van schedulingsproblemen.

# 3 Schedulingsprobleem

Dit hoofdstuk beschrijft het schedulingsprobleem dat centraal staat bij dit project. Eerst wordt uitgelegd hoe het probleem in de praktijk optreedt, in het bijzonder bij NedTrain. Vervolgens wordt er een formele definitie gegeven en wordt er aandacht besteed aan de methode die wordt gebruikt om het probleem op te lossen. In het resterende deel van dit verslag, en in het ontwikkelde programma, komen de concepten uit dit hoofdstuk regelmatig aan de orde.

## 3.1 Achtergrond

Over de begrippen *planning* en *scheduling* bestaat de nodige verwarring. Het Engelse werkwoord scheduling wordt door Van Dale vertaald als plannen, terwijl dit eigenlijk twee verschillende concepten zijn. Onder *plannen* verstaan we het identificeren van activiteiten die uitgevoerd moeten worden om een project te voltooien. Tijdens dit proces worden er echter nog geen specifieke tijden toegekend. De start- en eindtijden van de activiteiten worden pas bepaald in een volgende stap, genaamd *scheduling*. Planning is dus vereist voordat scheduling plaatsvindt, waardoor deze begrippen zeer nauw samenhangen.

Bij NedTrain is plannen een belangrijke stap voordat er onderhoud gepleegd kan worden aan treinen. Iedere trein wordt beschouwd als een project waarin onderhoud gepleegd moet worden. Er is per trein slechts een beperkte periode om het onderhoud uit te voeren – een interval begrensd door een *release date* en een *due date*. Na deze periode gaat een trein bijvoorbeeld weer in dienst. Het onderhoud bestaat uit verschillende taken. Hierbij kan bijvoorbeeld gedacht worden aan een controlebeurt of een reparatie aan een elektrische installatie. Deze taken kunnen afhankelijk zijn van het gebruik van een resource. Een resource is een middel met een eindige capaciteit, die gebruikt wordt tijdens het uitvoeren van een taak. Hierbij kan gedacht worden aan een spoor dat beschikbaar moet zijn in de werkplaats of een aantal monteurs dat aan een trein moet werken. Een planner identificeert alle taken die uitgevoerd moeten worden aan treinen en welke resources daarbij nodig zijn. Vervolgens moeten aan alle taken specifieke start- en eindtijden worden toegekend, zodanig dat alles binnen de beschikbare tijd wordt uitgevoerd en er geen resource-capaciteit wordt overschreden.

Omdat er tijdens het uitvoeren van een schedule onverwachte omstandigheden kunnen optreden, is het wenselijk dat de uitvoerder van de schedule hier beter op kan inspelen. Bij het onderzoek van de Technische Universiteit Delft worden er methoden ontwikkeld die gebruikt kunnen worden bij het oplossen van dit soort problemen. Hierbij wordt het bestaande probleem gegeneraliseerd tot een bekende, meer algemene, vorm. Deze vorm van het schedulingsprobleem wordt in de volgende paragraaf gedefinieerd.

## 3.2 Definities

Het schedulingsprobleem wordt gemodelleerd als het *Resource Constrained Multi-Project Scheduling Problem* (RCMPSP), waarbij ieder project een trein representeert. Bij het vak IN3305 Bachelorseminarium hebben we uitgebreid gekeken naar het *Resource Constrained Project Scheduling Problem* (RCPSP) [1], waarbij er slechts één project is. Het

RCMPSP kan worden gezien als een uitbreiding van het RCPSP [5], en een probleeminstantie van het RCMPSP kan worden omgezet naar een probleeminstantie van het RCPSP. De onderstaande definities maken duidelijk waar het probleem precies om draait.

**Definitie 3.2.1** (Voorrangsrelatie)**.** *Een voorrangsrelatie $a_i \prec a_j$ geeft aan dat de eindtijd van taak $a_i$ in alle gevallen moet plaatsvinden voor de starttijd van taak $a_j$. Met andere woorden: $s_j \geq s_i + d_i$, waarbij $s_i$ de starttijd van taak $i$ representeert en $d_i$ de tijdsduur van taak $i$.*

**Definitie 3.2.2** (Resource Constrained Multi-Project Scheduling Problem)**.** *Het Resource Constrained Multi-Project Scheduling Problem bestaat uit een verzameling resources $R$ en een verzameling projecten $P$, waarbij ieder project $p \in P$ bestaat uit een verzameling taken $A$. Iedere taak $a_i \in A$ wordt uitgevoerd met nul of meer resources $r_k \in R$, waarvan er per resource een bekend aantal eenheden $req(a_i, r_k)$ nodig is. Alle resources $r_k \in R$ hebben een eindige capaciteit $cap(r_k)$.*

## 3.3 Oplosmethode

Het schedulingsprobleem dat opgelost moet worden is een NP-volledig probleem [4]. Bij het vinden van een oplossing moet er rekening mee worden gehouden dat het gelijktijdig plaatsvinden van taken ertoe kan leiden dat resource-capaciteiten worden overschreden. In die gevallen moet er worden afgedwongen dat de taken plaatsvinden op disjuncte tijdsintervallen. Het vinden van een oplossing bestaat uit het toekennen van starttijden aan taken, zodanig dat de resource-capaciteiten niet worden overschreden en voorrangsrelaties, *release dates* en *due dates* in acht worden genomen:

- voor iedere resource is het benodigde aantal eenheden op tijdstip $v$ kleiner dan of gelijk aan de capaciteit:

$$\sum_{a_i:\ s_i \leq v < f_i} req(a_i, r_k) \leq cap(r_k);$$

- voor alle voorrangsrelaties ($a_i \prec a_j$) geldt:

$$s_j \geq s_i + d_i;$$

- voor iedere taak geldt dat de start- en eindtijd plaatsvindt op het tijdsinterval begrensd door de *release date* en *due date* van het bijbehorende project.

Bij het oplossen van dit probleem wordt gebruikgemaakt van *Precedence Constraint Posting* [1]. Hierbij wordt er gezocht naar taken met een conflicterend resource-verbruik. Vervolgens wordt er een voorrangsrelatie toegevoegd om dit te verhelpen. Het doel is dat de verzameling voorrangsrelaties afdwingt, dat een oplossing die de voorrangsrelaties in acht neemt, geen overschrijding van resource-capaciteiten heeft. Het is mogelijk dat dit niet geldt voor elke oplossing die aan de voorrangsrelaties voldoet. Dit hangt af van de methode die wordt gebruikt. De methode die op dit moment wordt gebruikt, garandeert alleen dat de *Earliest Starting Time* toegelaten is. Hierbij wordt aan iedere taak de vroegst mogelijke starttijd toegekend.

In plaats van het toevoegen van een voorrangsrelatie, kan er ook worden gekozen voor het groeperen van twee taken [6]. In deze groepen mogen taken niet tegelijkertijd plaatsvinden, zodat er geen conflicterend resource-verbruik kan zijn binnen een groep. Het

toevoegen van voorrangsrelaties en groepen gebeurt op een *greedy* manier en er worden *heuristieken* gebruikt om te bepalen welke actie er precies wordt ondernomen. Dit betekent dat er niet altijd een oplossing wordt gevonden, ook niet noodzakelijkerwijs als er wel een oplossing bestaat. Het toevoegen van voorrangsrelaties en het groeperen van taken wordt verder toegelicht in de volgende paragrafen.

Het probleem wordt tijdens het oplossen gerepresenteerd als een *Simple Temporal Network* (STN). Dit model maakt het mogelijk om voorrangsrelaties tussen taken te definiëren. Hoe het schedulingsprobleem als STN wordt gerepresenteerd wordt niet in dit verslag behandeld. Meer informatie hierover is te vinden in onze paper *Precedence Constraint Posting for Robust Scheduling* [1]. In deze paper wordt ook de techniek *Precedence Constraint Posting* uitgebreider behandeld.

### 3.3.1 Toevoegen van voorrangsrelaties

Om resource-conflicten te voorkomen kunnen er voorrangsrelaties worden toegevoegd. Stel dat de taken $a_1$ en $a_2$, temporeel gezien, tegelijkertijd kunnen plaatsvinden en samen een resource $r_1$ nodig hebben. Als resource $r_1$ capaciteit één heeft, dan zorgt het gelijktijdig plaatsvinden van $a_1$ en $a_2$ voor een conflict. Dit is bijvoorbeeld op te lossen door een voorrangsrelatie $a_1 \prec a_2$ toe te voegen, mits temporele consistentie behouden blijft. Hiermee wordt bedoeld dat er nog steeds een oplossing bestaat die de voorrangsrelaties en de tijdsintervallen waarbinnen de taken van de projecten moeten worden uitgevoerd, in acht neemt. Dit is niet noodzakelijkerwijs het geval. Stel dat $a_1$ en $a_2$ samen een tijdsduur hebben die groter is dan de beschikbare tijd van het project, dan is het niet mogelijk om de voorrangsrelatie $a_1 \prec a_2$ toe te voegen, omdat dit de tijdsduur van het project zou overschrijden.

### 3.3.2 Groeperen van taken

Bij het groeperen van taken worden taken toegekend aan een groep, waarbinnen taken niet tegelijkertijd mogen worden uitgevoerd. De volgorde van taken binnen een groep is niet van belang. Dit betekent dat iedere mogelijke volgorde van taken binnen een groep een geldige oplossing oplevert. De definitieve volgorde kan tijdens het uitvoeren van de schedule worden bepaald. Omdat taken niet gelijktijdig plaats mogen vinden binnen een groep, kan een groep van taken gebruikt worden om een resource-conflict te verhelpen. Tevens zorgt deze techniek ervoor dat er beter kan worden ingespeeld op onverwachte omstandigheden zoals het uitlopen van taken.

## 3.4 Solver

Om een probleeminstantie op te lossen wordt gebruikgemaakt van een solver. Deze is gemaakt door Michel Wilson, één van onze begeleiders. Bij het oplossen wordt gebruikgemaakt van de besproken technieken.

### 3.4.1 Invoerformaat

Voor het invoerformaat van de solver is een grammatica gedefinieerd. De onderstaande regels tekst specificeren entiteiten zoals projecten, taken en resources.

- `'R' <resource_id:int> <capacity:int> <name:string>`: Specificatie van een resource.
- `'J' <trein_id:int> <release_date:int> <due_date:int> <name:string>`: Specificatie van de trein (project).
- `'A' <trein_id:int> <taak_id:int> <duration:int> <name:string>`: Specificatie van een taak voor een trein.
- `'Q' <trein_id:int> <taak_id:int> <resource_id:int> <amount:int>`: Specificatie dat een taak een bepaald verbruik bij een resource heeft.
- `'P' <trein1_id:int> <taak1_id:int> <trein2_id:int> <taak2_id:int>`: Specificatie van een voorrangsrelatie tussen twee taken. Dit betreft een constraint die is toegevoegd door de planner en deze wordt aangeduid met *hard precedence constraint*.

Opties kunnen worden meegegeven als argument. Om ervoor te zorgen dat taken binnen een project niet gelijktijdig plaatsvinden moet het argument `-x` meegegeven worden. Groepering kan worden toegepast door `-m` mee te geven, direct gevolgd door een drempelwaarde [6].

Voor bepaalde functionaliteit van de planningsapplicatie was het nodig om de input van de solver aan te passen. De nieuwe versie van de solver bevat, naast de bovengenoemde entiteiten, de onderstaande regels. Hierbij zijn enkele bestaande regels uitgebreid met nieuwe, optionele, parameters.

- `'A' <trein_id:int> <taak_id:int> <duration:int> <name:string> <start:int>`: Specificatie van een taak voor een trein, met starttijd. Hierbij is de starttijd optioneel. De starttijd (het veld `start`) wordt door de solver genegeerd en is voor persistentie in de application.
- `'A' <trein_id:int> <taak_id:int> <duration:int> <name:string> <start:int> <earliest_starttime:int> <latest_starttime:int>`: Specificatie van een taak voor een trein, met starttijd en interval [EST, LST], dat de mogelijke starttijden die de activiteit kan krijgen begrenst. De starttijd en het interval zijn optioneel. De starttijd (het veld `start`) wordt door de solver genegeerd en is voor persistentie in de application.
- `'D' <trein_id:int> <release_date:int> <due_date:int> <name:string>`: Specificatie van een dummytrein. Een dummytrein wordt door de solver behandeld als een normale trein. In de planningsapplicatie wordt het echter gebruikt om de capaciteit van een resource op een bepaald interval bezet te houden. Hiermee wordt resource-verlaging gerealiseerd.
- `'S' <trein1_id:int> <taak1_id:int> <trein2_id:int> <taak2_id:int>`: Specificatie van een voorrangsrelatie tussen twee taken. Dit betreft een constraint die is toegevoegd door de solver voor het oplossen van het scheduling probleem. Deze constraints worden aangeduid met *soft precedence constraints*. Door onderscheid te maken tussen hard precedence constraints en soft precedence constraints is het mogelijk om bij het solven ervoor te kiezen of de eerder gegenereerde constraints verwijderd moeten worden.
- `'F' <trein_id:int> <taak_id:int> <flex:int>`: Specificeert de toegevoegde tijdsduur voor een taak. Dit wordt gebruikt wanneer extra flexibiliteit gewenst is.

- `'G' <trein_id1:int> <taak_id1:int> <trein_id2:int> <taak_id2:int>`: Specificatie van een merge-actie, waarbij de tweede gespecificeerde taak wordt toegevoegd aan de groep die de eerste gespecificeerde taak representeert.

### 3.4.2 Uitvoerformaat

Behalve debug-output bestaat de output van de solver uit de volgende typen regels:

- `PROGRESS`: geeft een indicatie (in procenten) van de voortgang van het solveproces. De syntax is `'PROGRESS:' <percentage>`.
- `ERROR`: geeft bericht van een interne fout in de solver (een invariant tijdens het solve proces blijkt niet waar te zijn). Dit zou niet mogen optreden.
- Een bericht of de instantie succesvol is opgelost. Dit is een regel `Instance solved.` of `Instance not solved.`.
- `PC`: duidt aan dat een precedence constraint gepost wordt. De syntax is als volgt: `'PC:' <i1> <j1> <i2> <j2>`, waarbij `<i1> <j1>` de voorgaangde taak aanduidt, en `<i2> <j2>` de opvolgende taak.
- `STATE`: Geeft een beschrijving van de huidige verzameling groepen (partitie in groepen van de taken) in de solver en de huidige EST en LST van elke groep. De syntax is `'STATE:' <groepen> '-1'`, waarbij `-1` als scheidingsteken wordt gebruikt. Het `<groepen>` blok bevat de specificaties van groepen.
  Specificatie per groep begint met `<treinID> <EST> <LST> <#taken>`.
  Daarna volgen `#taken` taken, aangeduid met `<projectID> <taakID>`.
- `PEAK`: Deze regel wordt geprint als het solven niet gelukt is, omdat er aan het einde een contention peak was die niet geresolved kon worden. Het formaat is `'PEAK:'<time> <resource> <capacity> <lijst taken en groepen> '-1'`. Eerst wordt aangegeven waar de peak zich bevindt en de capaciteit. Elke taak of groep wordt gerepresenteerd met `<treinID> <taakID>`. Voor groepen geldt dat een van de taken die in de groep zitten wordt gegeven als *representant*. Vanuit de laatste `STATE` kan worden afgeleid welke groep bedoeld wordt. Deze peak kan door het planprogramma worden gevisualiseerd.
- `MUTEX`: Deze regel wordt geprint als het solven niet gelukt is, omdat er niet kon worden voldaan aan de eis dat wederzijdse uitsluiting moet gelden. Het formaat is hetzelfde als bij `PEAK`.

## 3.5 Voorbeeld

In deze paragraaf zal er een kort voorbeeld worden gegeven om de besproken technieken te illustreren. Er zijn twee monteurs beschikbaar en er zullen twee treinen binnenkomen waarop onderhoud zal worden uitgevoerd.

- Trein 9426 komt binnen op tijdstip 10 en zal vertrekken op tijdstip 50. Er moeten wielen gecontroleerd worden. Dit duurt 15 tijdseenheden en hierbij zijn er twee monteurs nodig. Er moet een lamp worden vervangen gedurende 8 tijdseenheden. Hierbij is er één monteur nodig. Ook moet er een deur worden gerepareerd door

twee monteurs. Dit duurt 2 tijdseenheden. Bij al het onderhoud moet er één spoor beschikbaar zijn.

- Trein 2615 komt binnen op tijdstip 20 en zal vertrekken op tijdstip 45. Bij deze trein moet er alleen een controle van de wielen plaatsvinden. Dit duurt 12 tijdseenheden en hierbij is er één monteur nodig en één spoor.

## Oplossen van de instantie

In de onderstaande lijst is te zien welke stappen gezet kunnen worden om deze instantie op te lossen.

1) Bij trein 9426 kan het vervangen van de lamp en het controleren van de wielen worden gegroepeerd. Dit betekent dat de volgorde waarin dit onderhoud wordt gedaan, bepaald kan worden tijdens het uitvoeren.

2) Omdat er slechts een beperkt aantal monteurs beschikbaar is, mag het controleren van de wielen bij trein 2615 niet plaatsvinden tijdens het onderhoud binnen de groep van trein 9426. Dit kan worden afgedwongen door een voorrangsrelatie toe te voegen, die aangeeft dat de controle bij trein 2615 moet plaatsvinden nadat alle taken binnen de groep van trein 9426 klaar zijn.

3) Om dezelfde reden als bij het voorgaande punt, mag de reparatie van een deur ook niet gelijktijdig met de groep plaatsvinden. Ook dit probleem kan worden verholpen door het toevoegen van een voorrangsrelatie.

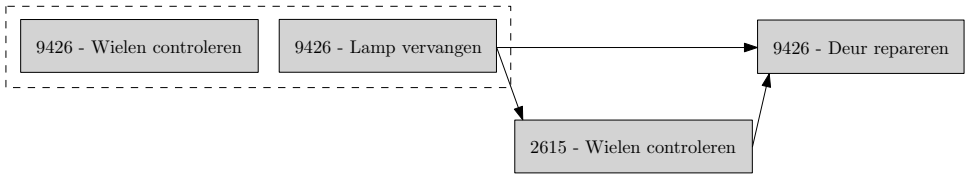
4) Er is nu alleen nog een conflict tussen de reparatie van de deur bij trein 9426 en de wielcontrole bij trein 2615. Ook dit kan worden opgelost door het toevoegen van een voorrangsrelatie. Vanwege de beperkte tijd voor trein 2615 zal de controle van de wielen als eerste moeten plaatsvinden.

De toegevoegde groepen en voorrangsrelaties leiden tot de onderstaande oplossing.

| **Treinnummer** | **Onderhoud** | **Tijdsduur** | **Tijdstip** |
|---|---|---|---|
| 9426 | Wielen controleren | 15 | 10 |
| 9426 | Lamp vervangen | 8 | 25 |
| 9426 | Deur repareren | 2 | 45 |
| 2615 | Wielen controleren | 12 | 33 |

Merk op dat de taken binnen de gemaakte groep ook in een andere volgorde mogen plaatsvinden. In dat geval start de wielcontrole bij trein 9426 op tijdstip 18 en het vervangen van de lamp op tijdstip 10. Verder zijn er meerdere oplossingen die voldoen aan de toegevoegde groepen en voorrangsrelaties. De reparatie van de deur van trein 9426 zou bijvoorbeeld ook later mogen plaatsvinden, zolang dit klaar is voordat de trein de werkplaats moet verlaten.

Een schematische representatie van de oplossing is te zien in figuur 3.1. De pijlen representeren een voorrangsrelatie, waarbij de taak waar de pijl naartoe wijst als laatst moet worden uitgevoerd. Het kader om de taken representeert een groep.




Figuur 3.1: Een schematische weergave van de oplossing.

## Opgeslagen instantie

De opgeslagen instantie zou er, na het oplossen, als volgt uit kunnen zien. De toegekende ID's mogen anders zijn, mits alle relaties tussen resources, treinen (projecten) en taken nog steeds in acht worden genomen.

