Ontwikkeling van visualisatiesoftware voor ERTMS HL3 mCRL2 simulatie.

ERTMS

Time to connect

ProRail

DE HAAGSE HOGESCHOOL

Auteur: Dennis van Gilst
Studentnummer:

E-mail:
Organisatie: ProRail B.V.
Locatie: Utrecht

Periode: 11/05/2020 – 5/10/2020
Opleiding: Software Engineering | Internet of Things | Applied Data Science

Organisatie:

Naam:	ProRail B.V.
Afdeling:	Assetmanagement (AM)
Adres:	Moreelsepark 3
Postcode:	3511 UP
Plaats:	Utrecht
Telefoonnummer:	

Opdrachtgever:

Naam:	mw Voorderhake-Borst
Voorletters:	M.
Functie:	Projectmanager / Innovator
Telefoonnummer:	
E-mail:	

Mentor:

Naam:	dhr Dragt
Voorletters:	S.
Functie:	Systeemspecialist Treinbeveiliging
Telefoonnummer:	
E-mail:	

Begeleidend examinator:	Doorn, E.M. (Ed) van
Expert examinator:	Bechet-Tjoonk, H.G.J. (Rianne)
Gecommitteerde:	Breumelhof, M (Maarten)
Afstudeercoördinator:	Janssen, P. (Poco)
Studieloopbaanbegeleider:	Vosteen, M.I.M. (Mirabai)

Voorwoord

Voor u op uw thuiswerklaptop of in uw gedesinfecteerde handen heeft u mijn afstudeerverslag. Dit document bevat de verslaglegging van mijn afstudeerproject bij ProRail voor de opleiding HBO-ICT aan de Haagse Hogeschool.

Ik geef een omschrijving van de opdracht en de context waarbinnen deze opdracht wordt uitgevoerd. Ook beschrijf ik hoe ik mijn aanpak bepaald en uitgevoerd heb.

Allereest zou ik mijn bedrijfsmentor, dhr Dragt, en de leden van het HL3 OVS team willen bedanken voor wekelijks aanhoren van mijn technisch gebrabbel en voor het helpen met mij in de juiste richting sturen.

Daarnaast wil ik ook mw Voorderhake bedanken voor het houden van een waakzaam oog op mij en de andere stagiaires. Ook voor het regelen van een zeer informatieve rondleiding van het Railcenter zodat wij ook nog iets meekrijgen van de werkzaamheden van ProRail wanneer we thuis werken.

Tevens wil ik mijn examinatoren bedanken voor het mopperen (hun woorden) op de eerdere versies van mijn verslag. Zo hebben zij mij geholpen ervoor te zorgen dat ik de beste versie van mijn verslag neerzet.

Dan wil ik uiteraard ook mijn ouders bedanken voor het geven van feedback en input zodat ik kan zorgen dat mijn verhaal duidelijk overkomt.

Tot slot wil ik dan nog mijn vriendin Lotte bedankt voor het zorgen dat ik zo af en toe ook nog eens naar buiten ging en aandacht kon besteden aan al het andere goede in de wereld.

Veel leesplezier.

Dennis van Gilst

Den Haag, 5 oktober 2020

Samenvatting

Dit verslag bevat een omschrijving van de ontwikkeling van een applicatie die visuele weergave geeft van een simulatie van het ERTMS HL3 systeem met de mCRL2 taal en toolkit.
De ontwikkeling van deze applicatie wordt uitgevoerd als afstudeeropdracht en is in opdracht van ProRail B.V. De opdracht is uitgevoerd binnen de afdeling Asset Management.

De Technische Universiteit Eindhoven heeft in opdracht van ProRail een simulator ontwikkeld van het European Rail Traffic Management System (ERTMS) Hybrid Level 3 (HL3) treinbeveiligingssysteem. Dit beveiligingssysteem moet de huidige systemen langs het spoor vervangen om te zorgen voor meer capaciteit op het spoor. De simulatie heeft geen grafische weergave en is lastig te begrijpen.

Om de simulator beter te kunnen begrijpen en de werking van het ERTMS HL3 te kunnen inzien wil ProRail een applicatie ontwikkelen die een visuele weergave geeft van de simulator. Deze visualisatie moet de verschillende onderdelen uit het HL3 systeem en de toestanden daarvan weergeven. Zo kan een gebruiker makkelijk inzien wat de status van het systeem is.

De ontwikkeling van de applicatie is volgens de Waterfall methode in vijf fases uitgevoerd.
Fase 0: In deze fase is het probleemdomein geanalyseerd en zijn de eisen van de stakeholders opgesteld.
Fase 1: Aan de hand van de opgestelde eisen zijn enkele concepten van de grafische interface van de applicatie en een ontwerp de interne opbouw van de software uitgewerkt. De software is ontworpen volgens het Domain Driven Design principe.
Fase 2: In deze fase is de software gerealiseerd met behulp van de ontwerpen uit de vorige fase. De applicatie is ontwikkeld met de C++ programmeertaal en SDL2 library.
Fase 3: Tijdens deze fase is de ontwikkelde software getest. Er is getest of de software correct functioneert en of de software de simulator correct visualiseert. Er is een testapplicatie ontwikkeld waarmee de functionaliteit van de visualisatie applicatie automatisch getest wordt.
Fase 4: In de laatste fase is de applicatie aan de stakeholders overgedragen door hen bekend te laten worden met de software, een handleiding voor het gebruik van de software te schrijven en ten slotte de applicatie met bijbehorende broncode en benodigde onderdelen over te dragen.

Het resultaat van deze werkzaamheden is een applicatie welke ProRail kan gebruiken om een visuele representatie van de simulator te krijgen. Ook kan ProRail de applicatie gebruiken om de simulator te beïnvloeden. Hiermee kunnen zij het gedrag van het ERTMS HL3 systeem gemakkelijk analyseren.

Verder beschrijft dit verslag een evaluatie van het opgeleverde product en de uitvoering van het project.

Inhoudsopgave

1. Inleiding

De treinbeveiligingssystemen die op dit moment langs het spoor liggen zijn in de jaren 45-90 ontwikkeld en geïnstalleerd. Ondertussen zijn er nieuwere en complexere systemen ontwikkeld die ten opzichte van het huidige systeem veiliger zijn en voor meer capaciteit op het spoor kunnen zorgen.
ProRail is de ontwerpvoorschriften aan het opstellen voor de implementatie van het nieuwe treinbeveiligingssysteem, het European Rail Traffic Management System (ERTMS) Hybrid Level 3 (HL3). Dit systeem gaat uiteindelijk langs alle spoorwegen aangebracht worden. Het gedrag van dit systeem is complex en daarom moeilijk te evalueren. De Technische Universiteit Eindhoven (TU/e) heeft het gedrag van het systeem vastgelegd in een model in de door hun ontwikkelde mCRL2 specificatietaal. Door middel van dit model is het gedrag te simuleren. Echter is het moeilijk in dit model terug te zien wat op een bepaald moment de status van het model is, omdat de status wordt bijgehouden in numerieke variabelen met namen en aanduidingen waarvan de betekenis niet direct duidelijk is.
ProRail heeft aan mij gevraagd een applicatie te ontwikkelen die een gebruiksvriendelijke visuele weergave genereert uit data van de simulator van de TU/e.
Dit afstudeerverslag beschrijft welke werkzaamheden ik heb uitgevoerd om een applicatie te ontwikkelen naar de wensen van de opdrachtgever en stakeholders binnen ProRail. Tot slot zal er een evaluatie plaatsvinden waarin de werkzaamheden en de ontwikkelde producten worden geëvalueerd.

2. Context

2.1 ProRail

ProRail is verantwoordelijk voor het aanleggen, onderhouden, beheren en beveiligen van het spoorwegnet van Nederland. Hieronder vallen naast de sporen zelf ook de stations, wissels, seinen en overwegen. ProRail verdeelt de beschikbare ruimte op het spoor aan de verschillende vervoerders van Nederland, zoals NS of Arriva, maar ook aan buitenlandse vervoerders, zoals DB Schenker of Thalys. Het beheren van het treinverkeer op het spoor wordt ook door ProRail gedaan, behalve op zogeheten lokaalspoorwegen. Daar is ProRail niet de hoofdbeheerder, maar werkt wel samen met de daadwerkelijke beheerder.

ProRail is ingericht als een organisatie die het spoor beheerd vanuit een centraal gedeelte en een aantal regionale gebieden. Het hoofdkantoor genaamd De Inktpot staat in Utrecht. In de regio's staan de verkeersleidingsposten, Schakel- en Meldcentra (SMC) en de regiokantoren.

De organisatie van ProRail is verdeeld in de twee stromen: Uitvoering en Bedrijfsvoering.
Onder Uitvoering vallen de afdelingen Vervoer & Dienstregeling, Projecten en Operatie. Vervoer & Dienstregeling verdeelt de capaciteit van het spoor en zorgt voor de publieke relaties met reizigers en goederenvervoerders. Projecten voert spoor- en andere bouwprojecten uit in opdracht van lokale of landelijke overheden. De afdeling Operatie is verantwoordelijk voor het spoorsysteem en het regelen van het treinverkeer.
Onder bedrijfsvoering vallen de afdelingen Finance en Staf & Ondersteuning. De afdeling Finance levert financiële cijfers en adviezen bij de organisatie. De afdeling Staf & Ondersteuning gaat over de interne zaken die betrekking hebben tot Human Resources (HR), bedrijfsstrategieën, communicatie, innovatie en duurzame ontwikkeling.

2.2 Assetmanagement

Mijn afstudeeropdracht wordt uitgevoerd binnen de bedrijfseenheid Assetmanagement (AM). AM valt onder de afdeling Operatie en is verantwoordelijk voor beheren van het spoor. Hieronder valt het uitvoeren van onderhoud, oplossen van storingen, vervangen van bestaande railinfrastructuur, maar ook het bepalen van de optimale besteding van het beschikbare geld.

De heer Dragt is de bedrijfsmentor tijdens deze opdracht. De heer Dragt is werkzaam binnen AM als ERTMS systeemspecialist. Hij werkt binnen het team dat verantwoordelijk is voor het opstellen van de ontwerpvoorschriften (OVS) van het ERTMS HL3 systeem. Dit team stelt aan de hand van de technische specificaties van het ERTMS HL3 systeem de ontwerpvoorschriften op welke gebruikt kunnen worden door de ingenieursbureaus wanneer zij een spoorweg met het ERTMS HL3 systeem aanleggen. De leden van dit team zijn de stakeholders van mijn project. Ik ontwikkel de applicatie voor hen.

2.3 ERTMS HL3

Het European Rail Traffic Management System (ERTMS) is een specificatie voor spoorwegseinen en treinbeïnvloedingssystemen ontwikkeld door het Spoorwegbureau van de Europese Unie. Het is ontwikkeld om de bestaande beveiligingssystemen te verbeteren op de gebieden van veiligheid, maximale snelheid van een trein en totale capaciteit van het spoor. Ook is het de bedoeling dat een systeem wat op Europees niveau ontwikkeld is het internationale spoorverkeer vergemakkelijkt.

Er bestaan verschillende versies binnen de ERTMS specificatie. De versies verschillen in de middelen waarmee het een trein kan detecteren en de positie kan bepalen, en de mate waarmee het de trein kan beïnvloeden. Deze verschillende versies worden binnen de specificatie *levels* genoemd. De versie waar de stakeholders geïnteresseerd in zijn heet Hybrid Level 3 (HL3). Dit systeem opereert normaal gesproken op Level 3, maar valt terug naar level 2 wanneer een trein niet de systemen aan boord heeft om op level 3 te opereren, of wanneer er een storing plaatsvindt.

Bij ERTMS spelen drie onderdelen de hoofdrollen: De treinen, de Trackside Train Detection (TTD) en de Virtual SubSection (VSS).

Een TTD is een sectie van het spoor gedefinieerd door de traditionele treinbeveiligingssystemen. Deze secties bevatten sensoren, zoals assentellers, om treinen binnen de TTD waar te nemen.
Een VSS is een virtueel sub-sectie van een TTD. Een TTD bevat vaak meerdere VSS'en. De bezetting van een VSS wordt bepaald door de TTD systemen in combinatie met positionele meldingen van de trein via een radioverbinding. Het gebruik van kleinere virtuele sub-secties stelt het systeem in staat het spoor te beheren op een kleiner niveau van granulariteit. Dit helpt de capaciteit van het spoor te vergroten.

2.4 mCRL2

mCRL2 is een specificatietaal met bijbehorende toolkit ontwikkeld aan de TU/e. De taal kan gebruikt worden voor het modeleren van systemen en protocollen. De bijbehorende toolkit bevat applicaties waarmee de modellen onder meer gevalideerd, geanalyseerd, of, zoals in het geval van dit project, gesimuleerd kunnen worden. De TU/e heeft voor ProRail een model van het ERTMS HL3 systeem ontwikkeld. Dit hebben zij gedaan op aanvraag van ProRail zodat ProRail het gedrag van de verschillende onderdelen van een HL3 systeem kunnen inzien. Deze modellen, de simulator en andere applicaties uit de mCRL2 toolkit worden in dit project gebruikt. Het model dat de TU/e ontwikkeld heeft bevat meerdere scenario's met verschillende hoeveelheden TTD's, VSS'en en treinen.

3. Opdracht

3.1 Situatie bij aanvang

Het model dat door de TU/e is uitgewerkt is vrij toegankelijk op het internet en dus ook voor de mensen binnen ProRail. Binnen ProRail was de afdeling bezig met het opstellen van een nieuwe opdracht voor de TU/e voor het uitbreiden van het model. Naast dit model zijn er geen andere applicaties voor het demonstreren van het gedrag van het ERTMS HL3 systeem. Dit model is met de mCRL2 toolkit onder andere te simuleren.
Geen van de stakeholders heeft op dit punt van het project dit model of de bijbehorende simulator bestudeerd of er in enige vorm mee gewerkt. De kennis van de stakeholders over hoe de simulatie werkt en wat er wel en niet in het model is uitgewerkt is daarom beperkt.

3.2 Probleemstelling

De leden van het OVS team willen het gedrag van het ERTMS HL3 systeem analyseren. Zoals hierboven al omschreven is heeft de TU/e in opdracht van ProRail hiervoor een model in de mCRL2 specificatietaal uitgewerkt. Het probleem is dat dit model een zeer gebruiksonvriendelijk manier gebruikt om weer te geven wat exact de situatie en status van het model is. De simulatie is moeilijk te begrijpen als je geen diepgaande kennis hebt van de specifieke uitwerking van het model en de werking van de simulator. Omdat het model door de TU/e ontwikkeld is hebben de leden van het OVS team deze kennis niet. Ook geeft de simulator slechts de status van een bepaald onderdeel uit het model aan wanneer deze veranderd door een bepaalde actie. De simulator geeft dus nooit een totaalbeeld van de situatie. Dit is onhandig voor de gebruikers omdat zij in één oogopslag duidelijk willen hebben wat de status is van alle onderdelen zonder terug te hoeven zoeken wat de laatste wijziging van een bepaald onderdeel was.

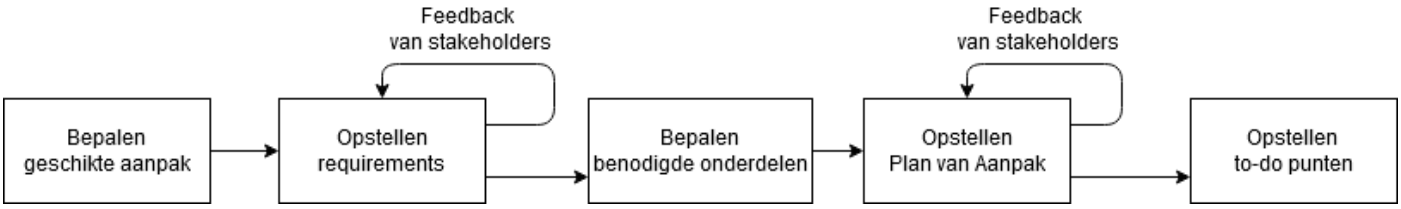
3.3 Doelstelling

Het doel van deze opdracht is het ontwikkelen van een applicatie die een visuele weergave geeft van de verschillende objecten in het model welke gesimuleerd wordt door de mCRL2 simulator. Deze applicatie houdt bij wanneer de status van een object veranderd en geeft een totale weergave van alle verschillende onderdelen. Ook geeft de applicatie aan welke acties er op een gegeven moment uitgevoerd kunnen worden. De gebruiker kan één van deze acties kiezen. De simulator zal deze actie uitvoeren wat als gevolg heeft dat de status van bepaalde onderdelen zal veranderen. De visualisatie zal bijgewerkt worden en de nieuwe statussen zullen weergegeven worden.

Met deze applicatie willen de stakeholders het gedrag van het totale systeem kunnen bestuderen. Ze krijgen inzicht hoe verschillende onderdelen reageren op het doorvoeren van een bepaalde verandering binnen het systeem.

4. Fase 0 (analyse)

De volgende werkzaamheden heb ik uitgevoerd tijdens deze eerste fase.



Figuur 1: Uitgevoerde werkzaamheden on Fase 0

Het is verstandig een project te beginnen met het bepalen van wat je precies moet doen en hoe je dat slim kan uitvoeren. Daarom ben ik begonnen met het kiezen van een geschikte aanpak en het opstellen van de requirements. Op basis hiervan kan ik een plan opstellen en verdere benodigde documenten voor de uitvoering opstellen. Op die manier ben ik op de volgorde van de punten in figuur 1 gekomen.

4.1 Bepalen aanpak

Om het project in zijn geheel goed te laten verlopen is het verstandig te werken volgens een bekende methode. De gekozen methode biedt inzicht in welken stappen er doorlopen moeten worden en welke producten er tussentijds gerealiseerd moeten worden. Ook geeft het mijzelf en de opdrachtgever een grof idee van een planning,

Ik heb voor mijn keuze drie populaire methodes overwogen. Ik heb deze drie als startpunt genomen omdat dit methodes zijn waarover ik geleerd heb tijdens de opleiding, en omdat deze drie een mooie verdeling bieden over het spectrum van iteratieve methodes naar lineaire methodes.
De drie methodes zijn:

1. Agile Development
2. Waterfall
3. Rapid Application Development (RAD)

Elke methode definieert zijn eigen manier van software ontwikkelen. Eén van de bovenstaande methodes zal het met geschikt zijn voor mij.

4.1.1 Agile Development

Het belangrijkste punt van Agile is dat de software ontwikkeld wordt in iteraties. Het einde van elke iteratie levert een leverbaar product op. Aan het begin van een project worden de requirements verzameld en aan de hand hiervan een backlog van to-do punten verzameld. Agile gaat flexibel om met de requirements. Gedurende de uitvoering van het project is het mogelijk de requirements te wijzigen.
Voor elke iteratie wordt er een korte tijdsperiode uitgetrokken. Vaak wordt hier twee weken voor genomen. Voor elke iteratie worden enkele to-do punten gekozen om uitgevoerd te worden. Aan het begin van elke iteratie vindt er een overleg plaats, en aan het einde een review. Ook vind er dagelijks een kleine bespreking van de voortgang plaats.

Werken volgens een Agile methode spreekt mij niet aan. Voornamelijk omdat de flexibiliteit van dit project wil beperken. Omdat de stakeholders beperkt en wispelturig beschikbaar zijn wil ik voorkomen dat een stakeholder later in het project een idee opgooit dat al eerder in het project meegenomen had moeten worden.

4.1.2 Waterfall

Bij deze methode zijn de fases die doorlopen moeten worden vooraf bekend. Men begint alleen aan een volgende fase wanneer de vorige compleet is afgerond. In het begin worden de requirements verzameld. Wanneer dit afgerond is wordt er een ontwerp opgesteld. Hierna wordt dit ontwerp uitgewerkt. Vervolgens wordt deze uitwerking door de stakeholders goedgekeurd. Het is niet de bedoeling dat er stappen terug worden gezet in de uitvoering.

Het gebruiken van deze methode voor de uitvoering van dit project spreekt mij aan. Om er zeker van te zijn dat ik het juiste product ontwikkel, lijkt het mij handig dat ik bevestigd krijg dat ik een fase correct heb uitgevoerd voordat ik aan de volgende fase begin. Ik wil niet aan het ontwerp beginnen voordat de stakeholders hebben kunnen bevestigen dat de requirements juist zijn. Ook wil ik niet aan het implementeren beginnen als de stakeholders het niet eens zijn met het ontwerp. Door enkel aan een volgende fase te beginnen wanneer de voorgaande helemaal zeker is afgerond voorkom ik dat er wijzigingen doorgevoerd moeten worden en ik stappen terug moet nemen.

4.1.3 Rapid Application Development

Deze methode zit tussen bovenstaande methodes in. Aan het begin van een project worden de requirements verzameld. Wanneer dit afgerond is gaat een herhalende fase in waarin het ontwerpen en ontwikkelen per onderdeel plaatsvindt. Per ontwerp en uitwerking geven de stakeholders feedback totdat de stakeholder bevestigt dat alle requirements correct zijn uitgewerkt.

Deze methode past ook enigszins bij mijn situatie. Echter spreek het maken van een ontwerp per onderdeel mij niet aan. Ik wil één keer één ontwerp maken wat dan ook maar één keer goedgekeurd hoeft te worden.

4.1.4 Keuze

Ik heb gekozen voor het volgen van de Waterfall methode. Deze keuze heb ik voornamelijk gemaakt omdat ik wil bevestigen dat een bepaalde fase correct is afgerond voordat ik aan de volgende fase begin.
Er zijn weinig contactmomenten, namelijk één keer per week, en niet alle stakeholders zullen elke week allemaal aanwezig zijn. Kiezen voor Waterfall helpt te voorkomen dat ik al een week bezig ben met het uitwerken van een bepaald punt en er achteraf blijkt dat een stakeholder het ergens niet mee eens is of iets toe te voegen heeft. Met Waterfall zal ik niet aan de volgende fase beginnen tenzij helemaal zeker is dat de huidige fase compleet is.

