Graph Analyse Op Mobile Grid Computing Netwerk

Tom van Amstel

20 maart 2017

Informatica – Haagse Hogeschool, locatie Zoetermeer

Eerste examinator: Dhr. R. Loke

Tweede examinator: Dhr. H. Al-Ers

Abstract

In dit document is het traject beschreven van de afstudeerstage van T. van Amstel in de periode november 2016 tot april 2017 bij het bedrijf Sogyo. De afstudeeropdracht gaat over het bouwen van een applicatie die als benchmarking tool gebruikt kan worden om te bewijzen dat door middel van parallelle collaboratie een complexe opdracht uitgevoerd kan worden door een grid computing netwerk.

Om dit te bereiken is er gekozen om het project te simuleren in plaats van met hardware te werken. Deze simulatie werkt met een customised game loop structuur die met behulp van sequentiële code een parallelle situatie kan simuleren. De onderlinge collaboratie tussen de nodes (devices) van het grid computing netwerk is de grootste focus van het project.

Als implementatie van het probleem dat opgelost moet worden door het gesimuleerde grid computing netwerk is er gekozen voor een graph die doorzocht wordt met behulp van een breadth-first-search algoritme. Dit algoritme is geparametriseerd en kan in stappen uitgevoerd worden, waardoor er per device CPU cycle een stap uitgevoerd kan worden. Hierdoor kan de sequentiële code gezien worden als parallel.

In het document zal dieper ingegaan worden op het proces dat gekozen is en de keuzes die gemaakt zijn in het project. Tevens zal het behaalde resultaat behandeld worden en zal het design van de applicatie en de daarbijhorende keuzes uitgelegd worden.

Keywords: mobile, grid computing, unreliable network, graph theory, graph analysis

Dankbetuiging

Na een periode van ongeveer 17 weken rond ik mijn afstudeerstage af. Het is een periode waar ik erg gegroeid ben, zowel persoonlijk als vakinhoudelijk. Met dit dankwoord wil ik graag stil staan bij de mensen die mij deze periode hebben ondersteund en geholpen.

Ik wil graag mijn collega's bij mijn stagebedrijf Sogyo bedanken voor een gezellige en fijne tijd. Hoewel het soms een beetje druk was, was het altijd gezellig. In het bijzonder wil ik stil staan bij mijn begeleiders: B. Klever en R. Wolter. Jullie hebben me tijdens het project bij zowel de technische als de procesmatige kant erg goed gesteund. De feedback die ik met regelmaat kreeg op mijn keuzes, producten en mijn scriptie heeft erg veel geholpen.

Daarnaast wil ik graag mijn studiebegeleiders, R.E. Loke en H. Al-ers, bedanken voor de fijne begeleiding. De hulp die ik kreeg voor de verduidelijking van mijn scriptie heeft een positief effect gehad op het eindresultaat.

Als laatste wil ik mijn familie bedanken. Ondanks de afstand die tussen ons ligt waren ze er altijd voor me, zowel als luisterend oor, klankbord of gewoon om over iets te praten wat niet mijn scriptie was.

Inhoudsopgave

Contents

1 Glossary

Term	Definitie
Grid computing	*"The ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across 'multiple' administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements.",* (IBM, 2002-2003)
Device	Het (echte of gesimuleerde) systeem dat een node is van een grid computing netwerk, niet te verwarren met de nodes van een graph. Meerdere devices vormen samen een grid computing netwerk. Deze systemen bevatten delen van de graph en zullen de analyse afhandelen.
Node	Een node is één van de twee basiselementen van een graph. Het is een aanknopingspunt van een edge en bevat types die het onderscheiden.
Edge	Een edge is één van de twee basiselementen van een graph. Een edge verbindt nodes aan elkaar, wat als gevolg heeft dat een edge twee nodes bevat.
Systeem	Een node van het type Systeem. Een systeem dat gebruikt wordt in een bepaald proces. Aan een systeem kunnen alleen hypotheses verbonden worden.
Hypothesis	Een node van het type hypothesis. Een hypothese van een probleem dat zou kunnen opspelen bij een systeem. De hypothese heeft evidence, sub-hypotheses of systemen eraan verbonden.

Term	Definitie
Evidence	Een node van het type hypothesis. Bevat een simpele waarde die als bewijs voor een hypothese kan worden gebruikt. Deze nodes kunnen alleen worden verbonden aan hypothesen.
Executable architecture	"*An executable architecture is a partial implementation of the system, built to demonstrate that the architectural design will be able to support the key functionality […]*", (Kruchten, 2004)
TimeStep	De meeteenheid die gebruikt wordt om de snelheid van een bepaald device te bepalen met betrekking tot de device CPU cycles.
Device CPU cycle	De term die gebruikt wordt voor een volledige cyclus van de game loop.
Parallelle collaboratie	Het parallel uitvoeren van een complexe opdracht in samenwerking met andere devices.

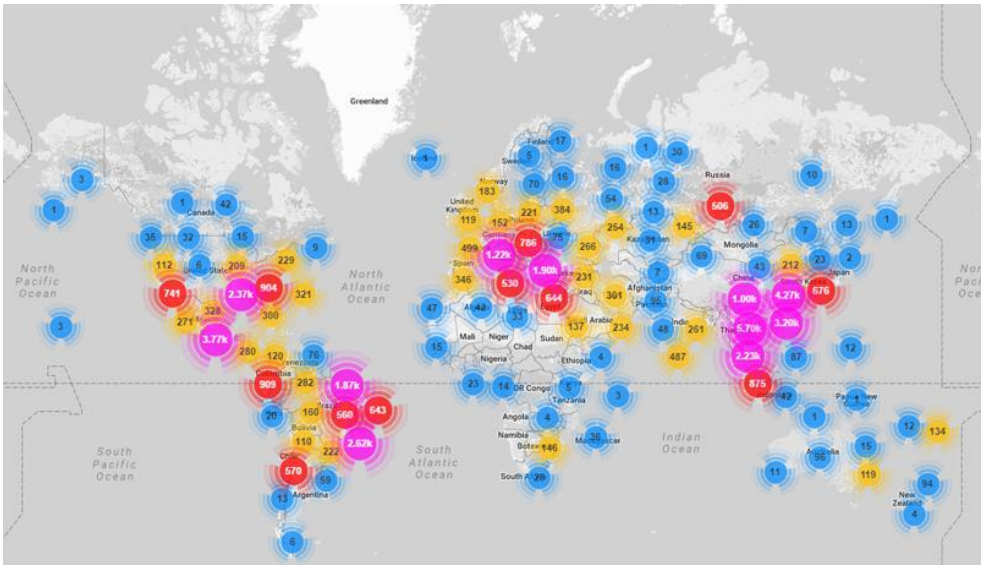
Sectie 1

Voorbereidende werkzaamheden

In deze sectie van het document zal het voorbereidende werk worden besproken. Er wordt dieper ingegaan op de achtergrond van het project, de methodieken en de technieken en tools die gebruikt worden. Ook zal het plan van aanpak behandeld worden en zal het bedrijf omschreven worden waar de opdracht voltooid word.

2 Introductie

Op 19 september 2016 werd er een DDoS aanval gerapporteerd door de CTO van het bedrijf OVH. De grootte van de aanval blies elke andere DDoS aanval uit het water: van de grotere aanvallen van 350-400 Gbps ging het naar een aanval die piekte op 1.1 Tbps. De reden dat deze aanval zo groot kon worden was omdat het gebruik maakte van Internet of Things devices: (semi-)geautomatiseerd apparaten. Er zijn veel van dit soort devices en ze zijn makkelijk te infecteren en over te nemen, wat resulteert in een gigantisch botnet. Het botnet dat de aanval uitvoerde is genaamd ‘Mirai’ en bestond op het moment van de aanval uit een geschatte 145607 devices (Oles, 2016)(Figuur 1).



Figuur 1, geo-locatie van alle Mirai geïnfecteerde devices

Deze DDoS aanval laat de kracht zien van grid computing: het inzetten van een hoeveelheid aan elkaar gekoppelde devices zodat er extreem veel processorkracht kan worden ingezet voor een bepaald doeleinde. Op dit punt is het gebruikelijk om grid computing vooral toe te passen op simpel parallelliseerbare problemen, waarbij er niet veel interactie tussen de devices zelf is. De vraag hoe een grid computing netwerk zich gedraagt met een complexer probleem, waarbij meer communicatie tussen devices nodig is, is de basis voor dit document. In concrete termen zal het project een benchmarking tool opleveren die gebruikt kan worden om te testen hoe een onbetrouwbaar mobiel grid netwerk zich gedraagt met lastig parallelliseerbare problemen. Dit wordt mogelijk gemaakt door de generatie van metrieken gebaseerd op instellingen die op de devices kunnen worden toegepast. Hierbij kan gedacht worden aan latency, de kloksnelheid van de CPU’s, packet loss en meer.

3 Bedrijfsoriëntatie

Deze afstudeeropdracht zal uitgevoerd worden in opdracht van Sogyo Information Engineering B.V. te De Bilt. Het werk zal op de hoofdlocatie worden volbracht.

3.1 Bedrijfsomschrijving

> The Japanese term for company birth is sogyo and many companies today speak of experiencing a second or third or 'new' sogyo. It is precisely at this moment of rebirth that long-term success or failure is determined. For if the new reborn firm is still organized along bureaucratic lines, like the old one it replaces, it may have a short and unhappy second life.
> (Toffler, 1991, p. 185)

Deze quote uit het boek Powershift is de inspiratie geweest voor de naam Sogyo. Het refereert zowel naar de geboorte van het bedrijf Sogyo als naar de hulp die Sogyo kan bieden bij de sogyo van een nieuw of bestaand bedrijf.