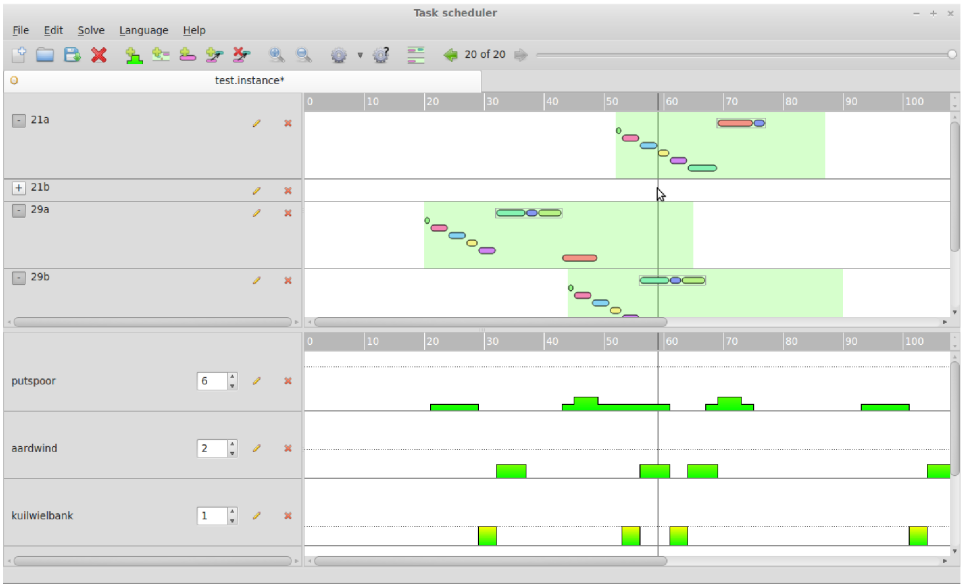
```
R 0 2 "Monteur"
R 1 1 "Spoor"

J 0 10 50 "9062"
A 0 0 15 "Wielen controleren" 10
Q 0 0 0 2
Q 0 0 1 1
A 0 1 8 "Lamp vervangen" 25
Q 0 1 0 1
Q 0 1 1 1
A 0 2 2 "Deur repareren" 45
Q 0 2 0 2
Q 0 2 1 1

J 1 20 45 "2615"
A 1 0 12 "Wielen controleren" 33
Q 1 0 0 2
Q 1 0 1 1

S 0 0 1 0
S 0 0 0 2
S 1 0 0 2
G 0 0 0 1
```

# 4 Opgeleverd product

In dit hoofdstuk wordt het opgeleverde product beschreven. Er wordt hierbij een beschrijving van het programma gegeven. Vervolgens wordt toegelicht welke functionaliteiten er aan het oorspronkelijke programma zijn toegevoegd.

## 4.1 Planningsapplicatie en solver

Het opgeleverde product bestaat uit een planningsapplicatie en een aangepaste versie van de solver. De planningsapplicatie is een uitbreiding van een reeds bestaand eenvoudig programma. De solver is ontwikkeld door één van de begeleiders voor het onderzoek naar oplosmethoden voor het schedulingsprobleem. Om goed samen te werken met de planningsapplicatie zijn er bugs opgelost en verbeteringen aangebracht. De planningsapplicatie is universeel en kan ook gebruikt worden met andere solvers. Voorwaarde is wel dat de solver het juiste in- en uitvoerformaat hanteert. Dit formaat staat beschreven in paragraaf 3.4. Er zijn functionaliteiten binnen de planningsapplicatie die nog niet volledig ondersteund worden door de solver. Dit wordt toegelicht in het resterende deel van dit hoofdstuk en in hoofdstuk 8. Het toevoegen van deze ondersteuning valt buiten het bereik van dit project.




Figuur 4.1: Het hoofdscherm van de applicatie.

In figuur 4.1 is het hoofdscherm van de applicatie te zien, waarin een instantie is ingeladen. Het bovenste deel van het scherm bevat alle projecten, met daarin taken. De projecten worden gerepresenteerd door een horizontale balk, waarbij het groene deel het interval tussen *release date* en *due date* representeert. De gekleurde blokjes zijn de taken die binnen een project uitgevoerd moeten kunnen worden. Het onderste deel van het scherm bevat de resources. Per resource staat het resource-profiel ingetekend. Dit laat zien hoeveel eenheden van de resource worden gevraagd door de schedule die wordt getoond in het bovenste deel van het scherm. Zodra een deel van het resource-profiel

rood is, betekent dit dat er resource-capaciteiten worden overschreden. De functionaliteiten van het programma zijn toegankelijk via het menu en de werkbalk. Bewerkingen op taken kunnen worden gedaan door te klikken of te slepen.

## 4.2 Basisfunctionaliteiten

Deze sectie beschrijft de basisfunctionaliteiten die al beschikbaar waren in het oorspronkelijke programma van Ronald Evers. In de nieuwe versie van het programma zijn deze functionaliteiten nog steeds aanwezig, soms echter in aangepaste vorm.

### Inladen van instanties

De instanties die opgelost kunnen worden door de solver zijn opgeslagen in een tekstbestand. Er is een contextvrije grammatica die de vorm beschrijft waarin de instantie wordt opgeslagen in het tekstbestand. De applicatie kan een instantie in deze vorm ook inlezen, zodat de bijbehorende taken, projecten en resources ingeladen kunnen worden in het programma.

### Oplossen instantie

Een instantie kan worden opgelost door een externe solver, die kan worden ingesteld via een configuratiescherm. Zodra de instantie opgelost moet worden, geeft het programma de instantie door aan de solver, die vervolgens als output aangeeft wat de oplossing is. Als er geen oplossing gevonden kan worden, wordt dit ook aan het programma doorgegeven. Zodra er een oplossing is, worden de taken in het programma op de juiste posities gezet, zodanig dat er geen resource-capaciteiten worden overschreden.

### Handmatig invoeren taken, projecten en resources

Instanties kunnen worden ingeladen uit een bestand, maar zijn ook handmatig in te voeren. Hierbij zijn er opties om taken, projecten en resources toe te voegen. Deze functionaliteit is echter zeer beperkt. Verwijderen na het toevoegen is bijvoorbeeld niet mogelijk.

## 4.3 Toegevoegde functionaliteiten

Deze paragraaf beschrijft alle functionaliteiten die zijn toegevoegd aan het bestaande programma. Per functionaliteit wordt toegelicht wat het precies betekent en waar het te vinden is in het programma.

### Aanpassen van een instantie

Het oorspronkelijke programma bood onvoldoende mogelijkheden om een instantie aan te passen. Dit zorgde voor problemen wanneer de gebruiker een fout maakte bij het

toevoegen van een project, taak of resource. Dit probleem is opgelost door overal verwijderopties toe te voegen en bewerkschermpjes verder uit te breiden, zodat de onderliggende instantie correct wordt aangepast.

### Ondersteuning voor meerdere talen

Het programma zal worden gebruikt voor verschillende doeleinden. Onderzoekers kunnen het gebruiken bij het ontwikkelen van nieuwe algoritmen en zijn op dat moment niet specifiek bezig met treinonderhoud. De begrippen die in het programma worden gebruikt, mogen in dat geval dus algemeen zijn. Voor planners bij NedTrain is het belangrijk dat de terminologie in het programma in overeenstemming is met de werkomgeving. De ingebouwde ondersteuning voor meerdere talen zorgt ervoor dat de gebruiker zelf het gewenste taalbestand kan kiezen. De vertalingen die worden meegeleverd zijn Engels en Nederlands, waarbij de Nederlandse versie gericht is op het plannen bij NedTrain. Deze functionaliteit is ingebouwd met het Qt Internationalization framework.

### Sjablonen voor taken

Het toevoegen van taken aan een project kan een tijdrovende klus zijn. Als dezelfde taak bij meerdere projecten moet worden toegevoegd, zal dit voor ieder project herhaald moeten worden. Om deze vaak voorkomende handeling te vereenvoudigen is er een mogelijkheid om sjablonen op te slaan. Een sjabloon bevat alle informatie die nodig is om een taak meerdere keren te kunnen toevoegen, zonder dat alles opnieuw ingevoerd moet worden door de gebruiker. Deze functie zou bij NedTrain bijvoorbeeld gebruikt kunnen worden om een specifiek type taak toe te voegen bij meerdere treinen. Sjablonen kunnen worden aangepast via een optie in het menu.

### Verbeterde gebruiksvriendelijkheid

De gebruiksvriendelijkheid van het programma is verbeterd. Om vaak uitgevoerde handelingen te vereenvoudigen is een werkbalk toegevoegd met duidelijke iconen. Ook worden waarschuwingen en problemen nu duidelijk gemeld, zodat de gebruiker precies weet wat er aan de hand is. Functionaliteiten die in de huidige toestand van het programma niet beschikbaar zijn, worden automatisch uitgeschakeld in de menu's en de werkbalk.

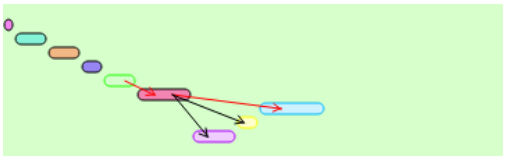
### Inzichtelijk maken van taakgegevens

Gegevens van taken waren in het oorspronkelijke programma alleen op te vragen door naar het bewerkscherm te gaan. Dit is in de praktijk onhandig, omdat je geen snel overzicht kunt krijgen van alle bestaande taken. Daarom is er aan elke taak een tooltip met de naam toegevoegd. Ook is er een optie toegevoegd om de eigenschappen van een taak op te vragen zonder die te bewerken. Dit is toegankelijk via de rechtermuisknop.

### Voorrangsrelaties van taken tonen

Door op een taak te klikken worden de betrokken voorrangsrelaties met andere taken getekend als een pijl of lijn. Een voorbeeld is te zien in figuur 4.2. Het is handig

om te zien welke voorrangsrelaties zijn toegevoegd tijdens het oplossen en welke voorrangsrelaties door de gebruikers zelf zijn toegevoegd. Er wordt onderscheid gemaakt tussen beide gevallen door een afwijkende kleur te gebruiken. Rood representeert een voorrangsrelatie die is toegevoegd door de gebruiker en zwart representeert een voorrangsrelatie die tijdens het oplossen is toegevoegd.



Figuur 4.2: Het tonen van voorrangsrelaties.

### Handmatig toevoegen van voorrangsrelaties

In sommige gevallen is het wenselijk om vooraf te specificeren dat een bepaalde taak in alle gevallen moet plaatsvinden voor een andere taak. In zo'n geval kan de gebruiker zelf een voorrangsrelatie toevoegen. Na het kiezen van deze functie in de werkbalk kan de relatie worden toegevoegd door de twee betrokken taken aan te klikken. Vervolgens wordt de voorrangsrelatie toegevoegd aan de onderliggende instantie.

### Taken groeperen

Gegroepeerde taken worden in het programma getoond als een blok. Dit is te zien in figuur 4.3. Taken binnen een groep mogen in iedere willekeurige volgorde uitgevoerd worden. Eigenlijk representeert een oplossing met groepen dus meerdere oplossingen. De gebruiker kan de groepen zelf beperkt aanpassen. De functionaliteit "taak verwijderen" werkt binnen groepen nog steeds en daarnaast kan men de volgorde van taken binnen de groep veranderen. Deze volgorde blijft behouden bij het opnieuw laden van de instantie. De gebruiker kan de groepen ook aanpassen door taken binnen de groep te verschuiven via een menuoptie.

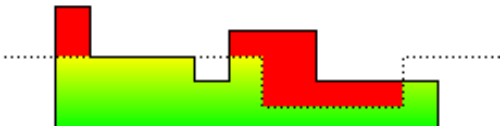
Figuur 4.3: Het tonen van groepen.

### Oplosproces simuleren

Voor educatieve doeleinden en voor onderzoek naar algoritmen is het handig om het oplosproces van de solver te kunnen simuleren. Alle tussenstappen die de solver gemaakt heeft, zijn stap voor stap terug te zien. Hierbij is ook te zien wat de solver heeft gedaan om een resource-conflict te verhelpen. Dit is te zien aan een voorrangsrelatie en het maken van groepen. Het doorlopen van stappen kan worden gedaan met opties in de werkbalk.

### Resource-capaciteit verlagen op een interval

Er is een optie die ervoor zorgt dat de capaciteit van een resource wordt verlaagd op een door de gebruiker gespecificeerd interval. De solver zal hier vervolgens rekening mee houden bij het oplossen van de instantie. Dit zou in de praktijk gebruikt kunnen worden als er bijvoorbeeld 's nachts minder personeel beschikbaar is. Capaciteiten kunnen verlaagd worden bij het bewerken van een resource. Ook in het resource-profiel wordt dit ingetekend. Dit is te zien in figuur 4.4.



Figuur 4.4: Het resource-profiel met verlaging.

### Versnelling voor grote instanties

Het bestaande programma was niet geschikt om grote instanties op te lossen. Zelfs wanneer de solver binnen zeer korte tijd een oplossing vond, duurde het tientallen seconden voordat het programma de oplossing presenteerde op het scherm. Door het intekenen van taken en het resource-profiel te verbeteren is een grote versnelling gerealiseerd.

### In- en uitzoomen

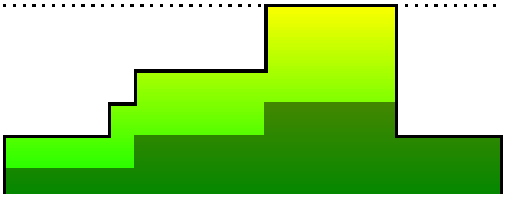
Het bekijken van instanties waarbij het verschil tussen de starttijd en eindtijd erg groot is, vereist veel scrollen. Om dit te vereenvoudigen is er een functionaliteit toegevoegd om in- en uit te zoomen. Hierbij wordt tevens de stapgrootte op de tijdlijn aangepast. In- en uitzoomen is mogelijk via knoppen op de werkbalk.

### Weergave van uren op de tijdbalk

In het oorspronkelijke programma had de tijdbalk geen eenheid. Voor onderzoek naar algoritmen is dit ook niet noodzakelijk, maar voor planners bij NedTrain is het handig om te werken met normale tijden. De weergave van de tijd kan via het bewerkmenu worden aangepast. Hierbij worden er normale tijden getoond op de tijdbalk en worden tijden van projecten en taken ingevoerd als een tijdstip uitgedrukt in uren en minuten.

### Inzichtelijk maken van het resource-verbruik van een project

Het resource-verbruik van een individueel project kan worden opgevraagd. Deze optie is toegankelijk via de rechtermuisknop. Vervolgens wordt het resource-profiel met een afwijkende kleur ingetekend in het bestaande resource-profiel. Dit is te zien in figuur 4.5. Het is ook mogelijk om op te vragen welke taken gebruikmaken van een resource op een bepaald moment. Als een resource-eenheid wordt aangeklikt, dan worden de betrokken taken gemarkeerd. Dit kan handig zijn bij het plannen wanneer de capaciteit van een resource wordt overschreden.



Figuur 4.5: Het resource-verbruik van een specifiek project.

### Importeren van instanties

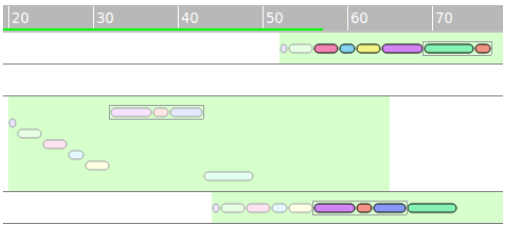
Eenvoudig combineren van instanties was in de oorspronkelijke software niet mogelijk. Hiervoor is een importeerfunctionaliteit gemaakt, die het mogelijk maakt om een bestaande instantie te importeren in een openstaande instantie. Hierbij kan worden aangegeven op welk tijdstip de instantie geïmporteerd moet worden. Dit tijdstip zal gebruikt worden als nulpunt van de geïmporteerde instantie. De importeerfunctie is een makkelijke methode om een extra trein, met een standaard takenpakket, toe te voegen aan de bestaande instantie. Dit kan worden gebruikt als er een extra trein binnenkomt.

### Voorrangsrelaties verwijderen

De verzameling voorrangsrelaties van de instantie kan ervoor zorgen dat er geen oplossing bestaat. In zo'n geval kan het wenselijk zijn om voorrangsrelaties te verwijderen, om op die manier eventueel toch een oplossing te vinden. Het verwijderen van voorrangsrelaties werkt op soortgelijke manier als het toevoegen: na het aanklikken van twee taken wordt de voorrangsrelatie tussen de twee taken verwijderd.

### Taken afmelden

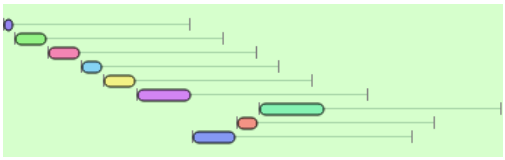
Tijdens het uitvoeren van een schedule wil de planner kunnen zien welke taken reeds afgerond zijn. Door op de tijdbalk te klikken worden alle taken die voor het aangeklikte tijdstip klaar zijn in een andere kleur getekend. Op die manier is het in een oogopslag te zien welke taken afgerond zijn en waaraan nog gewerkt moet worden. Dit is te zien in figuur 4.6.




Figuur 4.6: Een overzicht van afgemelde taken.

### Taken schuiven tussen EST en LCT

Om de planner inzicht te geven in de mate waarin taken kunnen worden verschoven, is er een functionaliteit geïmplementeerd die per taak het interval tekent tussen de vroegste starttijd en de laatste eindtijd. Deze intervallen zijn te zien in figuur 4.7. Dit toont aan in hoeverre taken verschoven kunnen worden, zonder dat dit ervoor zorgt dat er geen oplossing meer bestaat. De solver biedt op dit moment beperkte ondersteuning voor deze functionaliteit. Er wordt voor slechts één oplossing resource-consistentie gegarandeerd, namelijk de *Earliest Starting Time* oplossing. Op dit moment garanderen de getekende intervallen dus niet dat consistentie behouden blijft bij het schuiven. In de toekomst kan dit worden opgelost door *chaining* toe te passen [1]. Bij het schuiven wordt rekening gehouden met de voorrangsrelaties waarbij de taak betrokken is. Dit kan leiden tot andere verschuivingen.



Figuur 4.7: De taken met ingetekende intervallen.

### Tijdsduur van taken aanpassen

Als er tijdens het uitvoeren van een schedule uitloop plaatsvindt, kunnen de taken in de applicatie verlengd worden. Hierbij kan worden gecontroleerd of er in het geval van uitloop nog steeds een oplossing bestaat. Het verlengen van taken is mogelijk door het bijbehorende blokje uit te rekken.

### Oplossen naar een nieuw tabblad

Voor onderzoeksdoeleinden is het wenselijk om verschillende algoritmen of instellingen te kunnen uitproberen op dezelfde instantie. Om dit mogelijk te maken is er een venster toegevoegd waarin de argumenten voor de nieuwe oplospoging kunnen worden ingevoerd. Hierbij kan ervoor gekozen worden om de oplossing te tonen in een nieuw tabblad van de applicatie.

### Oplossingen vergelijken

Als er instanties zijn opgelost met verschillende algoritmen of instellingen, is het interessant om deze in een oogopslag te kunnen vergelijken. Hiervoor is een vergelijkingsvenster toegevoegd, waarin twee instanties kunnen worden getoond. Hierbij worden alle taken grijs getoond, behalve de taken waarvoor geldt dat de starttijden verschillend zijn. Hierdoor is het eenvoudig te zien op welke punten de oplossingen van elkaar afwijken.

### Oorzaak van niet oplosbare instanties

Als bij het oplossen van een instantie blijkt dat dit niet mogelijk is, kan een indicatie worden gegeven waardoor dit komt. Hierbij kan het gaan om een *resource peak*. In dat geval wordt dit ingetekend in het resource-profiel. Als de solver niet slaagt in het vinden van een oplossing vanwege de eis dat er wederzijdse uitsluiting moet zijn, dan wordt dit ook aangegeven.

### Taken vergrendelen

In sommige gevallen is het wenselijk dat een planner zelf aangeeft op welk tijdstip een taak moet plaatsvinden. Hiervoor kan de taak worden verschoven. Het is echter niet de bedoeling dat deze taak weer terugkeert naar zijn oude plek als de solver een oplossing zoekt. Door een optie te selecteren in het contextmenu kan een taak worden vastgezet op zijn huidige positie.

### Flexibiliteit van een taak verhogen

Voor sommige taken is het handig om extra flexibiliteit te hebben bij het uitvoeren. Dit houdt in dat er tijd vrij is tussen een taak en zijn opvolgers, zodat uitloop van de taak niet voor problemen zou zorgen. De gewenste mate van flexibiliteit kan worden gespecificeerd via een optie in het contextmenu. Vervolgens wordt dit bij het oplossen doorgegeven aan de solver, zodat hier rekening mee gehouden kan worden bij het vinden van een nieuwe oplossing.