Ik heb gewerkt met een aantal fasen die afwijken van traditioneel Waterfall. Deze zijn als volgt:

0. Analyse:
   In deze fase analyseer ik de algemene aanpak en benodigdheden van dit project.
   Ik bespreek met de stakeholders hun wensen en stel ik de softwarerequirements op. Daarnaast bestudeer ik de mCRL2 taal en toolkit omdat de applicatie hiervan afhankelijk zal zijn. Dit geeft mij technisch inzicht van de relevante applicaties en stelt mij in staat een idee te scheppen op welke manier de toolkit gebruikt kan worden. Daarnaast bestudeer ik ook hoe het ERTMS HL3 systeem in de taal is uitgewerkt. Ook stel ik een plan van aanpak op om een idee van de uitvoering van het project te scheppen. Ik sluit af met het opstellen van een overzicht met de verschillende punten die voor de applicatie ontwikkeld moeten worden.
1. Concept:
   Deze fase dient voor het ontwerpen van een klassediagram van de applicatie en voor het maken van een concept van de grafische interface van de applicatie. Deze punten kunnen nu getest worden tegen de wensen van de stakeholders. Wanneer de ontwerpen van de verschillende aspecten van de software duidelijk zijn kan de volgende fase beginnen.
2. Ontwikkel:
   Deze fase dient voor het realiseren van de applicatie aan de hand van de ontwerpen uit de vorige fase.
3. Test:
   Deze fase dient voor het testen van de applicatie. Allereest zal er een testplan opgesteld worden. Ik zal een tweede applicatie ontwikkelen die de werking van de visualisatie applicatie test volgens het testplan. Het resultaat hiervan zal ogenomen worden in een testrapport.
4. Overdracht:
   In deze fase zal ik enkele documenten opstellen die het correcte gebruik en de benodigdheden van de applicatie omschrijven. Ook zullen de ontwikkelde producten aan de stakeholder overgedragen worden.

4.1.5 Afwijkingen t.o.v. plan

In mijn afstudeerplan beschrijf ik dat ik zou gaan werken volgens een Agile methode. Op het moment dat ik dit plan schreef had ik andere verwachtingen van de uitvoering van het project. Ik had verwacht bij ProRail op locatie in het kantoor in Utrecht binnen een team te werken. Met dit team zou ik stand ups en andere besprekingen die bij Agile

methodes horen kunnen houden. Echter zijn er in de tussentijd een hoop beperkende maatregelen van kracht gegaan vanwege de COVID-19 pandemie. Deze maatregelen maken het voor mij niet mogelijk op locatie te werken en daarom heb ik ook opnieuw gekeken naar welke methode voor mij geschikt is.

4.2 Opstellen softwarerequirements

Het is belangrijk een goed beeld te hebben van de wensen van de stakeholders Deze applicatie wordt immers voor de stakeholder ontwikkeld. Dan is het ook de bedoeling dat de applicatie voldoet aan de wensen van de stakeholders. Ik heb de eerste paar contactmomenten gebruikt om met alle stakeholders te overleggen over wat precies hun requirements van de applicatie zijn. Zoals ik al eerder heb omschreven had ProRail al een nieuwe uitvraag naar de TU/e gestuurd voor het uitbreiden van het mCRL2 model. In deze uitvraag vragen zij ook of er de mogelijkheid is voor het visualiseren van de simulatie. Hierbij geven zij al een korte omschrijving van wat zij graag gevisualiseerd willen hebben. Dit biedt voor mij een mooi startpunt waar ik mijn vragen aan de stakeholders op kan baseren.

Voor het achterhalen van de softwarerequirements ben ik begonnen met een kleine brainstorm sessie met de stakeholders. Ik heb gekozen om te beginnen met een brainstorm omdat hier vaak grove ideeën naar boven komen. Aan het begin van een project kan het zo zijn dat de stakeholders hun ideeën nog niet helemaal uitgedacht hebben en nog geen consensus gevormd hebben. Een brainstorm stelt mij in staat in grove lijnen een beeld te vormen van het gewenste resultaat van het project.

De ideeën die bij de brainstorm naar boven zijn gekomen kan ik combineren met de beschrijving die in de hierboven genoemde uitvraag staat om meer gerichte vragen aan de stakeholders te stellen. In het volgende overlegmoment ben ik opnieuw met de stakeholders in gesprek gegaan en deze keer ben ik dieper op de verschillende punten ingegaan. Ik heb gevraagd om een meer exacte toelichting van de punten die zij gerealiseerd willen zien. Tussen deze bespreking en de brainstorm zat een week. In die tijd hebben de stakeholders hun ideeën verder uit kunnen werken.

Nadat alle requirements zijn opgesteld heb ik bij de stakeholders geïnformeerd naar prioriteit van de verschillende requirements. Voor elke individuele requirement is bepaald hoe belangrijk deze is voor het slagen van het project. Op basis hiervan zijn de requirements geprioriteerd volgens de MoSCoW methode. (Swart, N., 2017)

Na het uitwerken van de requirements heb ik een document opgesteld met daarin een overzicht van de verzamelde requirements en de prioritering. Zie bijlage A ‘Requirements’. In dit document licht ik toe hoe de requirements tot stand zijn gekomen en leg ik uit waarom sommige requirements geweigerd zijn. Dit document heb ik gedeeld met de stakeholders zodat zij deze konden doornemen en tijdens het volgende contactmoment feedback konden geven. Het document is aangepast op basis van de ontvangen feedback en opnieuw gedeeld. Wanneer de stakeholders geen feedback meer hadden om te geven heb ik het vergaren van de requirements afgerond door de huidige requirements als definitief neer te zetten.

De uiteindelijke requirements zijn als volgt:

#	Omschrijving	Prioriteit
R1	De visualisatie moet de status van een TTD of VSS (free/occupied/ambiguous/unknown) laten zien aan de hand van vier kleuren (groen/geel/oranje/rood respectievelijk).	M
R2	De visualisatie moet de huidige status van de statemachine van een VSS en de laatste overgang laten zien.	M
R3	De visualisatie moet duidelijk weergeven wat de voorkant van een trein is en wat de achterkant.	M
R4	De visualisatie moet de Movement Authority van een trein laten zien aan de hand van een lijn aan de voorkant van de trein.	M
R5	De visualisatie moet de status van een trein (integrity) laten zien.	M

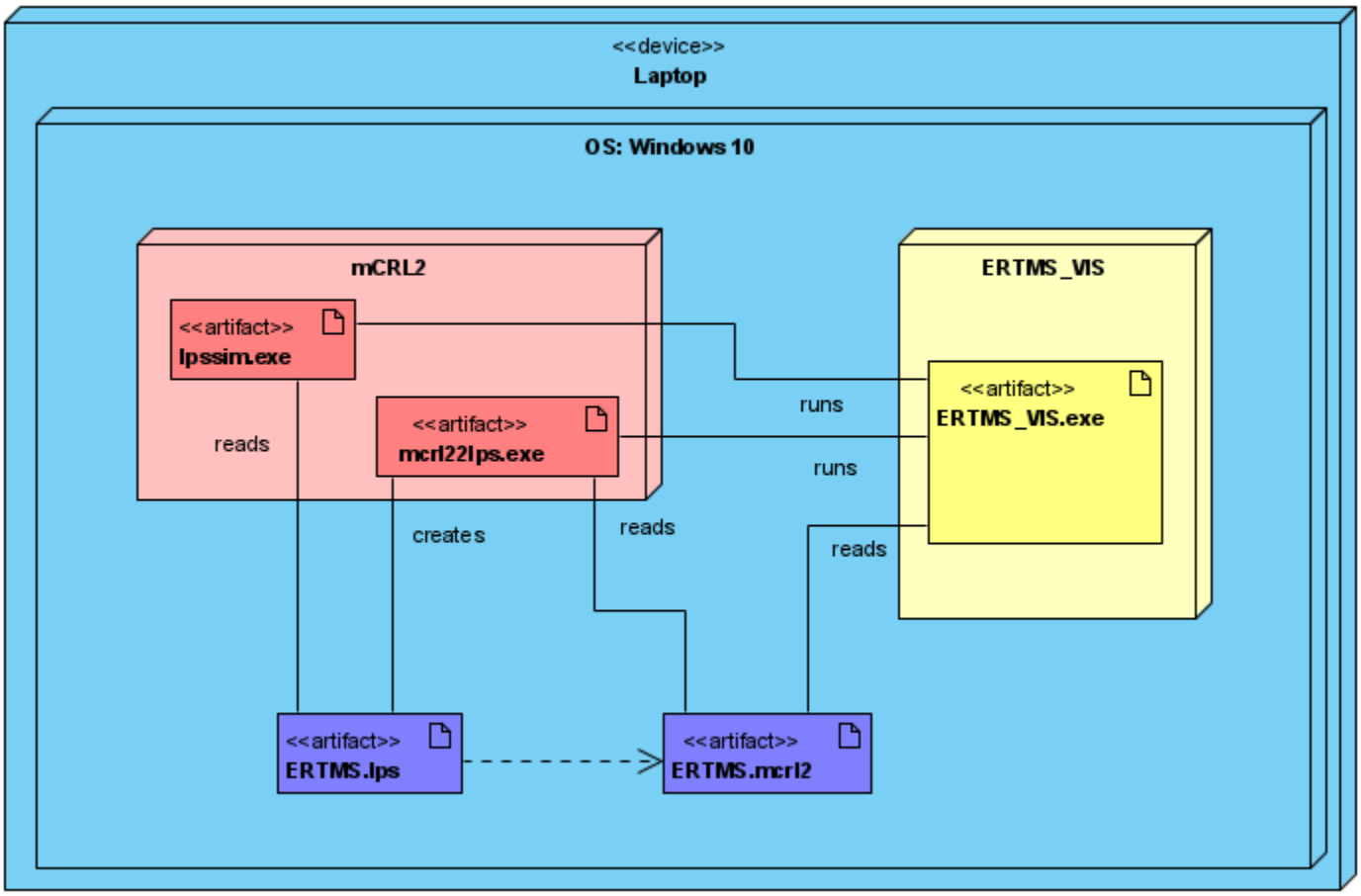
R6	De visualisatie moet de status van een Timer (stopped/running/expired) laten zien.	M
R7	De visualisatie moet laten zien bij welke VSS een Timer hoort.	S
R8	De visualisatie moet de gevolg van een bepaalde input duidelijk laten zien. Een verandering in de output moet duidelijk herleidbaar zijn naar de input.	S
R9	Het uiterlijk van de visualisatie dient zo veel mogelijk overeen te komen met de afbeeldingen in de HL3 specificatiedocumenten.	S
R10	De software moet alle gebeurtenissen in de simulatie bijhouden in een log.	M
R11	De software moet een trace-bestand kunnen inladen en automatisch kunnen doorlopen met instelbare periode tussen stappen.	C
R12	De software moet een scenario met automatische willekeurige overgangen kunnen doorlopen.	S
R13	De software moet de gebruiker de mogelijkheid geven de waardes van timers te veranderen.	W
R14	De visualisatie moet ook niet-hl3 timers (zoals T_NVCONTACT) kunnen visualiseren.	W
R15	De simulatie moet de rijweginstelling automatisch kunnen laten verlopen met de mogelijkheid voor de gebruiker om in te grijpen.	W

In dit overzicht is te zien dat enkele requirements geweigerd zijn (aangegeven met prioriteit 'W', voor *won't have*).

R13 is geweigerd omdat het HL3 model van de TU/e enkel de statussen (*running*, *expired* of *stopped*) van timers geïmplementeerd heeft. Er zijn geen timerwaardes van een x aantal seconden om in te stellen zoals dat in werkelijkheid voor het HL3 systeem wel is. Omdat de visualisatie gebruik maakt van de bestaande simulator kan deze requirement niet geaccepteerd worden.
R14 is geweigerd omdat er geen niet-HL3 timers zijn opgenomen in de uitwerking van het HL3 systeem door de TU/e.
R15 is geweigerd omdat de beschreven functionaliteit niet is opgenomen in het HL3 model van de TU/e.

4.3 Bepalen benodigde onderdelen

Zoals al eerder omschreven moet de visualisatie applicatie gebruik maken van de bestaande mCRL2 simulator. Dit betekent dat mijn applicatie afhankelijk zal zijn van andere onderdelen op hetzelfde computersysteem. De mCRL2 toolkit bevat verschillende applicaties met verschillende doeleinden. Ik heb een deployment diagram opgesteld om weer te geven welke applicaties nodig zijn voor het simuleren van een model. Deze heb ik gevonden in de documentatie van de mCRL2 toolkit.



Figuur 2 Deploymentdiagram

Dit diagram is tevens te vinden als bijlage B 'Deploymentdiagram'.

Het systeem waarop de applicatie draait is hier aangegeven als een Windows 10 laptop. De stakeholders hebben geen eis opgesteld wat betreft het besturingssysteem. Ik heb er zelf voor gekozen te ontwikkelen op en voor Windows 10 omdat de laptops die ProRail aan haar werknemers verstrekt dit besturingssysteem gebruiken.

De onderdelen onder mCRL2 bestaan reeds. Dit is de toolkit die de TU/e ontwikkeld heeft voor de mCRL2 taal. De hierbinnen weergegeven onderdelen zijn nodig voor het uitvoeren van een simulatie op basis van een .mcrl2 bestand.
Het onderdeel ERTMS.mcrl2 is ook een bestaand onderdeel. Dit bestand bevat de uitwerking van het ERTMS HL3 model van de TU/e. Dit model kan met de mCRL2 toolkit gelineariseerd en vervolgens gesimuleerd worden.

ERTMS_VIS.exe is afhankelijk van vier andere onderdelen:

1. ERTMS.mcrl2:
   Dit is een ASCII bestand waarin het model van ERTMS HL3 in de mCRL2 taal is gedefinieerd. Dit bestand wordt door ERTMS_VIS.exe uitgelezen om de initiële status van het model te bepalen.
2. ERTMS.lps
   Dit is het Linear Process Specification (LPS) bestand. Dit bestand is het resultaat van het uitrekenen van de alle mogelijke overgangen van de statussen van dit model. Dit wil zeggen dat van alle verschillende onderdelen in het model alle mogelijke statussen en de daarbij behorende mogelijke overgangen van te voren zijn bepaald. Dit proces heet linearisatie. Het bestand is een binair bestand en wordt gegenereerd door mcrl22lps.exe op basis van ERTMS.mcrl2. Een .lps bestand is het type bestand dat door de simulator gebruikt kan worden.
3. mcrl22lps.exe:
   Dit is een applicatie uit de mCRL2 toolkit die een .lps bestand genereerd op basis van een gegeven .mcrl2

bestand. Dit programma rekent van een model, vanuit een gegeven startpunt, alle mogelijke statusovergangen van de verschillende onderdelen uit. Deze applicatie wordt uitgevoerd door ERTMS_VIS.exe om een simuleerbaar .lps bestand te genereren aan de hand van een .mcrl2 bestand dat de gebruiker selecteert.

4. lpssim.exe:
   Deze applicatie is de simulator van de mCRL2 toolkit. Het simuleert een model uit een .lps bestand. Het neemt een aangewezen status als startpunt en biedt een gebruiker verschillende acties aan die uitgevoerd kunnen worden. Elke actie verandert de status van bepaalde onderdelen. Afhankelijk van de status van het gehele model zijn er beperkte acties mogelijk. ERTMS_VIS.exe fungeert in dit geval als een soort vertaallaag en geeft de status van het model visueel weer. Ook stuurt het stuurt commando's naar de simulator om bepaalde acties uit te voeren. Deze acties zijn geselecteerd door de gebruiker van ERTMS_VIS.exe.

4.4 Opstellen Plan van Aanpak

De ontwikkelmethode, softwarerequirements en benodigde onderdelen zijn nu bekend. Nu moet ik gaan bepalen welke werkzaamheden ik moet uitvoeren om de software te realiseren. Ik heb een plan van aanpak opgesteld waarin ik een omschrijving geef van de opdracht, welke risico's er bij dit project een rol spelen, welke producten er opgeleverd zullen worden, hoe ik de realisatie van de producten ga aanpakken, wat de requirements van de visualisatiesoftware zijn en hoe deze gewaarborgd zullen worden.

Dit document is met de stakeholders gedeeld. Zij konden dit doornemen en voorzien van feedback. In de eerste versie van mijn plan van aanpak heb ik de mogelijke risico's niet opgenomen. Sommige van deze risico's had ik al wel besproken met de stakeholders. De stakeholders hebben mij het advies gegeven deze risico's in het plan van aanpak op te nemen.

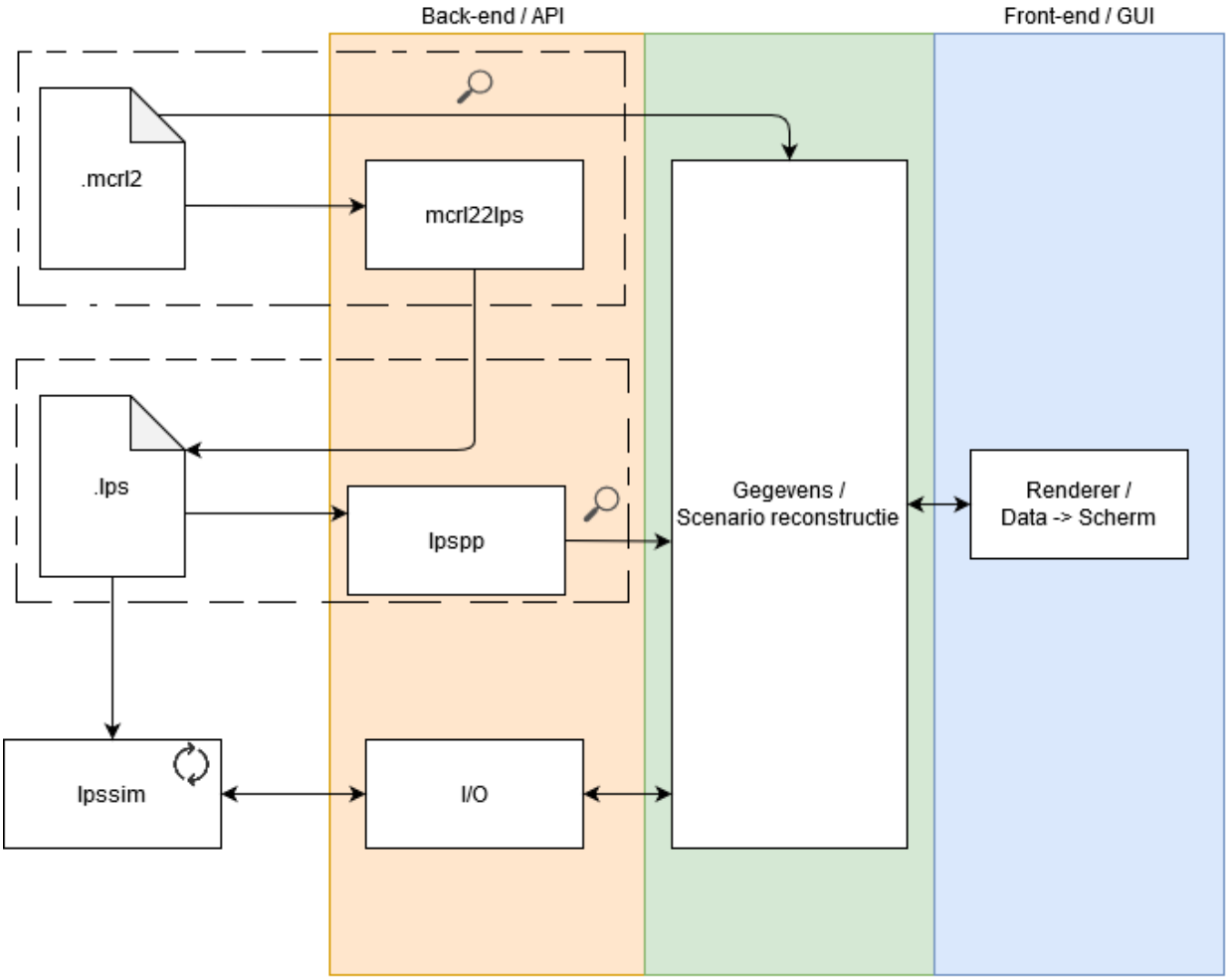
Ook in dit plan beschrijf ik gebruik te maken van een Agile ontwikkelingsmethode. Zoals ik al eerder omschreven heb was dit het oorspronkelijke idee maar is dit gewijzigd naar een lineaire methode.

4.4.1 Domain Driven Design

In het plan van aanpak (en ook al in het afstudeerplan) omschrijf ik gebruik te gaan maken van Domain Driven Design (DDD). In de opleiding, tijdens het uitstroomtraject 'Internet of Things', heb ik geleerd over dit ontwerpprincipe. Dit principe neemt het domein van de applicatie als 'hart' van het systeem. Het domein is te definiëren als het kennisgebied waar de logica van de applicatie om draait. In het geval van deze opdracht is het domein het ERTMS HL3 systeem. Als je software ontwikkeld volgens DDD leg je de onderdelen binnen het domein vast in een Domein laag. Vervolgens bouw je daar lagen met de benodigde logica omheen om de gewenste applicatie te ontwikkelen.

De applicatie moet gebruik maken van de mCRL2 simulator. Daarom moet er een Infrastructuur laag toegevoegd worden die logica bevat om te interacteren met de simulator.
Er zal ook een Applicatie laag moeten zijn. Deze laag bevat meer generieke logica die het algemeen functioneren van de applicatie bepaalt.
Daarbovenop zal er ook een User-Interface laag of Presentatie laag moeten zijn die de logica bevat voor het visualiseren van het domein en het geven van de grafische userinterface.