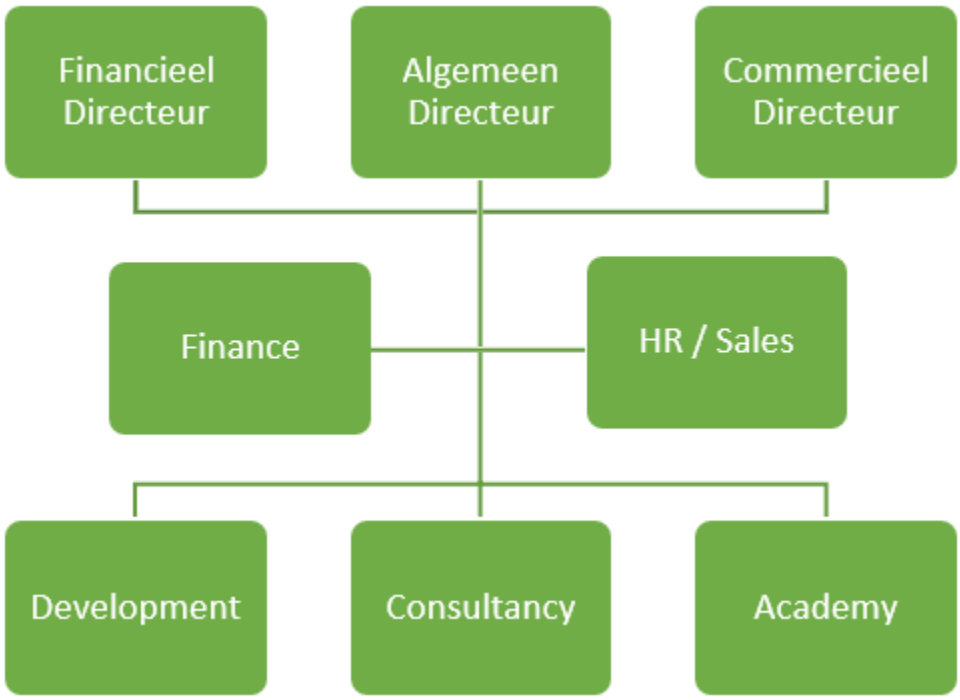
Sogyo is gespecialiseerd in software architectuur, softwareontwikkeling en software integratie. De kennis en kunde die Sogyo vanaf de oprichting in 1995 heeft opgebouwd wordt ingezet voor het ontwikkelen van IT-talent, IT-toepassingen en IT-organisaties. Ze is voor een groot gedeelte een for-profit organisatie, die diensten heeft geleverd aan onder andere ABN Amro, ING, Rabobank, Triodos en EBay.

Het vizier van Sogyo richt zich vooral op het werven, selecteren, ontwikkelen en begeleiden van jonge IT-talenten. Ook biedt ze verschillende cursussen en lectures aan voor bedrijven om het softwareontwikkelingsniveau op een hoger niveau te brengen. Verder maakt ze maatwerksoftware die in een agile aanpak wordt ontwikkeld binnen een afgebakend budget.

3.2 Bedrijfsstructuur

Sogyo bestaat op het moment van schrijven uit 90 medewerkers verdeeld over vijf afdelingen. Er ligt een grote focus op het voorop blijven lopen van de markt qua kennis over software engineering. Hierdoor kunnen ze klanten blijven helpen door hun kennis te delen. De human resource/sales afdeling is verantwoordelijk voor de werving en selectie van nieuw personeel bij

Sogyo. Ze onderhouden ook het contact met werknemers die zijn ingezet en met Sogyo's klanten. Verder matchen ze medewerkers van Sogyo aan opdrachtgevers. Finance zorgt ervoor dat alle financiële aspecten van Sogyo in goede banen worden geleid.



Figuur 2, organogram Sogyo

3.2.1 Development en consultancy

Het development team van Sogyo werkt aan projecten die zijn opgezet in opdracht van externe opdrachtgevers. Het gaat hier meestal om wat kleinere bedrijven die geen IT-afdeling hebben of niet de expertise in huis hebben. Als projecten zijn voltooid pakt de development afdeling ook een stuk van het beheer op. Er wordt gewerkt in een Scrum omgeving, waarbij daily stand-ups, planning poker en sprint retrospectives een grote rol spelen.

Ook wordt er consultancy gegeven aan externe partijen: leden van het development team bezoeken de partijen op locatie en geven deskundig advies op gebieden waar de externe partij dat nodig heeft.

3.2.2 Academy

In de Academy beginnen nieuwe developers met een opleidingstraject. Dit traject is meestal rond de twee tot vijf maanden. Elke maand starten er tussen de twee en zes nieuwe medewerkers. De experts van de Academy begeleiden de deelnemers en geven colleges en workshops; ze zijn ook verantwoordelijk voor het geven van gastcolleges op hogescholen en trainingen bij bedrijven. Afstudeerders komen in de Academy terecht, zodat ze genoeg technische begeleiding door de experts krijgen. Vaak is er ook een begeleider uit de humanresourcesafdeling die het scrum proces in de gaten houdt.

4 Theorie

De term grid computing is bedacht om de collectie van computer resources van meerdere locaties met als richtpunt het bereiken van een gemeenschappelijk doel te beschrijven. Het is een gedistribueerd systeem waarbij elke node een andere taak uitvoert, hierdoor onderscheidt het zichzelf van de andere (virtuele) supercomputers. Ook verschillen de apparaten die gelinkt zijn met elkaar in een grid netwerk meestal met elkaar qua hardware specificaties, ze zijn vaak niet fysiek gekoppeld.

In de vroege jaren van 1990 werd de term ‘grid computing’ bedacht. De vergelijking werd hiermee getrokken naar het elektriciteitsnet: de bedoeling van grid computing was namelijk om processing power net zo makkelijk bereikbaar maken als het elektriciteitsnet. Al gauw werd de term ‘grid computing’ de meest gebruikte term voor het concept, na jaren van het gebruik van de term ‘utility computing’.

De beginselen van het echte grid computing kunnen gevonden worden rond 1997 in de release van distributed.net, waar het vooral ging over CPU Scavenging en volunteer computing. De combinatie van deze twee concepten brengt als resultaat dat computers parallel kunnen werken aan oplossingen voor problemen die ze toegezonden krijgen. Dit gebeurt door een ‘grid’ op te zetten van de ongebruikte resources in het netwerk van gebruikers. Door de ongebruikte CPU cycles te gebruiken die anders verspild zouden zijn kunnen de computers in het ‘grid’ helpen om de oplossing te vinden voor hun probleem (Wang & Udoh, 2009).

De echte popularisatie van het concept grid computing begon in 1998 toen SETI@home werd aangekondigd. In één jaar waren er al 400,000 gebruikers geregistreerd en toen het uitgebracht werd in 1999 waren er binnen vier maanden 3,91 miljoen mensen die de client

gedownload en gedraaid hadden (Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002, p. 59). Het idee was om een netwerk van PCs op te bouwen en de processing power van deze PCs in te zetten om wetenschappelijk werk te doen en tegelijk de levensvatbaarheid en uitvoerbaarheid van volunteer computing te bewijzen.

De huidige ideeën van grid computing werden gerealiseerd door de zogenaamde ‘fathers of the grid’: Ian Foster, Steve Tuecke en Carl Kesselman (Foster, 2004). Deze mannen zitten achter de Globus Toolkit, wat de implementatie is van een combinatie van de ideeën over grid computing, distributed computing, object-oriented programming en webservices. Het idee van ‘grid computing’ is dus door de jaren heen ontwikkeld en veranderd, maar op het moment van schrijven houdt IBM de volgende definitie aan:

> The ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across ‘multiple’ administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements.
> (IBM, 2002-2003)

5 Project beschrijving

5.1 Aanleiding

“Is het mogelijk om een ‘zwerm’ van mobile devices aan elkaar te linken en met de samengevoegde processorkracht en onderlinge communicatie een opdracht uit te voeren?”, deze vraag wekte de interesse van een Senior IT Consultant bij Sogyo. Deze IT Consultant werkt ook bij een van de grote banken en is daar bezig om met behulp van graph theorie een diagnosesysteem te maken. Het doel van dit systeem is om een correcte diagnose te maken als er iets fout gaat in het systeem van deze bank. De manier waarop dit bereikt wordt is door een graph op te stellen met nodes die evidence, symptomen en hypotheses bevatten. Hierna kan met

behulp van een kortste pad algoritme de root oorzaak van het probleem worden geïdentificeerd. Deze aanpak resulteert in een erg grote graph, wat het systeem erg traag maakt. Hier komt het idee vandaan om de analyse met grid computing te doen. Het project is dus opgezet om meer kennis op te doen op het gebied van (mobile) grid computing, wat direct in lijn ligt met het doel van de organisatie: het vooruit blijven lopen op gebied van kennis in software engineering.

5.2 Aanvangssituatie

Omdat het een totaal nieuw project is, is de aanvangssituatie compleet theoretisch. Er is geen concrete grondlegging van het project, waardoor het project vrij in te richten is. Dit heeft zowel voor- als nadelen. Een van de voordelen is dat er geen omgeving is om het in te bouwen, wat ervoor zorgt dat er vrij gekozen kan worden tussen de verschillende programmeertalen en het globale design. Helaas is dit niet alleen maar positief, aangezien al deze keuzes ook tijd in beslag nemen. Dit wil zeggen dat de scope van het project goed in de gaten moet worden gehouden.

Waar ook rekening mee gehouden moet worden is de real-world applicatie waar de opdracht op gebaseerd is. Zoals hierboven beschreven staat is het project los gebaseerd op een lopend project. De structuur van de graphs moest hierdoor overeenkomen met de structuur die gebruikt wordt in het bestaande project.