# 5 Ontwerp

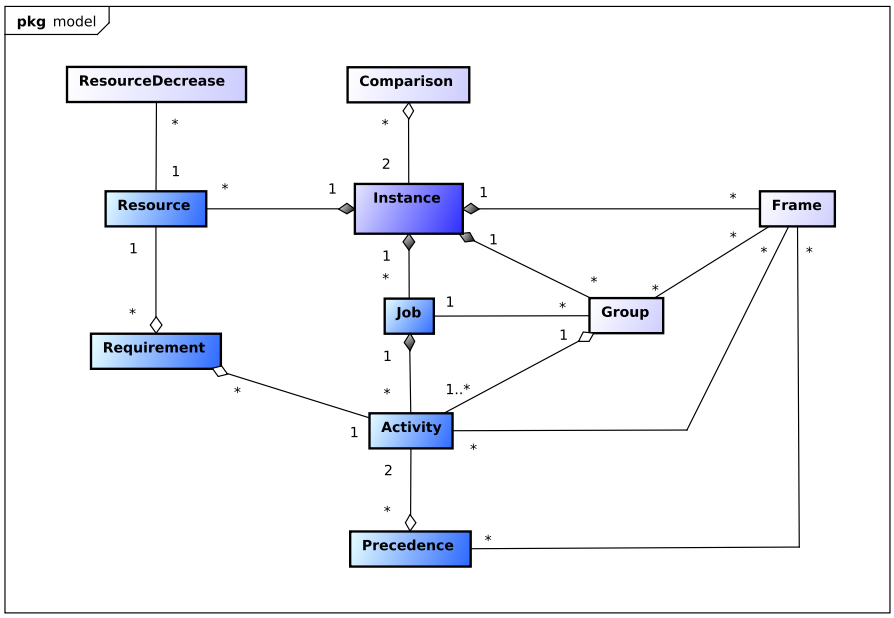
In dit hoofdstuk wordt het ontwerp van de applicatie besproken. Aan de hand van klassendiagrammen zullen de verantwoordelijkheden van de verschillende packages worden besproken. Daarna wordt met sequencediagrammen een deel van de interactie getoond. In de klassendiagrammen zijn de namen van variabelen en methoden weggelaten, omdat die de diagrammen te groot en onoverzichtelijk zouden maken. Voor een volledige specificatie van de methoden en variabelen verwijzen wij naar de Doxygendocumentatie van de applicatie.

## 5.1 Globaal ontwerp

Het systeem bestaat uit verscheidene packages, namelijk: `model`, `data`, `controller`, `widgets`, `solve` en `util`.

Het model representeert een instantie van het schedulingsprobleem en staat daardoor dicht bij de business-entiteiten. In de volgende paragraaf wordt hier dieper op ingegaan. De `widgets` zorgen voor de Graphical User Interface van het programma en hebben beschikking tot een aantal dialoogvensters. De `controller`-klasse staat tussen de `widgets` en het `model` in. De `controller` gebruikt `data`klassen voor toegang tot het bestandssysteem en de SQLite-database. `Util` bevat functies die die specifiek tot het `model` of de `widgets` behoren, maar overal gebruikt kunnen worden, zoals een color generator voor het bepalen van de kleur waarmee een taak wordt weergegeven.

## 5.2 Representatie van de probleeminstantie



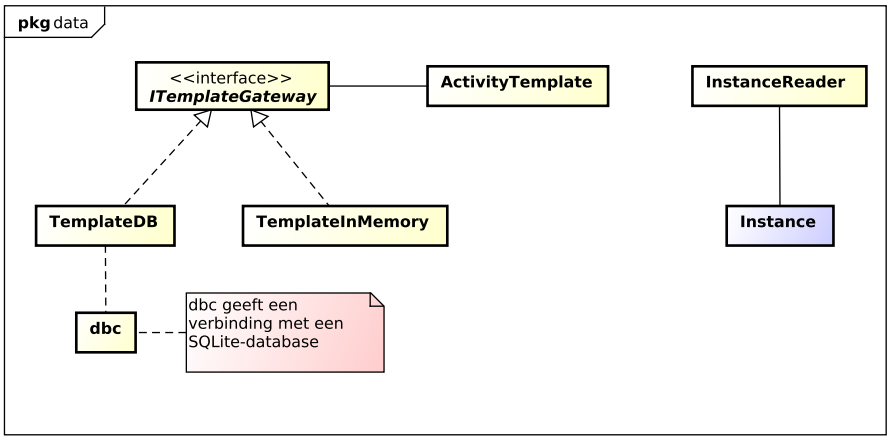

Figuur 5.1: De package `model` met klassen voor de probleeminstantie.

De probleeminstantie wordt gepresenteerd door de klasse `Instance`, die met donkerblauw is aangegeven in figuur 5.1. Essentiële onderdelen van een instantie zijn jobs, activities, resources, requirements en precedences. De bijbehorende klassen zijn in het klassendiagram met blauw aangegeven.

Met een job wordt hetzelfde bedoeld als een 'project' of 'trein', wat gedefinieerd is in paragraaf 3.1. Het is enigszins verwarrend dat hier verschillende benamingen gebruikt worden. De benaming 'job' is geïntroduceerd in de bestaande software van Ronald Evers. We hebben ervoor gekozen om dit te handhaven zodat namen van klassen, geschreven commentaar en functienamen in de software consistent blijven. Een job bevat activities. Activities is een synoniem voor 'taken'. Deze naam is om dezelfde reden anders dan de gebruikelijke benaming. Een instantie bevat daarnaast resources. De klasse `Requirement` geeft aan hoeveel eenheden van een resource vereist worden door een taak. Ten slotte kunnen er voorrangsrelaties (de klasse `Precedence`) bestaan. Deze kunnen inherent aan de probleeminstantie zijn, of ze zijn door een solver aan de instantie toegevoegd.

In lichtblauw zijn extra klassen weergegeven. De klasse `Group` representeert een groep van taken binnen een project. De klasse `Frame` representeert één stap van het oplosproces van de solver en bevat zodoende een 'snapshot' van de instantie. De klasse `Comparison` kan gebruikt worden om een vergelijking tussen twee opgeloste instanties te maken. Bij de klasse `ResourceDecrease` is het van belang om te vermelden dat een resource-verlaging gemodelleerd wordt met behulp van een dummy-project. Dit project bevat één taak die even lang duurt als de duur van het project zelf. Omdat de taak slechts één mogelijk interval heeft, weten we dat de resource waar de taak gebruik van maakt op dit interval minder beschikbaar zal zijn.

## 5.3 I/O-klassen



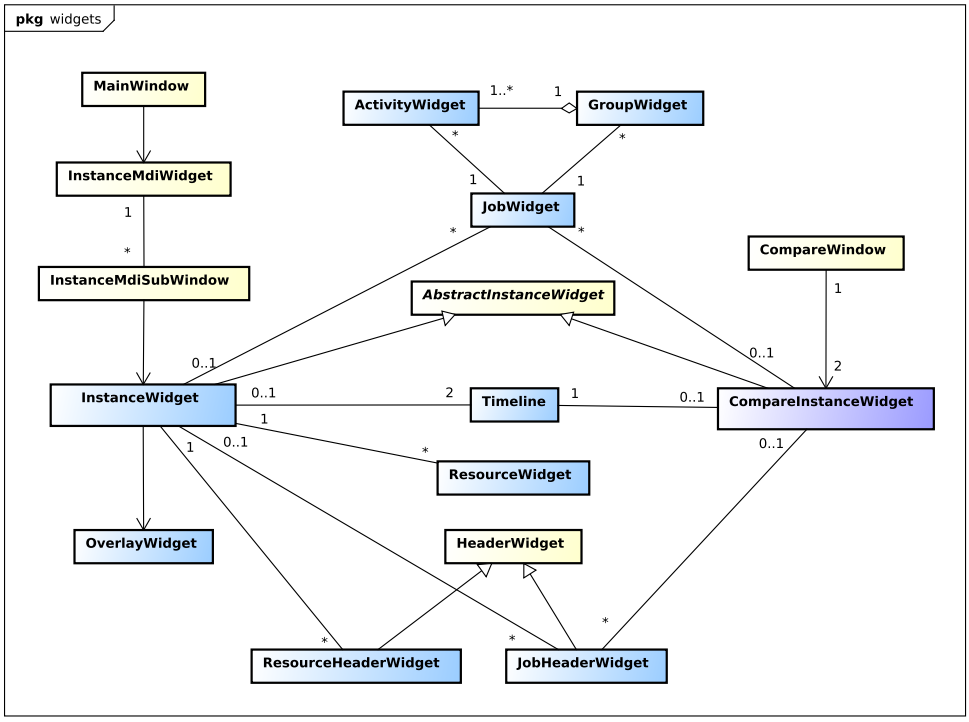

Figuur 5.2: Het klassendiagram van de package `data`.

De package `data`, schematisch weergegeven in figuur 5.2, bevat onder andere de klasse `InstanceReader`, die een instantie naar het bestandssysteem kan wegschrijven of kan inladen. Het formaat van een opgeslagen instantie is reeds besproken in paragraaf 3.4.

Daarnaast zijn er klassen voor het opslaan van objecten van de klasse `ActivityTemplate`.

Deze klasse representeert een sjabloon voor het aanmaken van nieuwe taken. Deze sjablonen kunnen in het programma worden aangemaakt, waarna ze worden opgeslagen in een database. Hierbij is gekozen voor de database-manager SQLite, omdat deze lichtgewicht is en geen server vereist. De interface `ITemplateGateway` wordt gebruikt voor het in- en uitlezen van sjablonen. De implementatie `TemplateDB` gebruikt hierbij de database. Door het specificeren van een interface is de mogelijkheid opengehouden om een implementatie te bieden die van een ander opslagmedium gebruikmaakt. Voor testdoeleinden is de implementatie `TemplateInMemory` gemaakt, die de sjablonen opslaat in het geheugen, waarbij ze verloren gaan na het sluiten van het programma.

## 5.4 Grafische interface

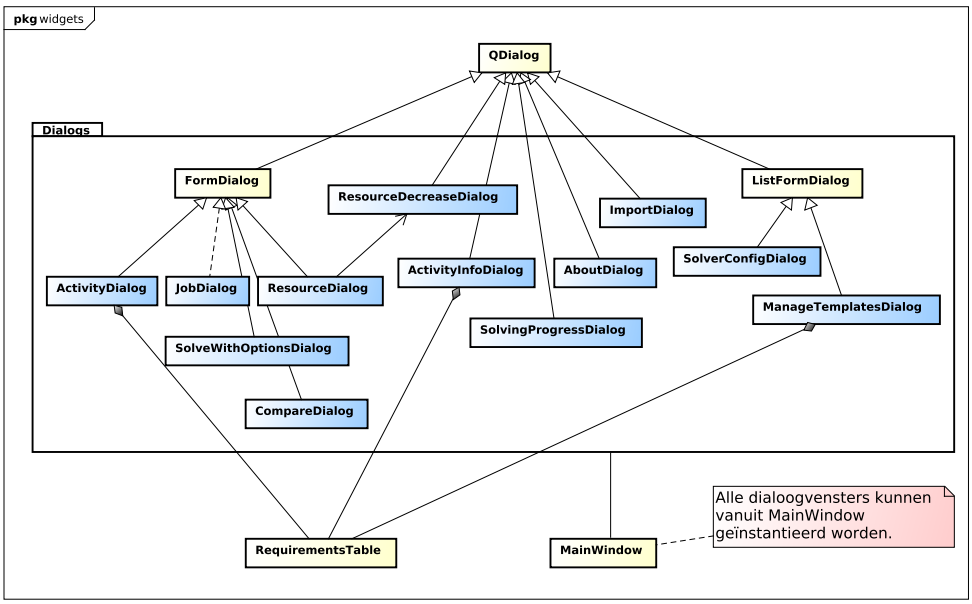



Figuur 5.3: Het klassendiagram van de package `widgets`.

De grafische inteface maakt gebruikt van het Qt-framework. Figuur 5.3 geeft de belangrijkste GUI-klassen weer. De `InstanceMdiWidget` initialiseert voor iedere instantie die geopend wordt een `InstanceWidget` en plaatst deze in een nieuw tabblad. Het bovenste gedeelte van de `InstanceWidget` bestaat uit het weergeven van projecten en taken. Hier zorgen `JobHeaderWidget` en `JobWidget` voor. Binnen `JobWidget` worden de taken getekend met behulp van `ActivityWidget`s. Het onderste gedeelte van de `Instancewidget` toont het resource-gebruik. Hierbij zijn `ResourceHeaderWidget` en `ResourceWidget` betrokken. `TimeLine` wordt gebruikt om de tijd weer te geven boven de `ResourceWidget` en de `JobWidget`. De `OverlayWidget` is een doorzichtige widget die over de `JobWidget` heen zit, die het mogelijk maakt om over de `JobWidget` heen te tekenen. Bijna alle klassen die gebruikt worden door `InstanceWidget`, worden ook gebruikt door `CompareInstanceWidget`, die twee instanties onder elkaar toont en daarbij

de verschillen aangeeft.

Vanuit het hoofdscherm (`MainWindow`) kunnen er verschillende dialoogvensters worden aangeroepen, bijvoorbeeld om taken en projecten toe te voegen. Het klassendiagram in figuur 5.4 geeft hier een goed overzicht van.




Figuur 5.4: Het klassendiagram van de package `widgets.dialogs`.

## 5.5 Overige klassen

De package `controller` bevat enkele klassen die een sleutelrol vervullen in de werking van het programma. Er is een algemene klasse `Controller` en een meer specifieke klasse `InstanceController`, waarvan elke geopende instantie zijn eigen object heeft. Deze klassen zijn in veel gevallen een brug tussen verschillende GUI-widgets en voorkomen dat de widgets teveel logica bevatten. Ten slotte wordt vanuit de controller het `MainWindow` gestart.

Dan is er nog de klasse `Solver`, die een externe solver kan aanroepen en zijn uitvoer kan verwerken. De klasse `Solver` zorgt ervoor dat de instantie bijgewerkt wordt na het solven en dat de berekeningsstappen worden opgeslagen.

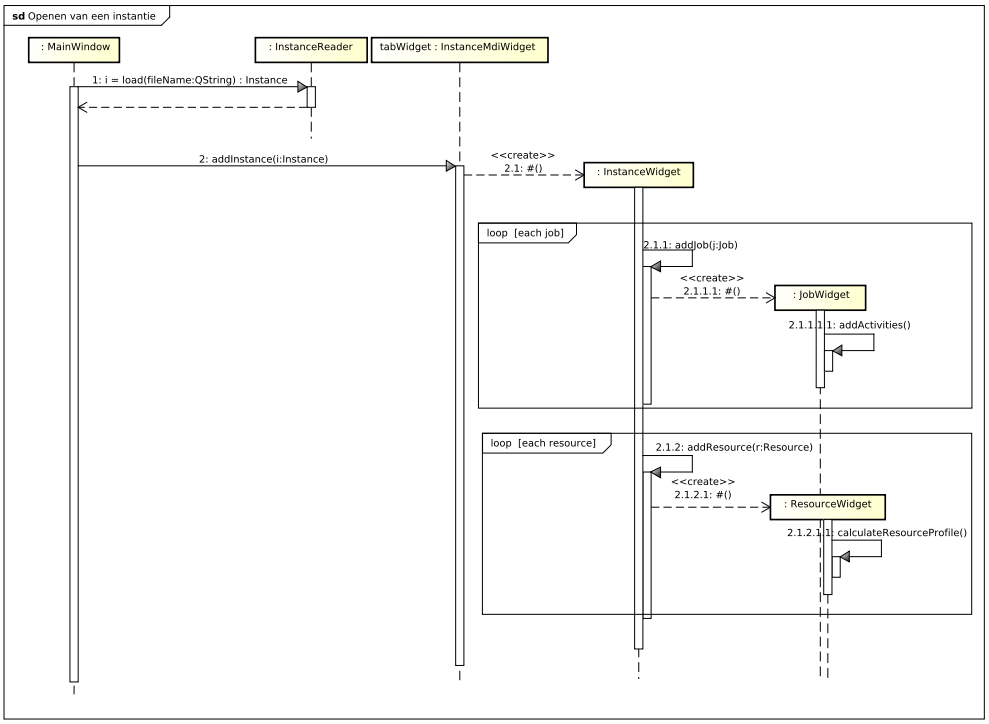
Ten slotte zijn er nog enkele hulpklassen in de package `util` zoals de klasse `Language` voor het representeren van de taal van de gebruikersinterface. Ook zijn er nog enkele exception-klassen die een nette foutafhandeling mogelijk maken.

## 5.6 Dynamische modellen

Nu het design van de verschillende packages en klassen is toegelicht, laten we zien hoe de interactie tussen verschillende componenten verloopt. Ten behoeve van de overzich-

telijkheid worden alleen de belangrijkste stappen van het proces weergegeven.

Het openen van een instantie verloopt zoals weergegeven in figuur 5.5. Als in `MainWindow` op een widget wordt geklikt (in het menu of in de werkbalk) om een instantie te openen, wordt eerst een dialoogvenster getoond om een bestand te selecteren. Dit is een standaard onderdeel van Qt en daarom niet getoond in het diagram. De `InstanceReader` creëert vervolgens een `Instance` object op basis van de inhoud van het geselecteerde bestand. Om de geopende instantie weer te geven in de applicatie wordt er een nieuw tabblad (`InstanceMdiWidget`) toegevoegd met als inhoud een `InstanceWidget`. Afhankelijk van het aantal resources en projecten in de instantie worden aan `InstanceWidget` de nodige `JobWidget`s en `ResourceWidget`s toegevoegd. Aan elk `JobWidget` worden ten slotte de bijbehorende taken (activities) toegevoegd.

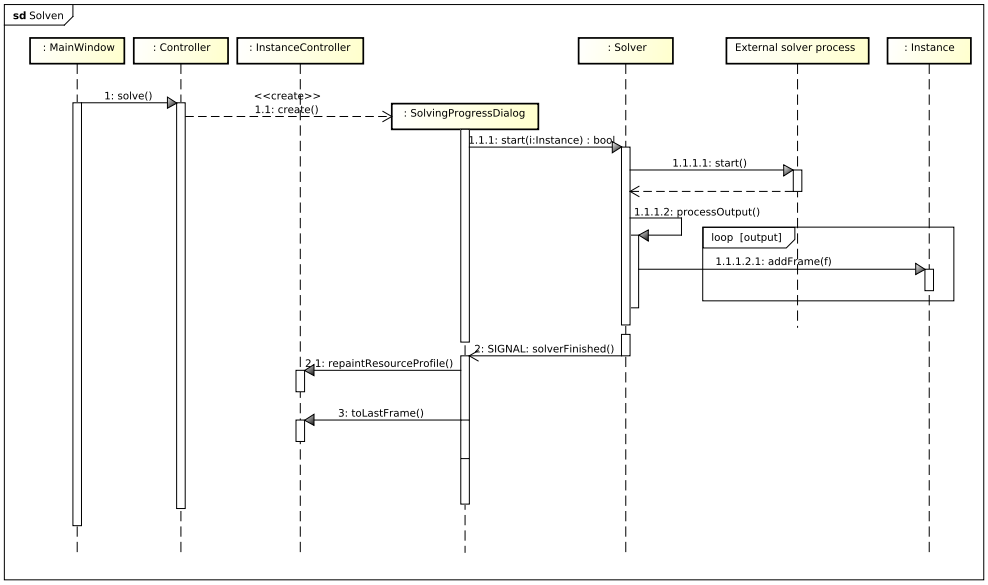



Figuur 5.5: Het sequencediagram van het openen van een instantie.

Het oplossen van een instantie wordt getoond in het sequencediagram van figuur 5.6. Zoals bij het openen het geval was, wordt de actie gestart in `MainWindow`. De controller opent vervolgens een dialoogvenster om het solven te starten. De `Solver` roept een extern proces aan die een oplossing voor de instantie genereert. Tijdens het oplossen worden steeds frames toegevoegd aan `Instance`. Deze frames maken het mogelijk om achteraf het solveproces terug te spoelen. Zodra de solver klaar is wordt een `SIGNAL` gestuurd, waardoor alle wijzigingen gepropageerd worden in de view.

In Qt worden `SIGNALS` en `SLOTS` gebruikt voor communicatie tussen objecten. Als er een wijziging plaatsvindt in het ene object is het vaak nodig dat een ander object daarvan op de hoogte wordt gesteld. Dit kan door middel van het verbinden van een `SIGNAL` van het ene object aan een `SLOT` van een ander object. In onze applicatie gebruikten we dit mechanisme voor de communicatie tussen het model en de view. Als er een wijziging plaatsvindt in het model, wordt de view hiervan automatisch op de hoogte gesteld.

Hierdoor is de communicatie tussen model en view mogelijk zonder een hoge *coupling* te veroorzaken tussen deze entiteiten.




Figuur 5.6: Het sequencediagram van het oplossen van een instantie.