De volgende figuur geeft een abstracte weergave van de lagenstructuur:



Figuur 3 Schets lagenstructuur

Deze schets is niet het ontwerp van de applicatie volgens de DDD methode. Dit ontwerp zal in de conceptfase gemaakt worden. Het doel van de schets is de stakeholders duidelijk maken dat de applicatie verdeeld is in verschillende onderdelen en dat deze onderdelen een eigen functie hebben.

De schets geeft weer dat er een back-end is welke interacteert met de simulator en de daarbij behorende onderdelen. Er is een middellaag waarin de applicatie-specifieke logica en de reconstructie van het scenario zich bevindt. Ook is er een front-end waar de logica in zit voor het visualiseren van het scenario.

4.5 Opstellen to-do punten

Ik heb voor mijzelf een overzicht gemaakt waarin de verschillende onderdelen staan die ontwikkeld moeten worden voor de applicatie. Dit overzicht kan ik gebruiken tijdens het ontwerpen en realiseren van de software.

Het overzicht ziet er als volgt uit:

TODO			Req #	OMSCHRIJVING
Front-end / Presentatielaag				
	Tekenen van VSS			
		status	R1, R9	Weergeven van de status van een VSS dmv kleur
		state machine	R2, R9	Weergeven van de huidige en vorige status en laatste overgang van een statemachine van een VSS
		onderbreking	R9	Weergeven van de grens van een VSS dmv het bijbehorende symbool
		timers	R6, R7	Weergeven van de status van de timers van een VSS
	Tekenen van TTD			
		status	R1, R9	Weergeven van de status van een TTD dmv kleur
		onderbreking	R9	Weergeven van de grens van een TTD dmv het bijbehorende symbool
		timers	R6, R7	Weergeven van de status van de timers van een TTD
	Tekenen van Trein			
		status	R3, R5	Weergeven van de status van een trein dmv kleur
		lengte authority	R4	Weergeven End of Authority van een trein dmv een lijn
		timers	R6	Weergeven van de status van de timers van een trein
	Transition mogelijkheden			
		knoppen		Weergeven van knoppen voor elke mogelijke actie
		preview	R8	Weergeven wat een mogelijke actie veranderd voordat de gebruiker de actie uitvoerd
	Legenda		R1 t/m R6	Weergeven van een legenda
	Geschiedenis (trace)		R10	Weergeven van de acties die plaatsgevonden hebben
Middellaag / Applicatielaag				
	Scenario opbouwen			Opbouwen van het scenario door middel van het lezen/parsen van het geselecteerde .mcrl2 bronbestand
	Bijwerken scenario			Output van lpssim parsen na het uitvoeren van een actie en de veranderingen in het scenario doorvoeren
	Selecteren bronbestand			Gebruiker een .mcrl2 bestand laten selecteren
	Gebruikersinput			Input van de gebruiker ontvangen en verwerken
	Compileren lps			Compileren van het geselecteerde .mcrl2 bestand tot een .lps bestand voor lpssim
Middellaag / Domeinlaag				
	Scenario			Datastructuur en benodigde functies van scenario
	VSS			Datastructuur van VSS
	TTD			Datastructuur van TTD
	trein			Datastructuur van trein
Back-end / Infrastructuurlaag				
	LPSSIM Interface			Ontwikkelen van lpssim wrapper
		subproces		Uitvoeren en interacteren met lpssim subproces
		undo		draai de laatste actie terug
		redo		voer ongedaangemaakte actie opnieuw uit
		initial		Ga terug naar de initiële status
		goto		Ga terug naar een bepaald punt in de trace
		load	R10, R11	Laad een trace-bestand in
		save	R11	Sla de huidige trace op
		quit		Stop
		transition		Kies een van de mogelijke acties

Figuur 4 Overzicht to-do punten

Het uitwerken van deze punten vormt de ontwikkeling van de ERTMS_VIS.exe applicatie uit het deployment diagram.

Voor enkele punten is er aangegeven bij welke requirement(s) het hoort en onder welke laag het hoort. Ook is er een korte toelichting gegeven.
Bij andere punten is er niet aangegeven bij welke requirement(s) ze horen. Dit betekent dat dit punt niet naar aanleiding van een wens van de stakeholders is toegevoegd. Ik heb besloten dat het onderdeel noodzakelijk is voor het realiseren van een werkende en gebruiksvriendelijke applicatie. In dit overzicht is goed te zien dat de meeste eisen van de stakeholders gaan over de manier waarop de simulatie gevisualiseerd wordt, en niet zo zeer over de verdere werking van de applicatie.

4.5.1 Afwijking t.o.v. plan

In mijn afstudeerplan noem ik ook het ontwikkelen van een API. Dit heb ik in dit verslag niet opgenomen. Mijn idee in het plan was dat ik een API zou ontwikkelen die een interface zou bieden voor de simulator uit de mCRL2 toolkit. Wat ik uiteindelijk ontwikkeld heb zou ik geen API maar een *wrapper* noemen. Hieronder beschrijf ik het verschil.

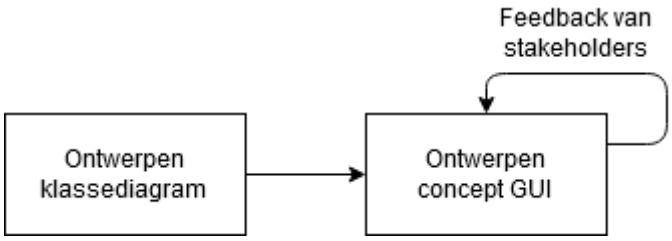
Een API biedt een interface voor een onderliggende applicatie. De functies of *entry-points* die beschikbaar worden gesteld door een API roepen vaak meerdere onderliggende functies uit een applicatie op. Ook zijn de functies van een API vaak op een hoger niveau qua abstractie. Denk bijvoorbeeld aan een REST API waar de functies aangeroepen worden via een HTTP request. Dit is een hoger niveau van abstractie dan de functies die daadwerkelijk in een applicatie zitten.

De mCRL2 simulator heeft al een API. Je kan als gebruiker bepaalde acties oproepen door data via een console te sturen. Ik heb een wrapper ontwikkeld voor deze API. De wrapper neemt alle functies van de API één op één over. Er

worden geen functies aangeboden die de simulator niet heeft en alle functies die de simulator heeft zijn beschikbaar via de wrapper. Het enige verschil is dat de API nu beschikbaar is door middel van functies binnen de applicatie zelf in plaats van via een console. Dit verlaagt feitelijk het abstractieniveau waarop de functies worden aangeroepen.

5 Fase 1 (concept)

In deze fase zijn de volgende werkzaamheden uitgevoerd:

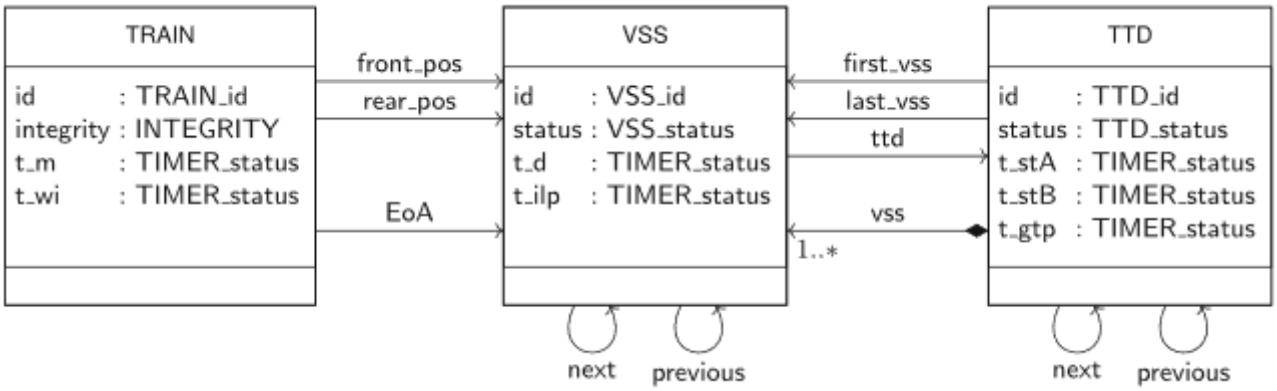


Figuur 5 Werkzaamheden Fase 1

5.1 Ontwerpen klassediagram

Zoals ik al eerder omschreven heb gebruik ik het principe Domain Driven Design voor het ontwerpen van de software. In dit project is het centrale domein de onderdelen uit het HL3 systeem dat gesimuleerd wordt. Hierin bevinden zich een bepaald aantal treinen, TTD's en VSS'en. Dit domein is vastgelegd in de Domain Layer, of de Domein laag.

Het ontwerp van de Domein laag (_3_DomainLayer) is gedeeltelijk gebaseerd op het volgende figuur:



Figuur 6 Datamodel mCRL2 ERTMS HL3
Uit Modelling and analysing ERTMS hybrid level 3 with the mCRL2 toolset (p.101), door M. Bartholomeus, B. Luttik, & T. Willemse, 2018, Springer.

Dit figuur omschrijft de structuur waarmee de onderdelen van het ERTMS HL3 systeem zijn vastgelegd in het mCRL2 model. Omdat mijn applicatie een reconstructie moet maken van deze structuur heb ik het in mijn ontwerp nauw overgenomen.

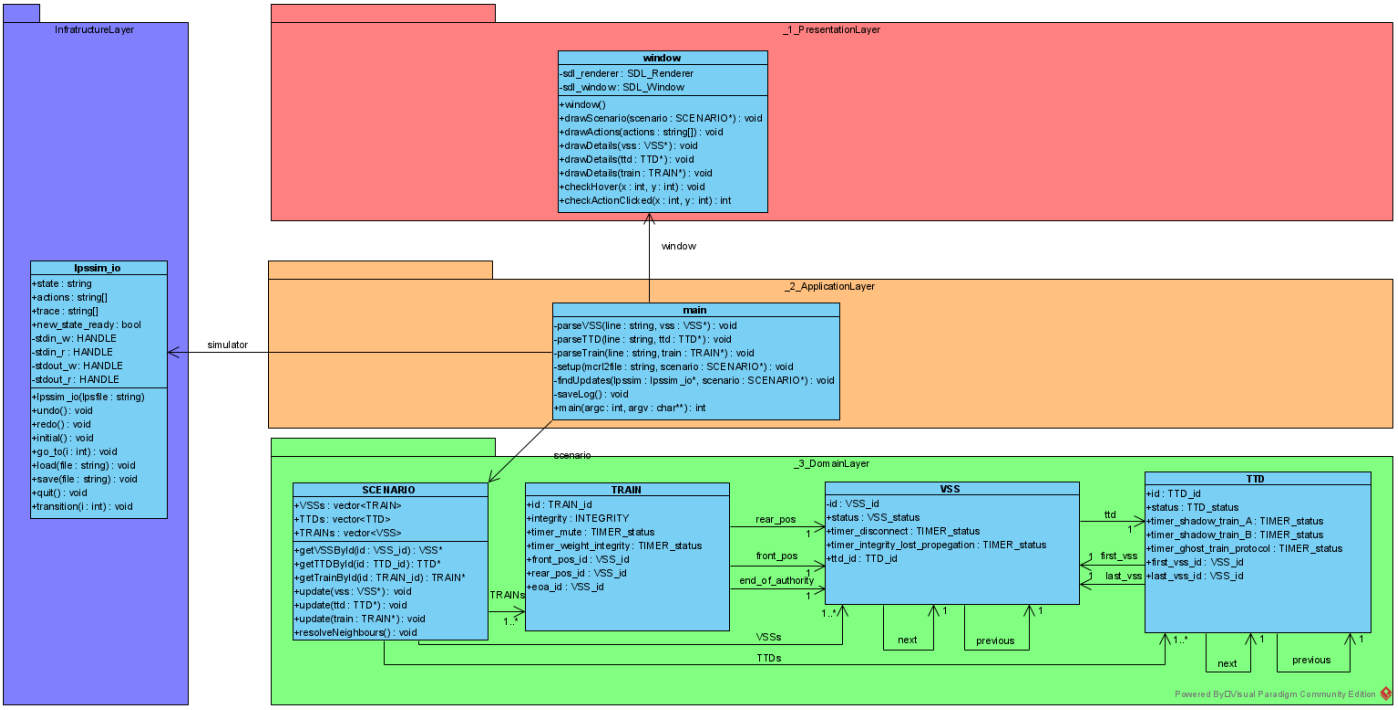
Om de centrale domein laag heen bevinden zich enkele andere lagen. Elke laag heeft een beperkte scope en verantwoordelijkheid.

De Infrastructuur laag (InfrastructureLayer) bevat een klasse 'lpssim_io' die de back-end vormt. Deze laag is verantwoordelijk voor de interactie met de simulator uit de mCRL2 toolkit. De lpssim_io klasse is een wrapper voor de lpssim.exe applicatie. De wrapper maakt alle functies van lpssim.exe beschikbaar tot de andere klasse binnen de visualisatie applicatie.
De lpssim_io klasse draait de simulator applicatie als een subproces. Er is gekozen voor het gebruiken van de binaire applicatie in tegenstelling tot, bijvoorbeeld, het implementeren van een library. Dit houdt het niveau van separatie tussen de mCRL2 simulator en mijn visualisatie applicatie hoog. Daarmee is de visualisatie minder afhankelijk van de simulatie. Dit zorgt ervoor dat de visualisatie niet bijgewerkt hoeft te worden als de TU/e in de toekomst de simulator bijwerkt.

De Presentatie laag (_1_PresentationLayer) bevat een klasse die de front-end van de applicatie vormt. Deze klasse is verantwoordelijk voor het tekenen van het venster van de applicatie en het visualiseren van het scenario zoals deze in het domein is opgeslagen.

De Applicatie laag (_2_ApplicationLayer) bevat de hoofdklasse van de applicatie en bevat meer generieke code. De klasse verbindt de drie andere lagen aan elkaar. Het bevat code die in de output van de simulator wijzigingen in het scenario herkent en deze doorvoert naar het interne opgeslagen scenario. Op deze manier bouwt de applicatie een reconstructie van het scenario in de simulatie. Daarnaast houdt deze klasse ook de input van de gebruiker bij en verteld het de klasse in de Presentatie laag wanneer deze het venster opnieuw moet tekenen.

Het ontwerp ziet er als volgt uit:



Figuur 7 Ontwerp klassediagram

Dit diagram is ook te vinden onder bijlage C ‘Klassediagram’.

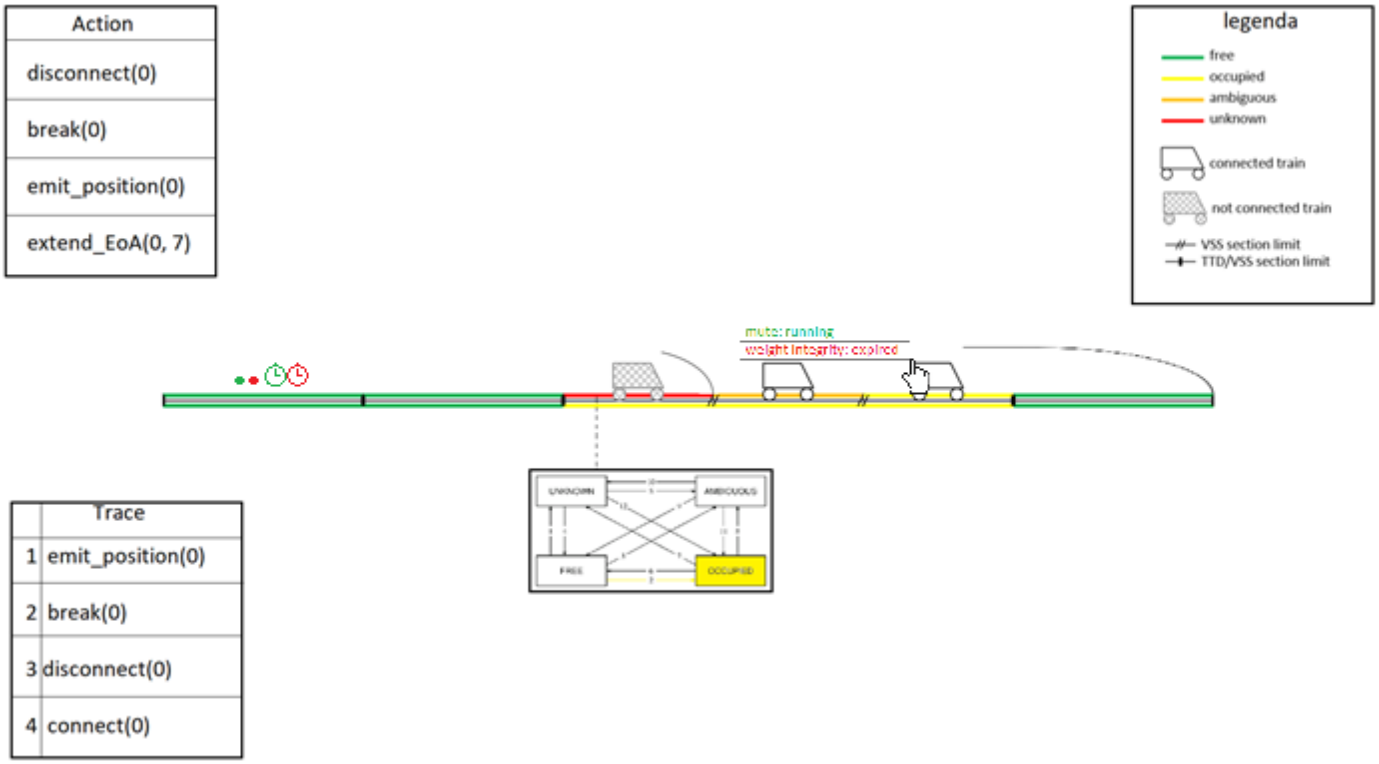
5.2 Ontwerp concept GUI

Een belangrijk onderdeel van de applicatie is de grafische user interface (GUI). Dit visualiseert de status van de simulatie en is de enige manier waarmee de gebruiker interactie heeft met applicatie.

Een van de requirements is dat de visuele weergave van onderdelen zo veel mogelijk overeen moet komen met de voorbeelden die te vinden zijn in de documenten waarin de HL3 specificatie gedefinieerd is. De stakeholders zijn bekend met deze documenten. Het zou praktisch zijn als de visualisatie hiermee overeenkomt zodat de onderdelen voor hen makkelijk te herkennen zijn. Daarom heb ik de illustraties uit het specificatiedocument gebruikt als inspiratie en voorbeelden voor het concept van mijn visualisatie.

Aan de hand van de opgestelde requirements is bepaald welke verschillende onderdelen er gevisualiseerd moeten worden. Voor elk onderdeel heb ik een of meerdere voorbeelden uitgewerkt. Deze voorbeelden heb ik samen met een korte toelichting en een voorbeeld van een complete visualisatie in een document verzameld. Dit document is gedeeld met de stakeholders zodat zij deze konden doornemen en eventueel van feedback konden voorzien. Dit document is te vinden als bijlage D ‘Voorbeeld GUI’.

Het voorbeeld van de complete GUI ziet er als volgt uit:

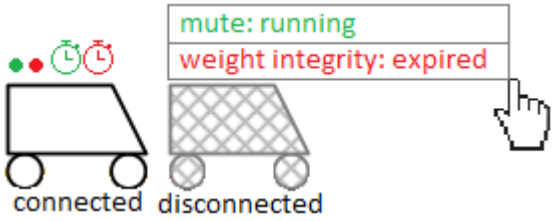


Figuur 8 Voorbeeld GUI

Dit is een schets van het venster van de visualisatie applicatie.
Links bovenaan zijn de acties weergegeven waarvan de gebruiker er één kan kiezen om uit te voeren. Wanneer een actie is uitgevoerd komen hier de volgende mogelijke acties te staan.
Links onderin is een lijst weergegeven waarin de acties die zijn uitgevoerd te zien zijn. Bovenaan de lijst staat de laatst uitgevoerde actie.
In het midden is de reconstructie van het scenario weergegeven. Hier zijn de VSS'en, TTD's, timers en treinen te zien.
Rechtsboven aan is een legenda van de visualisatie weergegeven.

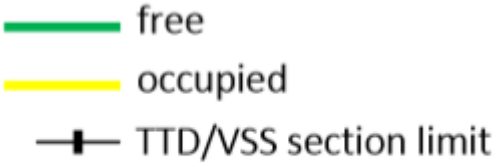
Voor de verschillende onderdelen heb ik ook meer gedetailleerde voorbeelden uitgewerkt:

Trein/Timers:



Een trein kan afgebeeld worden op dezelfde manier als in het specificatiedocument.
Timers kunnen weergegeven worden door middel van een stip of icoon dicht bij het onderdeel waar het bij hoort. Er zou een duidelijkere weergave gegeven kunnen worden door middel van tekst wanneer de cursor van de gebruiker op een onderdeel of timer staat.