5.3 Doelstelling

Zoals gezegd in de aanleiding is het project opgesteld op basis van een systeemdiagnose-systeem. Met behulp van hypothesen, evidence en symptoom nodes wordt de root oorzaak van een probleem in een graph gelokaliseerd. Deze graphs worden te groot, dus wordt er gezocht naar een oplossing voor dit probleem.

Hoewel het project gemaakt is op basis van dit voorbeeld is het belangrijkste punt van dit project het maken van een benchmark tool. Deze benchmarking tool moet gebruikt kunnen worden om aan te tonen dat een grid computing netwerk ook complexe problemen kan oplossen door gebruik te maken van parallelle collaboratie. Parallelle collaboratie wil zeggen dat de devices in het netwerk met elkaar moeten communiceren om tot een oplossing te komen voor het probleem. Om dit punt te kunnen bereiken moet het generieke probleem dat aan het netwerk wordt voorgelegd in stappen verdeelbaar zijn. Door deze verdeling kunnen veel problemen die verdeelbaar zijn in deze stappen afgehandeld worden door het netwerk, wat als gevolg heeft dat

de verwerking van het probleem de focus is. Gecombineerd met de andere doelstellingen kan er binnen Sogyo nieuwe kennis worden opgedaan, dit ligt in lijn met het business doel van Sogyo.

5.4 Programma van eisen

Om een project in goede lijnen te laten lopen zijn er voorwaarden nodig en moet er rekening gehouden worden met eventuele risicofactoren. Ook moet de scope worden opgesteld zodat het project meetbare grenzen heeft. Dit zorgt ervoor dat het project haalbaar en gestructureerd wordt. Verder zijn er een aantal projectgebonden risico's en niet projectgebonden activiteiten die gedaan zullen moeten worden.

5.4.1 Randvoorwaarden

De voorwaarden voor het project zijn voldoende begeleiding van Sogyo, medewerking met de gekozen projectmethodiek en de aanwezigheid van begeleider Sogyo bij gesprekken met de Haagse Hogeschool. Ook zal er support geboden worden vanuit Sogyo voor versiebeheer en zullen er vergaderingen gehouden worden zodat de voortgang gemonitord kan worden. Verder moet het project binnen 17 weken af te ronden zijn en zal er gezorgd worden voor een werkplek op kantoor.

5.4.2 Scope

Zoals in de doelstelling wordt vermeld focust het project zich op het maken van een simulatie die een complex probleem met behulp van parallelle collaboratie op kan lossen. Het probleem dat de graph gaat oplossen moet hiervoor op te delen zijn in stappen. Het generieke probleem wordt dan met communicatie tussen de devices stapsgewijs opgelost.

Het grid computing concept moet rekening houden met de limitaties van een onbetrouwbaar netwerk, hierbij kan gedacht worden aan latency en packetloss. Om concreet te zijn wordt het generieke probleem een graph analyse algoritme. Het algoritme moet de graph die in de devices gehouden wordt analyseren door er met een breadth-first-search algoritme stapsgewijs doorheen gaan. Ook moet de applicatie de graphs visueel weergeven, met de devices waar ze bij horen. Als het algoritme loopt moet er gevisualiseerd worden hoe de progressie gaat. Als laatste moeten er ook metrieken gegenereerd en getoond worden zodat het einddoel te meten is, hoewel dit een minder belangrijk aspect van het programma is. Dit wordt besproken in een later stadium van dit document.

5.4.2.1 Niet projectgebonden werkzaamheden

Er wordt van alle medewerkers van Sogyo verwacht om eens in de paar maanden corvee te draaien. Dit houdt in dat er boodschappen voor de lunch gedaan moeten worden en dat de gezamenlijke ruimte schoongemaakt moet worden. Ook moet er een ontwikkelstraat worden opgesteld. Er moet gekozen worden voor een versiebeheer tool, IDE, documentatieomgeving en testmethode.

5.4.3 Risicoanalyse

Risico	Maatregel
Onvoldoende kennis	Field research. Praten met experts op locatie. Doornemen literatuur.
SCRUM-sprintdoel wordt niet bereikt	Van tevoren de SCRUM-master en bedrijfsmentor op de hoogte stellen. Overleg in sprint retrospective hoe het beter kan gaan volgende keer. Niet gehaalde backlog items naar volgende sprint overhevelen of weghalen.
Onvoldoende hardware	Er is een zeer reële mogelijkheid dat de hardware onvoldoende krachtig is om op te ontwikkelen. In dit geval zal er in overleg met Sogyo een virtual machine worden ingesteld met krachtigere hardware.
Interne gitlab server faalt	De servers van Sogyo maken elke dag een back-up, die terug gezet kan worden.

Tabel 1, risico's van project

5.5 Requirements

Om het project wat hierboven beschreven wordt te omschrijven zijn er requirements opgesteld die te vinden zijn in Bijlage F. Deze requirements zijn geprioriteerd met behulp van het MoSCoW principe en zijn verdeeld in non-functional, functional en business requirements. Verder is er nog een verdeling gemaakt bij de functional requirements, zodat er een logische structuur ontstaat. De requirements worden vooral aangehaald bij hoofdstuk 9, Design. Hier

worden de requirements die gebruikt zijn voor een beslissing kort vermeld, met de redenering van de gemaakte keuze. Er zijn ook een aantal keuzes die hebben geleid tot de aanmaak van nieuwe requirements, deze zijn worden ook behandeld.

De meest belangrijke requirements zijn de business requirements, hier kan in gevonden worden wat het product vanuit een abstract viewpoint moet gaan doen. Van deze requirements is de belangrijkste BR01 (zie Tabel 2). Deze requirement komt overeen met de doelstelling en is de kern van het project. Voor de rest wordt de instelbaarheid en het abstractieniveau behandeld, waarbij ook gekeken moet worden naar het realiteitsniveau van de uiteindelijke benchmarking tool.

ID	Description
BR01	Er moet een product gemaakt worden dat als benchmark gebruikt kan worden om het concept ‘parallelle collaboratie’ te bewijzen
BR02	Het product moet voldoende instelbaar zijn
BR03	Het product moet andere implementaties van analyse algoritmes kunnen ondersteunen
BR04	Het product moet zo veel mogelijk op de realiteit lijken, zodat er een goede vergelijking getrokken kan worden met de realiteit

Tabel 2, uitgelichte requirements van Bijlage F.

6 Project methodiek

Om het project dat in hoofdstuk 5 beschreven staat te faciliteren moet de structuur van het project duidelijk zijn. Er zijn project ontwikkelmethodieken opgesteld om het verloop van een software ontwikkeltraject gestructureerd te laten verlopen. Om helder te krijgen wat dit project nodig heeft zullen er criteria worden opgesteld en de gekozen methodieken behandeld worden. Hierna zal een keuze gemaakt worden op basis van de methodieken en de resultaten.

Het projectteam heeft het meest ervaring met SCRUM, maar om een breed genoeg draagvlak te maken van de verschillende project ontwikkelmethodieken zal er naast SCRUM ook gekeken worden naar een aantal andere ontwikkelmethodieken. De gekozen methodieken waarnaar gekeken wordt zijn: SCRUM, OpenUP, de Watervalmethode en Extreme Programming.

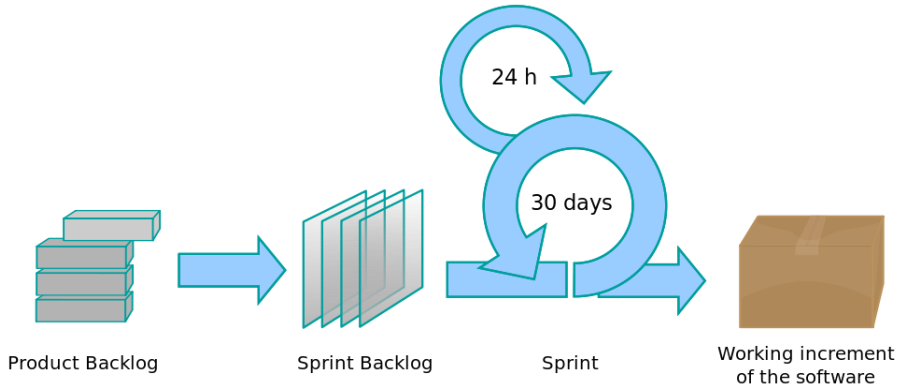
6.1 Criteria

Er zijn een aantal criteria waar de projectmethodiek aan moet voldoen. De keuze voor criteria is gevallen op: flexibiliteit, iteratieve en incrementele werkwijze, goede code coverage

met behulp van tests en het moet door één persoon uitvoerbaar zijn. Flexibiliteit en een iteratieve, incrementele werkwijze zijn gekozen omdat het project niet vanaf het begin duidelijk is. Er kunnen nog aspecten van het project veranderd worden en er kan worden overgegaan op een meer complexe manier van werken als het project goed verloopt. Dit brengt meteen het volgende punt ter sprake: er wordt vanuit Sogyo verwacht dat er voldoende code coverage is in het project. Dit zorgt ervoor dat het project voldoende getest is en dat de bruikbaarheid omhoog gaat. Als laatste moet er ook in het achterhoofd gehouden worden dat de methodiek goed uitgevoerd kan worden door één persoon. Een criteria die niet opgenomen wordt in de eindtabel is dat Sogyo eist dat er elke twee weken een demo moet worden gegeven en wordt besproken wat er is gedaan en gaat gebeuren.