# 6 Ontwikkelproces

Dit hoofdstuk beschrijft onze ervaringen met ontwikkelmethoden en de gebruikte hulpmiddelen. Eerst zal aan de orde komen hoe het toepassen van *scrum*-principes ons bevallen is. Vervolgens wordt per hulpmiddel toegelicht hoe dit tijdens het project werd gebruikt en hoe we dit hebben ervaren.

## 6.1 Ontwikkelmethode

Tijdens het project is gebruikgemaakt van principes uit de *scrum* ontwikkelmethodiek. Zoals beschreven in het oriëntatieverslag, was het niet mogelijk om volledig volgens de *scrum*-methodiek te werken. Dagelijks is er een *sprint meeting* gehouden. We hebben gemerkt dat dit bij de opzet van het project lastig toepasbaar was. Omdat we de hele dag samenwerkten, was er tijdens een bespreking aan het begin van de dag weinig te melden over de ontwikkelingen van de vorige dag. De meeste dingen waren immers al besproken. De tijdsduur van de sprints was ruim een week, afhankelijk van de hoeveelheid functionaliteiten op het sprint backlog en het aantal verplichte vrije dagen in die week. In het resterende deel van deze paragraaf wordt per sprint een korte beschrijving gegeven. De bijbehorende user stories zijn te vinden in het Requirements Analysis Document in appendix D.

### 6.1.1 Sprint 1

Het doel van de eerste sprint was het bekend raken met de ontwikkelmethode en de bestaande software. Het aantal functionaliteiten op het backlog werd bewust klein gehouden, omdat werd verwacht dat het ontwikkelen nog niet zo snel zou gaan.

#### Terugblik

Het verloop van de sprint was goed, maar achteraf bleek dat er te weinig taken waren toegekend aan de sprint. Hierdoor was de looptijd van de sprint aanvankelijk kort. Dit is snel opgelost door gerelateerde taken toe te voegen aan het backlog, in plaats van het direct starten van een tweede sprint. Dit leerpunt werd meegenomen naar de volgende sprint.

### 6.1.2 Sprint 2

Tijdens de tweede sprint werden er geavanceerdere functionaliteiten geïmplementeerd. Tevens werden er meer functionaliteiten op het backlog geplaatst om te voorkomen dat de sprint vroeg afgelopen zou zijn. De doelstelling tijdens deze sprint was het implementeren van functionaliteiten voor aanvang van de voortgangspresentatie.

#### Terugblik

Na de tweede sprint waren we erg tevreden over het verloop. Het implementeren verliep veel soepeler dan tijdens de eerste sprint en alle functionaliteiten die we voor ogen hadden konden worden afgerond. Tijdens deze sprint hebben we het houden van dagelijkse

besprekingen vervangen door korte overlegmomenten gedurende de dag. Het voordeel hiervan was dat we tijdens het ontwikkelen goed op de hoogte waren van elkaars werk. Dit was handig als er afhankelijkheden waren tussen het werk van meerdere teamleden.

### 6.1.3 Sprint 3

Tijdens de derde sprint werd er gewerkt aan een representatieve versie van de software om te gebruiken bij de *14th European Agent Systems Summer School* in Valencia. Vooraf zijn alle wensen besproken met de begeleiders, zodat deze functionaliteiten nog geïmplementeerd konden worden voor de summerschool. Tevens werd er veel aandacht besteed aan het oplossen van kleine problemen. Aan het einde van de sprint is er een bijeenkomst geweest met Bob Huisman om de nieuwste versie van het programma door te nemen.

#### Terugblik

Hoewel alle gewenste functionaliteiten aan het eind van de sprint klaar waren, verliep het ontwikkelen minder vlot dan gehoopt. Dit werd veroorzaakt door het constateren van nieuwe bugs, die voor het einde van de sprint opgelost moesten worden. Leerpunt was dat de geavanceerde functionaliteiten beter opgesplitst moesten worden om het overzicht te behouden.

### 6.1.4 Sprint 4

Het doel tijdens sprint 4 was het implementeren van functionalteiten die werden aangedragen tijdens de voortgangsbijeenkomst. Omdat er verder werd gewerkt op bestaande code, was er het gevaar dat de code minder overzichtelijk zou worden naarmate er meer dingen werden toegevoegd. Om dit terug te dringen is er aandacht besteed aan het verbeteren van de kwaliteit van de code.

#### Terugblik

Het verloop van deze sprint was erg goed en het ontwikkelen verliep zonder grote problemen. De korte overlegmomenten die werden geïntroduceerd tijdens de tweede sprint bleken effectief. Na afloop van deze sprint waren we nog steeds tevreden met onze keuze voor de principes uit de *scrum*-ontwikkelmethode. De extra flexibiliteit ten opzichte van andere ontwikkelmethoden is goed bevallen en resulteerde in een prettige werksfeer.

### 6.1.5 Sprint 5

De vijfde sprint was gericht op het implementeren van de laatste functionaliteiten en het verbeteren van de code. Deze verbeteringen konden voor een deel meegenomen worden in de versie die werd opgestuurd naar de Software Improvement Group. Tevens zijn er laatste wijzigingen gedaan aan de code van de solver.

**Terugblik**

Er zijn tijdens deze sprint geen problemen opgetreden. Inmiddels waren we geheel gewend aan de *agile* werkvorm, wat de productiviteit ten goede kwam. We hebben gemerkt dat het niet volledig toepassen van *scrum* niet optimaal is. Bij een volgend ontwikkelproject zou het beter zijn om de ontwikkelmethodiek volledig toe te passen. Helaas was dit wegens de opzet van het project niet mogelijk. Desondanks hebben we genoeg facetten kunnen uitproberen om een beeld te krijgen van de werkvorm.

## 6.2 Hulpmiddelen

In deze paragraaf zal besproken worden hoe wij het gebruik van de gekozen hulpmiddelen hebben ervaren. Hierbij wordt vooral aandacht besteed aan het framework, het continuous integration platform en het bijhouden van bugs en scrum backlogs.

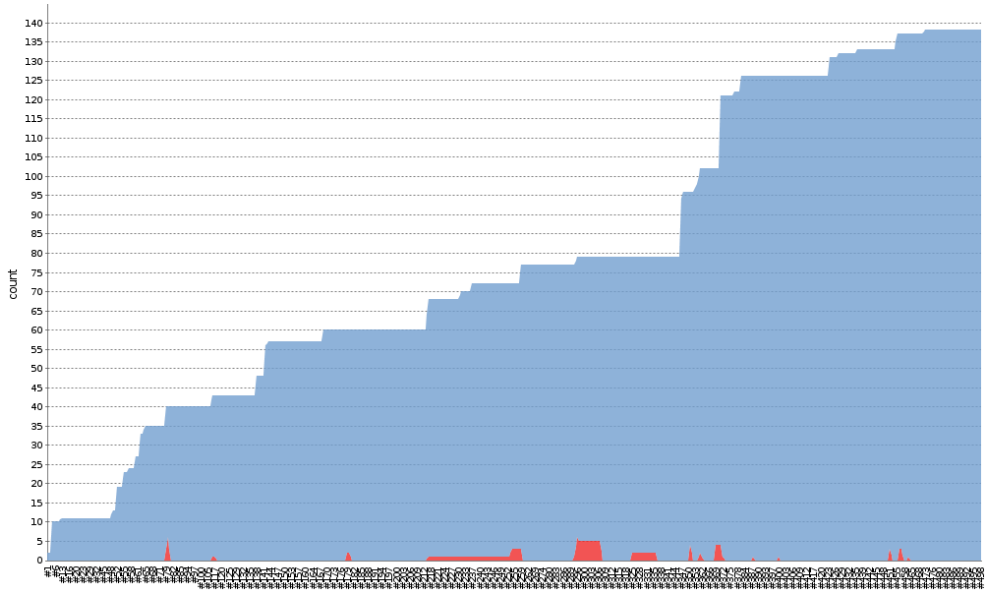
### 6.2.1 C++ en Qt

Voor de ontwikkeling van de software werd gebruikgemaakt van de taal C++ in combinatie met het Qt framework. Dit was voor ons een nieuwe ervaring en tevens een mooi moment om meer te leren over het programmeren met C++. Hoewel we vooraf vooral gewend waren om software met een grafische interface te ontwikkelen in Java, heeft dit niet tot grote problemen geleid. Problemen met C++ konden snel worden opgelost, omdat Jan al eerder projecten in deze programmeertaal had uitgevoerd. Hierdoor kon er tijdens de sprints goed worden doorgewerkt en werd er weinig tijd verspild aan het oplossen van problemen. Naarmate het project vorderde, verliep het ontwikkelen met C++ steeds soepeler. Het ontwikkelen van grafische interfaces met Qt was voor ons allemaal een nieuwe ervaring. Omdat er een bestaande applicatie als basis werd gebruikt, konden we de code daarvan gebruiken om te zien hoe bepaalde concepten werken in Qt. Dit werkte erg efficiënt om de beginselen onder de knie te krijgen.

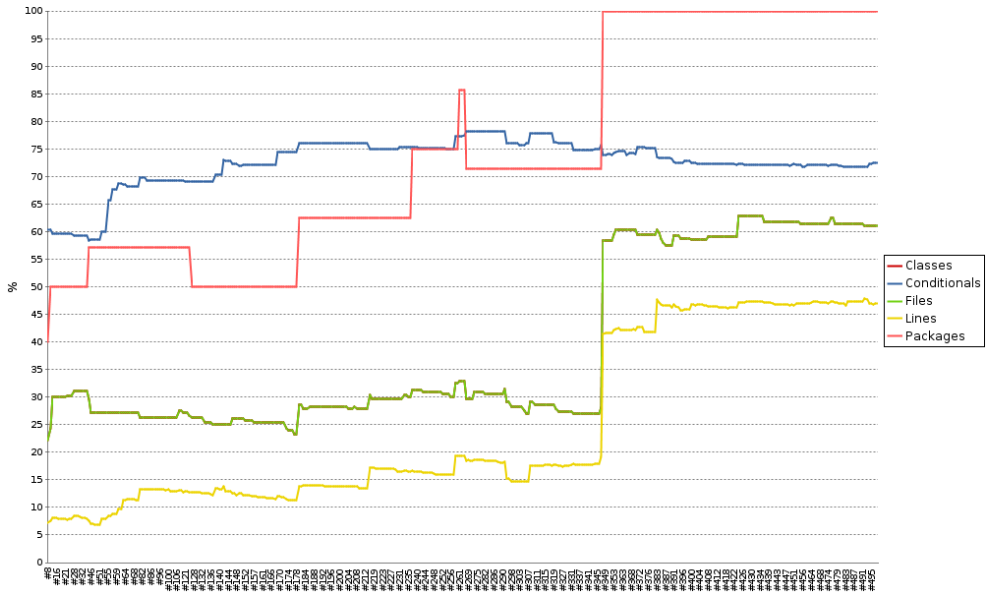
### 6.2.2 Google Test

Tijdens het ontwikkelproces is een uitgebreide testsuite gemaakt met behulp van het Google Test framework. Omdat we testcases tijdens het ontwikkelen hebben geschreven en niet achteraf, konden we telkens verifiëren of de werking van eerder geschreven code nog steeds dezelfde was. Falende testcases alarmeerden ons wanneer wij wijzigingen in de code aanbrachten die ongewenste consequenties hadden. Een grafiek van de ontwikkeling van tests en hun resultaten is te zien in figuur 6.1.

Aangezien wij hebben voortgebouwd op eerdere code die geen testsuite had, was er aanvankelijk een inhaalslag te maken wat betreft het schrijven van testcases. Dit komt ook tot uiting in de grafiek met codedekking in figuur 6.2, waarbij de dekking in het begin laag is. Het testen van GUI-elementen is erg tijdrovend en in ons geval niet eenvoudig, omdat er veel gebruik wordt gemaakt van elementen die niet standaard worden ondersteund door testbibliotheken voor grafische interfaces in Qt. Om die reden hebben we ons vooral gericht op het testen van alle onderliggende modelklassen en de operaties die op probleeminstanties kunnen worden uitgevoerd. Dit verklaart waarom de uiteindelijk dekking van de code niet maximaal is.




Figuur 6.1: De testresultaten.




Figuur 6.2: De codedekking.

Zo nu en dan is ook *test-driven development* (TDD) toegepast. Het was echter niet mogelijk om dit geheel in ons proces te integreren, mede door de inhaalslag die te maken viel. TDD houdt in dat men per in te bouwen functionaliteit eerst een testcase schrijft en dan pas de code. Men schrijft eerst een zo eenvoudig mogelijke testcase. Deze zou moeten falen, aangezien de implementatie nog ontbreekt. Men probeert dan de testcase te laten slagen door een implementatie te schrijven. Wanneer dit is gelukt, kan de code 'gerefactored' (d.w.z. mooi gemaakt) worden, aangezien de eerste prioriteit lag op het laten slagen van de testcase. Op diverse momenten hebben wij eerst een

testcase gemaakt en vervolgens de implementatie. Een voordeel is dat je de testcase en de implementatie ook door verschillende mensen kunt laten maken. Bovendien fungeert de testcase op deze manier ook als een specificatie van de functionaliteit die wordt ingebouwd.

### 6.2.3 Continuous integration met Jenkins

Tijdens het gehele ontwikkelproces is *continuous integration* toegepast met behulp van het programma Jenkins. Continuous integration houdt in dat om de zoveel tijd *automatisch* alle programmacode wordt gecompileerd en de werking daarvan geverifieerd wordt. Op die manier wordt continu gecontroleerd of het programma stabiel is, zonder dat je daar als programmeur iets voor hoeft te doen. In ons geval gebeurde dit elke 20 minuten, indien er veranderingen in de code waren.

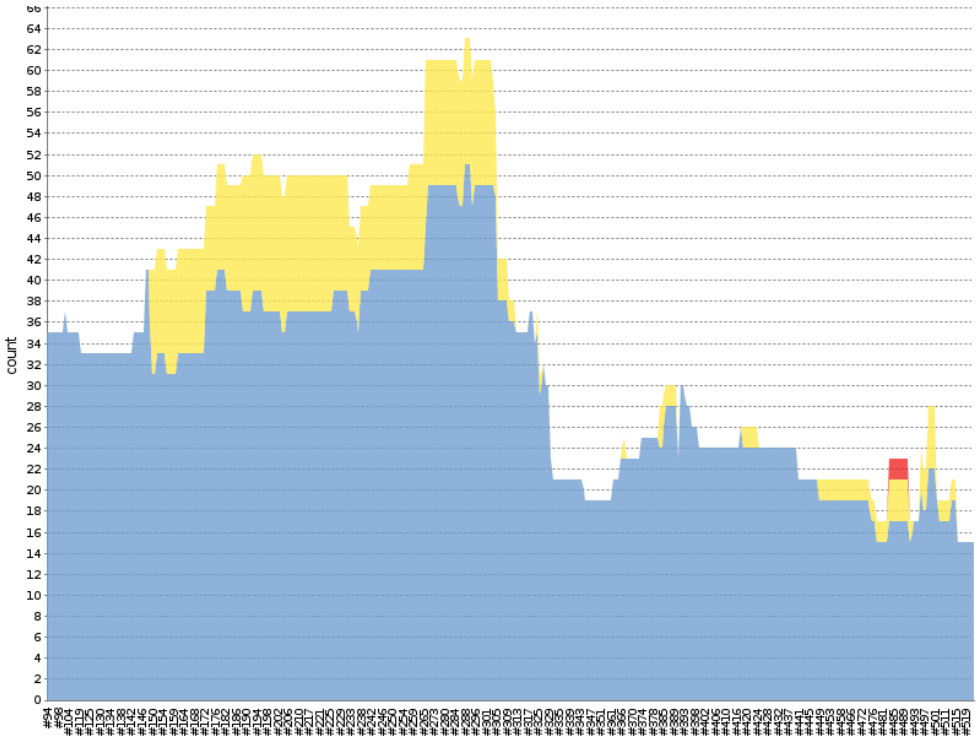
Wij hebben Jenkins bovendien uitgebreid met verscheidene plug-ins die *statische analyse* op de code verrichtten en zodoende nuttige statistieken opleverden. Zo is er gebruikgemaakt van de tool CppCheck, die waarschuwingen gaf over de stijl van de code en ongebruikte variabelen en methoden. Daarnaast detecteerde de plug-in PMD/CD duplicaatcode. Tijdens het schrijven van een groot programma wordt er al snel duplicaatcode toegevoegd, tenzij je actie onderneemt om dat te voorkomen met behulp van 'refactoring'. Doordat de plug-in het aantal duplicaten meette, konden we eenvoudig zien waar actie ondernomen moest worden. Hierdoor konden we het aantal duplicaten binnen de perken houden, hetgeen de kwaliteit ten goede kwam. Aan het begin van het ontwikkeltraject werd er minder gelet op duplicaatcode. Later hebben we dit beter onder controle gehouden. Dit is ook te zien in figuur 6.3, waarbij blauw staat voor het aantal duplicaten van $< 10$ regels, geel voor het aantal van $\geq 10$ regels en rood voor het aantal van $\geq 25$ regels.

Ten slotte is de nuttigste functie van Jenkins het automatisch kunnen laten uitvoeren van testcases telkens wanneer het programma gecompileerd wordt. Jenkins geeft vervolgens een waarschuwing af wanneer er testcases mislukken. Ook wordt de dekking van de code berekend. De diagrammen in de vorige paragraaf komen rechtstreeks uit Jenkins. Samenvattend kan gesteld worden dat Jenkins continu de gezondheid van onze code inspecteerde, wat ons behoedde voor fouten en ons een spiegel voorhield wat betreft onze programmeerstijl, zodat we kritisch konden zijn en ons konden verbeteren.

### 6.2.4 Bug tracking en backlogs met Mantis

Voor het monitoren van bugs werd gebruikgemaakt van Mantis. We hebben tijdens een eerder project met deze applicatie gewerkt en dit beviel dusdanig goed dat we ervoor gekozen hebben om dit weer te doen. Bugs en andere problemen werden bijgehouden in zogenaamde *issues*, die eventueel aan elkaar werden gekoppeld als ze onderling een verband hadden. Dit leverde een gestructureerd overzicht op van alle problemen, met per probleem een duidelijke omschrijving. Doordat de *issues* gekoppeld werden aan een persoon, was het altijd direct duidelijk wie op dat moment verantwoordelijk was voor het probleem.

We hebben Mantis ook gebruikt voor het bijhouden van product- en sprintbacklogs volgens de scrum ontwikkelmethode. Het was niet mogelijk om dit, volgens de scrumfilosofie, met post-it's op een bord te doen, omdat de ruimte waarin werd gewerkt niet exclusief beschikbaar was voor ons project. We hadden gehoopt dat de Mantis-plugin




Figuur 6.3: De duplicaatcode.

voor scrum een goed alternatief zou zijn, maar dit viel in de praktijk tegen. Een digitaal scrumbord werkt minder prettig en het is veel minder toegankelijk om dingen aan te passen. Hieruit is dus gebleken dat een fysiek scrumbord eigenlijk essentieel is. Uiteindelijk werden de product- en sprint backlogs bijgehouden in een aparte categorie, zodat het wel mogelijk was om de voortgang binnen een sprint bij te houden. Hoewel de uitvoering hiervan niet helemaal volgens de scrumfilosofie was, is het ons goed bevallen.

### 6.2.5 Versiebeheer

Voor versiebeheer werd tijdens het ontwikkelen van de software gebruikgemaakt van Subversion. Er was altijd een werkende internetverbinding beschikbaar, dus dit verliep zonder problemen. Bij het aanpassen van de solver van één van de begeleiders werd in eerste instantie geen versiebeheer gebruikt. Op dat moment was dit niet noodzakelijk, omdat het aantal wijzigingen erg klein was en de aanpassingen door een persoon werden gedaan. De aanpassingen werden verspreid in de vorm van een patch. Na enkele weken bleek dat er meerdere dingen aangepast moesten worden. Op dat moment werd besloten om, alleen voor de ontwikkeling van de solver, over te stappen op het gedistribueerde versiebeheersysteem Mercurial. Hierbij werd gewerkt in een *fork* van de solver van de begeleider, zodat er onafhankelijk doorontwikkeld kon worden. Door middel van een *pull request* konden we onze wijzigingen doorgeven. Hoewel Mercurial, in tegenstelling tot Subversion, gedistribueerd is, ging dit erg soepel.

## 6.3 Software Improvement Group

In de loop van het ontwikkelingsproces is de code opgestuurd naar de Software Improvement Group (SIG) om tussentijds beoordeeld te worden. In deze paragraaf zal toegelicht worden welke aanbevelingen werden gegeven en hoe dit is verwerkt in de uiteindelijke versie van het programma.