TTD:

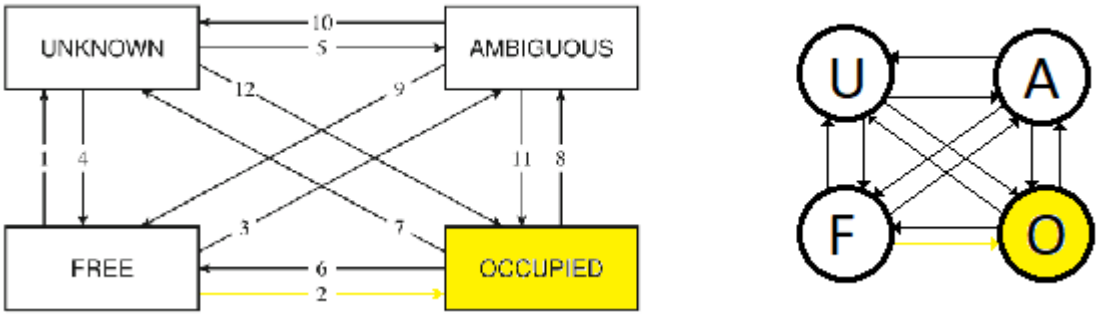


TTD's kunnen op dezelfde manier geïllustreerd worden als in het specificatiedocument. De kleur is afhankelijk van de status van de TTD. De lengte is afhankelijk van hoeveel VSS'en er onder de TTD vallen. De onderbreking van een TTD is ook volgens de voorbeelden in het specificatiedocument weer te geven. Een grens van een TTD is ook de grens van de laatste VSS van de TTD.

VSS:

Ook de VSS'en kunnen weergegeven worden zoals deze in het specificatiedocument weergegeven worden. De lengte van elke VSS is het zelfde. De lengte wordt bepaald aan de hand van hoeveel VSS'en er op het scherm moeten passen. VSS'en worden boven de TTD's geplaatst. De grens van een VSS wordt ook geïllustreerd volgens het voorbeeld uit het specificatiedocument.

VSS State Machine:



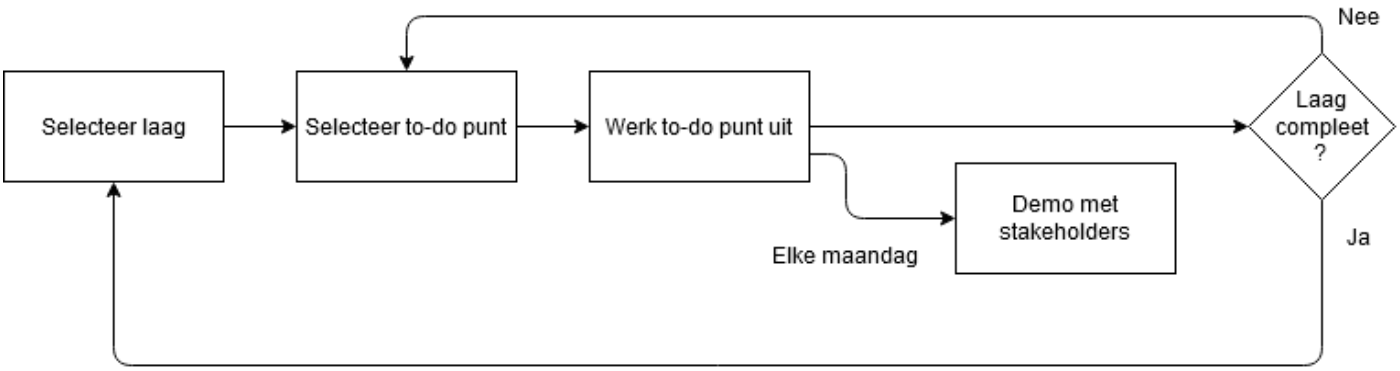
De state machine van een VSS kan weergegeven worden zoals deze in het specificatiedocument staat (links). Er kan ook een versimpelde versie gebruikt worden om minder ruimte op het scherm in te nemen (rechts).

Movement Authority / End of Authority:

De Movement Authority (MA, ook wel End of Authoirity of EoA) is de afstand waarvoor de trein toestemming heeft om af te leggen. Dit kan weergegeven worden door middel van een afbuigende lijn die begint bij de voorkant van de trein en rijkt tot het einde van de laatste VSS binnen de MA. Dit is volgens het specificatiedocument.

6 Fase 2 (ontwikkel)

Deze fase dient voor het daadwerkelijk programmeren van de verschillende onderdelen. In deze fase ben ik als volgt te werk gegaan:



Figuur 9 Werkwijze Fase 2

Allereerst kies ik één van de vier lagen om uit te werken. Uit deze laag kies ik één punt om uit te werken. Dit doe ik zolang er nog punten te doen zijn voor de gekozen laag. Als alle punten van een laag uitgewerkt zijn kies ik een volgende laag. Elke maandag heb ik een demo gegeven van de nieuwe functionaliteit aan de stakeholders. Zolang er nog geen visualisatie ontwikkeld was om een demo mee te geven heb ik gebruik gemaakt van het weergeven van debug-informatie in de console. Op deze manier kon ik zonder visualisatie de stakeholders inlichten over de werking van de applicatie en kon ik ook mijn voortgang laten zien.

De ontwikkeling lijkt volgens de bovenstaande werkzaamheden iteratief. Het wijkt echter af van Agile of andere iteratieve ontwikkelmethodes omdat de analyse- en ontwerpwerkzaamheden al uitgevoerd zijn en in deze fase niet nog eens uitgevoerd worden. Daarnaast is na het resultaat van het uitwerken van één to-do punt geen compleet of leverbaar product.

Voor het realiseren gebruik ik de C++ programmeertaal. Ik hiervoor gekozen omdat deze taal veel controle biedt voor het definiëren van structuren en objecten. Ook zijn er voor deze taal veel library's beschikbaar voor het implementeren van grafische interfaces. Ook leek mij dit project een geschikte manier om meer over deze taal te leren, aangezien deze weinig behandeld is in de opleiding.

6.1 Domeinlaag

Allereerst heb ik de Domein laag uitgewerkt. Het leek mij een goed idee met deze laag te starten, omdat deze relatief weinig code bevat maar meer datastructuren. Omdat de software ontworpen is volgens DDD is het ook handig het domein als eerste uit te werken omdat veel van de andere onderdelen hiervan afhankelijk zijn.

Het uitwerken van deze laag verliep soepel. Voor de VSS, TTD en TRAIN onderdelen hoefde enkel datastructuren in C++ header bestanden gedefinieerd te worden. Voor het SCENARIO onderdeel moest er wel functionele code geïmplementeerd worden, welke ook gecompileerd moest worden.

6.2 Infrastructuurlaag

Na de Domein laag ging ik aan de slag met de Infrastructuur laag. Deze laag bevat slecht één klasse, namelijk de lpssim_io klasse. Deze klasse dient als wrapper voor de functionaliteit van de onderliggende simulator. Deze klasse start de simulator applicatie (lpssim.exe) in een subproces en biedt de benodigde functionaliteit door de output van de simulator af te vangen en input naar de simulator te sturen. De simulator is een console applicatie, dus alle in- en output verloopt via ASCII stdin en stdout pipes. Normaal gesproken stuurt de console van de gebruiker de ASCII data die de gebruiker intypt over de stdin pipe naar de simulator. In dit geval stuurt en ontvangt het lpssim_io onderdeel dus deze data.

6.2.1 Uitdagingen

Dit bleek een van de uitdagendste onderdelen om uit te werken. Dat kwam voornamelijk door twee punten waar ik op vast liep.

6.2.1.1 Subproces in- en uitvoer

Voor dit onderdeel is het dus nodig dat de simulator als subproces uitgevoerd wordt, en dat de in- en uitvoer hiervan door de lpssim_io klasse wordt geregeld. Het draaien van subprocessen in Windows was voor mij nieuwe stof. Uiteindelijk bleken de buffers tussen de twee processen niet te werken zoals ik dacht. Het vergde veel tijd om te begrijpen hoe de buffers werken en hoe je er correct uit moet lezen en naar moet schrijven. Ook was er weinig documentatie beschikbaar die overeenkwam met mijn situatie.

Uiteindelijk heb ik kunnen leren over de werking van de buffers door veel te experimenteren en te leren van voorbeelden die op het internet te vinden zijn.

6.2.1.2 lpssim broncode

Tijdens het onderzoek naar de werking van de in- en uitvoer van de simulator wou ik de broncode van de simulator aanpassen om mij meer informatie te geven die mij kon helpen met debuggen. Het mCRL2 project is een open-source project dus de broncode aanpassen is mogelijk. Ook is er op de website van mCRL2 documentatie te vinden over het compileren van de software. Aanvankelijk lukte het mij niet om de broncode zelf te compileren. Contact met de TU/e leverde mij ook niet de juiste handvatten op om dit op te lossen.

Uiteindelijk is het compileren van de broncode toch gelukt door verschillende versies van een benodigde library uit te proberen en de juiste te vinden, wat veel tijd gekost heeft.

6.3 Applicatielaag

Na de Infrastructuur laag begon ik aan de Applicatie laag. Deze laag heeft één klasse en heeft als doel het samenbrengen en het aansturen van de overige lagen. Wanneer er data van de ene laag naar de andere moet, gaat dat via deze laag. In deze laag vindt ook conversie van de data plaats, mocht dit nodig zijn.

De functionaliteit van deze laag is samen te vatten in de volgende rollen:

1. Een initieel scenario opbouwen op basis van het model in het .mcrl2 bestand.
2. De simulator starten en aansturen op basis van input van de gebruiker.
3. Het scenario aanpassen op basis van data uit de simulator.
4. De visualisatie aanpassen op basis van data uit het scenario.

Ik zal voor deze punten uitleggen waarom ze nodig zijn en ik zal een omschrijving geven van de geïmplementeerde functionaliteit.

6.3.1 Initieel scenario

De simulator geeft nooit de data van het complete scenario in één keer. De simulator geeft alleen de status van een onderdeel wanneer deze verandert ten opzichte van het startpunt. Dit betekend dat mijn applicatie niet een complete reconstructie van het scenario kan maken wanneer het de simulator start. Het startpunt van het scenario moet dus ergens anders vandaan gehaald worden.

Het startpunt staat wel gedefinieerd in het .mcrl2 bestand. Dit bestand is een ASCII bestand dat uitgelezen kan worden door mijn applicatie. Voordat mijn applicatie de simulator start leest het dit bestand door en zoekt het naar het initiële scenario. De applicatie vindt de statussen van het scenario door te zoeken naar de sleutelwoorden “VSSs_config’, “TTDs_config” en “TRAINs_config”. Hiermee maakt de applicatie een reconstructie van het initiële scenario.

6.3.2 Scenario aanpassen

Elke keer als de simulator een actie uitvoert kan de status van een onderdeel van het scenario veranderen. De simulator gebruikt de console om in tekst aan te geven hoe het onderdeel verandert. De applicatie vangt deze uitvoer af en herkent aan de hand van de symbolen “ -> ” wanneer een onderdeel van status verandert. Vervolgens

leest het de tekst om dit symbool heen en achterhaald het wat de nieuwe status is. Daarna werkt de applicatie de reconstructie van het scenario bij met de nieuwe data.

De implementatie van dit onderdeel en het hierboven omschreven onderdeel was oorspronkelijk niet gepland. Deze bleken noodzakelijk nadat ik tegen een probleem aan liep. Dit licht ik verder toe in paragraaf 6.3.5.

6.3.3 Visualisatie aanpassen

Elke keer als de simulatie een actie uitvoert kan het scenario veranderen. Dit betekend dat de visualisatie ook na elke actie bijgewerkt moet worden. Na elke actie werkt de applicatie het scenario bij zoals hierboven is omgeschreven. Daarna laat het de klasse in de Presentatie laag de visualisatie bijwerken door het oproepen van een functie. Hierbij geeft het een verwijzing mee naar het scenario dat gevisualiseerd moet worden als parameter mee.

6.3.4 Simulator aansturen

Dit gedeelte van de applicatie start en stuurt de simulator aan. De gebruiker kan kiezen welke actie er uitgevoerd moet worden. De klasse in de Applicatie laag ontvangt de keuze van de gebruiker. Vervolgens stuurt het de simulator aan door de simulator de gekozen actie te laten uitvoeren. De klasse ontvangt input van de gebruiker door middel van het indrukken van toetsen op het toetsenbord of het bewegen of klikken van de muis binnen het venster.

6.3.5 Kwestie beperkte uitvoer

Zoals al eerder omschreven moet de applicatie de statussen van de verschillende onderdelen in het gesimuleerde scenario bijhouden. Echter bleek dat de simulator niet altijd de statussen van alle onderdelen in de uitvoer aangeeft. Het geeft enkel de status van een onderdeel dat van status veranderd. Ook geeft de simulator bij het starten van de simulatie geen compleet overzicht. Dit betekend dat de initiële status van het scenario niet te achterhalen is bij het starten van de simulatie. Dit betekend ook dat deze informatie ergens anders vandaan gehaald moet worden.

Ik heb onderzocht waar de initiële status van het scenario te vinden is. Dit staat in het .mcrl2 bronbestand gedefinieerd.

Deze kwestie heeft als gevolg dat er nieuwe functionaliteit in de applicatie moet worden opgenomen waarvan aan het begin niet duidelijk was dat dit nodig was. De benodigde functionaliteit “achterhaal de status van de verschillende objecten uit de simulatie” verandert naar “bouw het initiële scenario op met behulp van het .mcrl2 bronbestand. Compileer dit bestand en start de simulatie met het resulterende .lps bestand. Herken wanneer een object van status verandert en voer deze verandering door op de juiste plek binnen de interne reconstructie van het scenario”.

Deze revisie van de benodigde functionaliteit heeft ook effect op de haalbaarheid van het project. Om de effecten van deze revisie duidelijk te maken en om de manieren waarop ik hiermee om kan gaan te bespreken, heb ik een document opgesteld en deze heb ik met de stakeholders gedeeld. Met dit document kunnen zij een beeld scheppen van de situatie en hun eigen ideeën vormen over hoe ik hier het beste mee om kan gaan. In het contactmoment dat hier op volgde is besproken hoe ik verder zal gaan.
Dit document is te vinden onder bijlage G.

Uiteindelijk is besloten dat ik de functionaliteit van de applicatie van de back-end naar de front-end iteratief ontwikkel, in tegenstelling tot het eerst compleet ontwikkelen van de back-end, en daarna het compleet ontwikkelen van de front-end. Deze iteraties vinden plaats per to-do punt.
Bijvoorbeeld: Ik ontwikkel eerst de functionaliteit die van een VSS de status achterhaalt, daarna ontwikkel ik de functionaliteit die deze status van een VSS visualiseert. In tegenstelling tot het ontwikkelen van de functionaliteit voor alle statussen van alle verschillende soorten onderdelen, en vervolgens de functionaliteit voor het visualiseren van alle onderdelen.

Gelukkig kreeg ik het redelijk snel voor elkaar de extra benodigde functionaliteit te ontwikkelen waardoor de haalbaarheid van het project ook niet meer in gevaar was. Hierdoor was het ook niet meer nodig de ontwikkeling uit te voeren op de manier zoals ik die hierboven omschreven heb. Het was voor mijn duidelijk dat de applicatie compleet van back-end tot en met de front-end ontwikkeld kon worden. Dit heb ik gedaan volgens de manier waarop ik dit eerst bedacht had, namelijk eerst de complete back-end en daarna de complete front-end.

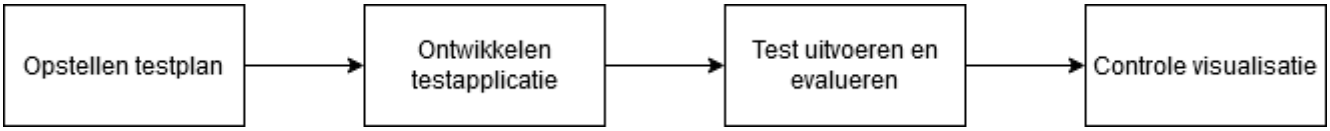
6.4 Presentatielaag

De Presentatie laag is nu de enige overgebleven laag om te ontwikkelen. Ook deze laag bevat maar één klasse. Deze klasse is verantwoordelijk voor het visualiseren van een scenario. Deze klasse krijgt een verwijzing naar een gereconstrueerd scenario om te visualiseren. Het loopt alle VSS'en, TTD's en treinen in het scenario langs en bepaalt hoe het deze onderdelen moet tekenen aan de hand van hun status. Daarnaast geeft het ook weer welke acties de gebruiker de simulator kan laten uitvoeren in de vorm van tekst. De weergave van deze visualisatie is gebaseerd op het voorbeeld dat in de conceptfase is uitgewerkt en besproken.

In het ontwerp uit de vorige fase staat dat de MA van een trein weergegeven wordt met een afbuigende lijn. Uiteindelijk is de implementatie gebleven bij een rechte lijn van de voorkant van de trein naar het einde van de laatste VSS onder de MA. Voor het tekenen van een afbuigende lijn zou een nieuwe library opgenomen moeten worden wat volgens mij en de stakeholders de moeite niet waard was. Ook een rechte lijn geeft de EoA duidelijk aan.

Het visualiseren gebeurt met behulp van de SDL2 library. Ik heb gekozen voor het gebruiken van SDL2 omdat dit een populaire library is bij hobby-programmeurs voor het ontwikkelen van videospellen. Er bestaat een grote actieve community waar ik oplossingen kan vinden voor de problemen waar ik tegen aan zal lopen. Ook geeft de populariteit onder beginners aan dat het een redelijk eenvoudige library is om onder de knie te krijgen.
De SDL2 library wordt gebruik om een venster op het scherm te creëren. In dit venster kan de library bepaalde vormen met een gegeven kleur tekenen, bijvoorbeeld een groene balk voor een vrije VSS. Ook kan de library PNG afbeeldingen tekenen om bijvoorbeeld een trein weer te geven. Daarnaast kan de library ook tekst naar het scherm tekenen om bijvoorbeeld de mogelijke acties weer te geven.

7 Fase 3 (test)



Figuur 10 Uitgevoerde werkzaamheden Fase 3

7.1 Opstellen testplan

De testfase begint met het opstellen van een plan. In dit plan staat welke onderdelen er getest worden en welke manier van testen je gebruikt. Er zijn meerdere manieren om het testen van software te benaderen.
Je kan van elk onderdeel en eventueel de daarbij behorende functies testen of deze correct geïmplementeerd zijn. Deze manier heet *Unit Testing* en wordt voornamelijk tijdens de ontwikkelfase van een project gebruikt om de implementatie te testen.
Je kan ook een niveau hoger testen. Bijvoorbeeld op *Component* niveau. Dan test je de verschillende onderdelen binnen een stuk software en de interactie tussen deze onderdelen.
Ook kan je op *System* niveau testen. Dan test je de werking van een compleet systeem binnen de infrastructuur waarop het moet draaien. Op deze infrastructuur kunnen andere systemen al aanwezig zijn.

Mijn doel voor deze testfase is testen of de verschillende onderdelen van mijn software, de Infrastructuur laag, de Domein laag, de Applicatie laag en de Presentatie laag, correct samenwerken. Deze onderdelen zijn elk op zichzelf ontwikkeld. ik wil testen of de interactie tussen deze onderdelen ook correct verloopt. Daarom ga ik een Componenten Integratie test uitvoeren.

Wanneer je software test kan je de interne werking van een onderdeel wel of niet meenemen in het testen.
Als je de interne werking, bijvoorbeeld de algoritmes en logica, van een klasse wilt testen, dan heet dat *White Box Testing*. Als dit niet belangrijk is en je bijvoorbeeld het gedrag of de externe interactie wilt testen, dan heet dat *Black Box Testing*.
Zoals ik al omschreven heb wil ik de interactie tussen de verschillende onderdelen testen. Ik beschouw elk onderdeel dus als een *Black Box*.

Daarnaast kan je kiezen of je de onderdelen incrementeel of in één keer gaat testen. Wanneer je in één keer test noem je dit *Big Bang*. Hierbij integreer je alle onderdelen die je wilt testen tegelijkertijd in je test. Daar tegenover, bij incrementeel testen, integreer je een onderdeel pas wanneer deze nodig is.
Ik heb er voor gekozen incrementeel te testen. Ik wil de interactie testen en hierbij zijn vaak maar twee onderdelen tegelijkertijd nodig. Ik integreer een onderdeel pas wanneer ik deze nodig heb.

Bij incrementeel testen kan je ook weer kiezen in welke volgorde je de onderdelen integreert. Dit kan *Bottom Up* of *Top Down*. Bij *Bottom Up* begin je met de onderdelen die weinig afhankelijkheden hebben en waarvan andere onderdelen afhankelijk zijn. Bij *Top Down* doe je dit omgekeerd.
Ik heb ervoor gekozen de *Bottom Up* volgorde aan te houden. Mocht de test van een onderdeel falen, dan weet je dat de fout in het geteste onderdeel zit, en niet in een onderliggend onderdeel dat nog niet getest is.

Ik heb een testplan opgesteld waarin ik de hierboven bedachte strategie, samen met de scope en criteria van deze test beschrijf. Dit document is te vinden als bijlage E ‘Testplan en rapport’. In dit plan staan de volgende testcases:

Testcase #	Requirement #	Getest component	Omschrijving
T1	n.v.t.	lpssim_io::lpssim_io()	Test constructie van lpssim_io object. Hiervoor moet de klasse lpssim.exe kunnen gebruiken.
T2	n.v.t.	SCENARIO::SCENARIO()	Test constructie van SCENARIO object.
T3	n.v.t.	main::setup()	Test of deze functie correct het initiële scenario opbouwt.
T4	n.v.t.	SCENARIO::resolveNeighbours()	Test of deze functie pointers maakt naar de correcte VSS’en/TTD’s.

T5	R10 (M)	main::saveLog()	Test of de applicatie een log met daarin de uitgevoerde acties naar een tekstbestand kan schrijven.
T6	R11 (C)	lpssim_io::load()	Test of het lpssim_io object een trace bestand kan inladen.
T7	n.v.t.	lpssim_io::transition()	Test of het lpssim_io object de correcte status krijgt na een transitie.
T8	n.v.t.	main::findUpdates()	Test of deze functie het SCENARIO object correct bijwerkt na de vorige transitie.
T9	n.v.t.	window::window()	Test constructie van window object. Hiervoor moet SDL2 geïnstalleerd zijn.
T10	R1 t/m R6 (M)	window::drawScenario()	Test of er naar een scenario naar het scherm getekend kan worden. Of dit scenario correct getekend word (juiste kleuren voor statussen) volgt in de 'correcte grafische weergave' test.