6.1.1 SCRUM

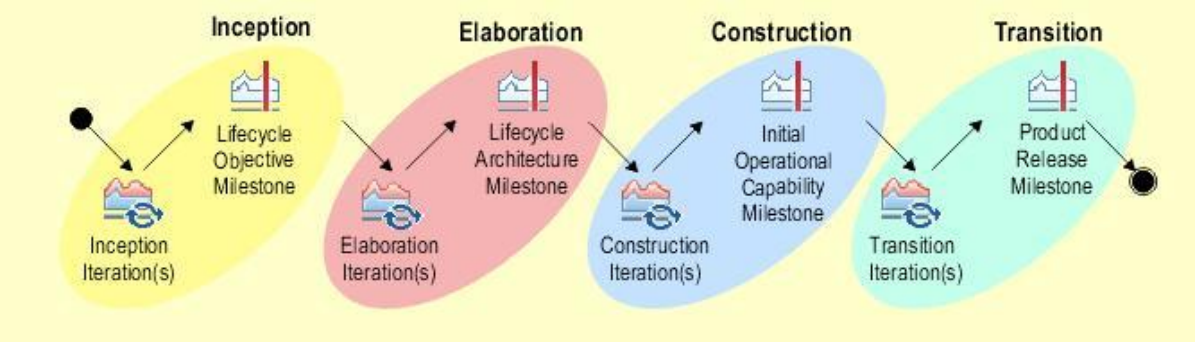
Scrum is een agile ontwikkelmethodiek die bedoeld is om zo flexibel mogelijk met projecten om te gaan. Het is een iteratieve en incrementele ontwikkelmethodiek waarbij gewerkt wordt met sprints. Het SCRUM-proces (Figuur 3, Scrum proces) bevat korte periodes van twee tot zes weken met bepaalde doelen die gehaald moeten worden, deze doelen worden ingepland en komen terecht in de sprint backlog. De in te delen doelen zijn te vinden in de product backlog. Binnen Sogyo wordt er gewerkt met sprints van twee weken met daily standups en een sprint retrospective aan het eind van de week. Hier wordt de afgelopen sprint besproken, de nieuwe sprint ingedeeld en demo's getoond. Hoewel het lastig is om SCRUM met één persoon te doen, kan het wel. De product- en sprintbacklogs kunnen gewoon gevuld worden en de retrospective zou gehouden worden met de maar één persoon van het ontwikkelteam, de scrum master en bedrijfsbegeleider.



Figuur 3, Scrum proces (Lakeworks, 2016)

6.1.2 OpenUP

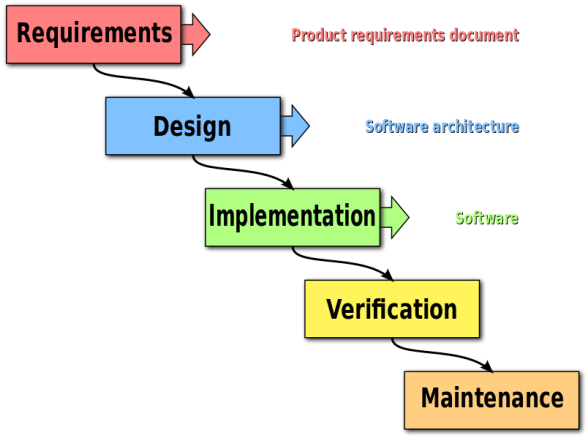
OpenUp is een methodiek die veel gelijkenissen heeft met RUP, maar is veel makkelijker over te nemen. OpenUP brengt een lichtere vorm van RUP, door de complexiteit en grootte van RUP te verlagen (Gustafsson, 2008). Het verschil met SCRUM zit vooral in de hoeveelheid documentatie en de algemene cultuur van de methodiek, SCRUM is in dit geval veel lichter. OpenUP gebruikt veel eigenschappen van RUP, zo ook de vier RUP-fases (Figuur 4, OpenUP proces), inception, elaboration, construction en transition. In de Inception wordt de scope en de doelstellingen gedefinieerd en de requirements gemaakt. In de Elaboration fase worden de requirements uitgewerkt en wordt er een executable architecture gemaakt. Daarna volgt de Construction fase, waar de functionaliteit van het systeem gemaakt wordt. Als laatste is de Transition fase, daar wordt het systeem overgedragen aan de eindgebruikers. OpenUP is wat meer gericht op grotere teams dan SCRUM, wat het lastiger maakt om het in je eentje te doen.



Figuur 4, OpenUP proces (GFLewis, 2016)

6.1.3 Watervalmethode

Het waterval model is een niet iteratief, niet incrementeel designproces. Zoals de naam al zegt wordt de progressie gezien als een waterval, het loopt gestaag door en naar beneden (Figuur 5). Deze methodiek is niet flexibel, het is heel lastig om wijzigingen aan het project te doen en het project moet van tevoren duidelijk zijn. Deze methode kan met één persoon gedaan



Figuur 5, waterval proces (Smith & Kemp)

worden, hoewel het dan wel lastig wordt om alle requirements aan het begin goed te krijgen.

6.1.4 Extreme programming

XP is een agile ontwikkelmethode die heel erg gericht is op flexibiliteit. Er is nagenoeg geen documentatie en er wordt gewerkt in coding pairs. Door te werken in pairs wordt de code kwalitatief erg goed. Om de code kwalitatief nog beter te maken wordt er geëist dat alle code coverage heeft door middel van unit tests. Helaas moet er rekening gehouden worden met de criteria dat de methodiek uitgevoerd kan worden met één persoon. Aangezien Extreme Programming erg gefocust is op coding pairs, zal dit lastig zijn.

6.2 Conclusie

Naar aanleiding van de opgestelde criteria is er een overzicht gemaakt van de verschillende projectmethodieken en de beoordeling die ze gekregen hebben voor dit project. In Tabel 3 is het overzicht te zien, waaruit blijkt dat er twee methodieken zijn die erg goed uitpakken voor dit project: SCRUM en XP. De flexibiliteit is erg groot ten opzichte van de twee meer formele methodieken. Dit is mede dankzij de gewenste documentatie: bij OpenUP en Waterval is er veel documentatie dat gericht is op het helder krijgen van een project voor een grote organisatie. Omdat het project door één persoon zal worden uitgevoerd, is het belang van deze documentatie laag.

De code coverage van OpenUp en XP zijn erg hoog, dit komt doordat het ingebakken zit in de ontwikkelmethodiek. Omdat Sogyo verwacht dat er goede code coverage is in het project is dit een pluspunt voor de beide ontwikkelmethodieken. Bij zowel Scrum en Waterval word je aardig vrijgelaten in de keuze voor testtechniek, waardoor het in dit geval als een negatief punt gezien wordt.

	SCRUM	**OpenUP**	**Waterval**	**XP**
Flexibiliteit	++	+-	--	++
Documentatie	+	-	--	+-
Iteratief/Incrementeel	++	++	--	++
Code coverage	+-	++	+	++
Eén persoon	+-	-	-	--

Tabel 3, overzicht plus- en minpunten van methodieken

Zoals eerder vermeld zijn er twee methodieken die het best uit het overzicht komen. Omdat XP altijd met minimaal twee personen gedaan wordt en het feit dat Scrum iets beter uit de test kwam, is er gekozen als basis methodiek Scrum te gebruiken. Bij Scrum word je vrijgelaten in de keuze voor testtechniek en omdat XP ook heel goed beoordeeld was is er gekozen om de testtechniek strategie van XP te nemen: unittests.

Omdat de projectindeling bij SCRUM gedaan wordt per sprint is er weinig kennis over de latere fases van het project. Hierom is er gekozen om de fase-indeling van RUP / OpenUP te gebruiken. Met behulp van deze fases wordt er gedwongen om structuur te geven in het begin van het project.

Met behulp van SCRUM worden de requirements ingedeeld in user stories, wat in de product backlog terecht komt. Om de verdeling per sprint wat makkelijker te maken worden de requirements geprioriteerd met behulp van de MoSCoW methode.

7 Technieken en tools

Nadat het project is ingericht moeten er technieken en tools gekozen worden om de applicatie te kunnen maken. Sommige van deze technieken of tools zijn tijdens het project gekozen. Er zal per techniek of tool kort worden besproken waarom het is gekozen en hoe het in het project is geïmplementeerd.

7.1 Visualisatie

Op basis van onder andere de requirements UR06 en UR07 (zie Tabel 4) moet er technieken en/of tools gekozen worden waarmee de visualisatie afgehandeld kan worden. Helaas waren er problemen met het vinden van een goede graph visualisatie library. De keuze die volgde was dat het project met een front-end van Javascript en Jquery gebouwt wordt. Deze keuze wordt beschreven in hoofdstuk 9.4 .

ID	Description	MoSCoW priorisation
UR06	De graph moet gevisualiseerd worden	Must have
UR07	De simulatie moet gevisualiseerd worden	Must have

Tabel 4, uitgelichte requirements van Bijlage E.

CytoscapeJS is de gekozen graph visualisatie library, dit was een belangrijke keuze die wordt uitgelegd in hoofdstuk 9.4. In het project wordt er gebruik gemaakt van verschillende lay-

outs voor CytoscapeJS. Voor wat kleinere graphs kan er gekozen worden voor de Cola lay-out. Dit is een physics based layout, wat ervoor zorgt dat het geen snelle layout is. Dit maakt het lastiger om grotere graphs te visualeren. De andere keuzes is de Cose layout. Deze lay-out is erg snel en kan een grote graph visualiseren. Deze lay-outs en library hebben support voor asynchrone animatie, wat betekent dat het highlighten van edges naar mate het kortste pad algoritme vordert mogelijk is.

Aangezien er al ervaring is met MVC5 is er gekozen om het project op te zetten met behulp van MVC5. Hierdoor kan er een snelle start gegeven worden aan het project. Door een design decision die wordt uitgelegt in hoofdstuk 10.1 wordt de functionaliteit van MVC niet erg veel meer gebruikt. Het fungeert meer als een doorgeefluik voor de backend applicatie die in C# geschreven wordt.