SIG heeft het volledige programma beoordeeld. Hierbij is dus ook de code meegenomen die al bestond bij aanvang van het bachelorproject. Aangezien deze bestaande code niet alleen is uitgebreid, maar ook herzien en gerefactored, was het niet mogelijk een duidelijke scheiding aan te brengen tussen de bestaande code en de eigen code. In het opgestuurde bestand werd een toelichting bijgevoegd waarin, voor zover mogelijk, duidelijk werd gemaakt welke delen van de code al aanwezig waren.

### Oordeel

Bij de tussentijdse evaluatie scoorde het programma bijna vier sterren op het onderhoudbaarheidsmodel van SIG. Dit betekent dat de code over het algemeen bovengemiddeld heeft gescoord. De hoogst haalbare score bij deze evaluatie is vijf sterren. Om ons te helpen bij het verbeteren van de code, heeft SIG aangegeven op welke punten er lager werd gescoord. In het vervolg van deze paragraaf zullen deze punten worden uitgelicht.

### Component Independence

Bij het controleren van Component Independence wordt gekeken naar de hoeveelheid code die alleen binnen een bepaald deel van het programma wordt gebruikt, en niet wordt aangeroepen vanuit andere delen van het programma. Als er veel code is die wordt aangeroepen vanuit andere componenten, dan is de kans groter dat aanpassingen binnen een bepaald deel van het programma leiden tot aanpassingen in andere delen van het programma. Dit zou kunnen zorgen voor een lagere productiviteit.

In ons programma viel de cyclische afhankelijkheid op tussen de componenten *controller* en *widgets*. Er werd aangeraden om kritisch te kijken of deze afhankelijkheid kan worden verwijderd.

De software die als startpunt werd gebruikt bevatte geen controller. Voor alle communicatie werd gebruikgemaakt van `SLOTS` en `SIGNALS`. Deze aanpak is minder geschikt voor alle communicatie, omdat het zeer complex wordt naarmate er meer functionaliteit wordt toegevoegd. Er is besloten om dit alleen toe te passen op plaatsen waar het *Observer pattern* wordt gebruikt. Voor alle andere communicatie is er een controller geïntroduceerd. Gevolg hiervan is dat er geen strikte scheiding is tussen de verantwoordelijkheden van de widgets en de controller, omdat widgets aanroepen doen naar de controller en omgekeerd. Een mogelijke oplossing hiervoor is het gebruikmaken van het MVC-ontwerppatroon, waarbij de controller reageert op acties binnen de widgets. Dit is een ingrijpende verandering binnen het programma, die niet eenvoudig is door te voeren. Vanwege de beperkte overgebleven tijd hebben wij besloten om dit niet te verwerken in het programma. Het zou onverantwoord zijn om in dit stadium van het project dergelijke grote wijzigingen door te voeren. Het advies om dit alsnog te doen zal worden opgenomen in de lijst met aanbevelingen.

### Unit Size

Bij het beoordelen van Unit Size wordt er gekeken naar het percentage van de code dat bovengemiddeld lang is. Door methoden op te splitsen wordt elk onderdeel makkelijker te begrijpen, te testen en ook eenvoudiger te onderhouden.

Ons programma bevatte enkele lange methoden, waarbinnen er losse functionaliteiten te vinden waren die opgedeeld kon worden in aparte methoden. Het afhandelen van de uitvoer van de solver kon bijvoorbeeld opgesplitst worden door onderscheid te maken tussen verschillende soorten uitvoer. We hebben lange methoden binnen het programma vereenvoudigd en opgedeeld in losse methoden met een autonome functionaliteit. Dit is ook de leesbaarheid van de code ten goede gekomen.

### Unit Complexity

Bij dit onderdeel wordt er gekeken naar het percentage van de code dat bovengemiddeld complex is. Ook hier zorgt het opsplitsen ervoor dat de code beter te begrijpen is, en het testen en onderhouden zal eenvoudiger zijn.

Bij de evaluatie is gebleken dat de meest complexe methoden binnen het programma ook het langste waren. We hebben dit probleem dus kunnen oplossen door het voorgaande probleem op te lossen.

# 7 Conclusies

Terugkijkend concluderen we dat we de gestelde doelen hebben behaald en dat we tevreden kunnen zijn over het resultaat. Alle requirements die vooraf werden verzameld, hebben we kunnen verwerken in het programma. Daarnaast hebben we ook tussentijds toegevoegde requirements kunnen inwilligen. Naar onze mening hebben wij een solide applicatie afgeleverd, die een duidelijk geheel is en geen half werk bevat. Naast dat de applicatie in veel behoeften van de opdrachtgevers zou moeten voorzien, is er nog alle ruimte voor verdere uitbreiding en doorontwikkeling, waarbij we in het volgende hoofdstuk zullen stilstaan.

We hebben allemaal ook veel geleerd door het gebruik van de programmeertaal C++ en het Qt framework, welke voor ons redelijk onbekend waren. Ook hebben we tijdens het project de aangeleerde concepten uit vakken zoals Software Engineering en Softwarekwaliteit en Testen kunnen toepassen. Hierdoor hebben we gezien hoe dit in de praktijk gebruikt kan worden.

De samenwerking tijdens het uitvoeren van het project was goed en het ontwikkelen verliep voorspoedig. De enthousiaste reacties van de begeleiders zorgden voor extra stimulans om door te werken aan een mooi eindproduct. Ook de rustige werkomgeving heeft positieve invloed gehad op het resultaat. Er zijn geen problemen opgetreden en we hebben goed door kunnen werken.

Samenvattend zijn wij erg tevreden met het uiteindelijke eindproduct en het doorlopen proces. Wij vonden dit bachelorproject dan ook een leerzame en geslaagde onderneming.

# 8 Aanbevelingen

In dit hoofdstuk geven we enkele aanbevelingen voor de toekomst. Hierbij wordt onderscheid gemaakt tussen de solver en de opgeleverde applicatie.

## 8.1 Solver

Hoewel wij kleine dingen hebben aangepast in de code van de solver, wordt de ontwikkeling hoofdzakelijk gedaan door Michel Wilson. De ontwikkeling hiervan is op dit moment nog niet afgerond. Dit betekent dat enkele functionaliteiten in ons programma nog niet optimaal kunnen werken. Het programma biedt de mogelijkheid om per taak een interval te tekenen, waarbinnen de taak verschoven kan worden, zonder dat dit leidt tot een resource-conflict. De oplossing en de intervallen die de solver doorgeeft aan het programma, garanderen dit echter niet. Dit betekent dat een verschuiving binnen het interval wel kan resulteren in een resource-conflict. Dit probleem kan worden opgelost door de solver te laten garanderen dat alle temporeel toegelaten oplossingen ook resource toegelaten zijn. Hierbij kan *chaining* toegepast worden [1]. Op dit moment wordt dat slechts voor één oplossing gegarandeerd, namelijk de *earliest starting time* oplossing.

De oorspronkelijke solver van masterstudent Ronald Evers was ontwikkeld in de programmeertaal C. Vervolgens heeft Michel Wilson verder ontwikkeld in deze programmeertaal. Hoewel onze aanpassingen in de solver relatief klein zijn, zijn we overgestapt op de programmeertaal C++. Hierdoor zijn delen van de code veel korter en overzichtelijker geworden, wat de leesbaarheid ten goede is gekomen. Als dit niet tot een grote afname van de performance leidt, zou een definitieve overstap naar C++ overwogen kunnen worden.

## 8.2 Applicatie

Het door ons gemaakte programma bevat op dit moment geen *undo*-functionaliteit om uitgevoerde acties snel ongedaan te maken. Als dit wordt ingebouwd, kan dit worden gecombineerd met de functionaliteit om het planproces van de solver te bekijken. Hiermee is het nu al mogelijk om stap voor stap terug te kijken wat de solver heeft uitgevoerd. Alle handelingen die de gebruiker doet, zouden ook toegevoegd kunnen worden aan deze lijst met stappen. In dat geval zullen er bij het stap voor stap 'terugspoelen' zowel handelingen van de solver als van de gebruiker te zien zijn.

Het is in de huidige versie alleen mogelijk om oplossingen te vergelijken op basis van de starttijd van taken. Het is wellicht interessant om een vergelijkingsfunctie te maken voor de stappen die gezet zijn tijdens het oplossen. Dit geeft nog meer inzicht in de manier waarop een oplossing tot stand komt.

In het programma kan een functionaliteit worden ingebouwd om te koppelen met een LP-solver. Dit kan bijvoorbeeld gebruikt worden om flexibiliteit evenredig te verdelen over de taken. Als de koppeling met deze solver algemeen wordt gehouden, kan dit eventueel ook gebruikt worden om andere toekomstige optimaliseringsproblemen op te lossen in het programma.

Daarnaast is het een goed idee om een solve-functie te maken die alleen de starttijden van taken herberekent. Op dit moment gebeuren er bij het oplossen twee dingen: er worden voorrangsrelaties toegevoegd aan de instantie en de starttijden worden berekend. Door dat laatste onderdeel ook apart uitvoerbaar te maken, kunnen wijzigingen van de gebruiker zoals het verschuiven van taken of aanpassen van tijdsduren gepropageerd worden door het temporele netwerk, zodat er mogelijk een nieuw tijd- en resource-feasible schedule gevonden wordt, zonder het herberekenen van de gehele schedule en het toevoegen van nieuwe voorrangsrelaties. Oftewel, naast de volledige solve-optie, moet ook een lichtgewichte doorrekening mogelijk zijn.

Bij de evaluatie door de Software Improvement Group is gebleken dat de code lager scoort op Component Independence. Hierbij werd er gekeken naar de hoeveelheid code die alleen binnen een bepaald deel van het programma wordt gebruikt, en niet wordt aangeroepen vanuit andere delen van het programma. Vooral de cyclische afhankelijkheid tussen de componenten *widgets* en *controller* is hierbij opgevallen. Een oplossing voor dit probleem is het scheiden van de verantwoordelijkheden, door de controller te laten reageren op acties van de gebruiker binnen de widgets. Deze ingrijpende wijziging hebben wij niet meer kunnen doorvoeren in de opgeleverde versie. Er wordt aangeraden om dit in de toekomst alsnog te doen, zodat de onderhoudbaarheid van het programma verbetert.

Er zijn plannen om een nieuw bachelorproject te laten uitvoeren om het programma verder te ontwikkelen. Omdat het programma inmiddels zo ver is doorontwikkeld, zal het maken van een requirements analyse en een ontwerp slechts bestaan uit het aanvullen van ons werk. Het is dus belangrijk om een duidelijk afgebakende opdracht te formuleren, die het mogelijk maakt om een compleet traject van softwareontwikkeling uit te voeren.

# 9 Bibliografie


[1] Erik Ammerlaan, Jan Elffers, Erwin Walraven, and Wilco Wisse. Precedence constraint posting for robust scheduling. In *Seminarium IN3305, Student Papers*, pages 75–88, 2012.

[2] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98*, pages 214–223. AAAI Press, 1998.

[3] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *In AAAI/IAAI*, pages 742–747, 2000.

[4] Ronald Paulus Evers. Algorithms for scheduling of train maintenance. Master's thesis, Delft University of Technology, Faculty of EEMCS, 2010.

[5] S. Katsavounis. Scheduling multiple concurrent projects using shared resources with allocation costs and technical constraints. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1 –6, april 2008.

[6] Michel Wilson, Cees Witteveen, and Bob Huisman. Enhancing predictability of schedules by task grouping. In *Proceedings of the 20th European Conference on Artificial Intelligence*. IOS Press, aug 2012 (to appear).

# Appendix A: Opdrachtomschrijving

# Opdrachtomschrijving

Het bachelorproject zal worden uitgevoerd in opdracht van NedTrain en de Technische Universiteit Delft. De opdrachtgevers doen samen onderzoek naar efficiëntere methoden om treinonderhoud te plannen. Inmiddels zijn er enkele algoritmen ontwikkeld die onderhoud kunnen plannen. Het doel van het project is om een grafische interface te ontwikkelen die het resultaat van het planningsproces op een visuele manier kan tonen. Hierbij worden de algoritmen en reeds bestaande software als startpunt gebruikt. Deze software is ontwikkeld door andere studenten in het kader van een afstudeerproject en een bachelorproject.

Het resulterende programma zal gebruikt worden voor verschillende doelen:

- De applicatie kan gebruikt worden bij het onderzoek naar nieuwe algoritmen. De interface kan hierbij bijvoorbeeld inzichtelijk maken wat de verschillen zijn in het resultaat wanneer er verschillende planningsalgoritmen worden toegepast. De doelgroep die hiermee wordt bediend bestaat dus uit onderzoekers.
- Voor werknemers van NedTrain kan de applicatie dienen als 'proof of concept'. Mensen zonder achtergrond in de informaticawereld zijn zich vaak niet bewust van alle mogelijkheden op dit gebied. Het programma kan dus gebruikt worden om te laten zien wat software allemaal kan betekenen voor de planningswerkzaamheden. De doelgroep bestaat in dit geval uit werknemers van NedTrain op het gebied van onderhoudplanning.
- Tijdens een summerschool die in de zomer geörganiseerd zal worden is er een planningsapplicatie nodig die gebruikt kan worden om verschillende aspecten van planning te laten zien. Het publiek bij deze summerschool zal bestaan uit internationale studenten. Voor deze doelgroep moet de applicatie dus een educatief karakter hebben, en het moet niet specifiek gericht zijn op de toepassing bij NedTrain.

Omdat er al meerdere applicaties ontwikkeld zijn, is het niet de bedoeling om de volledige applicatie vanaf de grond op te bouwen. Tijdens de opdracht zal geïnventariseerd moeten worden welke delen van de bestaande applicaties te gebruiken zijn en welke gebreken er precies zijn. Er zullen aanvullende eisen gesteld worden, die samen met de bestaande functionaliteiten verwerkt moeten worden in de te bouwen applicatie.

Vanuit de opdrachtgevers worden er enkele ideeën voorgedragen die mogelijk verwerkt kunnen worden in de applicatie. Er wordt niet verwacht dat alles wordt geïmplementeerd. Tijdens het project kan er naar eigen inzicht een selectie gemaakt worden hieruit. Enkele voorgestelde functionaliteiten zijn:

- Het stap voor stap kunnen simuleren van een planningsproces, zodat het duidelijk te zien is welke beslissingen het algoritme in iedere stap neemt.
- Zichtbaar maken welke constraints er bestaan tussen verschillende taken.
- Het verwijderen van constraints nadat een planning is gemaakt, of om ervoor te zorgen dat de set van constraints weer toegelaten is.
- Het handmatig kunnen wijzigen van een planning nadat het algoritme een oplossing heeft bepaald. Hierbij kan een planner bijvoorbeeld naar eigen inzicht een taak verschuiven.
- Ondersteuning voor verschillende talen, zodat de software voor verschillende doelen kan worden gebruikt.
- Een starttijdstip toekennen aan een gemaakte planning, zodat getoond kan worden welke taken reeds uitgevoerd zouden moeten zijn. Eventueel kunnen op basis hiervan aanpassingen worden gedaan.

## Appendix B: Oriëntatieverslag

IN3405 BACHELORPROJECT

# Oriëntatieverslag

Bron: http://www.qii12.com/nl/img/content/extra/cases/Nedtrain_leidschendam.jpg

Erik Ammerlaan
Jan Elffers
Erwin Walraven
Wilco Wisse

4 juli 2012

# Inhoudsopgave

# 1 Inleiding

Dit document is onderdeel van het bachelorproject aan de Technische Universiteit Delft. Gedurende dit project wordt er een onderhoudsplanner ontwikkeld voor NedTrain. Hierbij zullen eerder ontwikkelde applicaties als startpunt worden gebruikt.

NedTrain is een bedrijf dat verantwoordelijk is voor het onderhoud aan treinstellen van NS. Hierbij speelt het plannen van het onderhoud een belangrijke rol. Op dit moment vindt er, in samenwerking met de Technische Universiteit Delft, een onderzoek plaats naar het efficiënt plannen van onderhoud. De opdracht die uitgevoerd zal worden is het ontwikkelen van een onderhoudsplanner. Het doel van deze applicatie is het laten zien van de mogelijkheden die er zijn op het gebied van software om de bestaande bedrijfsprocessen te ondersteunen.

Dit verslag beschrijft het vooronderzoek dat is uitgevoerd in de eerste week van het project. Hierbij is gekeken naar de achtergrond van het bedrijf NedTrain, de bestaande software en de ontwikkelmethoden die gebruikt kunnen worden tijdens het project. In sectie 2 wordt een beschrijving van het bedrijf NedTrain geschreven. Sectie 3 en 4 geven een overzicht van de bestaande software en de solver die daarbij wordt gebruikt. Sectie 5 beschrijft de oriëntatie die is uitgevoerd naar geschikte applicaties en tools om te gebruiken bij de ontwikkeling.

# 2 NedTrain

In deze sectie wordt een kort overzicht gegeven van de bedrijven NedTrain en NS. Hierbij zullen hun werkzaamheden die relevant zijn voor het project worden besproken.

## 2.1 Organisatie

De NS groep bestaat uit twee bedrijfsonderdelen: reizigersvervoer en knooppuntontwikkeling. Reizigersvervoer bestaat uit dochterondernemingen zoals NS reizigers (verantwoordelijk voor het rijden van treinen en serviceverlening), NS Hispeed (hogesnelheidsvervoer naar internationale bestemmingen) en NedTrain. Knooppuntontwikkeling bestaat uit dochterondernemingen van NS die zich bezig houden met activiteiten in en rond stations. Ondernemingen zoals NS Fiets en Servex behoren hier toe [5].

De taak van NedTrain betreft het onderhoud aan treinstellen. Onderhoud aan treinstellen kan worden onderverdeeld in drie categorieën: servicebeurten, technisch onderhoud en renovatie. Bij servicebeurten worden controles afgehandeld, wordt er schoongemaakt en als het mogelijk is worden kleine technische mankementen opgelost. Technisch onderhoud betreft het doen van uitgebreidere onderhoudswerkzaamheden aan treinstellen. Renovatie behelst het uitvoeren van alle vormen van onderhoud die niet regelmatig voorkomen, zoals modernisering van treininterieurs en het uitvoeren van langetermijnrevisie. NedTrain heeft meer dan 30 locaties, verspreid over heel Nederland. De onderhoudscentra verschillen in grootte en hebben hun eigen specialisatie. Zo vind het langetermijnonderhoud plaats bij de werkplaats ‘Refurbishment en overhaul’ in Haarlem [6].

## 2.2 Onderhoudsplanning

In dit bachelorproject richten we ons op het plannen van technisch onderhoud. NedTrain heeft vier locaties waar technisch onderhoud wordt uitgevoerd: in Amsterdam, Leidschendam, Onnen en Maastricht.

Voor onderhoud aan treinstellen heeft NedTrain de beschikking over een aantal resources. Elke werkplek gebruikt één of meerdere speciale spoortypes om dit onderhoud uit te voeren [4]. Deze spoortypes zijn:

**Putspoor:** een spoor waarbij de vloer verlaagd of het spoor verhoogd is. Het meeste onderhoud maakt hier gebruik van. Het putspoor maakt onderhoud aan de onderkant van de trein makkelijker;

**Kuilwielbank:** op dit spoor worden wielen van de trein opnieuw vorm gegeven;

**Aardwind:** een spoor waarbij er een lift en een kraan aanwezig zijn om draaistellen te plaatsen;

**ATB:** ATB staat voor automatische trein beïnvloeding. Deze gespecialiseerde rail is gemaakt om veiligheidssystemen van treinen te testen.

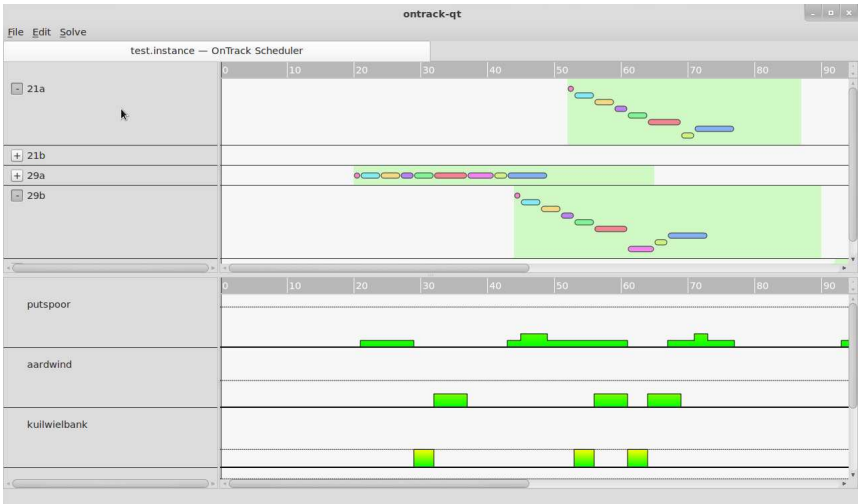
Momenteel wordt de onderhoudsplanning handmatig uitgevoerd. Hierbij moet rekening worden gehouden met de capaciteit van de resources en het tijdbestek waarin het onderhoud aan een trein moet worden uitgevoerd.