De functies die getest worden zijn uitgekozen omdat dit de functies zijn waarbij verschillende onderdelen met elkaar interacteren.
Bij de functies main::saveLog() en lpssim_io::load() worden geen andere onderdelen betrokken. Omdat de functies wel interactie hebben met bestanden (schrijven en lezen) op de computer zie ik het wel als een vorm van interactie en uitwisseling van data welke getest moet worden. Vooral het testen van de functie main::saveLog() is belangrijk omdat deze onder een requirement met prioriteit Must valt.

7.2 Ontwikkelen automatische testapplicatie

Voor bovenstaande testcases ontwikkel ik een applicatie die de testen automatisch uitvoert en evalueert. ProRail kan dit testprogramma op een systeem draaien als zij willen testen of de applicatie ook op dat systeem zou werken. Mocht in de toekomst de visualisatie applicatie niet werken, dan kunnen zij dit testprogramma ook gebruiken om inzicht te krijgen in welk onderdeel niet werkt.

Het testprogramma maakt gebruik van een bekend scenario. Wanneer het bepaalde functies uitvoert is de resultaat daarvan dus ook bekend. Het programma vergelijkt het resultaat van een functie met het verwachte resultaat. Als dit niet overeenkomt is de test gefaald. Het testprogramma geeft hier een melding van. Ook geeft het aan wat het verwachte en daadwerkelijke resultaat was op het moment van falen.

Het testprogramma is een console applicatie. Ik voeg hier geen grafische weergave aan toe omdat dit veel tijd zou kosten en de resultaten van de testen ook makkelijk door middel van tekst in de console weer te geven zijn.

7.3 Uitvoeren en evalueren

Het uitvoeren van het testprogramma levert de volgende resultaten op:

Testcase #	Requirement #	Invoer	Verwachte uitvoer	Daadwerkelijke uitvoer
T1	n.v.t.	n.v.t. (object construction)	lpssim_io != NULL, Construction geeft geen exception	lpssim_io != NULL
T2	n.v.t.	n.v.t. (object construction)	SCENARIO != NULL	SCENARIO != NULL
T3	n.v.t.	test.mcrl2	TTD[0].status == OCCUPIED TTD[1].status == FREE TTD[2].status == FREE	TTD[0].status = OCCUPIED TTD[1].status =FREE TTD[2].status = FREE
T4	n.v.t.	VSS[0].next_id	VSS[0].next.id == VSS[1].id	VSS[0].next.id = VSS[1].id

T5	R10 (M)	“transition_log.txt”	Trace is geschreven naar bestand ‘transition_log.txt’	Trace is geschreven naar bestand ‘transition_log.txt’
T6	R10 (M)	“trace.trace”	lpssim_io.status == “[1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, single, 0, 0, 0, true, true, [], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)]”	lpssim_io.status = “[1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, single, 0, 0, 0, true, true, [], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)]”
T7	n.v.t.	4	lpssim_io.status == “*TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed, true, single, running, running)]]*”	lpssim_io.status bevat “TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed, true, single, running, running)]]”
T8	n.v.t.	lpssim.status = “[2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 7, 0, true, true, single, [0, 0], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed, true, single, running, running)]]”	TRAIN[0],eoa_id == 7	TRAIN[0],eoa_id == 7
T9	n.v.t.	n.v.t. (object construction)	window != NULL	window != NULL
T10	n.v.t.	SCENARIO	Functie geeft geen exception	Functie geeft geen exception

Alle testen zijn uitgevoerd en geslaagd. Daarmee wordt voldaan aan de criteria van deze test.

7.4 Controle visualisatie

Van sommige onderdelen wordt de status aangegeven door middel van een kleur. In de specificatiedocumenten worden illustraties gebruikt waarbij ook bepaalde kleuren gebruikt worden om bepaalde statussen aan te geven. Een van de eisen is dat de visualisatiesoftware zo veel mogelijk de gebruikte kleuren en vormen uit deze illustraties overneemt.
Een stakeholder merkte op dat het handig zou zijn als zij een bevestiging zouden krijgen dat de visualisatie ook daadwerkelijk de juiste kleuren gebruikt. Daarom heb ik er voor gekozen in de testfase een controle uit te voeren door middel van een handmatige test. Ook gaan de meeste requirements met prioriteit Must, over de manier waarop een onderdeel wordt gevisualiseerd. Het is dus handig dit te controleren.
Dit had ik oorspronkelijk niet gepland uit te voeren, maar heb ik vanwege de suggestie van de stakeholder uitgevoerd. Voor deze test zijn aan hetzelfde document als de vorige test een nieuw hoofdstuk toegevoegd met opnieuw een omschrijving van de scope, criteria en strategie. Dit is later in het project, dicht tegen de deadline aan toegevoegd. Ook dit onderdeel is opgenomen in bijlage E ‘Testplan en rapport’.

De weergave van de volgende onderdelen zijn opgenomen in de scope van de test:

- TTD
    - status
    - shadow_train_A timer
    - shadow_train_B timer

    - ghost_train_protocol timer
- VSS
    - status
    - disconnect timer
    - integrity_lost_propegation timer
- Trein
    - integrity
    - movement authority (End of Authority, EoA)
    - rear end

Het criterium is opgesteld als “Alle boven staande onderdelen moeten weergegeven worden zoals in de requirement aangegeven is”.

De volgende strategie is bedacht:
De status van een onderdeel en de weergave daarvan zal handmatig gecontroleerd worden. De visualisatie applicatie zal in debugmodus uitgevoerd worden. Hierdoor kan de output van de mCRL2 simulator bekeken worden waarmee de status van een onderdeel achterhaald kan worden. Vervolgens kan de manier waarop dit onderdeel is weergegeven vergeleken worden met de bijbehorende requirement.

De weergave is gecontroleerd door een nieuw scenario te definiëren waarin de verschillende mogelijke statussen voorkomen en vervolgens de visualisatie software dit te laten tekenen. Zo kan ik controleren of een bepaald onderdeel juist gevisualiseerd is.

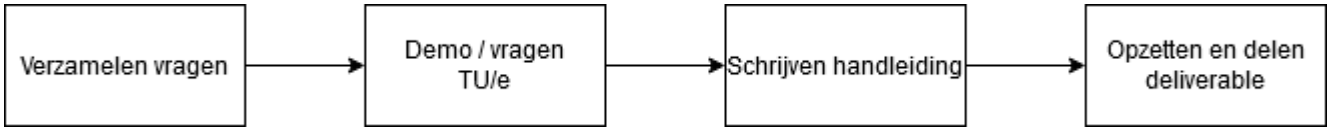
Dit onderdeel is moeilijk automatisch te testen. Omdat de data die naar het scherm gestuurd wordt uiteindelijk het makkelijkst te controleren is door handmatig te kijken naar wat er daadwerkelijk op het scherm staat.

Het resultaat van de test is de volgende rapportage:

Testcase #	Requirement #	onderdeel	status	verwachte weergave	werkelijke weergave
T11	R1 (M)	TTD.status	FREE	groen	groen
T12	R1 (M)		OCCUPIED	geel	geel
T13	R6 (M)	TTD.timer_shadow_train_A	RUNNING	groen	groen
T14			EXPIRED	rood	rood
T15			STOPPED	geel	geel
T16	R6 (M)	TTD.timer_shadow_train_B	RUNNING	groen	groen
T17			EXPIRED	rood	rood
T18			STOPPED	geel	geel
T19	R6 (M)	TTD.timer_ghost_train_protocol	RUNNING	groen	groen
T20			EXPIRED	rood	rood
T21			STOPPED	geel	geel
T22		VSS.status	FREE	groen	groen
T23			OCCUPIED	geel	geel
T24			AMBIGUOUS	oranje	oranje
T25			UNKNOWN	rood	rood
T26	R6 (M)	VSS.timer_disconnect	RUNNING	groen	groen
T27			EXPIRED	rood	rood
T28			STOPPED	geel	geel
T29	R6 (M)	VSS.timer_lost_propegation	RUNNING	groen	groen
T30			EXPIRED	rood	rood
T31			STOPPED	geel	geel
T32	R5 (M)	TRAIN.integrity	CONFIRMED	groen	groen
T33			LOST	oranje	oranje
T34			UNKNOWN	rood	rood

T35	R4 (M)	TRAIN.EoA	7	lijn tot einde EoA VSS (7)	lijn tot einde EoA VSS
T36	R3 (M)	TRAIN.rear_end	0	lijn tot begin rear_end VSS (0)	lijn tot begin rear_end VSS

8 Fase 4 (overdracht)



Figuur 11 Uitgevoerde werkzaamheden fase 4

8.1 Verzamelen vragen

Tegen de tijd dat er een redelijk complete versie van mijn visualisatie ontwikkeld was, heb ik de applicatie gedeeld met de stakeholders. Zo hebben zij al een beetje kunnen experimenteren met mijn applicatie en hebben zij ook een idee kunnen krijgen over hoe de simulator van de TU/e werkt.
Een reden waarom de stakeholders vroegen om een visualisatie is dat zij het gedrag van het HL3 systeem dat de simulator representeert beter willen begrijpen. Zoals ik al eerder omschreven heb is de simulator van de TU/e niet erg gebruiksvriendelijk en moeilijk te begrijpen. Nu zij mijn visualisatie applicatie hebben kunnen gebruiken, wordt de werking van de simulator ook duidelijker voor hen. Dit heeft ook als gevolg dat de stakeholders nu meer vragen hebben over waarom de simulator op bepaalde manieren werkt. Voor sommige stakeholders was het onduidelijk of de antwoorden op hun vragen aan de kant van de simulator of aan de kant van de visualisatie liggen. Daarom wisten zij ook niet bij wie zij voor bepaalde vragen moesten aankloppen.

Toevallig nam rond de zelfde tijd te TU/e contact op met mijn mentor. Zij hadden een nieuwe opdracht van ProRail voor het aanpassen van de simulator geaccepteerd. Ook wisten zij van mijn project en waren benieuwd naar mijn voortgang. Dit leek mij een goede kans om een overleg met de mensen van de TU/e in te plannen, waar ik kon vertellen over mijn visualisatie, en ook de vragen van de stakeholder kon stellen.

Ik heb aan de stakeholders gevraagd de vragen die zij hebben naar mij toe te sturen. Dit hebben zij gedaan via chatberichten in Microsoft Teams. De meeste vragen die ik ontving gingen over de werking van de simulator, maar er waren ook vragen over mijn applicatie. De vragen heb ik verzameld in een kladblok.

8.2 Demo / vragen TU/e

Voor dit overleg waren, naast mijzelf, drie mensen van ProRail en twee van de TU/e uitgenodigd. Er kon geen moment gevonden worden waarop iedereen aanwezig kon zijn. De afwezigen heb ik op een later moment ingelicht over wat er besproken is. Als zij vragen naar mij toe hebben gestuurd heb ik die ook met hen besproken.

Allereest heb ik een demo gegeven van mijn visualisatie applicatie. Ik heb laten zien hoe de applicatie de mCRL2 simulator gebruikt en hoe het op basis daarvan een reconstructie maakt van het scenario en dit visueel weergeeft. Daarna hebben ik en de andere aanwezigen van ProRail enkele vragen kunnen stellen, waarop de heren van de TU/e antwoord hebben gegeven. De antwoorden die ik heb gekregen heb ik genoteerd. Bij het eerst volgende overleg moment met de stakeholders heb ik deze antwoorden met hen gedeeld.

8.3 Schrijven handleiding

Om er voor te zorgen dat de stakeholders ook tijdens mijn afwezigheid correct gebruik kunnen maken van de visualisatie schrijf ik voor hen een handleiding.
In deze handleiding beschrijf ik hoe zij als gebruiker mijn software kunnen gebruiken voor het visualiseren van een simulatie, maar ook hoe zij de broncode kunnen compileren, mochten zij in de toekomst aanpassingen willen aanbrengen. Voor het compileren zijn weer enkele andere onderdelen nodig, zoals GNU Make en de juiste C++

compiler. Van deze benodigdheden geef ik in de handleiding ook een beschrijving.
De handleiding is te vinden als bijlage F ‘Handleiding’.

Daarnaast geef ik ook een korte uitleg hoe zij enkele tools uit de mCRL2 toolkit kunnen gebruiken. In principe is het voor de eindgebruiker niet nodig om kennis te hebben van de tools uit de toolkit omdat de TU/e het model ontwikkeld en mijn software weet hoe het met de bestanden, waarin zulke modellen staan, om moet gaan. Toch leek het mij handig een uitleg te geven van de tools die ik handig vond om te gebruiken tijdens het ontwikkelen. Zodat de stakeholders ook weten hoe zij met mCRL2 simulator om moeten gaan buiten mijn visualisatie applicatie om.

8.4 Opzetten en delen deliverable

Nu rest mij alleen nog ervoor te zorgen dat de eindgebruikers ook daadwerkelijk mijn applicatie kunnen uitvoeren op hun systemen. Op het systeem waarop ik de software ontwikkeld heb staan een hoop omgevingsvariabelen ingesteld en libraries geïnstalleerd. Ik kan er niet vanuit gaan dat elk systeem waarop de stakeholders mijn applicatie willen uitvoeren deze benodigdheden ook correct heeft geïnstalleerd. Daarom heb ik uitgezocht welke Dynamic Linked Libraries (DLL’s) mijn applicatie nodig heeft. Deze DLL’s heb ik in een map verzameld en deze zullen met het delen van mijn applicatie meegeleverd worden.

Een map met daarin de ontwikkelde applicatie, de broncode en de benodigde onderdelen zoals .mcrl2 bestanden of DLL’s vormt de *deliverable* van de applicatie die uiteindelijk met de stakeholder gedeeld wordt. De ontwikkelde tussenproducten worden hierbij ook meegeleverd.

9 Evaluatie / Reflectie

9.1 Productevaluatie

Het doel van de opdracht van ProRail was het ontwikkelen van een visualisatie applicatie voor de ERTMS HL3 mCRL2 simulator. In deze paragraaf zal ik terugblikken op de eisen die aan deze applicatie gesteld zijn en zal ik evalueren in hoeverre deze eisen gehaald zijn.

Hieronder volgt een overzicht van de requirements. Per requirement is aangegeven of deze gehaald is of niet. Een groene markering betekend dat de requirement gehaald is. Een gele markering geeft een requirement aan die niet gehaald is maar dat dit ook niet verplicht is. Een rode markering geeft een niet gehaald maar wel verplichte requirement aan. De geweigerde requirements zijn niet opgenomen in deze evaluatie.

#	Omschrijving	Prioriteit
R1	De visualisatie moet de status van een TTD of VSS (free/occupied/ambiguous/unknown) laten zien aan de hand van vier kleuren (groen/geel/oranje/rood respectievelijk).	M
R2	De visualisatie moet de huidige status van de statemachine van een VSS en de laatste overgang laten zien.	M
R3	De visualisatie moet duidelijk weergeven wat de voorkant van een trein is en wat de achterkant.	M
R4	De visualisatie moet de Movement Authority van een trein laten zien aan de hand van een lijn aan de voorkant van de trein.	M
R5	De visualisatie moet de status van een trein (integrity) laten zien.	M
R6	De visualisatie moet de status van een Timer (stopped/running/expired) laten zien.	M
R7	De visualisatie moet laten zien bij welke VSS een Timer hoort.	S
R8	De visualisatie moet de gevolg van een bepaalde input duidelijk laten zien. Een verandering in de output moet duidelijk herleidbaar zijn naar de input.	S
R9	Het uiterlijk van de visualisatie dient zo veel mogelijk overeen te komen met de afbeeldingen in de HL3 specificatiedocumenten.	S
R10	De software moet alle gebeurtenissen in de simulatie bijhouden in een log.	M
R11	De software moet een trace-bestand kunnen inladen en automatisch kunnen doorlopen met instelbare periode tussen stappen.	C
R12	De software moet een scenario met automatische willekeurige overgangen kunnen doorlopen.	S

R8 is niet gehaald omdat er geen tijd was deze functionaliteit te implementeren. Wel is de mogelijkheid om het uitvoeren van een actie ongedaan te maken. Hiermee kan de gebruiken het effect van een actie inzien.
R11 is deels gehaald. De software kan een trace-bestand inladen maar daar niet automatisch doorheen stappen. Deze functionaliteit bestaat ook niet in de mCRL2 simulator dus dit zou veel tijd gekost hebben om te implementeren.
R12 is niet gehaald omdat er geen tijd was deze functionaliteit te implementeren.

9.2 Reflectie producten

Ik ben tevreden over het product dat ik ontwikkeld heb. Er is aan alle Must-have requirements voldaan. Over de documenten die ik daaromheen ontwikkeld heb durf ik niet veel te zeggen. De stakeholders hebben weinig feedback hierop gegeven dus ik ga ervan uit dat zij tevreden zijn over de documenten die ik hen geleverd heb. Verder ben ik

ook tevreden over de testapplicatie die ik ontwikkeld heb. Deze applicatie is niet alleen handig voor het testen van de werking van de visualisatie, maar ook om te testen of de visualisatie zou werken op een computersysteem.

Ik de toekomst zal ik ervoor moeten zorgen dat er meer bekend is over de wensen op het gebeid van documentatie. Zo kan ik ervoor zorgen dat ik vooraf weet wat ik in een document moet opnemen, en niet achteraf van de feedback afhankelijk ben.

9.3 Procesevaluatie

In dit stuk zal ik het mijn uitvoering van het proces en de verschillende fases evalueren.

Ik geloof dat werken volgens de Waterfall methode de juiste keuze was. Ik vond het prettig om per fase met één taak bezig te zijn. Het is niet altijd gelukt geen stappen terug te zetten. De main::saveLog() functie is een functie die oorspronkelijk niet in het ontwerp opgenomen was. Tijdens de testfase merkte een stakeholder op dat het trace-bestand wat de de simulator kan opslaan niet handig is in het gebruik omdat dit niet leesbaar is zonder bepaalde tools uit de mCRL2 toolkit. Daarom heb ik de main::saveLog() functie toegevoegd. Deze functie slaat een leesbaar log op. Gelukkig was dit geen groot onderdeel en kon het eenvoudig in het ontwerp opgenomen worden en geïmplementeerd worden.

Fase 0 verliep wat rommelig. In deze fase was vooral het opstellen van de requirements belangrijk. Er was maar één keer per week een overlegmoment en daarom zat er veel tijd tussen de momenten waarop ik de requirements kon bespreken. Deze tijd heb ik geprobeerd te benutten door te leren over het ERTMS systeem en de mCRL2 taal en toolkit.
Wanneer ik het plan van aanpak had opgesteld had ik deze gedeeld met de stakeholders zodat zij feedback konden geven. Van hen heb ik weinig feedback gekregen, wat ik niet prettig vond. Zij vertelden dat zij druk waren, wat ik zeker kan begrijpen, maar het is wel onprettig als ik een week op feedback moet wachten en vervolgens weinig feedback krijg.

De concept fase verliep soepeler. Hierin heb ik als eerste de software ontworpen volgens het DDD principe. Ik vond het prettig dit principe te gebruiken omdat dit goed binnen de situatie van het project past. Het ontwerpen van een voorbeeld van de GUI verliep ook goed. Ik kon veel voorbeelden uit het principedocument gebruiken voor het ontwerpen van de GUI. Dit maakte deze taak vrij makkelijk. Het overleggen van het ontwerp met de stakeholders was wel wat moeilijker omdat er maar één keer per week overlegt kon worden.

De ontwikkel fase verliep moeilijker. In deze fase liep ik tegen meerdere problemen aan, zoals het interacteren met de simulator als subproces, het aanpassen van de mCRL2 broncode, of het bepalen van het beginpunt van de simulator. Tijdens deze fase twijfelde ik aan de haalbaarheid van het project. Dit heb ik ook met de stakeholders besproken en hier hebben wij goed over gediscussieerd. We hebben een revisie van de werkzaamheden besproken om een zo compleet mogelijk product neer te zetten. Uiteindelijk kon in de ontwikkeling toch zoals gepland doorzetten nadat ik na een keertje flink doorwerken de problemen had opgelost.

De testfase was een bijzondere fase. Het was al een tijd geleden dat ik over testen geleerd heb in de opleiding en veel stof was weggezakt. Ik heb met behulp van de slides van de colleges uiteindelijk een testplan geschreven.
Na overleg met de stakeholders is hier het onderdeel ‘Test correcte grafische weergave’ aan toegevoegd. Dit was in principe geen probleem, omdat ik nog in de testfase zat. Maar de deadline van het project naderde en de overdracht moest ook nog uitgevoerd worden.

De overdracht fase verliep soepel, vooral omdat in deze fase weinig producten ontwikkeld moesten worden. Ik heb een goed gesprek gehad met de TU/e over de werking van mijn applicatie en hun simulator. In dit gesprek heb ik vragen van mijzelf en van de stakeholders kunnen beantwoorden. In deze fase heb ik ook een handleiding geschreven waarin staat hoe een gebruiker mijn software moet gebruiken. Daarnaast heb ik ook omschreven hoe andere handige tools binnen de context van dit project gebruikt kunnen worden. Hiermee kunnen de stakeholders niet alleen leren over het gebruik van mijn software, maar ook over bepaalde tools die zij in de wellicht in de toekomst moeten gebruiken wanneer de TU/e de simulator of het model aanpast.