7.2 Backend

De structuur van MVC laat het niet toe om een worker thread te maken, wat nodig is om de game loop oplossing uit te voeren (zie 9.2.2 en 9.3). Om te vermijden dat de structuur van MVC aangepast wordt, is er gekozen om gebruik te maken van een tweede project waar de backend code in staat. Door middel van web sockets (zie 10.1) kan er gecommuniceerd worden tussen de backend code en de front-end. Als library voor web sockets wordt SignalR gebruikt.

7.3 Kwaliteitsbewaking en testen

Om de kwaliteitsbewaking op peil te houden is er een continuous integration omgeving opgezet. Er wordt hiervoor gebruik gemaakt van een interne GitLab server. Continuous integration wordt meestal ingezet om integratie problemen te voorkomen. In extreme programming is het bijvoorbeeld bedoeld om test-driven development te faciliteren. Dit wordt bereikt door alle unit tests te draaien voor de commit naar de main branche. Dit voorkomt dat builds van andere developers falen en forceert het team om rekening te houden met de kwaliteitsbewaking.

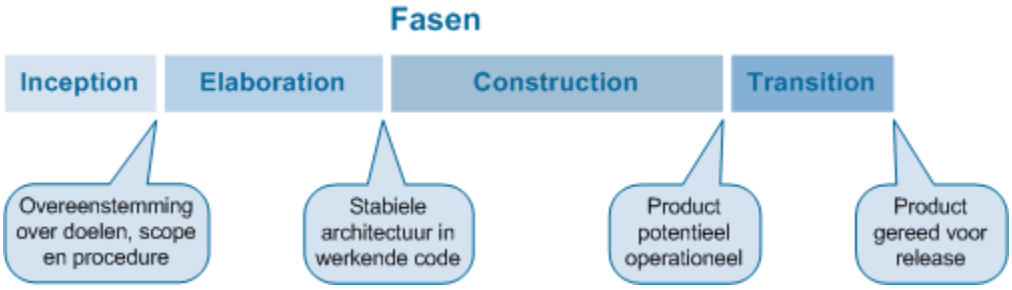
Om de code coverage af te dwingen met behulp van de continuous integration environment is een unit testing framework uitgekozen: NUnit. Om te garanderen dat de unit tests correct zijn worden ze gemaakt op basis van de requirements. De unit tests zelf zijn te vinden in Bijlage F. Zoals eerder gezegd worden deze unit tests automatisch gedraait als er een commit gedaan wordt. Dit zorgt ervoor dat de code quality op peil wordt gehouden, aangezien het

resultaat van alle builds in het bedrijf op een scherm komt te staan bij het koffiezetapparaat. Naast de unit tests en CI omgeving wordt er ook gebruik gemaakt van Opencover en Reportgenerator. Deze libraries zorgen ervoor dat er code coverage berekend kan worden voor de solution, waarna een report gegenereerd wordt door Reportgenerator.

Als laatste wordt er in het oogpunt van kwaliteitsbewaking een applicatie gebruikt genaamd Redmine. Dit is een managementsapplicatie waar de user stories, sprints, sprint backlogs en product backlogs bijgehouden kunnen worden. Elke sprint worden hier de user stories die zijn uitgekozen in de sprint backlog gezet en deze wordt geüpdatet tijdens de sprint. Aan het einde van de sprint zouden alle items ideaal gezien op closed moeten staan, wat wordt beoordeelt met behulp van de definition of done. Als er iets open blijft staan, kan het terug naar de product backlog of naar de volgende sprint. Ook bugs en andere issues worden in deze backlogs gezet.

8 Fasering

Zoals vermeld bij de ontwikkelmethodiekkeuze wordt de fasering van OpenUP aangehouden (zie Figuur 6), wat inhoudt dat het project bestaat uit vier fasen. In deze fasen wordt SCRUM toegepast, hierdoor is er een mooi overzicht van het project, terwijl de beste methodiek gebruikt wordt.



Figuur 6, fasen van UP (Collaris & Dekker, 2016)

8.1 Inception

De inception fase vindt plaats van 14 november t/m 20 november. Het project wordt hier duidelijk gemaakt door middel van het opstellen van requirements en het plan van aanpak.

In deze fase worden de volgende producten opgeleverd:

- Plan van Aanpak
- Keuze simulatie of hardware

- Sprint planning
- Requirements

8.2 Elaboration

Deze fase wordt uitgevoerd in de periode van 21 november t/m 30 december. In deze fase zal er worden gewerkt aan het duidelijk krijgen van de opdracht. Ook zal er een tracer bullet gemaakt worden van de applicatie. Dit houdt in dat alle basis functionaliteit van de applicatie gemaakt zal worden, zodat alle code en functionaliteit traverseable is.

In deze fase worden de volgende producten opgeleverd:

- Keuze graph visualisatie library
- Tracer bullet applicatie
- Drie sprint plannings

8.3 Construction

Deze fase wordt uitgevoerd in de periode van 2 januari t/m 10 maart. In deze fase wordt de focus gelegd op het coderen van alle benodigde functionaliteit van het systeem. De verschillende design keuzes die hieruit komen worden gedocumenteerd.

In deze fase worden de volgende producten opgeleverd:

- Vijf sprint plannings

8.4 Transition

Deze fase wordt uitgevoerd in de periode van 13 t/m 20 maart. De applicatie wordt klaargemaakt voor eventuele overdracht naar een volgend team en er wordt gewerkt aan de scriptie. Ook wordt een eindpresentatie voorbereid voor de werknemers van Sogyo.

In deze fase worden de volgende producten opgeleverd:

- Sprint planning
- Post implementation document eindgebruikers

8.5 Planning

In Tabel 5 staat de sprintplanning en tijdsverdeling in fases van het project. Voor elke sprint staat er aangegeven in welke fase die zich bevindt, erboven staat de verdeling in weken. Zoals te zien is wordt de sprint lengte van twee weken aangehouden.

Tabel 5, tijdsverdeling project

- - - \ Week	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
Fase \ *Sprint*	*1*		*2*		*3*		*4*		*5*		*6*		*7*		*8*		*9*		*10*	
Inception	S1																			
Elaboration			S2		S3		S4													
Constructio n									S5		S6		S7		S8		S9			
Transition																			S10	

Sectie 2

Design en realisatie

In deze sectie van het document zullen de belangrijke design keuzes worden benoemd en toegelicht. Deze keuzes zijn zowel voor als tijdens het project gemaakt en zullen in chronologische volgorde worden besproken. Het hoofdstuk realisatie zal gaan over de implementatie van het product.

9 Design

In dit hoofdstuk zullen de design keuzes behandeld worden en verdere werkzaamheden die verricht zijn. Er zal per design keuze kort worden uitgelegd wat het probleemdomein is, waarna de verschillende keuzes behandeld zullen worden. Als laatste wordt de keuze die gemaakt is verdedigd en wordt er kort teruggegrepen op het probleemdomein.

9.1 Simulatie of hardware

Eén van de eerste design keuzes ging over de vraag of het project gesimuleerd zou worden, of dat er met behulp van hardware een daadwerkelijk prototype zou worden gebouwd voor het project. Dit is een erg belangrijke keuze, omdat het de gehele vormgeving van het project beïnvloedt. Deze keuze vloeide voort uit de business requirements die gaan over de instelbaarheid, het abstractieniveau en het doel van het product. Andere requirements die van belang zijn gaan over het realismeniveau van de simulatie en wat specifiekere requirements over instelbaarheid. Deze requirements zijn te vinden in Tabel 6.

ID	Description
BR01	Er moet een product gemaakt worden dat als benchmark gebruikt kan worden om het concept ‘parallelle collaboratie’ te bewijzen
BR02	Het product moet voldoende instelbaar zijn
BR03	Het product moet andere implementaties van analyse algoritmes kunnen ondersteunen
BR04	Het product moet zo veel mogelijk op de realiteit lijken, zodat er een goede vergelijking getrokken kan worden met de realiteit
SR01	Packet loss moet controleerbaar zijn
SR02	Latency moet controleerbaar zijn
SR03	Individuele device snelheid moet controleerbaar zijn

Tabel 6, uitgelichte requirements van Bijlage F

9.1.1 Hardware

Het project doen met behulp van hardware zorgt voor een reëlere werking van het systeem, wat overeenkomt met requirement BR04 in Tabel 6. Als er met behulp van hardware gewerkt gaat worden, zal de hardware keuze vallen op de Raspberry Pi. Dit is een kleine, goedkope computer waar de software op geïnstalleerd kan worden. Hierdoor kan er een ad-hoc netwerk opgezet worden waardoor de Raspberries met elkaar kunnen communiceren. Op deze manier is de oplossing erg gericht op de real-world, waarbij de enige limiterende factor de

kwaliteit van de hardware is: in de real-world applicatie zou geen gebruik worden gemaakt van Raspberry Pis, maar van betere hardware. Het lastige van de hardware oplossing is dat het erg lastig is om een goede test op te kunnen zetten met de applicatie die opgezet wordt. Om packet-loss te simuleren moet er gedacht worden aan fysiek een kabeltje eruit trekken, of er moet tussen de netwerk stack en de applicatie geprogrammeerd worden. Voor de rest is de snelheid ook lastiger te controleren, aangezien er echte apparaten worden gebruikt. Bij deze nadelen komt nog kijken dat er ook nog real-life problemen kunnen optreden met de devices. Er kan bijvoorbeeld een device moeite hebben met het netwerk, terwijl het op dat moment niet gewenst is. Door al deze factoren wordt de applicatie lastig om te gebruiken, omdat er geen goede controle over is. Ook hangen er kosten en andere limitaties aan de hardware oplossing. Er zou bijvoorbeeld gekozen moeten worden voor een van de ondersteunde talen en de scope zou vergroot moeten worden, omdat een ad-hoc netwerk opgezet zou moeten worden.