# 3 Bestaande software

In deze sectie wordt reeds bestaande software beschreven die als startpunt gebruikt zal worden voor de planningssoftware. Er wordt een analyse gegeven van de functionaliteiten, hun tekortkomingen en mogelijke verbeteringen. Beide programma's zijn ontwikkeld in de taal C++ met behulp van het crossplatform Qt framework.

## 3.1 OnTrack Scheduler

De OnTrack Scheduler is een eenvoudige interface die ontwikkeld is door Ronald Evers [4]. Een overzicht hiervan is te zien in figuur 3.1. Het biedt een visuele representatie van jobs en bijbehorende activiteiten, alsmede de resources en het verbruik daarvan. Het is mogelijk om eenvoudige instanties te maken door resources, jobs en activiteiten toe te voegen. Nadat een externe solver is geselecteerd kan de probleeminstantie opgelost worden.




Figuur 3.1: OnTrack Scheduler interface

De interface is op sommige plaatsen niet intuïtief, waardoor er relatief snel fouten worden gemaakt. Het herstellen van deze fouten is echter niet mogelijk, wat het gebruik van het programma lastig

maakt in de praktijk. Het programma sluit soms af, als de solver wordt aangeroepen nadat er wijzigingen zijn gedaan aan de instantie.

Het is duidelijk dat de software niet de noodzakelijke basisfunctionaliteiten heeft en niet gebruiksvriendelijk genoeg is. Het is echter wel een mooi startpunt omdat de code relatief compact en netjes is en de interface inspiratie biedt voor een verbeterde versie.

## 3.2 Voorgaand bachelorproject

Edwin Bruurs en Cis van der Louw hebben in het kader van een eerder bachelorproject een alternatieve interface ontwikkeld [2]. Hierbij is de scheduler van Ronald Evers als uitgangspunt gebruikt. Het idee achter de software is dat het gebruiksvriendelijker moet zijn om planningen te maken, waarbij jobs en activiteiten kunnen worden toegevoegd. Deze planningen kunnen worden opgeslagen in een MySQL-database. De functionaliteiten werken redelijk goed, maar de software is duidelijk niet klaar en niet robuust genoeg. Het programma is op sommige plaatsen inconsistent op het gebied van taal; Nederlands en Engels worden door elkaar gebruikt.

Na het bekijken van de code is gebleken dat er tijdens het ontwikkelen niet getest is. De structuur in de code is niet in orde, omdat verantwoordelijkheden niet goed gescheiden zijn. Er wordt een model-view-controller-model toegepast, maar niet geheel op de juist wijze. Code die gerelateerd is aan views is ook in controller klassen terug te vinden. Dit had beter gescheiden moeten blijven. Ook is niet op alle plaatsen het OO-principe correct toegepast. Het zou beter zijn om het afhandelen van excepties abstracter aan te pakken. Ondanks de tekortkomingen, kunnen bepaalde delen van de code voor de grafische interface opnieuw gebruikt worden in een verbeterde versie.

## 3.3 Bruikbaarheid

Omdat het niet wenselijk is om volledig vanaf de basis te beginnen, is er gekeken naar de bruikbaarheid van de huidige programma's. De OnTrack Scheduler is een goede basis, omdat de functionaliteiten die er zijn goed werken en de bijbehorende code is netjes. Verder bouwen op de software van het vorige bachelorproject is lastig, omdat de functionaliteiten niet goed genoeg werken en er een hoop verbetering noodzakelijk is voordat er nieuwe functionaliteiten toegevoegd kunnen worden. Door het ontbreken van tests is het onmogelijk om te zien of het toevoegen van nieuwe functionaliteiten heeft geleid tot nieuwe problemen. De code van de interfaces kan wel opnieuw gebruikt worden en kan eventueel worden verbeterd op plaatsen waar dat nodig is.

# 4 Oplosmethode

In deze sectie wordt het planningsprobleem dat centraal staat in de applicatie toegelicht. Er zijn verschillende entiteiten in het probleem: we onderscheiden *resources*, *treinen*, *taken*, *requirements* en *precedence constraints*. In dit probleem zijn er een aantal treinen waarop taken moeten worden uitgevoerd. Hiervoor moeten we een *schedule* vinden. Elke trein heeft een *release date* en *due date*, wat inhoudt dat taken op een trein vanaf de *release date* kunnen worden uitgevoerd en klaar moeten zijn op de *due date* in een geldig schedule. Verder zijn er *resources* (grondstoffen) die herbruikbaar zijn maar een beperkte (parallelle) capaciteit hebben. Taken kunnen resources gebruiken (dit wordt gemodelleerd door *requirements*) en in geldige schedules mag de capaciteit van de resources nooit overschreden worden. Ten slotte zijn er nog *precedence constraints*, die specificeren dat er een afhankelijkheid tussen twee taken is. Dit houdt in dat een bepaalde taak klaar moet zijn voordat een andere begint. In dit probleem geldt verder dat op elke trein maximaal één taak tegelijk kan worden uitgevoerd.

Het bestandsformaat waar instanties van dit probleem in opgeslagen worden bestaat uit regels die individuele entiteiten specificeren:

- `'R' <resource_id:int> <capacity:int> <name:string>`: specificatie van de resource.
- `'T' <trein_id:int> <release_date:int> <due_date:int> <name:string>`: specificatie van de trein.
- `'A' <trein_id:int> <activity_id:int> <duration:int> <name:string>`: specificatie van een taak voor een trein.
- `'Q' <trein_id:int> <activity_id:int> <resource_id:int> <amount:int>`: specificatie dat een bepaalde taak een bepaald verbruik bij een resource heeft.
- `'P' <trein1_id:int> <activity1_id:int> <trein2_id:int> <activity2_id:int>`: specificatie van een precedence constraint tussen twee taken.

Het doel is dus om een *goed* feasible schedule te maken waarin al deze taken worden gekoppeld aan een begintijd, zodat de herbruikbare resources nooit overbelast zijn, aan de precedence constraints voldaan is en slechts één taak tegelijk per trein plaatsvindt.

Dit probleem kan worden geformuleerd als een Resource Constrained Project Scheduling Problem (RCPSP) probleem. Dit probleem is hetzelfde gedefinieerd, behalve dat de activiteiten daar niet gegroepeerd zijn (zoals hier, per trein). De eis dat er slechts één taak tegelijk mag plaatsvinden per trein kan worden vertaald door voor elke trein een resource met capaciteit 1 toe te voegen, en voor elke taak een requirement van 1 van deze kunstmatige resource van de bijbehorende trein toe te voegen. Voor een formele definitie van het RCPSP en uitleg over oplossingsmethoden verwijzen we naar de paper van het Bachelorseminarium [1].

We weten af van twee bestaande solvers, een van Ronald Evers (ontwikkeld in het kader van zijn afstuderen, zie ook zijn master these [4]) en een van Michel (een van onze begeleiders). Beide zijn gebaseerd op het ESTA-algoritme [3], en het verschil zit vooral in de efficiëntie. De solver van Michel is veel sneller. Het ESTA-algoritme geeft een relatief goed schedule in beperkte rekentijd maar in het algemeen geen optimaal schedule. In de uitvoer van de solver staan ook de precedence constraints die zijn toegevoegd tijdens het oplossen.

# 5 Softwareontwikkeling

In deze sectie wordt het vooronderzoek beschreven naar ontwikkelmethodieken die gebruikt kunnen worden tijdens het uitvoeren van het project. Tevens wordt beschreven welke tools zijn overwogen.

## 5.1 Methodiek

Tijdens het ontwikkelen willen wij bepaalde principes van *scrum* toepassen. Scrum is een procesraamwerk dat gebruikt wordt om complexe productontwikkeling te managen, zodat de effectiviteit van een ontwikkelteam toeneemt [7].

Scrum definieert drie rollen: een *product owner*, *scrum master* en een *ontwikkelteam*. De scrum master is een persoon die controleert of iedereen zich aan de regels van scrum houdt. De product owner is één persoon die de belangen van de klant behartigt en met het ontwikkelteam de producteisen bespreekt. Het belangrijkste begrip bij scrum is een *sprint*. Een sprint is een periode van een maand of minder waarbinnen een increment wordt gemaakt, die een werkend product is. Bij elke sprint wordt er nieuwe functionaliteit aan het product toegevoegd, zodanig dat er aan het einde van elke sprint een werkende versie met nieuwe functionaliteit is. Aan het begin van een sprint wordt een *planningsbijeenkomst* gehouden om te beslissen wat er binnen de sprint gaat gebeuren. De taken voor een sprint worden geselecteerd uit een *product backlog*. De product backlog bevat

alle eisen, features en taken voor het hele product. De product owner is verantwoordelijk voor het bijhouden van deze lijst en mag aan taken een prioriteit of waardering toekennen. Ontwikkelaars mogen aan taken een schatting van de benodigde werktijd geven. Taken op een product backlog moeten klein zijn, zodat ze binnen een sprint zijn af te maken. Tijdens het ontwikkelproces kan de product backlog veranderen, als de product owner met extra eisen of wensen komt.

Aan het begin van een sprint worden taken uit de product backlog geselecteerd die tijdens de sprint geïmplementeerd gaan worden. Deze taken worden overgeheveld naar een *sprint backlog*. Nadat beslist is wat er gedaan gaat worden, moet er nog besloten worden hoe deze toe te voegen functionaliteit uiteindelijk resulteert in een werkende productincrement aan het einde van de sprint. Er wordt alvast een planning voor de eerste dagen van de sprint gemaakt. Tevens wordt er een *sprintdoel* geformuleerd, die aangeeft wat aan het einde van de sprint bereikt moet zijn. De sprint backlog kan tussenstijds veranderen, als het echt moet, maar het sprintdoel moet altijd worden nagestreefd.

Het ontwikkelteam houdt een *dagelijkse scrum*. Dat is een bespreking van ongeveer 15 minuten waarin een plan wordt gemaakt voor de komende 24 uur en waarin de voortgang ten opzichte van de sprint backlog wordt besproken. Aan het einde van een sprint wordt een *sprint review* gehouden, waar de nieuwe productversie wordt gepresenteerd aan de product owner en waarbij de product backlog wordt gesproken. Uiteindelijk volgt er ook nog een *sprint retrospective*, waarin het team reflecteert op hoe de scrum-techniek is toegepast. Daarna kan aan een nieuwe sprint worden begonnen.

Wij zullen delen van scrum toepassen tijdens ons ontwikkelproces. In het verleden hebben wij meestal gewerkt volgens het watervalmodel en wij hopen ons ontwikkelproces nu meer *agile* te maken. Wij zullen gaan werken met één- of tweewekelijkse sprints, waarbij we aan het eind van iedere sprint een nieuwe versie van het programma hebben met toegevoegde functionaliteit. We zullen voorafgaand aan een sprint een planningsbijeenkomst houden en daarna een dagelijkse scrum houden. De rol product owner vervalt, omdat we niet van onze opdrachtgevers willen eisen dat zij handelen volgens de scrum-principes. We hebben daarnaast geen scrum master; we moeten dus zelf in de gaten houden of we ons aan de regels houden. De sprint review en sprint retrospective kunnen hierdoor ook moeilijk uitgevoerd worden. Wel zullen we regelmatig de werkende versie van de laatste sprint tonen aan de projectbegeleiders.

## 5.2 Hulpmiddelen

De software wordt geprogrammeerd in C++, aangezien de reeds bestaande versie ook al in C++ is geschreven. Hierbij wordt gebruik gemaakt van het Qt-framework voor het programmeren van de grafische gebruiksinterface. Wij zullen de ontwikkelomgeving Eclipse gebruiken. Met behulp van de 'Qt Eclipse Integration for C++' is het maken van een grafische interface vereenvoudigd. De software zal ontwikkeld en gecompileerd worden op Linux. In theorie zou de software ook op andere besturingssystemen moeten werken, aangezien C++ en Qt cross-platform zijn, maar wij zullen ons in eerste instantie op Linux richten. Mocht er voldoende tijd over zijn, dan kunnen we de mogelijkheden op Windows verkennen.

De programmeercode zal beheerd worden door een 'version control system', namelijk SVN. De code zal toegelicht worden met commentaar dat voldoet aan de Doxygen-syntax, zodat er met Doxygen externe documentatie van de code gegenereerd kan worden. Om meldingen over bugs bij te houden zal Mantis gebruikt worden. Deze applicatie niet geheel compatibel met de Scrum filosofie, maar er kunnen Scrum-plugins worden gebruikt om functionaliteiten toe te voegen. Het voordeel hiervan is dat meldingen gestructureerd kunnen worden bijgehouden op een Scrum board.

Om de kwaliteit van de code te bewaken zullen we diverse hulpmiddelen inzetten. Allereerst zullen wij unit tests schrijven met het framework Google Test. Het voordeel van dit framework is dat

het compact is en compatibel is met het continuous integration platform. Daarnaast gebruiken we de plugin Cppcheck voor het doen van statische analyse. Cppcheck detecteert fouten die de C++-compiler standaard niet detecteert. Zo controleert Cppcheck bijvoorbeeld op geheugenlekken, ongeïnitialiseerde variabelen en functies en overschrijdingen van arrays.

Om het proces van codekwaliteit te automatiseren maken we gebruik van een Jenkins-server voor 'continuous integration'. Jenkins kan de SVN-server in de gaten houden voor updates. Wanneer er updates zijn, zal Jenkins het programma automatisch opnieuw bouwen, alle unit tests draaien en statische analyse met behulp van Cppcheck doen. Op die manier is er altijd na te gaan of er een werkende versie is en zijn er automatisch coverage-statistieken beschikbaar. Wanneer wij dat tijdens het ontwikkelproces handig vinden, is Jenkins met plugins nog verder uit te breiden.

# Bibliografie


[1] Erik Ammerlaan, Jan Elffers, Erwin Walraven, and Wilco Wisse. Precedence constraint posting for robust scheduling. In *Seminarium IN3305, Student Papers*, pages 75–88, 2012.

[2] Edwin Bruurs and Cis van de Louw. Onderhoudsplanner Nedtrain. 2011.

[3] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *AIPS*, pages 214–231, 1998.

[4] Ronald Paulus Evers. Algorithms for scheduling of train maintenance. Master's thesis, Delft University of Technology, Faculty of EEMCS, 2010.

[5] NS groep. Over NS. `http://www.ns.nl/over-ns`, 2012.

[6] NedTrain. Over NedTrain. `http://nedtrain.nl/over-nedtrain/`, 2012.

[7] Ken Schwaber and Jeff Sutherland. De Scrumgids – de definitieve gids voor Scrum: de regels van het spel. 2011.

# Appendix C: Plan van aanpak

IN3405 BACHELORPROJECT

# Plan van aanpak

Bron: http://www.treinennieuws.nl/images/OB-Amsterdam-22_2007-5-10-584_kl.jpg

Erik Ammerlaan
Jan Elffers
Erwin Walraven
Wilco Wisse

4 juli 2012

# Inhoudsopgave

# 1 Inleiding

In dit verslag is het Plan van Aanpak opgesteld. In het Plan van Aanpak worden alle afspraken tussen opdrachtgever en de projectleden vastgelegd. Verder wordt beschreven op welke manier het project uitgevoerd gaat worden. Het plan van aanpak wordt voorgelegd aan de begeleiders. Indien gewenst kunnen tussentijdse wijzigingen besproken worden met de opdrachtgever. Het plan van aanpak moet zowel de goedkeuring hebben van opdrachtgever als opdrachtnemer.

Dit project wordt uitgevoerd voor NedTrain. NedTrain is een dochteronderneming van NS groep die verantwoordelijk is voor het onderhoud van van treinmaterieel. In het Oriëntatieverslag wordt NedTrain en het takenpakket van deze onderneming nader toegelicht. NedTrain werkt samen met de Technische Universiteit Delft om een applicatie te ontwikkelen om het onderhoud aan treinstellen zo efficiënt mogelijk in te plannen. Het achterliggende algoritme is reeds ontworpen en geïmplementeerd. In dit project ontwikkelen we een grafische interface, zodat de planner het algoritme kan gebruiken om zijn werkzaamheden te ondersteunen.

In hoofdstuk 2 wordt de opdracht omschreven zodat de opdrachtgever en de projectmanager helder hebben waaraan de applicatie moet voldoen. Vervolgens wordt in hoofdstuk 3 de aanpak beschreven om aan deze opdrachtomschrijving te voldoen. Daarna wordt de inrichting van het project besproken in hoofdstuk 4. Ten slotte wordt in hoofdstuk 5 uitgelegd welke maatregelen getroffen worden om de kwaliteit van het product te waarborgen.

# 2 Opdrachtomschrijving

In deze sectie zal een overzicht gegeven worden van de opdracht die uitgevoerd zal worden tijdens het project. Hierbij wordt vastgelegd wat de betrokken partijen zijn en wat hun belang bij het project is.

## 2.1 Opdrachtgever

De opdrachtgever voor dit project bestaat uit twee partijen: NedTrain en de Technische Universiteit Delft. NedTrain voert, samen met de universiteit, onderzoek uit naar het efficiënt plannen van onderhoud. Hieronder een overzicht van de contactpersonen die betrokken zijn bij het project.

- Cees Witteveen – begeleider – c.witteveen@tudelft.nl
- Michel Wilson – begeleider – m.wilson@tudelft.nl
- Bob Huisman – begeleider – b.huisman@nedtrain.nl
- Peter van Nieuwenhuizen – coördinator – p.r.vannieuwenhuizen@tudelft.nl
- Gerd Gross – coördinator – h.g.gross@tudelft.nl

## 2.2 Probleemstelling

NedTrain houdt zich bezig met het onderhouden van treinen voor NS. Om dit onderhoud te plannen wordt er in samenwerking met de Technische Universiteit Delft onderzoek gedaan. Hierbij is het wenselijk om een grafische applicatie te hebben die kan werken met het algoritme dat de planning maakt. Voor het onderzoek naar verbeteringen in algoritmen is een interface nuttig, omdat het op een intuïtieve en eenvoudige manier kan laten zien hoe een oplossing eruit ziet, en waarom er bepaalde keuzes zijn gemaakt. Voor NedTrain is het wenselijk om inzicht te krijgen in de mogelijkheden die software biedt voor hun bedrijfsproces. De wens voor een geschikte grafische interface komt dus vanuit beide betrokken partijen.

## 2.3 Opdracht en doelstelling

Tijdens het project zal er een gebruiksvriendelijke grafische interface ontwikkeld worden die gebruik kan maken van bestaande algoritmen. Het voornaamste doel hiervan is het aantonen van de mogelijkheden die software heeft op het gebied van plannen. Tevens dient het als ondersteuning bij het onderzoek naar nieuwe of verbeterde algoritmen.

- Het programma zal geen productieversie zijn die wordt ingepast in een bestaand bedrijfsproces. Het doel is om werknemers te laten ervaren hoe het is om te werken met een planningsapplicatie en in welk opzicht dit een aanvulling vormt op het werk dat er plaatsvindt. Het moet inzichtelijk maken welke stappen er tijdens het plannen worden gezet en hoe de invoer van de gebruiker de resulterende planning beïnvloedt. De applicatie die wordt gemaakt zal dus dienen als een ‘proof of concept’.
- Het programma zal gedurende de zomer gebruikt worden tijdens een summerschool met internationale studenten. Deze doelgroep moet de software kunnen gebruiken om verschillende algoritmen te kunnen uittesten. Hierbij is het belangrijk dat de gebruikte terminologie in het programma algemeen is en niet specifiek gericht op NedTrain.

- De interface die wordt ontwikkeld moet intuïtief zijn, zodat mensen die minder thuis zijn met computerprogramma's er eenvoudig mee om kunnen gaan. De interface moet voldoende duidelijkheid geven, zodat de gebruiker kan ervaren wat er bij de planning van het onderhoud komt kijken. Er zijn geen belangrijke eisen op het gebied van platform. Het is wel wenselijk om meerdere mogelijkheden open te houden. Op het gebied van taal moet er rekening gehouden worden met de doelgroep. Dit kan tijdens het project naar eigen inzicht worden ingevuld.
- Omdat het programma gebruikt zal worden om concepten te illustreren wordt er niet verwacht dat het programma in een professionele bedrijfsomgeving wordt ingepast. Er zijn echter wel functionaliteiten die in de toekomst mogelijk toegevoegd moeten worden. De software zal dus geschikt moeten zijn om verder uitgebreid te worden.