9.4 Reflectie proces

Ik ben redelijk teleurgesteld in de uitvoering van de opdracht. Een afstudeeropdracht wordt vrijwel altijd uitgevoerd op locatie bij een bedrijf. Dit stelt een student ook in staat meer van de werkzaamheden van een bedrijf mee te krijgen en te leren over het bedrijf. Ik heb de gehele opdracht thuis uitgevoerd waardoor ik weinig over ProRail geleerd heb. Dit vind ik jammer omdat ik geloof dat er een hoop interessants te leren is over ProRail en de werkzaamheden die zij uitvoeren.

Daaromheen ging het ontwikkelen van de applicatie en het uitvoeren van de opdracht verder prima. Ook heb ik mij zo veel mogelijk aan de Waterfall methode gehouden. Overleg hebben via videobellen vind ik niet ideaal en was ook niet altijd even makkelijk. Het maakt het opstellen van de requirements moeilijker omdat gesprekken minder soepel verlopen.

Ik geloof dat ik een hoop geleerd heb over dingen die relevant waren voor mijn opdracht, zoals leren over een formele specificatietaal, het abstraheren en simuleren van modellen, het ontwikkelen van een grafische applicatie en het programmeren van C++ (voor Windows).

Om samen te vatten: De uitvoering van het project verliep lang niet altijd soepel, maar nu het project af is geloof ik wel dat ik het project correct heb uitgevoerd. De applicatie voldoet aan de wensen van de stakeholders.

9.5 Evaluatie beroepstaken

A1 Analyseren Probleemdomein (verplicht)

Situatie

Opstellen van een plan van aanpak.

Taak

Ik moet een plan van aanpak opstellen waarin, onder andere, een probleemstelling en een overzicht van risico's voor het project moet worden opgenomen.

Activiteit

Ik heb tijdens meerdere overlegmomenten met de stakeholders gesproken over de probleem- en doelstelling van het project en wat de gewenste resultaten zijn. Aan de hand van deze gesprekken heb ik een plan van aanpak opgesteld. Het viel mij op dat sommige eisen afhankelijk zijn van factoren die tijdens de periode van dit project beperkt beschikbaar zijn. Bijvoorbeeld: Een deployment omgeving, of personen die bepaalde kennis met mij kunnen delen. Dit vormt een risico voor het behalen van de doelstelling van dit project.

Resultaat

Het resultaat van deze werkzaamheden is het plan van aanpak. Dit plan is gedeeld met de stakeholders zodat zij het document kunnen inzien en eventueel feedback kunnen geven. Uiteindelijke heeft mijn begeleider feedback gegeven wat ik verwerkt heb. Daarnaast heb ik vragen gesteld over enkele feedbackpunten maar daar heb ik geen antwoord op gekregen.

Reflectie

Ik ben zelf tevreden over het plan van aanpak. De punten die volgens mij een risico vormen voor het uitvoeren van het project staan er in. Ook zijn er voor mij voldoende requirements opgesteld om de applicatie mee te ontwikkelen. Ik vind het jammer dat er weinig contactmomenten zijn waarop ik kan vragen naar feedback en dat het geven van feedback vaak vergeten lijkt op de momenten waarop het besproken zou worden. Tegelijkertijd heb ik er begrip voor gezien de drukte van mijn begeleider en de stakeholders en de beperkte communicatiemogelijkheden in verband met de COVID-19 pandemie. In de toekomst zou ik proberen de feedback actief te achterhalen wanneer de communicatiemogelijkheden beperkt zijn.

A3 Vergaren en analyseren requirements (e-CF niveau 3)

Situatie

Verzamelen requirements van de stakeholders en verwerken in centraal document.

Taak

Voor het correct ontwikkelen van de visualisatiesoftware is het nodig alle eisen die de stakeholders hebben te verzamelen en te beoordelen. Het eindresultaat is een document waarin de geaccepteerde en afgewezen requirements terug te vinden zijn.

Activiteit

Tijdens de overlegmomenten met de stakeholders heb ik gevraagd hoe zij bepaalde aspecten van de visualisatie uitgewerkt willen zien. Hiervoor heb ik eerst een brainstorm sessie gehouden met de stakeholders. Hieruit zijn de grove ideeën van de applicatie gekomen. Vervolgens ben ik, ook met behulp van de informatie uit de eerdere uitvraag aan de TU/e, dieper op de ideeën van de verschillende stakeholders ingegaan. Zodoende is er tijdens meerdere overlegmomenten in overleg met de stakeholders een lijst van softwarerequirements opgesteld. Deze requirements zijn vervolgens in overleg geprioriteerd volgens MoSCoW.

Resultaat

Het resultaat is het genoemde document. Dit document is gedeeld met de stakeholders zodat zij het kunnen beoordelen, vragen kunnen formuleren en commentaar kunnen geven. Het document is opgesteld naar mijn eigen ideeën, de stakeholders hebben daarna geen feedback of commentaar gegeven.

Reflectie

Ik ben zelf van mening dat het opgestelde document compleet is. Ik vind het jammer dat de stakeholders weinig feedback gegeven hebben wat ik zou kunnen gebruiken om de requirements eventueel te verbeteren. Dit is besproken met de stakeholders en zij vertelden dat zij weinig tijd hebben gehad het document te lezen en te beoordelen.

C6 Ontwerpen Software

Situatie

Maken concept klassendiagram visualisatiesoftware.

Taak

Om voor mijzelf een overzicht te krijgen van de benodigde onderdelen en om de stakeholders inzicht in de ontwikkeling te bieden is er een klassendiagram ontworpen. Dit ontwerp is een concept omdat er nog niet volledig duidelijk is welke onderdelen er nodig zijn en omdat er tijdens het daadwerkelijk ontwikkelen van de software andere uitwerkingen handiger kunnen blijken.

Activiteit

Voor het ontwerpen van de software heb ik gekozen om Domain Driven Design te volgen. Dit principe komt mooi overeen met welke data er nodig is voor de visualisatie om de simulatie correct weer te geven.
De benodigde objecten zijn in de Domein laag geplaatst met daaromheen de beoogde benodigde onderdelen om de data uit de simulator te verwerken naar de attributen van domeinklassen, en om de data in de domeinklassen te ‘vertalen’ naar een visuele weergaven.

Resultaat

De werkzaamheden hebben geleid tot het klassediagram dat gedeeld is met de stakeholders.

Reflectie

Ik ben tevreden over het ontwerp. Het domein is netjes vastgelegd en daaromheen zijn verschillende onderdelen en lagen toegevoegd om een werkend geheel te maken. De verschillende onderdelen hebben elk een eigen verantwoordelijkheid en hebben geen overlap op het gebied van functionaliteit. De separatie

tussen de onderdelen is ook vrij groot. Je zou bijvoorbeeld de Presentatie laag kunnen vervangen met een andere manier om met de gebruiker te interacteren zonder de overige onderdelen aan te moeten passen.

C11 Ontwerpen van Human Computer Interaction (e-CF niveau 3)

Situatie

Voor de applicatie moet er een grafisch onderdeel ontwikkeld worden die een gebruik kan gebruiken om het scenario van een simulatie in te zien en om de simulator te besturen

Taak

Het grafische onderdeel moet voor de gebruiker goed te begrijpen zijn. Ze moeten onderdelen en de status daarvan eenvoudig kunnen herkennen. Ook moeten zij makkelijk kunnen zien wat zij kunnen doen om de simulator te beïnvloeden.

Activiteit

Ik ben met de stakeholders in gesprek gegaan over het grafische onderdeel. Ik heb gevraagd hoe ik de visualisatie voor hen makkelijk te begrijpen kan maken. De stakeholders verwezen naar een document waarin het ERTMS HL3 systeem gespecificeerd staat. In dit document wordt er ook gebruik gemaakt van illustraties om bepaalde situaties aan te geven. Omdat zij dit document goed kennen en de illustraties daaruit begrijpen zou het handig zijn als de stijl van de visualisatie de stijl van de illustraties overneemt.

Resultaat

Het resultaat is een visualisatie die de illustraties uit het specificatiedocument nauw heeft overgenomen. Dit zorgt voor een grafische weergave die de bedoelde eindgebruikers goed kunnen begrijpen en makkelijk in gebruik kunnen nemen. Ook is er in de software een legenda opgenomen waarin is aangegeven wat bepaalde symbolen en kleuren van de visualisatie betekenen. De stakeholders hebben al proefversies van de visualisatiesoftware kunnen uitproberen en de feedback die hieruit is gekomen ging maar weinig over de manier waarop onderdelen gevisualiseerd zijn.

Reflectie

Zelf ben ik tevreden over het ontwerp van de grafische interface. De manier waarop het scenario uit de simulator wordt gevisualiseerd is duidelijk en komt goed overeen met de illustraties uit het specificatiedocument. Daaromheen zijn de onderdelen van de grafische interface ook duidelijk leesbaar en zichtbaar wat het gebruik van de software makkelijker maakt. De software lijkt soms traag. Dit komt vooral omdat de onderliggende simulatie vrij traag is.

D15 Testen (e-CF niveau 3)

Situatie

De applicatie moet getest worden zodat de stakeholders bewijs hebben dat de applicatie correct werkt.

Taak

De applicatie bestaat uit verschillende onderdelen (lagen). Er moet getest worden of deze onderdelen correct met elkaar samenwerken.

Activiteit

Voor het testen van de applicatie heb ik een Black-Box Bottom-Up Component integratietest uitgevoerd. Het gebruiken van deze strategie heb ik bedacht omdat dit goed de interactie tussen en correct functioneren van de verschillende onderdelen test. Omdat de stakeholders weinig eisen hebben gesteld aan de interne werking van de applicatie waren er maar weinig requirements om in deze test mee te nemen. Ik heb zelf veel testcases bedacht om de applicatie te testen.

Resultaat

Het resultaat hiervan is de bijbehorende testrapportage. Alle testcases zijn uitgevoerd en geslaagd en daarmee voldoet de test aan de gestelde criteria.

Reflectie

Ik geloof dat ik dat de test bewijst dat mijn applicatie goed functioneert. Lang niet alle functies van de applicatie zijn getest maar dat is ook niet de bedoeling van een componenten integratie test. Ook heb ik veel van de testcases zelf bedacht omdat er geen requirements waren om tegen te testen (veel requirements heb ik later handmatig getest in

de ‘controle visualisatie’ test). In de toekomst zal ik er voor zorgen dat er meer meetbare requirements duidelijk zijn die getest kunnen worden, mits ik weer werk volgens een lineaire methode, anders zal testen heel anders benaderd worden dan ik in dit project gedaan heb.

Gc Kritisch, onderzoekend en methodisch werken (verplicht, e-CF niveau 3)

Situatie

Mijn afstudeeropdracht komt in aanraking met verschillende domeinen waar ik weinig kennis van heb. Bijvoorbeeld het ERTMS systeem, het specificeren en simuleren van systemen met behulp van een specificatietaal.

Taak

Om de opdracht goed uit te kunnen voeren moest ik kennis opdoen van de hierboven genoemde onderwerpen.

Activiteit

Ik heb over het ERTMS systeem geleerd door de officiële specificatiedocumenten te bestuderen . Ook heb ik materiaal gebruikt dat ProRail gebruikt bij het opleiden van personeel dat met het ERTMS systeem gaat werken. Voor het leren over de mCRL2 taal heb ik tutorials gevolgd die de ontwikkelaars van de taal op hun website beschikbaar maken. Op deze website is ook documentatie over de taal te vinden. Ook heeft de TU/e een paper met mij gedeeld waarin beschreven staat hoe het ERTMS HL3 systeem in de mCRL2 taal is vastgelegd. Daarnaast heb ik ook met twee ontwikkelaars van de mCRL2 taal van de TU/e gesproken om over mCRL2 te leren.

Resultaat

Ik heb redelijk wat tijd besteed aan het bestuderen van de hierboven genoemde onderwerpen.

Reflectie

Ik heb veel geleerd over de verschillende onderwerpen. Vooral de werking het ERTMS systeem en de verschillende Levels zijn nu veel duidelijker. Bij het bestuderen van de mCRL2 taal heb ik lang niet alles kunnen begrijpen, maar ik geloof wel dat ik een hoop geleerd heb over het specificeren lineariseren van modellen. Ik geloof dat ik de juiste bronnen heb gebruikt bij het bestuderen van de onderwerpen.

Gf Leren leren (verplicht, e-CF niveau 3)

Situatie

Ik had mijzelf het doel gesteld een uitdagende opdracht te kiezen voor mijn afstuderen. Hier koos ik voor dat ik toch nog iets nieuws zou leren, ondanks dat ik dit laatste half jaar van school niet op school ben om nieuwe stof te leren. De vacature van ProRail sprak mij aan omdat daarin omschreven was dat ik met het ERTMS systeem zou kunnen werken.

Taak

Zoals bekend speelt ERTMS een grote rol binnen deze opdracht. Toen er voor mij meer duidelijk werd over de opdracht heb ik mijzelf het doel gesteld gebruik te maken van C++ en SDL2. De C++ programmeertaal is tijdens de opleiding vrij weinig behandeld. Ook is het ontwikkelen van een 2D grafische applicatie maar weinig behandeld. Daarnaast leken mij de lessen over DDD in de opleiding ook erg nuttig. Daarom wou ik ook graag dit principe toepassen in de praktijk.

Activiteit

Tijdens de opdracht heb ik een hoop geleerd over het ERTMS systeem, wat ik erg leuk vond. Daarnaast heb ik natuurlijk ook de applicatie ontwikkeld met C++/SDL2 waarvoor ik een hoop moest leren over deze taal en library.

Resultaat

Ik heb tijdens het ontwerpen en ontwikkelen gebruik gemaakt van DDD, C++ en SDL2.

Reflectie

Ik geloof dat het product een goed werkende applicatie is. De applicatie functioneert goed. Ook ervaar ik het makkelijk de functionaliteit van de applicatie uit te breiden omdat de structuur goed in elkaar zit. Daarnaast zijn de stakeholders ook tevreden over de applicatie. Dit laat zien dat ik de hierboven genoemde onderwerpen goed heb kunnen toepassen bij het ontwikkelen van de applicatie.

Bronnen

mCRL2 user documentation — mCRL2 202006.0 documentation. (2020). mCRL2.org. https://mcrl2.org/web/user_manual/user.html

Bartholomeus, M., Luttik, B., & Willemse, T. (2018). Modelling and analysing ERTMS hybrid level 3 with the mCRL2 toolset. In F. Howar, & J. Barnat (editors), Formal Methods for Industrial Critical Systems - 23rd International Conference, FMICS 2018, Proceedings (blz. 98-114). (Lecture Notes in Computer Science; Vol. 11119). Springer. https://doi.org/10.1007/978-3-030-00244-2_7

Swart, N. (2017). Handboek Requirements (2de editie, p.227). Reed Business Education.

EEIG ERTMS Users Group. (2017, juli). Principles Hybrid ERTMS/ETCS Level 3 (Nr. 16E0421A). http://www.ertms.be/sites/default/files/2018-03/16E0421A_HL3.pdf

Afkortingen en Termen

ERTMS – European Rail Traffic Management System:
Internationale standaard voor treinbeveiliging.

HL3 – Hybrid Level 3
Standaard binnen ERTMS

(Trein)beveiligingssysteem
Systeem dat helpt machinisten te voorkomen sneller of verder te rijden dan het spoorwegsein toestaat.

TU/e – Technische Universiteit Eindhoven

TTD – Trackside Train Detection:
Een sectie spoor waarbinnen treinen gedetecteerd kunnen worden.

VSS – Virtual SubSection:
Virtueel sub-sectie van een TTD.

Timer:
Onderdeel binnen ERTMS systemen met stopwatch functie.

Statemachine:
Abstract of wiskundig model van een systeem wat bestaat uit de mogelijkse statussen van het systeem en overgangen tussen de statussen.

MA - Movement Authority – EoA – End of Authority:
Afstand waarvoor een trein toestemming heeft om af te leggen.

mCRL2:
Specificatietaal met bijbehorende toolkit ontwikkeld aan de TU/e. De taal kan gebruikt worden voor het modeleren van systemen en protocollen.

LPS – Linear Process Specification:
Digitale enkelvoudige definitie van een proces.

API – Application Programming Interface:
Verzameling definities waarmee een computerprogramma aangeeft hoe het kan communiceren.

Wrapper:
Vertaallaag voor een API.

DDD – Domain Driven Design:
Principe voor het ontwerpen van software.

GUI - Grafische User-Interface - Front-end:
Grafisch onderdeel van software waarmee de gebruiker interacteert.

Back-end:
Verstopt onderdeel van software. Bepaalt interne werking

Subproces:
Een programma dat binnen een ander programma uitgevoerd wordt.

Pipe:
Communicatiemiddel tussen twee draaiende computerprogramma's.

Library:
Een verzameling code en functies die door een programma geïmplementeerd kunnen worden.

Intergrity:
Geeft aan of een trein wel of niet kan bevestigen dat de trein heel is. Dat wil zeggen dat alle wagons van een trein nog aan elkaar vast zitten.

Trace:
Bestandstype uit de mCRL2 toolkit. Bevat een log van alle uitgevoerde acties en transities.

Bijlagen

A Requirements
B Deploymentdiagram
C Klassediagram
D Voorbeeld GUI
E Testplan en rapport
F Handleiding
G Revisie werkzaamheden

Requirements

Dennis van Gilst — dennisrene.vangilst@prorail.nl — 16056302@student.hhs.nl

Introductie

Dit document is opgesteld om een overzicht te bieden van de eisen die aan de visualisatie voor de mCRL2 ERTMS HL3 simulatie gesteld worden.

Uitvraag

Sander Dragt heeft in een uitvraag gestuurd naar de TU/e waarin hij een uitbreiding van de mCRL2 uitwerking beschrijft. Hierin vraagt hij ook naar de mogelijkheid om enkele punten te kunnen visualiseren. Deze punten bieden de basis van welke aspecten er gevisualiseerd moeten worden en zijn als volgt:

- Status van TTD en VSS visualiseren
- Visualisatie van treinen (kan ook op basis status TTD/VSS)
- Visualisatie MA (status afgegeven per VSS)
- Timers visualiseren (wel/niet actief zijn van timers)
- Statussen visualiseren (per trein integriteit, verbinding)
- Status van statemachine visualiseren

ERTMS OVS team

De leden van het ERTMS OVS team zijn de stakeholders van dit project. Voor hen wordt de visualisatiesoftware ontwikkeld.
Door hen zijn de volgende requirements genoemd:

TTD/VSS:

- De visualisatie moet de status van een TTD of VSS (free/occupied/ambiguous/unknown) laten zien aan de hand van de vier kleuren (groen/geel/oranje/rood respectievelijk).
- De visualisatie moet de huidige status van de statemachine van een VSS en de laatste overgang laten zien.

Trein:

- De visualisatie moet duidelijk weergeven wat de voorkant van een trein is en wat de achterkant.
- De visualisatie moet de Movement Authority van een trein laten zien aan de hand van een afbuigende lijn aan de voorkant van de trein.
- De visualisatie moet de status van een trein (connection/integrity) laten zien

Timer:

- De visualisatie moet de status van een Timer (running/expired) laten zien.
- De visualisatie moet laten zijn bij welke VSS een Timer hoort.
- De software moet de gebruiker de mogelijkheid geven de waardes van timers te veranderen (bijv. d.m.v. invulvelden).
- De visualisatie moet ook niet-hl3 timers (zoals T_NVCONTACT) kunnen visualiseren.

Overig:

- De visualisatie moet de gevolg van een bepaalde input duidelijk laten zien. Een verandering in de output moet duidelijk herleidbaar zijn naar de input.
- Het uiterlijk van de visualisatie dient zo veel mogelijk overeen te komen met de afbeeldingen in de HL3 specificatiedocumenten.
- De software moet alle gebeurtenissen in de simulatie bijhouden in een log.
- De software moet een trace-bestand kunnen inladen en automatisch kunnen doorlopen met instelbare periode tussen stappen.
- De software moet een een scenario met willekeurige input kunnen doorlopen.
- De simulatie moet de rijweginstelling automatisch kunnen laten verlopen met de mogelijkheid voor de gebruiker om in te grijpen.

Eindresultaat

Het volgende overzicht is opgesteld als eisen aan de visualisatiesoftware:

TTD/VSS:

- De visualisatie moet de status van een TTD of VSS (free/occupied/ambiguous/unknown) laten zien aan de hand van de vier kleuren (groen/geel/oranje/rood respectievelijk). [MUST]
- De visualisatie moet de huidige status van de statemachine van een VSS en de laatste overgang laten zien. [MUST]

Trein:

- De visualisatie moet duidelijk weergeven wat de voorkant van een trein is en wat de achterkant. [MUST]
- De visualisatie moet de Movement Authority van een trein laten zien aan de hand van een lijn aan de voorkant van de trein. [MUST]
- De visualisatie moet de status van een trein (connection/integrety) laten zien [MUST]

Timer:

- De visualisatie moet de status van een Timer (running/expired) laten zien. [MUST]
- De visualisatie moet laten zijn bij welke VSS een Timer hoort. [SHOULD]

Overig:

- De visualisatie moet de gevolg van een bepaalde input duidelijk laten zien. Een verandering in de output moet duidelijk herleidbaar zijn naar de input. [SHOULD]
- Het uiterlijk van de visualisatie dient zo veel mogelijk overeen te komen met de afbeeldingen in de HL3 specificatiedocumenten. [SHOULD]
- De software moet alle gebeurtenissen in de simulatie bijhouden in een log. [MUST]
- De software moet een trace-bestand kunnen inladen en automatisch kunnen doorlopen met instelbare periode tussen stappen. [COULD]
- De software moet een een scenario met willekeurige input kunnen doorlopen [SHOULD]

Geweigerde requirements

De software moet de gebruiker de mogelijkheid geven de waardes van timers te veranderen
Deze requirement is niet mogelijk te implementeren. De uitwerking van de HL3 specificatie implementeert enkel de statussen (running/expired) van timers. Er zijn geen waardes voor de gebruiker om in te stellen. De visualisatie gebruikt de bestaande mCRL2 HL3 simulatie.