9.1.2 Simulatie

Omdat er grote nadelen zijn bij de implementatie van de eerdergenoemde hardware oplossing is er nagedacht over een tweede mogelijkheid: simulatie. Met behulp van een simulatie wordt het veel makkelijker om controle te hebben over de tool, wat overeenkomt met de requirements SR01, SR02 en SR03 die te vinden zijn in Tabel 6.

Het voordeel van een simulatie opzetten is dat de kosten voor de Raspberry Pis, de levertijd en dergelijke zaken die nodig zijn voor de hardware oplossing niet in de weg zitten van het project. Er kan meteen worden begonnen en de ontwikkelomgeving kan geheel volgens de wensen van de programmeur worden ingedeeld. Het grootste voordeel is echter dat er veel controle is over een simulatie. Het doet precies wat het gedesigned is om te doen. Dat wil zeggen dat er allemaal instellingen gemaakt kunnen worden die controleerbaar zijn, zoals packet-loss, latency en device snelheid (zie Tabel 6). Met de hardware optie is dit veel moeilijker te beïnvloeden en de network laag kan bijvoorbeeld lastig gecontroleerd getest worden. Met deze instellingen en de invloed die je hebt over de simulatie kunnen er goede metrieken en meetgegevens gegenereerd worden om te bewijzen dat het systeem werkt. Een nadeel aan het simuleren van het grid netwerk is dat de real-time problemen zoals beschreven in de hardware oplossing er hier niet zijn. Dit zorgt ervoor dat het lastig is om te beslissen wat voor problemen er in de simulatie worden nagebootst. In een ideale situatie zou alles worden nagebootst, maar dat is niet haalbaar, dus moet er goed nagedacht worden over scope.

9.1.3 Conclusie

Alhoewel de hardware oplossing het meest reële beeld geeft van de doelstelling wordt er gekozen voor de simulatie. Dit is gedaan om het project in scope te houden en zodat de onbetrouwbare kant van het netwerk goed getest kan worden door middel van een instelbare netwerk layer in de simulatie. Hierdoor worden ook de kosten teruggedraaid en kan de ontwikkelomgeving vrij worden ingericht.

Nadat er is gekozen voor de simulatie, werden er basis requirements opgesteld op basis van deze keuze. De belangrijkste requirements zijn hierbij UR01, UR02 en UR03 die in Tabel 7 te vinden zijn. Deze requirements gaan in op de basis van het project: het simuleren van de devices en de collaboratie tussen de devices. De laatste requirement, UR04, is ietwat minder belangrijk. Het breadth-first-search algoritme is zeker relevant, maar het is een specificatie van een probleem dat beschouwd wordt als generiek voor dit project. Dit wil zeggen dat deze implementatie heel goed iets anders zou kunnen zijn, hoewel het project gescoped is op graphs.

ID	Description	MoSCoW priorisation
UR01	Een grid computing network moet gesimuleerd worden	Must have
UR02	Nodes van een grid computing network (genoemd devices) moeten gesimuleerd worden	Must have
UR03	De devices van de simulatie moeten samen kunnen werken	Must have
UR04	De devices van de simulatie moeten een graph kunnen doorzoeken door middel van een breadth-first-search algoritme	Must have

Tabel 7, basis requirements volgend uit design decision, Bijlage F

9.2 Multi-thread vs. game loop

Nadat de keuze gemaakt was om een simulatie te maken moest er bedacht worden hoe deze in elkaar zou zitten. De belangrijkste requirements waren bij deze keuze de business requirements over realisme en parallelle collaboratie (Tabel 8). Er zijn uiteindelijk twee ideeën uitgekomen, beide ideeën houden de genoemde requirements in het oog, waardoor er gekeken moet worden naar andere aspecten om een keuze te kunnen maken. De belangrijkste aspecten bij deze beslissingen zijn scope en de moeilijkheidsgraad van de oplossing. De twee verschillen dan ook qua scope en moeilijkheidsgraad. Het eerste idee was gefocust op het zo realistisch mogelijk houden, door de core van de applicatie multi-threaded te maken. Door elk device een eigen thread te geven wordt de applicatie zeer realistisch. Deze keuze vergroot de scope en introduceert een hoge moeilijkheidsgraad. Het tweede idee is om sequentieel te werken, in plaats van parallel.

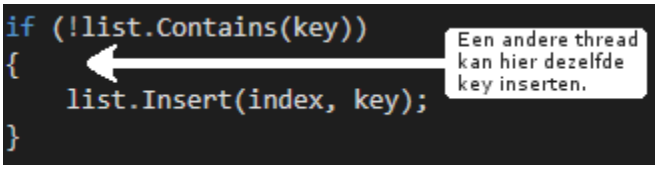
Dit is een meer onrealistische representatie van de werkelijkheid, maar beperkt de scope en moeilijkheidsgraad.

ID	Description
BR01	Er moet een product gemaakt worden dat als benchmark gebruikt kan worden om het concept 'parallelle collaboratie' te bewijzen
BR04	Het product moet zo veel mogelijk op de realiteit lijken, zodat er een goede vergelijking getrokken kan worden met de realiteit

Tabel 8, uitgelichte requirements van Bijlage F

9.2.1 Multithreading

Zoals hierboven vermeld was multithreading het eerste idee dat op de tafel belandde. De basis van het idee was om elk device op een eigen thread te laten draaien. Dit zorgt ervoor dat de simulatie heel erg overeenkomt met de werkelijkheid, aangezien de parallelle collaboratie daadwerkelijk aangetoond wordt door middel van parallelliteit. Er kan namelijk op deze manier gekeken worden naar de devices als aparte objecten die asynchroon werken, wat in de realiteit ook gebeurt.



Figuur 7, voorbeeld van race condition

Het grote nadeel van het werken met multithreading is de moeilijkheidsgraad. De reden hiervoor is dat threads werken met shared resources. Er kunnen bijvoorbeeld race conditions optreden (zie Figuur 7), of andere problemen zoals deadlocks, starvation en livelocks. Als gevolg van de implementatie van multithreading zou de complexiteit van het project omhoogschieten en er moet een goede reden zijn om dit te rechtvaardigen.

9.2.2 Game loop

Het idee om een game loop te gebruiken ontstond uit de discussie over multi-threading. Multi-threading is namelijk geen makkelijke oplossing, de moeilijkheidsgraad ligt hoog en het vergroot hiermee de scope van het project. Het voordeel van de game loop oplossing is dus ook dat de scope verkleind en de moeilijkheidsgraad erg naar beneden gaat. Dit gaat helaas wel ten koste van de waarheidsgetrouwheid van het project. De structuur zelf is een aanpassing van een

veelgebruikt design dat veel in games wordt gebruikt. Met behulp van het game loop design kan parallelle collaboratie worden gesimuleerd, door sequentieel alle devices aan te roepen en daarna een periode te wachten. Hoe deze structuur is opgezet is een van de belangrijkste onderdelen van de architectuur en zal hieronder worden beschreven. Het vormt de core functionaliteit van het programma, waar alles wat betreft de simulatie wordt afgehandeld.

9.2.3 Conclusie

Helaas zijn de nadelen van de grote scope en complexiteit van de multithreading oplossing een te groot obstakel om ervoor te kiezen voor dit project. Hoewel het de werkelijkheid zeer reëel nabootst, moet de vraag gesteld worden wat het praktisch verschil hiervan is. Uiteindelijk is hier de conclusie uit getrokken dat er te weinig praktisch verschil is met andere, sequentiële oplossingen om de complexiteit te rechtvaardigen.

9.3 Uitleg game loop

Zoals te zien in Pseudocode 1 is de core game loop geen moeilijk stuk code. Er zit een simpele while loop in die gedurende de simulatie gedraaid wordt. Als deze while loop één keer doorlopen is, zijn er een aantal dingen gebeurd: alle devices die volgens hun timestep een stap mogen zetten in het algoritme doen dat en eventuele calls tussen de devices worden afgehandeld. Verder wordt de informatie van de verwerkte graph nodes doorgegeven aan de front-end, waar asynchroon de visualisatie wordt afgehandeld. Als laatste wordt er nog gekeken of het algoritme klaar is met het analyseren van de graph, dit zorgt ervoor dat de simulatie zichzelf kan afsluiten door uit de while loop te breken en de thread te beëindigen. Deze reeks van gebeurtenissen wordt vanaf nu aangeduid met de term ‘device CPU cycle’.

In Pseudocode 1 worden deze gebeurtenissen afgehandeld door drie belangrijke method calls, namelijk: ProcessInput(), UpdateComms() en CheckProgress(). UpdateComms() stuurt de lijst met verwerkte graph node ids naar de front-end waar het gevisualiseerd word en zoals eerder gezegd kijkt CheckProgress() naar de progressie in het algoritme en eindigt de simulatie als het nodig is. De functie ProcessInput() is een soort doorgeefluik naar een belangrijke communicatiemethode wat hieronder beschreven zal worden.

```
double cycleCounter  = 0;
WHILE true
      IF status EQUAL stop THEN
            break while loop
      ENDIF

      IF wait THEN
            Continue while loop
      ENDIF

      CALL ProcessInput() with ++cycleCounter RETURNING processedIds
      CALL  UpdateComms() with processedIds
      CALL CheckProgress()
ENDWHILE
END thread
```

Pseudocode 1, versimpeld pseudocode fragment van de main game loop

9.3.1 Time step

De game loop op zichzelf is maar een deel van het core design van de applicatie. Er moet ook rekening gehouden worden met de individuele snelheid van devices en de snelheid van de simulatie. Voor de rest zijn er nog de requirements die een wat lagere priorisering hebben: latency moet kunnen worden ingesteld en de implementatie van een analyse algoritme moet makkelijk uit te breiden zijn. Hoewel het buiten de scope van dit project valt moet er als laatste nog rekening gehouden worden met de mogelijkheid dat devices kunnen uitvallen. De gerefereerde requirements zijn te vinden in Tabel 9.