Het uiteindelijk op te leveren product zal bestaan uit een grafische interface, die gebruikt kan worden in combinatie met bestaande algoritmen. Indien wenselijk kan er een korte handleiding bijgeleverd worden die het mogelijk maakt om de software eenvoudig te installeren. De tussentijdse versies zullen voorgelegd worden aan de opdrachtgever, maar zijn geen officieel product.

De suggesties die door de opdrachtgevers worden gedaan voor functionaliteiten zijn geen strikte eisen. Er kan naar eigen inzicht een selectie gemaakt worden die geïmplementeerd kan worden gedurende het project. Eigen inbreng of nieuwe ideeën zijn ook welkom. Dit zal wel vooraf besproken worden met de opdrachtgevers. De volgorde waarin de functionaliteiten worden geïmplementeerd zal in een volgende fase vastgelegd worden in samenspraak met de opdrachtgevers.

## 2.4 Randvoorwaarden

Van de deelnemers aan het project wordt er verwacht dat er tot de eerste week van juli wordt gewerkt aan het project. Er wordt verwacht dat er een geschikte planning wordt nageleefd zodat de op te leveren producten op tijd klaar zijn en aan de gestelde eisen voldoen. Een randvoorwaarde is dat we dienen uit te gaan van code die eerder door andere studenten is geschreven en dat de software moet werken met externe solvers, waaronder die van Michel Wilson. Van de begeleiders wordt verwacht dat er gedurende het project feedback wordt geleverd, zodat dit verwerkt kan worden voordat de definitieve versie wordt opgeleverd. Ook dienen de begeleiders op afspraak beschikbaar te zijn om feedback te leveren, eventuele problemen te bespreken of informatie te geven over de gestelde eisen aan het product.

# 3 Aanpak

In deze sectie geven we aan welke manier we gebruiken om aan de in de vorige sectie gestelde eisen te kunnen voldoen. Hierbij wordt de fasering van het project besproken.

## 3.1 Aanpak

We onderscheiden bij het project de volgende activiteiten:

**Oriëntatie** Tijdens de oriëntatiefase wordt het Oriëntatieverslag en het het Plan van Aanpak opgesteld. Naast het schrijven van deze documenten wordt geïnventariseerd welke bestaande programmatuur er is en wat de kwaliteit hiervan is. Verder wordt er gekeken naar de tools die we gaan gebruiken bij de implementatiefase en de bijbehorende programmeertalen.

**Requirements analyse** Bij de requirements analyse wordt geïnventariseerd aan welke eisen het systeem moet voldoen en welke functionaliteit aanwezig moet zijn. De opgestelde eisen worden voorgelegd aan de opdrachtgever, zodat eventuele wijzigingen naar wens kunnen worden toegevoegd.

**Technisch ontwerp** Het technisch ontwerp betreft het globaal ontwerp van het gehele systeem.

**Implementatie** Op basis van de requirements en het technisch ontwerp wordt het systeem geïmplementeerd en worden tests geschreven. Hierbij wordt Scrum gebruikt. Voor elke sprint wordt gekozen welke functionaliteit geïmplementeerd gaat worden. Na de sprint is het van belang dat we feedback krijgen van de opdrachtgever, zodat we snel kunnen inspelen op extra eisen.

Tussentijds moet de programmacode twee keer worden ingeleverd bij de Software Improvement Group (SIG). Bij SIG wordt de code beoordeeld op kwaliteit, deels met geautomatiseerde tools en deels met de hand. De eerste deadline hiervoor is 10 juni 2012, 23:59. Na ongeveer een week krijgen we feedback die gebruikt moet worden om de code te verbeteren. De definitieve code moet uiterlijk 5 dagen voor de datum van de eindpresentatie opnieuw worden ingeleverd.

**Afronding** Onder afronding valt het schrijven van het eindverslag en het voorbereiden van de eindpresentatie.

Op basis van een schatting van de nodige resources en tijd voor de verschillende activiteiten wordt een tijdsplanning gegeven in appendix A.

# 4 Projectinrichting

In deze sectie zal besproken worden welke personen betrokken zijn bij het project. Ook zal worden ingegaan op de faciliteiten die tijdens het uitvoeren van het project gebruikt zullen worden. Ten slotte wordt kort besproken welke informatiebronnen geraadpleegd kunnen worden.

## 4.1 Betrokkenen

Vanuit NedTrain en de Technische Universiteit zijn er meerdere personen betrokken bij het project. De begeleiding vanuit de universiteit is in handen van professor Cees Witteveen en Michel Wilson. Namens NedTrain zal Bob Huisman ons begeleiden. De coördinatoren van het bachelorproject zijn Peter van Nieuwenhuizen en Gerd Gross.

## 4.2 Faciliteiten

Tijdens het uitvoeren van het project wordt gebruikgemaakt van de computerfaciliteiten van de universiteit. Deze zijn, voor zover mogelijk, gereserveerd door de coördinatoren. Er kan eventueel in Utrecht worden gewerkt bij NedTrain, maar de faciliteiten zijn daar matig. Er is bijvoorbeeld geen stabiele internetverbinding beschikbaar en er zijn geen systemen waarop Linux is geïnstalleerd. Voor de ontwikkeling zal gebruikgemaakt worden van open source software. Er zijn dus geen extra softwarelicenties noodzakelijk.

## 4.3 Informatie

Voor de informatie met betrekking tot de gebruikte algoritmen kunnen we terugvallen op de paper die het team heeft geschreven als onderdeel van het vak IN3305 Bachelorseminarium [1]. Verdere vragen in dit opzicht kunnen worden gesteld aan Michel Wilson. Voor vragen met betrekking tot de processen bij NedTrain kunnen we terecht bij Bob Huisman.

Er zijn enkele relevante boeken met betrekking tot het programmeren in C++ met Qt. Boeken die we gaan gebruiken zijn ‘C++ GUI programming with Qt 4’ van Jasmin Blanchette [2] en ‘An introduction to design patterns in C++ with Qt’ van Alan Ezust [3].

# 5 Kwaliteitsborging

In deze sectie zal besproken worden hoe de kwaliteit gewaarborgd zal zijn tijdens het uitvoeren van het project. Hierbij speelt continuous integration een belangrijke rol. Ook wordt de documentatie, versiebeheer en de teststrategie kort besproken.

## 5.1 Kwaliteitbewaking

Om de kwaliteit tijdens het proces te bewaken zal gebruikgemaakt worden van een platform voor continuous integration. Dit biedt als voordeel dat het testen geautomatiseerd kan worden en dat problemen relatief snel aan het licht komen. Tevens zal er gebruikgemaakt worden van een bugtracker, zodat meldingen op een gestructureerde manier worden bijgehouden en inzichtelijk zijn voor het hele team.

## 5.2 Documentatie

De ontwikkelde software zal gedocumenteerd worden met in-line commentaar, dat gebruikt kan worden om documentatie te genereren. Om ervoor te zorgen dat dit consistent gebeurt zal er een tool gebruikt worden die alle code doorlicht en bekijkt of er commentaar staat op de juiste plaatsen. Dit zal bij iedere build op het continuous integration platform worden gecontroleerd.

## 5.3 Versiebeheer

Er zal gebruikgemaakt worden van een server waarop Subversion geïnstalleerd staat. Deze wordt ter beschikking gesteld aan studenten door de faculteit. Er is gekozen om Subversion te gebruiken omdat de leden van het team hier al mee bekend zijn. Een andere keuze zou het gedistribueerde versiebeheersysteem Git kunnen zijn, maar omdat Subversion voldoet aan alle eisen is dit alternatief niet uitgebreid overwogen.

## 5.4 Testen

Tijdens het ontwikkelen zal er uitgebreid getest worden. Er wordt naar gestreefd om test driven development toe te passen, waarbij tests worden ontwikkeld voordat de bijbehorende functionaliteit wordt geïmplementeerd. Er zullen eenvoudige unit tests geschreven worden om de software op laag niveau te kunnen controleren. Om te controleren of basisfunctionaliteiten volledig werken zal acceptance testing worden toegepast. Een robuuste testsuite, in combinatie met continuous integration, zorgt ervoor dat nieuw geïntroduceerde problemen snel aan het licht komen en verholpen kunnen worden. Het doel is dat er altijd een werkende versie van de software beschikbaar is. Dit wordt gewaarborgd door het automatisch testen bij iedere commit.

## 5.5 Evaluatie

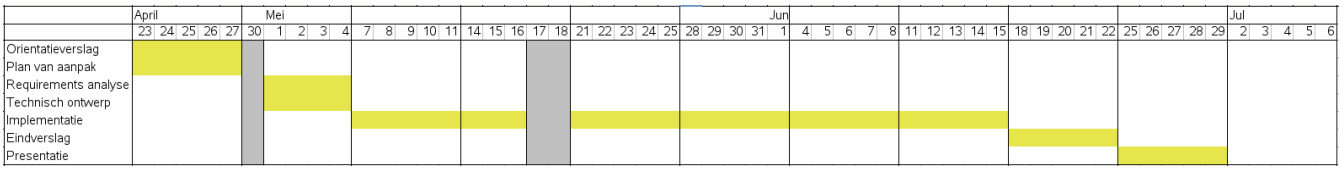
Tijdens het project zullen er tussentijds evaluaties gehouden worden met Michel Wilson en Bob Huisman. Omdat er voortdurend een werkende versie beschikbaar zal zijn, is het hierbij mogelijk de functionaliteiten te evalueren die op dat moment af zijn. De feedback die hieruit volgt kan in

de resterende tijd worden verwerkt. Aan het einde van het project zal worden geëvalueerd of het product aan de gestelde eisen voldoet.

## 5.6 Pilots

Gedurende het project zal er een bezoek gebracht worden aan een werkplaats van NedTrain. Hier kan een voorlopige versie van de software worden getoond aan werknemers. Het doel hiervan is het verkrijgen van inzicht in de werkprocessen bij NedTrain. Deze informatie kan gebruikt worden om bestaande functionaliteiten te verbeteren of nieuwe functionaliteiten toe te voegen.

# Appendix A: Planning




Figuur 5.1: Project planning

# Bibliografie

[1] Erik Ammerlaan, Jan Elffers, Erwin Walraven, and Wilco Wisse. Precedence constraint posting for robust scheduling. In *Seminarium IN3305, Student Papers*, pages 75–88, 2012.

[2] Jasmin Blanchette. *C++ GUI programming with Qt 4*. Prentice Hall, 2008.

[3] Alan Ezust. *An introduction to design patterns in C++ with Qt*. Prentice Hall, 2011.

# Appendix D: Requirements Analysis Document

IN3405 BACHELORPROJECT

# Requirements Analysis Document

Bron: http://www.treinenfotoforum.nl/Nedtrain_701_Ldd.jpg

Erik Ammerlaan
Jan Elffers
Erwin Walraven
Wilco Wisse

# Definities

- **Planning:** toekenning van taken aan resources zodanig dat iedere taak een begin- en eindtijd heeft en aan alle constraints is voldaan, de capaciteit van de resources mag hierbij niet overschreden worden;
- **Resource:** de middelen die nodig zijn voor een taak, zoals gereedschap of personeel;
- **Taak:** een handeling aan een trein. Een taak gebruikt doorgaans één of meerdere resources;
- **Trein:** de verzameling taken die aan één trein moet worden uitgevoerd binnen een bepaald tijdinterval.

# Functionele requirements

1. Probleeminstanties (met treinen, taken en resources) kunnen worden ingeladen in een voorgeschreven formaat.
2. Probleeminstanties moeten grafisch worden weergegeven.
3. Probleeminstanties moeten opgelost kunnen worden.
4. Als er geen oplossing voor een probleeminstantie bestaat, moet het programma duidelijk aangeven wat het probleem is.
5. Er moeten meerdere probleeminstanties tegelijkertijd kunnen zijn ingeladen.
6. Er moet een externe solver gekozen kunnen worden, die het programma gebruikt voor het oplossen.
7. Treinen, taken en resources van probleeminstanties moeten gewijzigd of verwijderd kunnen worden.
8. Treinen, taken en resources moeten aan probleeminstanties toegevoegd kunnen worden.
9. De stappen die een algoritme neemt, moeten visueel doorlopen kunnen worden.
10. Constraints moeten visueel zichtbaar gemaakt kunnen worden, nadat een oplossing gegenereerd is.
11. Constraints moeten verwijderd kunnen worden.
12. Taken moeten verplaatst kunnen worden qua tijd.
13. Taken zouden door algoritmen gegroepeerd moeten kunnen worden. Dit resultaat moet dan ook in de grafische interface zichtbaar zijn.
14. De gebruiker moet interactief met het programma om kunnen gaan. Hij moet aan kunnen geven wanneer wordt afgeweken van de eigen constraints. Bijvoorbeeld: de gebruiker kan beslissen dat er wordt overgewerkt, zodat er een betere planning komt.
15. Oplossingen voor een probleeminstantie, gemaakt met verschillende algoritmen, moeten door de gebruiker vergeleken kunnen worden.
16. De capaciteit van een resource moet op een bepaald tijdinterval verlaagd kunnen worden.
17. De eenheid van de tijdschaal moet ingesteld kunnen worden door de gebruiker, afhankelijk van zijn of haar voorkeur.

## Niet-functionele requirements

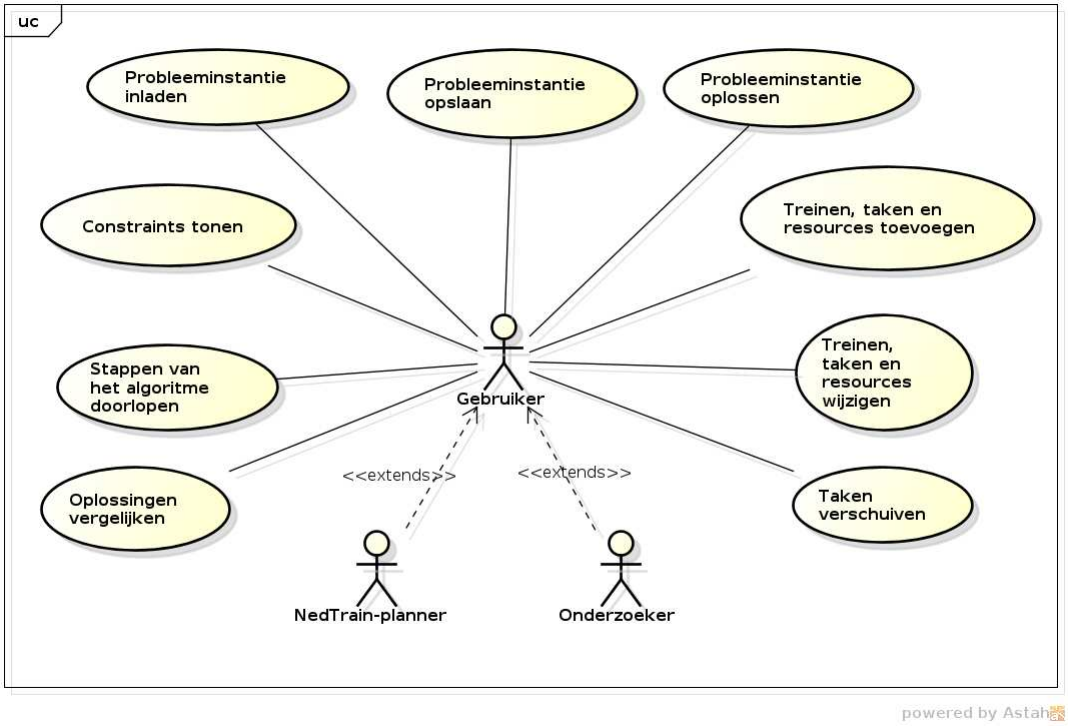
18. Er moeten algoritmen gebruikt worden die robuuste schedules opleveren.
19. De software moet meertalig zijn.
    - Het programma moet Engels zijn om bruikbaar te zijn in het onderwijs.
    - Het programma moet ook een Nederlandse versie met NedTrain-terminologie hebben.
20. Het programma moet intituïtief en overzichtelijk zijn.
21. Het programma moet een oplossing voor een rele probleeminstantie in korte tijd (hooguit 10 minuten) kunnen oplossen.

## Constraints (pseudo-requirements)

22. Het programma moet op een redelijke pc van 500 à 600 euro te draaien zijn.
23. Het programma moet een uitbreiding zijn op de eerder gemaakt software.

## Use cases

In figuur 1 is een use case diagram weergegeven met use cases die volgen uit de requirements. In het diagram zijn tevens twee actoren onderscheiden die beide gebruiker van het systeem zijn. Deze twee actoren representeren de twee doelgroepen van het programma: planners bij NedTrain en onderzoekers in de algoritmiek.




Figuur 1: Use case diagram

Bij de use cases zijn scenario's te maken. Deze scenario's zullen wij tijdens het ontwikkelproces opstellen als *user stories*. Aan het begin van een sprint, waarin we nieuwe functionaliteit implementeren, zullen wij user stories maken voor de functionaliteit die voor die sprint gepland staat. Op deze manier kunnen we agile en volgens de Scrum-methodologie werken. Het maken van alle mogelijke scenario's vooraf zou ons ontwikkelproces teveel op het watervalmodel doen lijken. Op deze manier kunnen wij ook rekening houden met eisen van de klant die pas op een later moment naar voren komen. Alle user stories zullen uiteindelijk hieronder worden toegevoegd.

# User stories voor sprint 1

## User story 1.1

**Als** een gebruiker **wil ik** de taal van het programma kunnen wijzigen **om** algemene, Engelse termen of Nederlandse, NedTrain-specifieke termen te zien.

*Wanneer* ik in een menu op een taal klik, *dan* krijg ik een melding dat wijzigingen zichtbaar worden nadat het programma opnieuw gestart is.

*Gegeven* dat het programma is afgesloten en dat de taal een vorige keer was ingesteld, *wanneer* ik het programma start, *dan* zie ik het programma in de ingestelde taal.

## User story 1.2

**Als** een gebruiker **wil ik** sjablonen voor taken kunnen opslaan **om** snel een bepaalde taak met een bepaald resource-verbruik aan verschillende treinen toe te kunnen voegen.

*Gegeven* dat er resources in het programma staan, *wanneer* ik in het menu kies voor 'Manage templates', *dan* zie ik een lijst met eerder gemaakte sjablonen.

*Gegeven* dat er resources in het programma staan, *wanneer* ik in het menu kies voor 'Manage templates' en klik op 'Add' en een naam, duur en resource-verbruik invul, *dan* wordt een sjabloon met deze gegevens aangemaakt.

*Gegeven* dat er resources in het programma staan, *wanneer* ik in het menu kies voor 'Manage templates', een sjabloon uit de lijst kies en klik op 'Remove', *dan* wordt de sjabloon verwijderd.

*Gegeven* dat er sjablonen zijn, *wanneer* ik een nieuwe taak ga aanmaken en een sjabloon uitkies, *dan* worden de velden naam, duur en resource-verbruik automatisch ingevuld.

*Gegeven* dat er sjablonen zijn, *wanneer* ik een nieuwe taak ga aanmaken en een sjabloon uitkies met resources die niet in de instantie staan, *dan* krijg ik een melding dat de sjabloon niet gebruikt kan worden.

## User story 1.3

**Als** een gebruiker **wil ik** de gegevens van een taak snel kunnen zien **om** te weten wat alle getoonde blokjes in het rooster betekenen.

*Gegeven* dat er een taak op de planning staat, *wanneer* ik met de muis over de taak beweeg, *dan* verschijnt er een tooltip met de naam van de taak.

*Gegeven* dat er een taak op de planning staat, *wanneer* ik er met de rechtermuisknop op klik en kies voor 'Properties', *dan* verschijnt er een venster met de naam, duur en het resource-verbruik van de taak.

### User story 1.4

**Als** een gebruiker **wil ik** een resource kunnen verwijderen **om** het toevoegen van de resource ongedaan te maken.

*Gegeven* dat er een resource bestaat en er geen taken zijn die de resource nodig hebben, *wanneer* ik klik op de verwijderknop van de resource, *dan* wordt de resource uit de instantie verwijderd en verdwijnt het resource-profiel.

*Gegeven* dat er een resource bestaat en er wél taken zijn die de resource nodig hebben, *wanneer* ik klik op de verwijderknop van de resource, *dan* krijg ik een melding dat de resource niet verwijderd kan worden.