De specificatie is uitgewerkt in een .mcrl2 bestand. Om dit te simuleren met dit eerst naar een .lps (linear process specification) bestand gecompileerd worden. Het compileren rekent alle mogelijke statusveranderingen van de objecten in de simulatie door. De simulator gebruikt het .lps bestand om de statussen van alle objecten en mogelijke acties te simuleren.

De visualisatie kan niet gebruikt worden om de specificatie aan te passen aangezien de visualisatie op de lps simulatie draait. Een gebruiker kan de bestaande mCRL2 toolkit gebruiken de specificatie in het .mcrl2 bestand naar zijn wensen aan te passen.

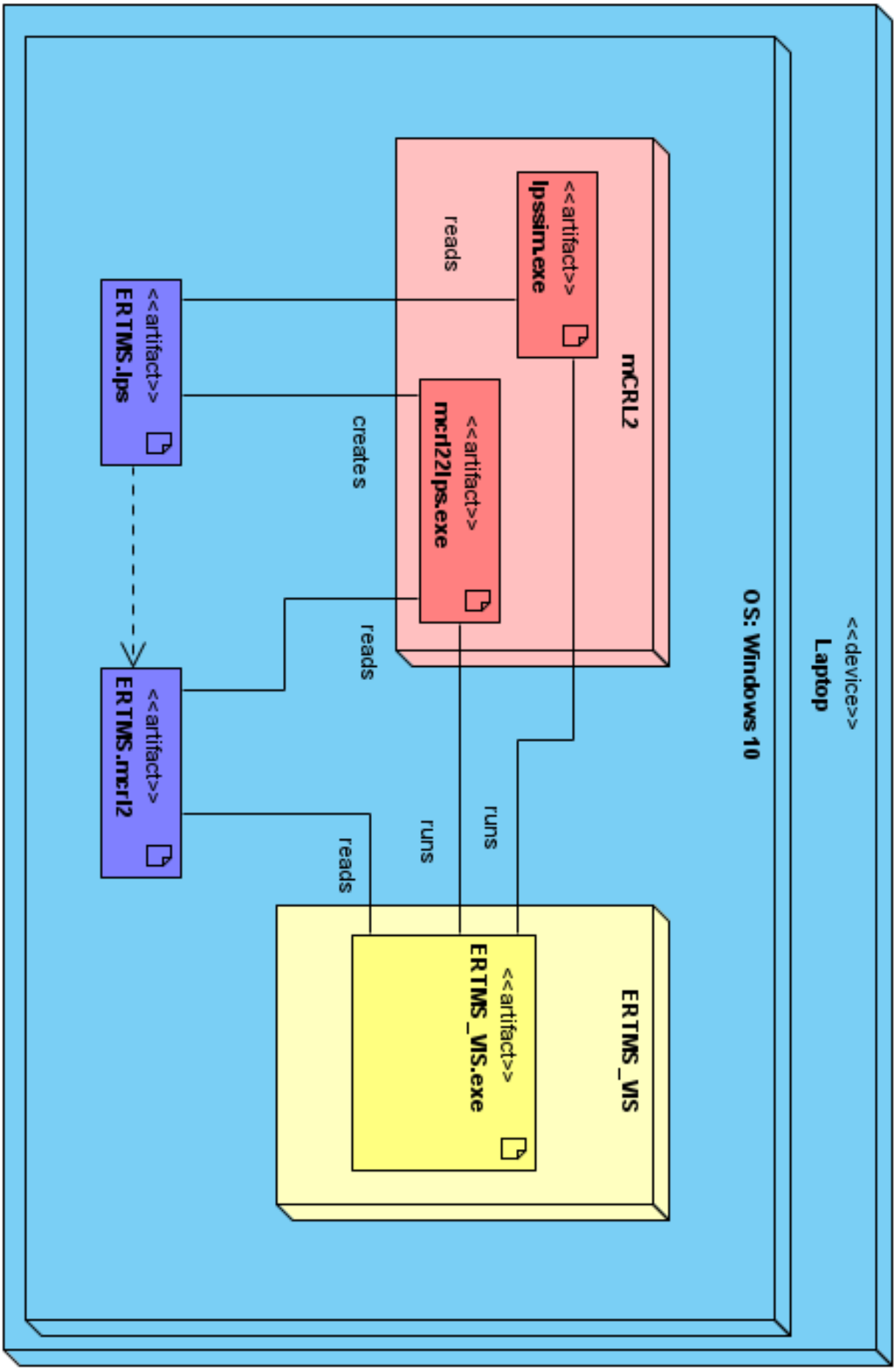
De visualisatie moet ook niet-hl3 timers (zoals T_NVCONTACT) kunnen visualiseren.
De uitwerking van de specificatie heeft geen niet-HL3 timers geïmplementeerd.

De simulatie moet de rijweginstelling automatisch kunnen laten verlopen met de mogelijkheid voor de gebruiker om in te grijpen.
Deze functionaliteit is niet opgenomen in de bestaande uitwerking van de HL3 specificatie.
De bestaande simulator (lpsxsim) kan wel tijdens het simuleren willkeurige overgangen kiezen.

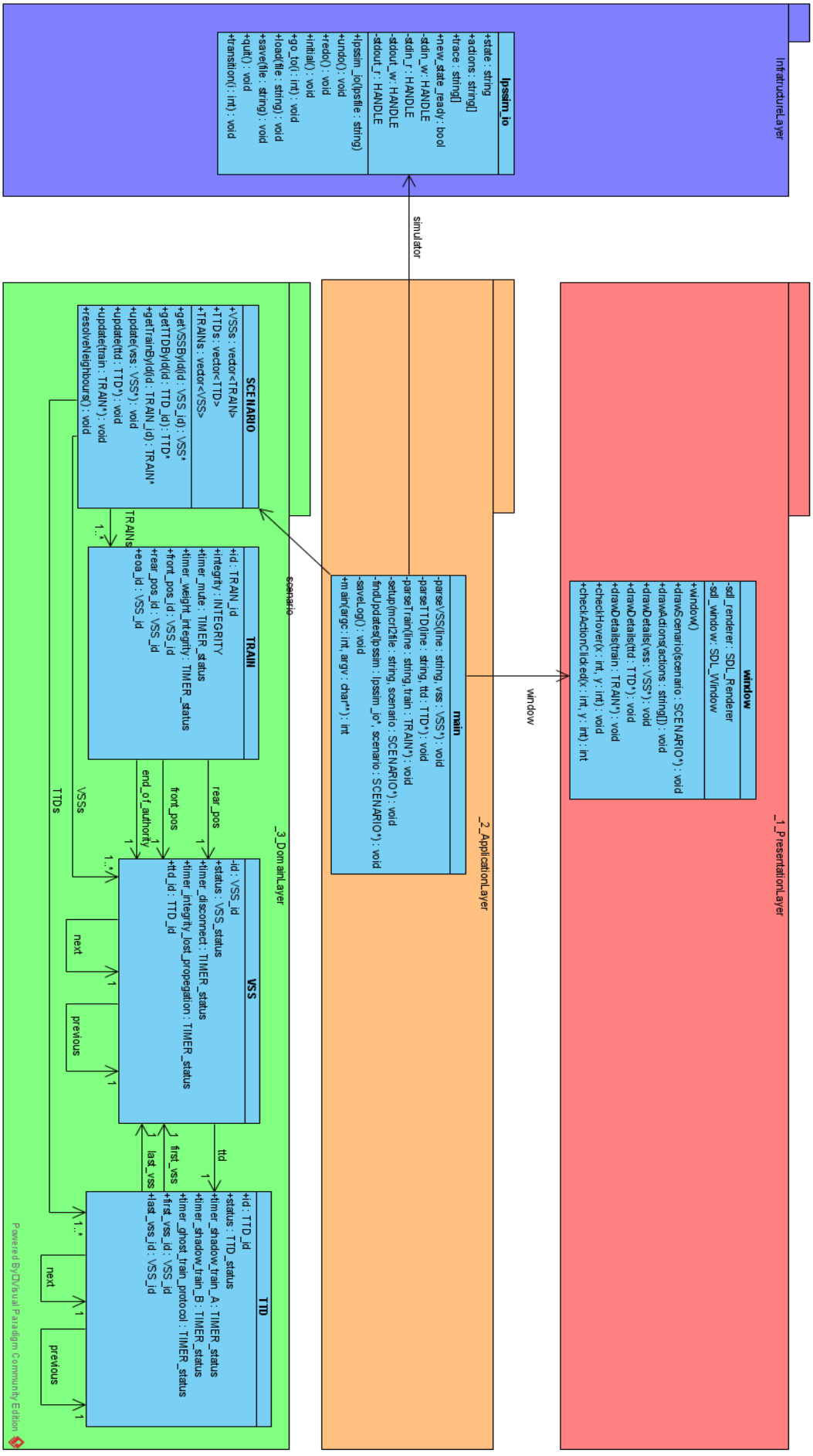
#	Omschrijving	Prioriteit
R1	De visualisatie moet de status van een TTD of VSS (free/occupied/ambiguous/unknown) laten zien aan de hand van vier kleuren (groen/geel/oranje/rood respectievelijk).	M
R2	De visualisatie moet de huidige status van de statemachine van een VSS en de laatste overgang laten zien.	M
R3	De visualisatie moet duidelijk weergeven wat de voorkant van een trein is en wat de achterkant.	M
R4	De visualisatie moet de Movement Authority van een trein laten zien aan de hand van een lijn aan de voorkant van de trein.	M
R5	De visualisatie moet de status van een trein (connection/integrity) laten zien.	M
R6	De visualisatie moet de status van een Timer (running/expired) laten zien.	M
R7	De visualisatie moet laten zien bij welke VSS een Timer hoort.	S
R8	De visualisatie moet de gevolg van een bepaalde input duidelijk laten zien. Een verandering in de output moet duidelijk herleidbaar zijn naar de input.	S
R9	Het uiterlijk van de visualisatie dient zo veel mogelijk overeen te komen met de afbeeldingen in de HL3 specificatiedocumenten.	S
R10	De software moet alle gebeurtenissen in de simulatie bijhouden in een log.	M
R11	De software moet een trace-bestand kunnen inladen en automatisch kunnen doorlopen met instelbare periode tussen stappen.	C
R12	De software moet een scenario met automatische willekeurige overgangen kunnen doorlopen.	S
R13	De software moet de gebruiker de mogelijkheid geven de waardes van timers te veranderen.	W
R14	De visualisatie moet ook niet-HL3 timers (zoals T_NVCONTACT) kunnen visualiseren.	W

R15	De simulatie moet de rijweginstelling automatisch kunnen laten verlopen met de mogelijkheid voor de gebruiker om in te grijpen.	W

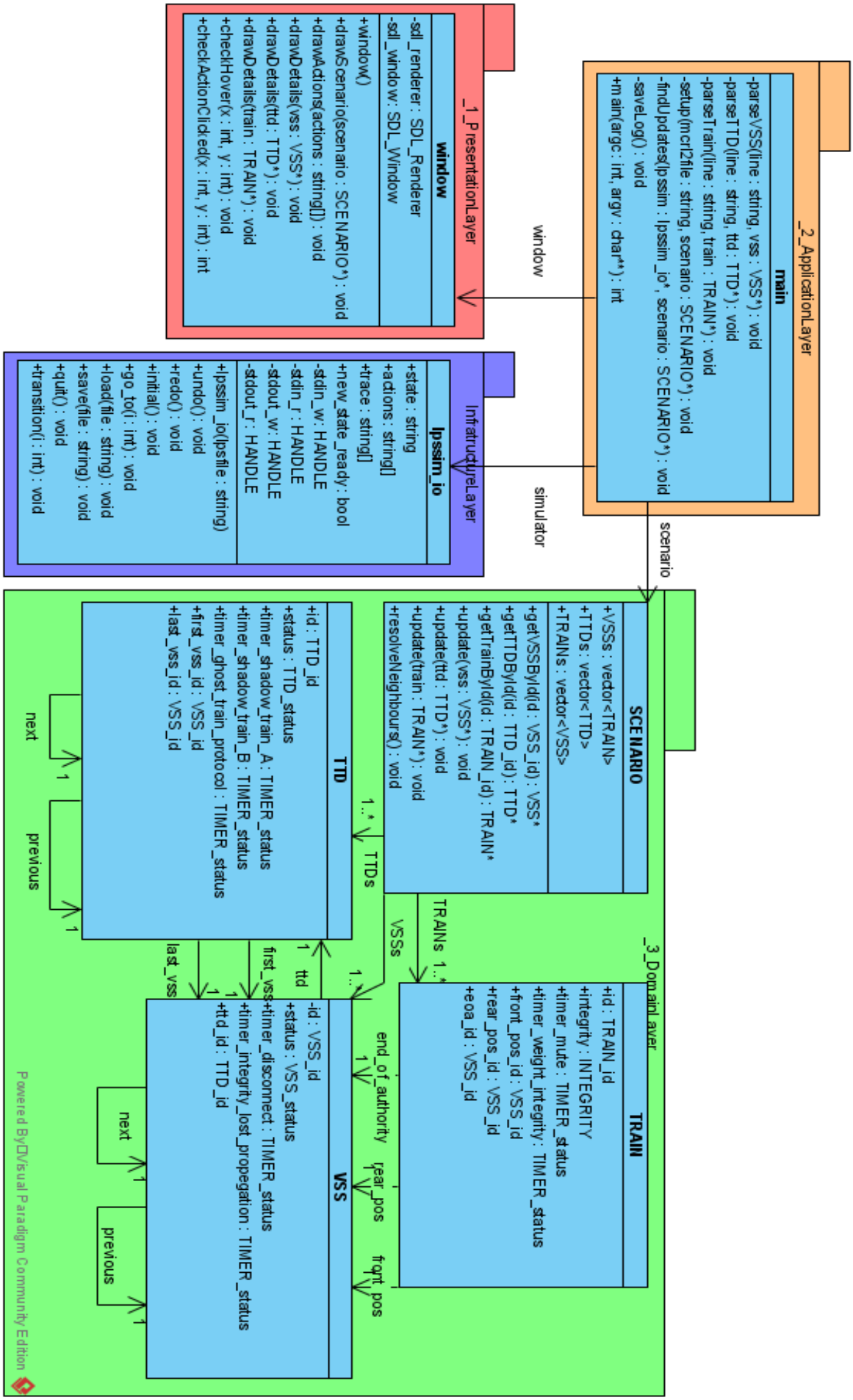
B Deploymentdiagram



C Klassediagram



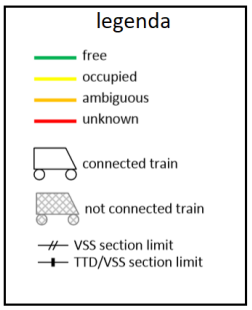
Dit is compacte variant van het ontwerp om de leesbaarheid van de tekst te verbeteren:

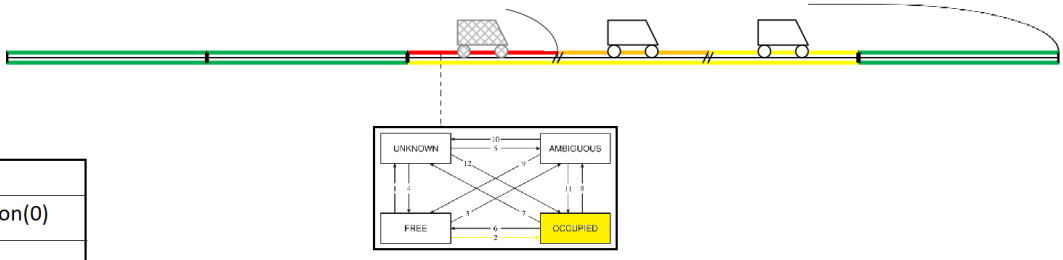


Voorbeeld Grafische Interface

Compleet

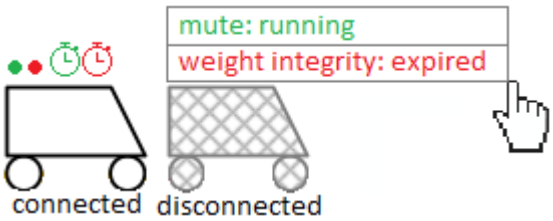
Action
disconnect(0)
break(0)
emit_position(0)
extend_EoA(0, 7)





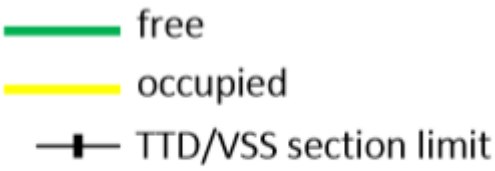
	Trace
1	emit_position(0)
2	break(0)
3	disconnect(0)
4	connect(0)

Trein/Timers



Afbeelding trein volgens specificatiedocument.
Voor timer kan een gekleurde stip of icoon of en venster met timer naam en status.
Te combineren door normaal een stip weer te geven en een venster wanneer de muis over de trein geplaatst is.

TTD



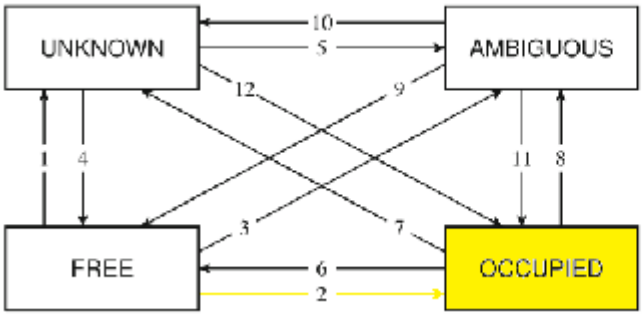
TTD's kunnen op dezelfde manier geïllustreerd worden als in het specificatiedocument. De kleur is afhankelijk van de status van de TTD. De lengte is afhankelijk van hoeveel VSS'en er onder de TTD vallen.

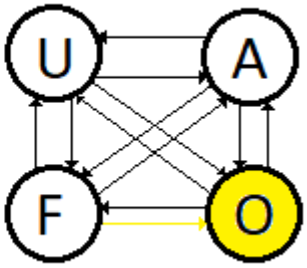
De onderbreking van een TTD is ook volgens de voorbeelden in het specificatiedocument weer te geven. Een grens van een TTD is ook de grens van de laatste VSS van de TTD.

VSS

Ook de VSS'en kunnen weergegeven worden zoals deze in het specificatiedocument weergegeven worden. De lengte van elke VSS is het zelfde. VSS'en worden boven de TTD's geplaatst. De grens van een VSS wordt ook geïllustreerd volgens het voorbeeld uit het specificatiedocument.

State Machine





De statemachine kan volledig weergegeven worden zoals deze in het spec.document staat. Met een kleur kan aangegeven worden wat de huidige status en de laatste overgang is. Dit kan ook versimpeld weergegeven worden.

Movement Authority

De MA van een trein wordt weergegeven met aan afbuigende lijn vanaf de voorkant van een trein tot het einde van de laatste VSS in de MA. (volgens spec.document)

Testplan en rapportage

Inleiding

Dit document geeft een omschrijving van de testen die zijn uitgevoerd voor de software van het ERTMS visualisatie project. Er zijn twee verschillende testen uitgevoerd. Voor beide testen is het plan omschreven. Dit beschrijft welke onderdelen er getest worden en welke strategie er gebruikt wordt. Ook is een rapportage van de testresultaten gegeven.

Automatische componenten integratie test

Plan

Scope

Voor deze test worden de onderdelen van de ontwikkelde software getest. Deze onderdelen zijn:

1. De lpssim_io klasse
2. De SCENARIO klasse
3. De main klasse
4. De window klasse

Criteria

Aan het accepteren van de test zijn een aantal eisen gebonden.
De test moet een run-rate van 100% hebben. Dat wil zeggen dat alle testcases die hieronder beschreven zijn uitgevoerd moeten zijn.
Verder moeten de testcases die bij een requirement met de prioriteit Must (M) horen een pass-rate van 100% hebben. Dit betekend dat alle Must test-cases moeten slagen.

Strategie

Deze test is een componenten integratie test. De ontwikkelde onderdelen en de uitwisseling van data tussen deze onderdelen wordt getest. De test wordt uitgevoerd volgens de *bottom up* en *black box* strategieën. *Bottom up* wil zeggen dat de meest concrete klassen en de klassen met de minste afhankelijkheden eerst getest worden. Er wordt niet gefocust op de implementatie van de klasse, maar op de in- en uitvoer van data en de interactie tussen verschillende onderdelen.

Test cases

Testcase #	Requirement #	Getest component	Omschrijving
T1	n.v.t.	lpssim_io::lpssim_io()	Test constructie van lpssim_io object. Hiervoor moet de klasse lpssim.exe kunnen gebruiken.
T2	n.v.t.	SCENARIO::SCENARIO()	Test constructie van SCENARIO object.
T3	n.v.t.	main::setup()	Test of deze functie correct het initiële scenario opbouwt.
T4	n.v.t.	SCENARIO::resolveNeighbours()	Test of deze functie pointers maakt naar de correcte VSS'en/TTD's.
T5	R10 (M)	main::saveLog()	Test of de applicatie een log met daarin de uitgevoerde acties naar een tekstbestand kan schrijven.
T6	R11 (C)	lpssim_io::load()	Test of het lpssim_io object een trace bestand kan inladen.
T7	n.v.t.	lpssim_io::transition()	Test of het lpssim_io object de correcte status krijgt na een transitie.

T8	n.v.t.	main::findUpdates()	Test of deze functie het SCENARIO object correct bijwerkt na de vorige transitie.
T9	n.v.t.	window::window()	Test constructie van window object. Hiervoor moet SDL2 geïnstalleerd zijn.
T10	R1 t/m R6 (M)	window::drawScenario()	Test of er naar een scenario naar het scherm getekend kan worden. Of dit scenario correct getekend word (juiste kleuren voor statussen) volgt in de 'correcte grafische weergave' test.