ID	Description	MoSCoW priorisation
SR04	Devices moeten kunnen uitvallen	Won't have
SR06	Individuele device snelheid moet ingesteld kunnen worden	Must have
SR07	De gebruiker moet de snelheid van de visualisatie kunnen instellen	Must have
SR08	Latency moet door de gebruiker ingesteld kunnen worden op de simulatie	Should have
SR09	De implementatie van een graph analyse algoritme moet makkelijk uit te breiden zijn.	Should have

Tabel 9, uitgelichte requirements van Bijlage F

In Pseudocode 2 is te vinden wat er in het belangrijkste stuk van ProcessInput() gebeurt. Zoals eerder gezegd wordt ProcessInput als doorgeefluik gebruikt om per device de Communicate() methode aan te roepen. Om deze reden is ProcessInput() niet expliciet genoemd in de pseudocode. Er is wel te zien dat er per device de Communicate() functie wordt aangeroepen. De Communicate() methode is geïmplementeerd in de 'device' abstract class, zodat het default behaviour is van de devices. Met behulp van deze abstract class is het erg makkelijk om een andere implementatie aan te leveren, zodat de requirement SR09 voldaan is. In de Communicate() functie wordt allereerst gecheckt of er voor het device in kwestie nodes klaar staan om aan de queue toegevoegd te worden. De nodes die binnen komen in devices mogen namelijk niet meteen in de queue gezet worden, aangezien de latency gesimuleerd moet worden.

```
FOR each device in devices
      FUNCTION Communicate()
      BEGIN FUNCTION with outEdges AND cycle count
            IF OutEdgeQueue count NOT null THEN
                  CALL ResolveLatency()
            ENDIF
            IF timestep * cyclecount passes unique whole number THEN
                  CALL ResolveStep() RETURNING listCheck
            ENDIF
      END FUNCTION Communicate() RETURNING processedIds
END FOR
```

Pseudocode 2, versimpeld pseudocode fragment van de functie ProcessInput()

Na de latency stap wordt er gekeken of een device mag activeren. Om dit uit te leggen wordt er een kleine stap terug gedaan, zodat er een goede schets kan worden gegeven van de parallellisatie van de simulatie.

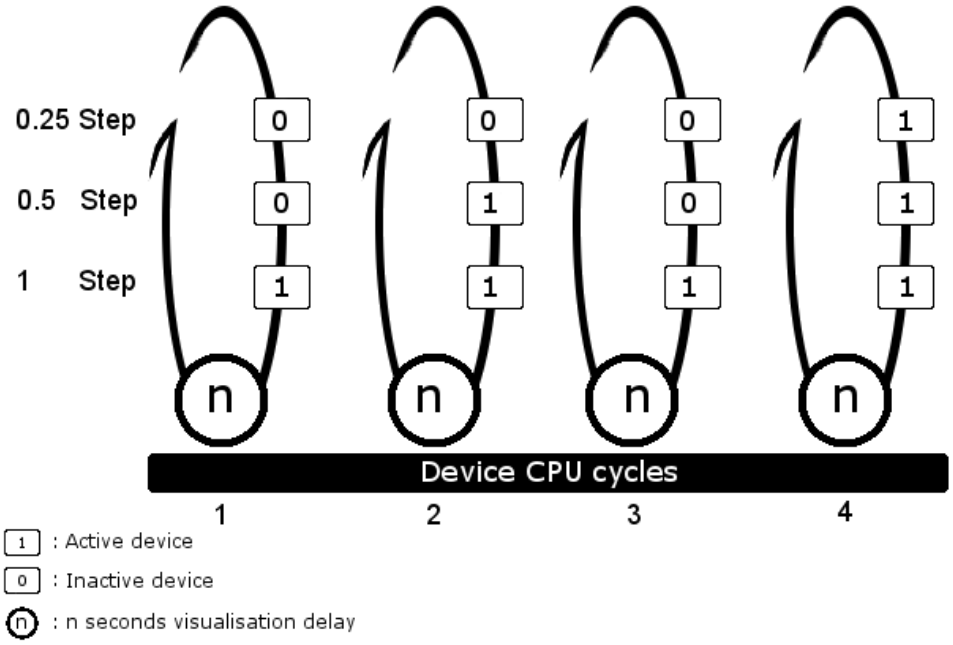
Om de gameloop uit te breiden met de benodigde functionaliteit beschreven in Tabel 9 is het concept timestep bedacht. Het houdt in dat elk device een timestep waarde toegewezen krijgt die altijd tussen de 0 en 1 ligt. Het kan worden vergeleken met de snelheid van het device, in normale processoren zou het in (Giga)Hertz worden weergegeven. Met behulp van deze waarde en de hoeveelheid device CPU cycles die er zijn geweest kan er worden berekend of een device mag activeren. In Pseudocode 2 staat dit aangegeven met 'IF timestep * cyclecount passes unique whole number'. Dit wil zeggen dat er voor elk heel nummer dat gepasseerd wordt het desbetreffende device geactiveerd wordt. Als voorbeeld kan naar een timestep van 0.6 worden

gekeken. Het begint met een berekening van cyclecount (1) * timestep (0.6) = 0.6, wat wil zeggen dat de device niet activeert. Bij een volgende cycle komt de berekening uit op 1.2, waarbij dus een nieuw uniek heel nummer gepasseerd is: de device activeert. Om als visueel voorbeeld te dienen is Tabel 10 opgezet.

Cycles	Timestep	Cycle * Timestep	Activatie
1	0.6	0.6	0
2	0.6	1.2	1
3	0.6	1.8	0
4	0.6	2.4	1
5	0.6	3	1

Tabel 10, timestep activatie uitleg

Nadat er is gedetermineerd dat een device mag activeren, wordt de ResolveStep() methode in het device uitgevoerd. ResolveStep() analyseert de graphs. Dit wil zeggen dat de methode bepaalt welke node als volgende behandeld wordt. Omdat alle devices aangesproken worden per device CPU cycle, kan deze sequentiele code gezien worden als parallel met als eenheid de cycles. Het enige verschil met een truly parallel oplossing zou zijn dat bij truly parallel een call op elk moment binnen kan komen bij elk device, wat kan betekenen dat de nodes op een ietwat andere volgorde behandeld worden. Omdat er niet gezocht wordt voor een kortste pad, maar de hele graph wordt doorlopen maakt dit niet uit. Om een beter beeld te geven van dit hele proces is het gevisualiseerd in Figuur 8, hier is ook meteen de visualisatie snelheid aan toegevoegd, in de vorm van n seconds visualisation delay aan het einde van de device CPU cycle (SR07, Tabel 9).



Figuur 8, uitleg timestep principe

9.4 Graph visualisatie library

Om de benchmarking tool goed te kunnen gebruiken moet de simulatie gevisualiseerd worden. Dit komt overeen met de requirements UR06 en UR07 (zie tabel Tabel 11). Met een visualisatie word bedoelt dat zowel de graph en de progressie van het algoritme in de graph (de simulatie) gevisualiseerd moeten worden.

Omdat graphs visualisatie een project op zich kan zijn is er gekozen om deze functionaliteit af te handelen met behulp van een library. Het is belangrijk dat de library makkelijk te begrijpen is met voldoende documentatie en een redelijke community er achter, deze criteria zorgen er voor dat er geen onnodige tijd wordt besteed aan het leren van een library. Verder moet de library de graphs asynchroon kunnen animeren, wat vooral gebruikt gaat worden voor het highlighten van de edges tijdens de uitvoering van het kortste pad algoritme. Ook moet de library grouping ondersteunen zodat de nodes die onderdeel uitmaken van de verschillende devices gevisualiseerd kunnen worden en moet de library open-source zijn.

ID	Description	MoSCoW priorisation
UR06	De graph moet gevisualiseerd worden	Must have
UR07	De simulatie moet gevisualiseerd worden	Must have

Tabel 11, uitgelichte requirements van Bijlage F

Na een kort field research zijn er twee libraries uit gekomen. Om een goed beeld te kunnen schetsen van de mogelijkheden van de libraries worden er prototypes gemaakt, wat voor praktijkervaring en diepere kennis van de libraries zal zorgen. In deze prototypes zal naar de eerder genoemde criteria gekeken worden en wordt er getracht om deze criteria zo snel, maar zo zorgvuldig mogelijk te evalueren.

Door deze aanpak is het niet mogelijk om meerdere libraries te evalueren, omdat het beschikbare timeframe niet toereikend was. Er zal gepraat worden over de ervaringen met beide libraries en de criteria zullen besproken worden. Als laatste zal er een conclusie worden getrokken uit de verkregen informatie.

9.4.1 Quickgraph / Graph#

Als eerste kandidaat is er een library uit gekomen voor een C# Windows applicatie. Er is veel ervaring in het projectteam met C# Windows applicaties, wat meegenomen wordt als een pluspunt. De library in kwestie is eigenlijk een combinatie van libraries, genaamd: Quickgraph

en Graph#. Deze libraries leverden een redelijk resultaat op, de graph was gevisualiseerd en er kon met de nodes geïnteracteerd worden. Helaas bleek het erg lastig te zijn om dynamisch edges te highlighten om de progressie van het analyse algoritme te visualiseren. Een zeer belangrijk element voor dit project werd ook niet gesupport: nested graphs, ook wel hypergraphs genoemd. Dit is zeer belanrijk voor dit project, omdat het gaat om de parallelle collaboratie tussen devices die een probleem gaan oplossen. Als deze devices niet gevisualiseerd kunnen worden is dat een groot minpunt.

In de prototype fase waren er ook al problemen verrezen door de lage quantiteit van support. Er is vrij weinig goede documentatie te vinden en de community is te klein om veel problemen al opgelost te hebben.