### User story 1.5

**Als** een gebruiker **wil ik** een project kunnen verwijderen **om** het toevoegen van het project ongedaan te maken.

*Gegeven* dat er een project bestaat zonder taken, *wanneer* ik klik op de verwijderknop van het project, *dan* wordt het project uit de instantie verwijderd.

*Gegeven* dat er een project bestaat met taken, *wanneer* ik klik op de verwijderknop van het project, *dan* wordt alle taken van het project en het project zelf uit de instantie verwijderd.

### User story 1.6

**Als** een gebruiker **wil ik** een taak kunnen verwijderen **om** het toevoegen van de taak ongedaan te maken.

*Gegeven* dat er een taak bestaat, *wanneer* ik met de rechtermuisknop op de taak klik en de verwijderfunctie kies, *dan* wordt de taak uit de instantie verwijderd.

### User story 1.7

**Als** een gebruiker **wil ik** op een snelle manier de capaciteit van een resource aanpassen **om** de capaciteit aan te passen naar mijn wensen.

*Gegeven* dat er een resource bestaat, *wanneer* ik in het scherm bij de resource op de pijltjes omhoog of omlaag klik, *dan* wordt de capaciteit verhoogd dan wel verlaagd en het resource-profiel opnieuw getekend.

## User stories voor sprint 2

### User story 2.1

**Als** een gebruiker **wil ik** de tijdseenheid kunnen aanpassen **om** op de tijdbalk echte tijden te zien.

*Gegeven* dat er een instantie geladen is en de tijdschaal nog niet op uren staat, *wanneer* ik in het menu klik op ‘Uren op de tijdlijn tonen’, *dan* worden er uren op de tijdlijn getoond.

*Gegeven* dat er een instantie geladen is de tijdschaal op uren staat, *wanneer* ik een project of taak wil toevoegen of bewerken, *dan* kan ik uren en minuten gebruiken bij het invullen van tijden.

### User story 2.2

**Als** een gebruiker **wil ik** het schedulen stap voor stap uitvoeren **om** inzicht te krijgen in de gemaakte beslissingen door het algoritme.

*Gegeven* dat een instantie is opgelost, *wanneer* ik de schuifbalk verplaats of de knopjes naar links of rechts gebruik, *dan* wordt het nummer van de huidige stap getoond en worden de taken op de tijd behorende bij die berekeningsstap gezet.

### User story 2.3

**Als** een gebruiker **wil ik** dat taken gegroepeerd kunnen worden **om** meer flexibiliteit te hebben in de planning.

*Gegeven* dat de solver groepen ondersteunt, *wanneer* ik een instantie laat oplossen en de solver groepen teruggeeft, *dan* wordt er een kader om de taken van iedere groep getekend.

*Gegeven* dat een instantie is opgelost en er is een groep gemaakt, *wanneer* ik met de rechtermuisknop klik op een taak binnen de groep en kies voor 'Naar links' of 'Naar rechts', *dan* wisselen de taken in die groep van volgorde.

### User story 2.4

**Als** een gebruiker **wil ik** zelf een precedence constraint kunnen toevoegen **om** de volgorde tussen twee taken vast te leggen.

*Gegeven* dat er een instantie geladen is met ten minste twee taken, *wanneer* ik op de werkbalk de opties kies om een constraint toe te voegen en achtereenvolgens op twee taken klik, *dan* verschijnt er een pijl van de ene naar de andere taak en is de constraint aan de instantie toegevoegd.

### User story 2.5

**Als** een gebruiker **wil ik** constraints kunnen zien **om** inzicht te krijgen in de relaties die een taak heeft met andere taken.

*Gegeven* dat er een instantie geladen is met een taak die precedence constraints heeft, *wanneer* ik klik op die taak, *dan* verschijnen de constraints in de vorm van pijlen naar andere taken.

*Gegeven* dat een instantie is opgelost, *wanneer* ik klik op een taak, *dan* worden de constraints die door het algoritme zijn toegevoegd getoond met een rode pijl, en de constraints die al in de instantie zaten met een zwarte pijl.

*Gegeven* dat de constraints van een taak getoond worden, *wanneer* ik klik op die taak, *dan* verdwijnen de constraints weer.

### User story 2.6

**Als** een gebruiker **wil ik** de capaciteit van een resource op een bepaald interval kunnen verlagen **om** aan te geven dat een resource op bepaalde tijden (bijvoorbeeld 's nachts) minder capaciteit heeft.

*Gegeven* dat er een resource is, *wanneer* ik ga naar 'Bewerken' en een verlaging toevoeg door een interval en verlaging in te vullen, *dan* is de verlaging aan de instantie toegevoegd en wordt deze getekend in het resource-profiel.

*Gegeven* dat er een resource-verlaging is, *wanneer* ik een instantie oplos, *dan* wordt er een oplossing gevonden waarbij de verlaging niet overschreden wordt, of er is geen oplossing.

# User stories voor sprint 3

## User story 3.1

*Extensie van user story 2.2.*

*Wanneer* ik terugspoel, *dan* worden ook de voorrangsrelaties getoond die in de desbetreffende stap zijn toegevoegd.

## User story 3.2

**Als** een gebruiker **wil ik** de soft precedences (toegevoegd door het algoritme) kunnen opslaan **om** ze terug te krijgen als ik de instantie opnieuw open.

*Gegeven* dat er soft precedences zijn toegevoegd door de solver, *wanneer* ik de instantie opsla, *dan* worden de soft precedences ook opgeslagen.

*Gegeven* dat er een instantie met soft precedences is opgeslagen, *wanneer* ik die instantie open, *dan* kan ik de soft precedences van taken weergeven.

*Gegeven* dat een instantie soft precedences heeft, *wanneer* ik de solver aanroep, *dan* worden de soft precedences door de solver hetzelfde behandeld als hard precedences.

## User story 3.3

**Als** een gebruiker **wil ik** kunnen zien wat het resource-verbruik van een taak of project is **om** bijvoorbeeld te kunnen zien waar pieken in het resource-profiel door veroorzaakt worden.

*Gegeven* dat er resources en taken zijn, *wanneer* ik klik op een blokje uit het resource-profiel, *dan* worden de taken die voor de bezetting verantwoordelijk zijn gearceerd.

*Gegeven* dat er resources en taken zijn, *wanneer* ik met rechtermuisknop klik op een project en kies voor 'Resource-verbruik tonen', *dan* wordt het resource-profiel uitgelicht op het interval waarop het project de resource verbruikt.

## User story 3.4

**Als** een gebruiker **wil ik** kunnen in- of uitzoomen **om** een grote instantie overzichtelijk te kunnen bekijken.

*Gegeven* dat er een instantie geladen is, *wanneer* ik klik op de de knop 'Inzoomen' of 'Uitzoomen', *dan* worden de taken en resources horizontaal geschaald.

# User stories voor sprint 4

## User story 4.1

**Als** een gebruiker **wil ik** individuele voorrangsrelaties kunnen verwijderen **om** de restricties voor het oplossen te beperken.

*Gegeven* dat er twee taken zijn met een voorrangsrelatie ertussen, *wanneer* ik klik op 'Voorrangsrelatie verwijderen' en eerst de ene en dan de andere taak aanklik, *dan* wordt de voorrangsrelatie verwijderd.

## User story 4.2

**Als** een gebruiker **wil ik** instanties kunnen importeren **om** twee instanties samen te voegen.

*Gegeven* dat er een instantie geopend is, *wanneer* ik klik op 'Importeren', een instantie kies en een tijd invul als invoegpunt, *dan* worden alle projecten van de geïmporteerde instantie toegevoegd aan de huidige instantie, waarbij alle begintijden relatief aan het gekozen invoegpunt zijn.

## User story 4.3

**Als** een gebruiker **wil ik** voor elke taak de *earliest starting time* en *latest completion time* kunnen zien **om** te kunnen zien hoeveel speling taken hebben.

*Gegeven* dat een instantie is geopend, *wanneer* ik klik op 'Toegelaten intervallen tekenen', *dan* wordt er bij elke taak een lijn getekend vanaf de EST naar de LCT.

## User story 4.4

**Als** een gebruiker **wil ik** resource-verlagingen kunnen opslaan **om** ze terug te krijgen als ik de instantie opnieuw open.

*Gegeven* dat er resource-verlagingen zijn toegevoegd door de gebruiker, *wanneer* ik de instantie opsla, *dan* worden de resource-verlagingen ook opgeslagen.

*Gegeven* dat er een instantie met resource-verlagingen is opgeslagen, *wanneer* ik die instantie open, *dan* zie ik resource-verlagingen in het resource-profiel.

## User story 4.5

**Als** een gebruiker **wil ik** eventueel extra parameters kunnen toevoegen bij het solven **om** niet telkens de solverconfiguratie te moeten aanpassen.

*Gegeven* dat een instantie is geopend, *wanneer* ik klik op 'Plannen met opties', *dan* kan ik extra parameters meegeven aan de solver.

*Gegeven* dat een instantie is geopend, *wanneer* ik klik op 'Plannen met opties' en ik vink de optie 'Plannen naar nieuw tabblad' aan, *dan* komt de opgeloste instantie in een nieuw tabblad, terwijl het huidige tabblad ongewijzigd blijft.

### User story 4.6

**Als** een gebruiker **wil ik** de tijdsduur van een taak kunnen aanpassen door de taak uit te rekken **om** op een snellere en intuïtievere wijze de tijdsduur aan te kunnen passen.

*Gegeven* dat er een taak in de instantie is, *wanneer* ik de linkermuisknop indruk op de rechterkant van een taak en de muiscursor naar rechts verplaats, *dan* wordt de taak uitgerekt en de tijdsduur overeenkomstig aangepast.

### User story 4.7

**Als** een gebruiker **wil ik** taken vóór een bepaald tijdspunt kunnen afmelden **om** taken in het verleden een inactieve status te geven.

*Gegeven* dat er een instantie geopend is, *wanneer* ik klik op de tijdlijn op een bepaald tijdstip, *dan* krijgen taken vóór die tijd een grijze achtergrondkleur.

## User stories voor sprint 5

### User story 5.1

**Als** een gebruiker **wil ik** oplossingen met elkaar kunnen vergelijken **om** inzicht te krijgen in wat het kiezen van een solver voor invloed heeft op de oplossing.

*Gegeven* dat ik twee opgeloste instanties in tabbladen heb, *wanneer* ik klik op 'Vergelijken' en de twee tabs uitkies, *dan* worden de twee instanties onder elkaar getoond in een nieuw venster, waarbij de taken die in beide instanties op dezelfde tijd zijn gezet een grijze kleur krijgen, en taken die qua begintijd verschillen per instantie een opvallende kleur krijgen.

*Gegeven* dat ik twee instanties met compleet verschillende taken, resources, enz. heb geopend, *wanneer* ik klik op 'Vergelijken' en de twee tabs uitkies, *dan* wordt een melding getoond dat er alleen vergeleken kan worden tussen dezelfde instanties die slechts afwijken qua begintijden.

### User story 5.2

**Als** een gebruiker **wil ik** een taak kunnen verplaatsen tussen de earliest starting time en de latest starting time **om** zelf een opgeloste instantie te kunnen beïnvloeden.

*Gegeven* dat een instantie een taak bevat en dat hij is opgelost, *wanneer* ik de linkermuisknop ingedrukt houd op de taak en de muiscursor versleep, *dan* wordt de taak naar links of rechts verplaatst en ook wordt de keten van taken met voorrangsrelaties meegeschoven.

*Gegeven* dat ik een taak aan het verplaatsen ben, *wanneer* ik te taak vóór de earliest starting time of na latest starting time probeer te zetten, *dan* blijft de taak staan op de earliest starting time dan wel de latest starting time.

### User story 5.3

**Als** een gebruiker **wil ik**, als een instantie niet kan worden opgelost, weten wat de oorzaak is **om** te weten hoe de instantie kan worden aangepast en niet opgescheept te zitten met een foutmelding dat oplossen niet mogelijk is.

*Gegeven* dat de solver een instantie oplost, *wanneer* de instantie niet kan worden opgelost door een onoplosbaar resource-conflict, *dan* worden de taken of groepen genoemd die de 'contention peak' veroorzaken.

*Gegeven* dat ik een voorrangsrelatie aan het toevoegen ben, *wanneer* ik daarmee een cycle van voorrangsrelaties zou introduceren, *dan* krijg ik een melding dat de voorrangsrelatie niet kan worden toegevoegd.

### User story 5.4

**Als** een gebruiker **wil ik** de begintijd van een taak vast kunnen zetten, **opdat** de solver de taak niet van begintijd verandert.

*Gegeven* dat ik een taak versleept heb, *wanneer* ik er met de rechtermuisknop op klik en een vinkje zet voor 'Vergrendelen' en de instantie opnieuw oplos, *dan* wordt de taak niet verplaatst, maar andere taken wel.

*Gegeven* dat ik een taak heb vergrendeld, *wanneer* ik er met de rechtermuisknop op klik en het vinkje voor 'Vergrendelen' weghaal en de instantie opnieuw oplos, *dan* wordt de taak na het solven op de earliest start time gezet.

*Gegeven* dat ik een taak heb vergrendeld, *wanneer* ik de muisknop op de taak ingedrukt houd en de cursor verplaats, *dan* wordt de taak *niet* verplaatst.

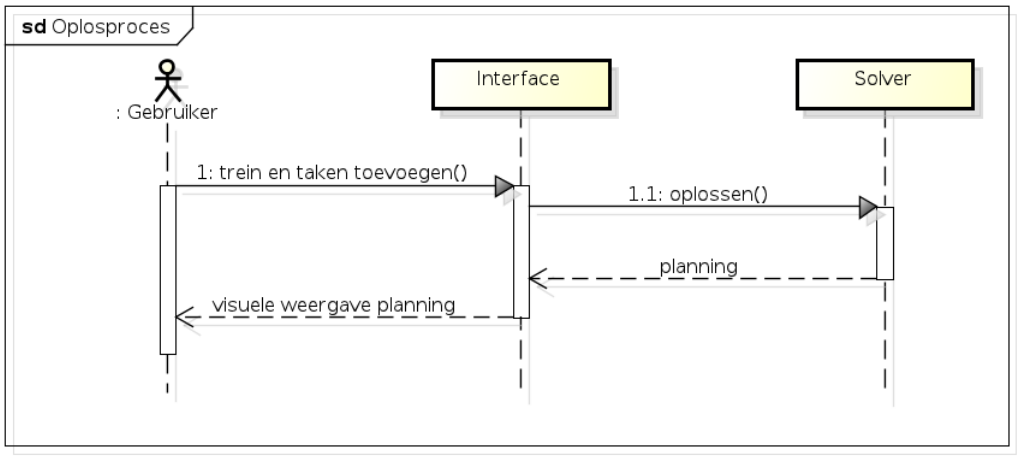
### User story 5.5

**Als** een gebruiker **wil ik** een taak flexibiliteit kunnen geven, **opdat** de solver er geen taken direct achter plaatst, zodat er geen problemen optreden als de taak uitloopt.

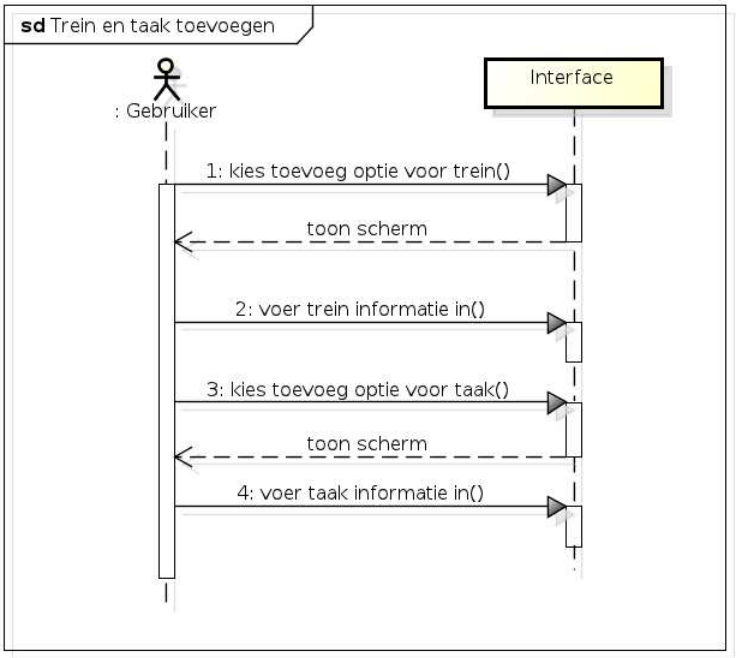
*Gegeven* dat er een taak $a$, een taak $b$ en een voorrangsrelatie $a \prec b$ is, *wanneer* ik met de rechtermuisknop op taak $a$ klik en een flexibiliteit instel van $x$ tijdseenheden, *dan* wordt na het solven taak $b$ $x$ tijdseenheden achter taak $a$ gezet, of de instantie kan niet worden opgelost.

## Dynamische modellen

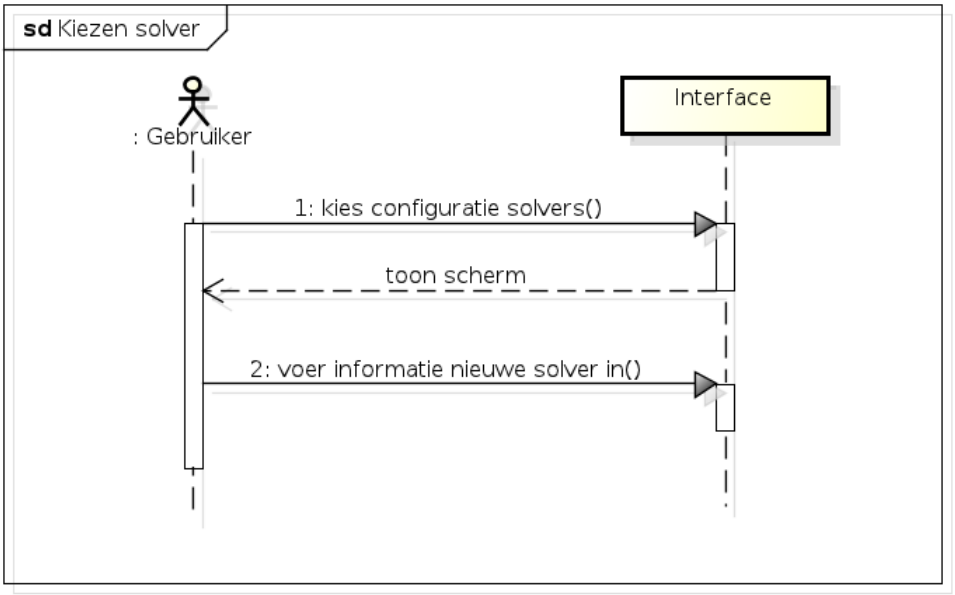
In deze paragraaf worden sequencediagrammen voor enkele basisfunctionaliteiten gegeven. Voor de optionele functionaliteiten wordt geen diagram gepresenteerd, omdat in een volgende fase bepaald zal worden welke functionaliteiten geïmplementeerd worden.



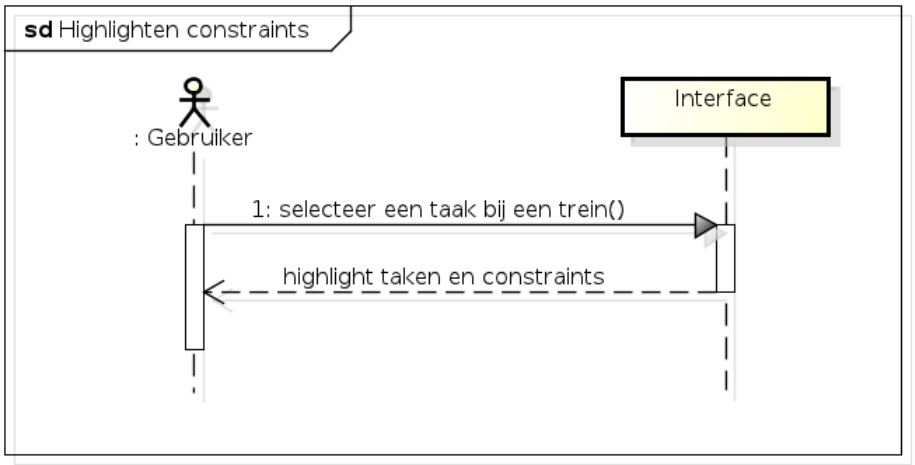

Figuur 2: Het sequencediagram van het oplosproces.




Figuur 3: Het sequencediagram van het toevoegen van treinen en taken.




Figuur 4: Het sequencediagram van het toevoegen van de solver.




Figuur 5: Het sequencediagram van het highlighten van voorrangsrelaties.