Rapportage

Testcase #	Requirement #	Invoer	Verwachte uitvoer	Daadwerkelijke uitvoer
T1	n.v.t.	n.v.t. (object construction)	lpssim_io != NULL, Construction geeft geen exception	lpssim_io != NULL
T2	n.v.t.	n.v.t. (object construction)	SCENARIO != NULL	SCENARIO != NULL
T3	n.v.t.	test.mcrl2	TTD[0].status == OCCUPIED TTD[1].status == FREE TTD[2].status == FREE	TTD[0].status = OCCUPIED TTD[1].status =FREE TTD[2].status = FREE
T4	n.v.t.	VSS[0].next_id	VSS[0].next.id == VSS[1].id	VSS[0].next.id = VSS[1].id
T5	R10 (M)	"transition_log.txt"	Trace is geschreven naar bestand 'transition_log.txt'	Trace is geschreven naar bestand 'transition_log.txt'
T6	R10 (M)	"trace.trace"	lpssim_io.status == "[1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, single, 0, 0, 0, true, true, [], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)]"	lpssim_io.status = "[1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, single, 0, 0, 0, true, true, [], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)]"
T7	n.v.t.	4	lpssim_io.status == "*TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed, true, single, running, running)]]*"	lpssim_io.status bevat "TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed, true, single, running, running)]]"
T8	n.v.t.	lpssim.status = "[2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 7, 0, true, true, single, [0, 0], 1, 0, ptd_info(0, 0, confirmed, true), VSSs_config(29, 1), VSSs_config(29, 1), TTDs_config(29, 1), TRAINs_config(29, 1)[0 -> train_info(0, 0, 7, confirmed,	TRAIN[0],eoa_id == 7	TRAIN[0],eoa_id == 7

		true, single, running, running)]]”		
T9	n.v.t.	n.v.t. (object construction)	window != NULL	window != NULL
T10	n.v.t.	SCENARIO	Functie geeft geen exception	Functie geeft geen exception

Test correcte grafische weergave

Plan

Scope

Voor deze test wordt er gecontroleerd of de applicatie de gesimuleerde onderdelen correct weergeeft en gebruik maakt van de juiste kleuren. Deze test is geen automatische test, dus er is ook geen testapplicatie voor ontwikkeld.

Van de volgende onderdelen wordt de weergave gecontroleerd:

- TTD
  - status
  - shadow_train_A timer
  - shadow_train_B timer
  - ghost_train_protocol timer
- VSS
  - status
  - disconnect timer
  - integrity_lost_propegation timer
- Trein
  - Integrity
  - Movement authority (End of Authority, EoA)
  - Rear end

Criteria

Alle boven staande onderdelen moeten weergegeven worden zoals in de requirement aangegeven is.

Strategie

De status van een onderdeel en de weergave daarvan zal handmatig gecontroleerd worden. De visualisatie applicatie zal in debugmodus uitgevoerd worden. Hierdoor kan de output van de mCRL2 simulator bekeken worden waarmee de status van een onderdeel achterhaald kan worden. Vervolgens kan de manier waarop dit onderdeel is weergegeven vergeleken worden met de bijbehorende requirement.

Rapportage

Testcase #	Requirement #	onderdeel	status	verwachte weergave	werkelijke weergave
	R1	TTD.status	FREE	groen	groen
	R1		OCCUPIED	geel	geel
	R6	TTD.timer_shadow_train_A	RUNNING	groen	groen
			EXPIRED	rood	rood
			STOPPED	geel	geel
	R6	TTD.timer_shadow_train_B	RUNNING	groen	groen
			EXPIRED	rood	rood
			STOPPED	geel	geel
	R6	TTD.timer_ghost_train_protocol	RUNNING	groen	groen
			EXPIRED	rood	rood
			STOPPED	geel	geel

		VSS.status	FREE	groen	groen
			OCCUPIED	geel	geel
			AMBIGUOUS	oranje	oranje
			UNKNOWN	rood	rood
	R6	VSS.timer_disconnect	RUNNING	groen	groen
			EXPIRED	rood	rood
			STOPPED	geel	geel
	R6	VSS.timer_lost_propegation	RUNNING	groen	groen
			EXPIRED	rood	rood
			STOPPED	geel	geel
	R5	TRAIN.integrity	CONFIRMED	groen	groen
			LOST	oranje	oranje
			UNKNOWN	rood	rood
	R4	TRAIN.EoA	7	lijn tot einde EoA VSS (7)	lijn tot einde EoA VSS
	R3	TRAIN.rear_end	0	lijn tot begin rear_end VSS (0)	lijn tot begin rear_end VSS

Handleiding ERTMS_vis

Gebruik ERTMS_vis applicatie

Uitleg scherm

ERTMS_vis toont het volgende scherm:

1 mogelijk acties

Links bovenin worden de acties weergegeven die je kan uitvoeren. Je kan een actie uitvoeren door er op te klikken of door het aangegeven nummer op het toetsenbord te gebruiken.

2 Legenda

Onderdeel 2 laat een legenda van de verschillende onderdelen in de visualisatie zien.

3 Scenario

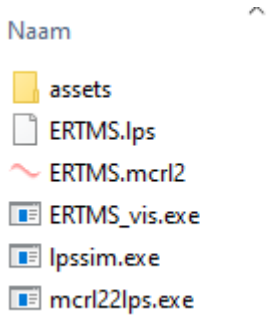
Onderdeel 3 is de visualisatie van het scenario van de simulator. Hierin worden de statussen van de VSS'en, TTD's en treinen weergegeven.

4 Detail

Onderdeel 4 laat meer details van een bepaald onderdeel zien. Dit venster komt tevoorschijn wanneer de gebruiker zijn cursor boven een onderdeel plaatst. Dit werkt voor VSS'en, TTD's en treinen.

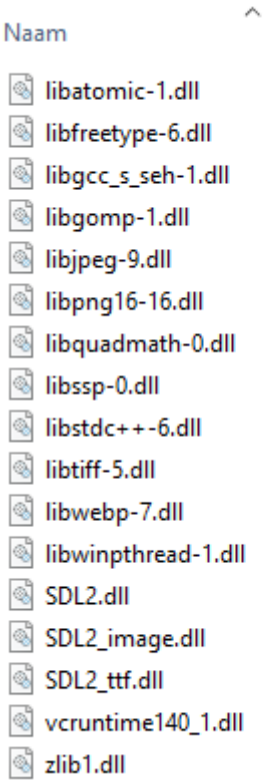
Benodigde onderdelen

De visualisatiesoftware is in de uitvoering afhankelijk van een aantal bestanden en programma's. Het is van belang dat deze onderdelen op de juiste locatie ten opzicht van de applicatie staan.



De visualisatieapplicatie is ERTMS_vis.exe. Het gewenste .mcrl2 bestand en het bijbehorende .lps bestand moeten in dezelfde map staan als de applicatie.
De assets map moet ook in deze map staan.
De toepassingen lpssim.exe en mcrl22lps.exe moeten ook in deze map staan, of in het systeempad vermeld staan. Dit laatste wordt verzorgd wanneer de mcrl2 toolkit met de installeerwizard geïnstalleerd wordt.

Daarnaast vergt de applicatie veel DLL's. Deze moeten ook in dezelfde map als de applicatie staan of op het systeem geïnstalleerd zijn.



Uitleg acties

Functienaam(Parameters)		**Toelichting**
break(TRAIN_id)	gebruiker	Verander de integrity van een trein naar LOST.
disconnect(TRAIN_id)	gebruiker	Laat een trein de verbinding met het RBC verliezen.
connect(TRAIN_id)	gebruiker	Laat een trein weer verbinding met het RBC maken.
move(TRAIN_id, [front_end, rear_end])	gebruiker	Verplaats de front en rear end van een trein naar de aangegeven VSS'en (VSS id aangegeven door front_end & rear_end).

enter(TTD_id)	systeem	Modelleert het TS dat merkt dat er een trein de TTD binnenrijdt.
leave(TTD_id)	systeem	Modelleert het TS dat merkt dat er een trein de TTD verlaat.
emit_status(TTD_id, TTD_status)	systeem	Een TTD rapporteert zijn status aan de TS.
emit_postition(TRAIN_id, ptd_info(front_end, rear_end, integrity, length_changed))	gebruiker	Een train meldt zijn status aan het RBC
Extend_EoA(TRAIN_id, VSS_id)	gebruiker	Verleng de MA/EoA van een trein naar de verste mogelijke VSS
ptd_stable()	systeem	Bevestig de informatie in het RBC
stable(VSS_statuslist)	systeem	bevestig de statussen van alle VSS'en
change(VSS_id, VSS_oldstatus, VSS_newstatus)	systeem	Verander de status van een VSS.

Sneltoetsen

De gebruiker kan gebruik maken van een aantal sneltoetsen op het toetsenbord om acties uit te voeren:

0 – 9:		Laat de simulator de actie met het ingedrukte nummer uitvoeren.
u:	undo	Draai de laatst uitgevoerd actie terug.
r:	redo	Voer de laatst ongedaan gemaakte actie opnieuw uit.
d:	dump	Schrijf de uitgevoerde acties naar een log genaamd 'tranisition_log.txt'
s:	save	Sla de trace van de simulator op naar 'trace.trace'.
l:	load	Laad een tracebestand met de naam 'trace.trace' in.
q:	quit	Stop de applicatie.

Veranderen scenario

In het .mcrl2 bestand staan meerdere scenario's gedefinieerd. Voorbeeld:

```
% section of page 9 (with the addition that there is an extra free VSS at the end of TTD 1)
VSSs_config(9,1)(0) = vss_info(free_vss,0,running,running);
VSSs_config(9,1)(1) = vss_info(unknown_vss,1,running,running);
VSSs_config(9,1)(2) = vss_info(ambiguous_vss,1,running,running);
VSSs_config(9,1)(3) = vss_info(occupied_vss,1,running,running);
VSSs_config(9,1)(4) = vss_info(free_vss,1,running,running);
TTDs_config(9,1)(0) = ttd_info(free_ttd, 0, 0, running, running, running);
TTDs_config(9,1)(1) = ttd_info(occupied_ttd, 1, 4, running, running, running);
TRAINs_config(9,1)(0) = train_info(2,2,2,confirmed,true,single,running,running);
TRAINs_config(9,1)(1) = train_info(3,3,3,confirmed,true,single,running,running);
Constants(9,1) = c(0,4,0,1,0,1);

% section of page 29, instance 1
VSSs_config(29,1)(0) = vss_info(occupied_vss,0,running,running); %VSS nr. 11
VSSs_config(29,1)(1) = vss_info(free_vss,0,running,running);     %VSS nr. 12
VSSs_config(29,1)(2) = vss_info(free_vss,1,running,running);     %VSS nr. 21
VSSs_config(29,1)(3) = vss_info(free_vss,1,running,running);     %VSS nr. 22
VSSs_config(29,1)(4) = vss_info(free_vss,1,running,running);     %VSS nr. 23
VSSs_config(29,1)(5) = vss_info(free_vss,2,running,running);     %VSS nr. 31
VSSs_config(29,1)(6) = vss_info(free_vss,2,running,running);     %VSS nr. 32
VSSs_config(29,1)(7) = vss_info(free_vss,2,running,running);     %VSS nr. 33
TTDs_config(29,1)(0) = ttd_info(occupied_ttd, 0, 1, running, running, running);
TTDs_config(29,1)(1) = ttd_info(free_ttd, 2, 4, running, running, running);
TTDs_config(29,1)(2) = ttd_info(free_ttd, 5, 7, running, running, running);
TRAINs_config(29,1)(0) = train_info(0,0,0,confirmed,true,single,running,running);
Constants(29,1) = c(0,7,0,2,0,0);
```

Je kan het gebruikte scenario veranderen door onderin het bestand de waardes van de variabelen ‘Page’ en ‘Instance’ te veranderen naar die van het gewenste scenario.

```
1657 %%%%%%%%%%%%
1658 %% System %%
1659 %%%%%%%%%%%%
1660
1661 map Page,Instance: Nat;
1662 eqn Page = 29; Instance = 1;
1663
1664 init Layout(Page,Instance);
1665
```

Wanneer dit bestand aangepast is moet het opnieuw gecompileerd worden door mcrl22lps.exe.

mCRL2 toolkit

(mcrl22lps, lpssim, mcrl2 ide?)

Hier volgt een uitleg over enkele tools uit de mCRL2 toolkit die handig kunnen zijn bij het gebruik van de visualisatieapplicatie.

mcrl22lps.exe

Deze applicatie kan gebruikt worden om een .mcrl2 bestand te compileren naar een .lps bestand.
De applicatie moet vanuit een console uitgevoerd worden. Het eerste argument is het .mcrl2 bestand dat gecompileerd moet worden. Het tweede argument is de gewenste naam van het nieuwe .lps bestand.

```
PS D:\> mcrl22lps.exe ertms.mcrl2 ertms.lps
```

Meer info hier.

lpssim.exe

Deze applicatie simuleert een .lps bestand.

```
PS D:\> lpssim.exe ertms.lps
```

De applicatie presenteert dezelfde acties die in in de visualisatieapplicatie zou zien. Na het vraagteken kan je de index van de actie die je wilt uitvoeren invullen. Voer in plaats van de index de letter ‘h’ in om een overzicht te krijgen van de mogelijke opdrachten.

```
initial state: [2, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, true, true, single, [], 1, 0, ptd_info
0: disconnect(0)  ->  [2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, false, true, single, [0, 0], 1,
1: break(0)  ->  [2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, true, false, single, [0, 0], 1, 0, p
2: emit_position(0, ptd_info(0, 0, unknown, false))  ->  [2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 0,
ig(29, 1)]
3: emit_position(0, ptd_info(0, 0, confirmed, false))  ->  [2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0,
config(29, 1)]
4: extend_EoA(0, 7)  ->  [2, 0, 0, 2, 0, 0, 2, 0, 0, 1, 0, 0, 7, 0, true, true, single, [0, 0],
 7, confirmed, true, single, running, running)]]
?
```

Meer info hier.

Broncode compileren

Benodigdheden

De volgende tools zijn nodig om de broncode te compileren:

- GNU Make (GnuWin32) 3.81
- MinGW-W64 (x86_64 posix-seh-rt_v6-rev0) 8.1.0

De software maakt gebruik van de volgende libraries:

- SDL2
- SDL2_ttf
- SDL2_image

Let op dat de Include en Library paden in de Makefile kloppen.

Compileren

De visualisatieapplicatie kan gecompileerd worden met de volgende opdracht:

```
PS D:\> make main
```

De testapplicatie kan gecompileerd worden met de volgende opdracht:

```
PS D:\> make test
```

Deze opdrachten moeten in de map waarin het bestand ‘Makefile’ staat uitgevoerd worden.

Revisie werkzaamheden

Inleiding

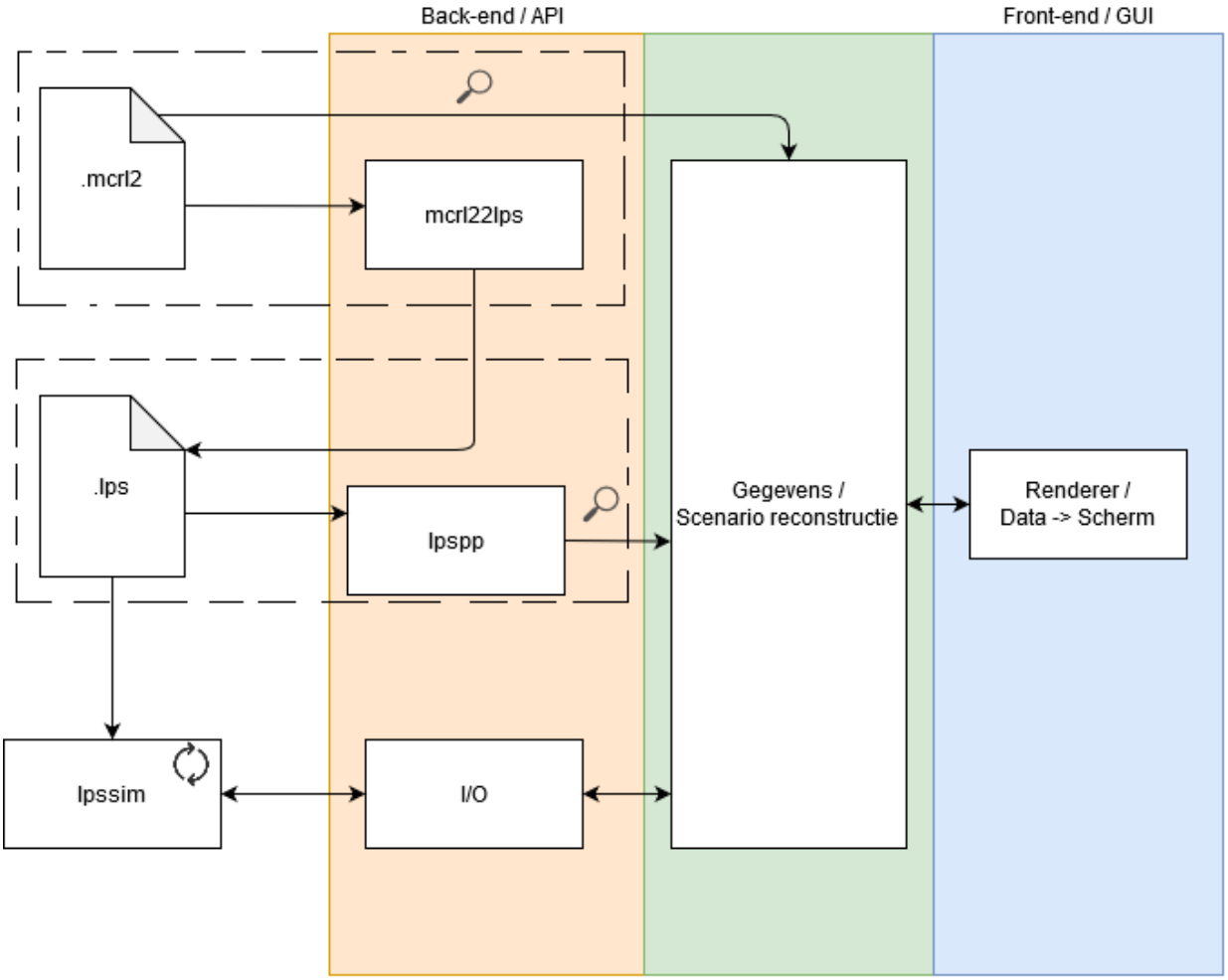
Dit document bevat een omschrijving van de verschillende onderdelen de te ontwikkelen visualisatie. Er zal omschreven worden welke twee keuzes er gemaakt kunnen worden met betrekking tot het prioriteren van de ontwikkeling van bepaalde onderdelen en het focussen van de werkzaamheden.

Aanleiding

Dit project wordt uitgevoerd als een afstudeerproject. De beoogde inleverdatum hiervan is 5 oktober. Op 28 augustus vindt een tussentijds assessment plaats. Voor dit assessment dient het product en het bijbehorende verslag voor ongeveer 80% compleet te zijn.

De visualisatie dient een grafische weergave van de bestaande ERTMS HL3 mCRL2 simulatie te geven. De visualisatie moet uit de simulatie gegevens ontvangen en verzamelen om een scenario op te bouwen om deze vervolgens weer te geven.
De bestaande simulator maakt niet alle benodigde gegevens beschikbaar. Dit maakt dat de visualisatie een combinatie van bronnen moet gebruiken om het scenario van de simulatie volledig te reconstrueren.
Aanvankelijk bestond het idee dat de simulatie alle benodigde gegeven vrijgaf en dat deze vrij simpel verzameld konden worden. Nu blijkt dus dat er een extra onderdeel nodig is dat uit meerdere bronnen de gegevens verzameld en vervolgens tijdens het uitvoeren van de simulatie veranderingen in deze gegevens bijhoudt en op de juiste plekken de gegevens aanpast.

Schets onderdelen



Back-end

De back-end maakt een reconstructie van het scenario in de simulatie aan de hand van data uit de simulatie zelf, en data uit de bestanden waarin de specificatie vastgelegd is.
De data die vooraf verzameld moet worden (die niet uit de simulatie te halen is) kan uit de .mcrl2 bestanden of de .lps bestanden gehaald worden. Een .lps bestand is een binaire blob. Dit bestand kan met het lpspp (lps pretty print) programma ‘vertaald’ worden naar begrijpbare en ‘parse-bare’ data.

Opslag gegevens

In de middelste laag wordt de data opgeslagen. Hier is de reconstructie van het scenario tijdens runtime in opgeslagen.

Front-end

De front-end bevat functies die de data van het scenario gebruiken voor het bouwen van de grafische weergave.

Benadering

Het prioriteren kan op drie manieren benaderd worden:

Een optie is het focussen op de front-end. Hiervoor ontwikkel je de functies die aan de hand van de reconstructie van het scenario het beeld vormt.

De tweede optie is focussen op de back-end. Hiervoor ontwikkel je de functies die uit de mcrl2 bestanden en de simulatie de juiste gegevens halen en deze op de juiste plaats invullen en aanpassen.

Een derde optie is het iteratief ontwikkelen van beide kanten tegelijkertijd. Hiervoor ontwikkel je per benodigd datapunt de manier om de juiste data te achterhalen en de manier om het te tekenen. Wellicht kunnen dan niet alle punten uitgewerkt worden, maar je hebt wel voor bepaalde punten een complete werkende uitwerking van de back-end t/m de front-end. De punten die uitgewerkt moeten worden kunnen geprioriteerd worden.