9.4.2 CytoscapeJS

De tweede kandidaat is een Javascript library genaamd CytoscapeJS. Het is een javascript library die Cytoscape Web opgevolgd heeft. De library is zeer geoptimaliseerd en de graphs kunnen geheel ge(de)serialiseerd worden met Json, verder is de animatie loop asynchronous en is er support voor grouping en vele andere features. Ook de community is groot, dit is waarschijnlijk het geval omdat het gebaseerd is op een bekend programma: Cytoscape. Voor de rest is er een grote hoeveelheid documentatie, er zijn voorbeelden te vinden voor bijna elke functie en er zijn verschillende graph layouts die out of the box gesupport worden.

9.4.3 Conclusie

Hoewel het projectteam veel ervaring heeft met C# Winforms applicaties is er toch gekozen voor CytoscapeJS. Zoals te zien in Tabel 12 is de beste keus voor de C# libraries niet toereikend. Vooral het highlighten van edges en grouping waren slecht gesupport of lastig te implementeren.

	Quickgraph / Graph#	**CytoscapeJS**
Documentatie	+-	++
Community	--	+-
Asynchroon	+-	+
Grouping	--	++
Open-source	++	++

Tabel 12, overzicht plus- en minpunten van visualisatie libraries

De community achter deze combinatie van libraries was ook niet groot, waardoor er weinig hulp te vinden was. De javascript libraries waren hier veel beter in, waar vooral CytoscapeJS goed uitpakte. Om toch C# te kunnen gebruiken voor de backend is er gekozen om een MCV omgeving op te zetten met een frontend die bestaat uit een combinatie van Javascript en Jquery, er kan dan met behulp van websockets gecommuniceerd worden met de backend applicatie.

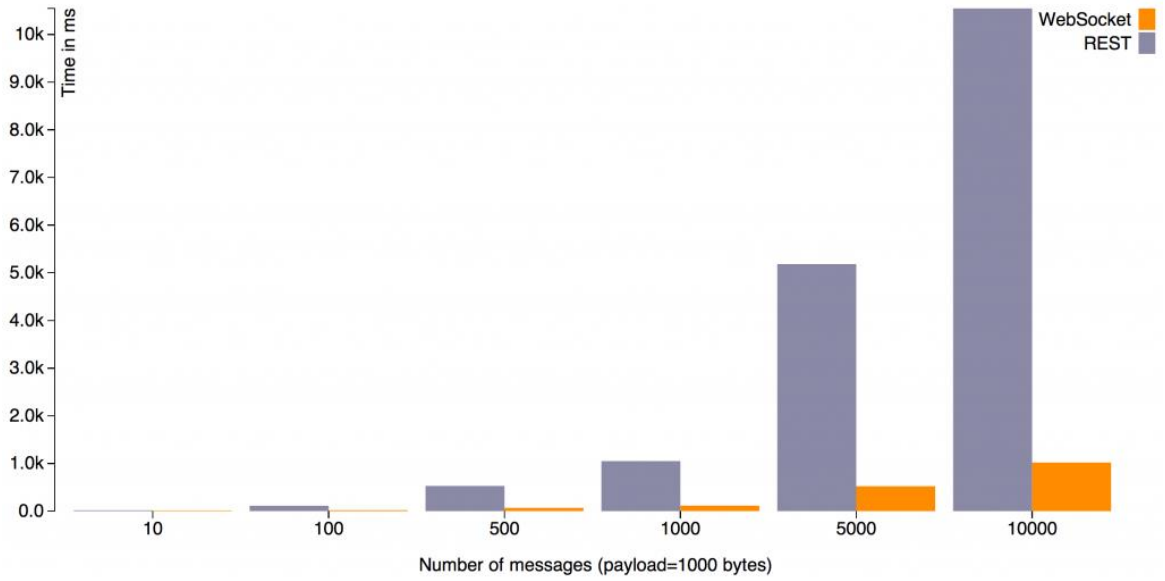
10 Realisatie

10.1 ASP.NET en multiple start up

Door de keuze voor CytoscapeJs in hoofdstuk 9.4 werd er gekozen om het ASP.NET MVC framework te gebruiken voor de front-end. Helaas moest de simulatie op een eigen thread gedraaid worden door de keuze voor de gameloop architecture beschreven in hoofdstuk 9.2. Door de eigen thread werd het lastig om verder met MVC te werken, aangezien MVC alleen multithreading toepast met behulp van async functions. Bij dit project zou het gaan om een meer complexe situatie, waar deze functions niet van waarde zouden zijn. Op dat punt moest er een beslissing worden genomen: opnieuw beginnen met een ander framework dan ASP.NET, of een andere oplossing voor het probleem bedenken. Om niet te veel tijd kwijt te raken is er besloten om niet opnieuw te beginnen, maar in plaats daarvan de code van de simulatie te abstraheren naar een ander soort project.

Omdat er veel bidirectionele communicatie nodig was tussen de server en de client was het idee al naar voren gekomen om met websockets te werken. Een websocket is een communicatie protocol die ‘full-duplex’ werkt. Een full duplex systeem staat communicatie in allebei de richtingen toe, wat gelijktijdig gebeuren mag. Met deze techniek kan je in real-time data overdragen naar, en van de server.

Websocket was niet de enige keuze die er was, er is ook kort nagedacht over een RESTful API. Het probleem met REST is dat het een grote overhead heeft, wat niet gewenst is voor het project. In Figuur 9 is te zien dat REST veel meer tijd in beslag neemt dan WebSockets doen. Hier komen wel allemaal leuke snufjes bij zoals caching, routing, en multiplexing (Arun, 2016). Voor dit project, waar latency belangrijk is, is een RESTful API dus niet geschikt, gezien de hoeveelheid messages die er verstuurd worden.



Figuur 9, de benodigde hoeveelheid tijd om N messages van een constante payload te sturen (Arun, 2016)

Er was dus besloten om met websockets te gaan werken, helaas was er niet genoeg tijd om goed naar andere oplossingen te kijken zoals bijvoorbeeld een dedicated windows service. Hierdoor werd er besloten om het te verplaatsen naar een tweede project, die als een multiple startup project werd aangegeven. Hierdoor was er een tweede 'thread' (namelijk een geheel ander project) verkregen en kon alle communicatie tussen de projecten via SignalR (een websocket library) worden geregeld.

Jammer genoeg betekende dit dat er heel veel overhead was gecreëerd in de vorm van het ASP.NET MVC framework. Het framework wordt op dit punt gebruikt als een verheerlijkt doorgeefluik, maar geeft wel goede support voor uitbreiding op dit project.

10.2 Json (de-)serialisatie

Om de graph te kunnen inladen zijn er twee elementen die hier besproken worden: de Json (de-)serialisatie en de opdeelmethode van de graph naar hypergraphs. Hierbij wordt gewerkt met Jason-bestanden en de backend representatie van de graph. Om deze data te krijgen wordt er verwacht dat een Json-bestand wordt ingeladen in de front-end.

Om van de Json-file naar een backend structuur met objecten te komen moest er een tussenobject komen. Dit is gedaan zodat de default behaviour van Json.NET kan worden gebruikt voor de deserialisatie. Het tussenobject, genaamd 'RootObject', bevat alle Json-elementen die voor kunnen komen in de file. De file wordt direct gedeserialiseerd naar een lijst van deze

objecten. Als deze objecten goed zijn geserialiseerd is er nog een probleem, er is namelijk geen structuur in deze elementen die overeenkomt met een graph.

Hiervoor is de ‘CastToGraph()’ methode voor bedacht. Deze methode kijkt naar elke instantie van het RootObject in de lijst die geleverd wordt en parsed dit naar de juiste backend structuur. Hier worden de edges ook meteen toegevoegd. Ook wordt er bijgehouden hoeveel nodes en edges er zijn geserialiseerd zodat dit resultaat op de output window kan worden getoond.

Om de graph te visualiseren moet er een Json-file terug gestuurd worden naar de front-end. In plaats van de Json-file in te laden wordt de backend objecten structuur geserialiseerd naar Json, zodat het 100% overeenkomt. Ook worden bij deze serialisatie wat dingen toegevoegd, zoals een CompoundLabel. De CompoundLabel wil zeggen dat de timestep van het device en de device naam samen worden gevoegd en als een extra Json-element worden meegestuurd. Op deze manier kan er meer dan alleen de naam aan de devices worden meegegeven voor de visualisatie.

Om deze serialisatie goed te laten verlopen moet er een custom implementatie van de serialiser van Json.NET worden geschreven. Dit komt door de complexiteit van de objecten die geserialiseerd moeten worden. De objecten zijn bijvoorbeeld naar aanleiding van de graph requirements GR02 en GR04 (zie Tabel 13), circulair met elkaar verbonden, wat voor problemen zorgt bij de default behaviour van Json.NET.

ID	Description	MoSCoW priorisation
GR02	In de graph mogen nodes met het type System alleen een edge hebben naar nodes met het type Hypothesis	Must have
GR04	In de graph mogen nodes met het type Hypothesis edges hebben naar nodes met de types System, Hypothesis en Evidence	Must have

Tabel 13, uitgelichte requirements van Bijlage F

Voor de serialisatie van de structuur van de backend naar Json-formaat wordt eerst het volledige data object gedeserialiseerd met de default serialiser. Deze data staat in standaard Json formaat en is makkelijk te serialiseren. Vanuit dit startpunt worden de ‘group’, ‘grabbable’ en ‘marked’ Json-elementen geserialiseerd. Deze elementen worden bij elkaar gevoegd en er wordt gekeken naar het type node. Als het een CompoundNode is wil het zeggen dat het de visualisatie is van het device in de front-end. Hierbij hoort wat meer informatie zoals hierboven is

beschreven: de CompoundLabel. Als dit gebeurd is worden als laatste nog de verbindingen tussen de nodes geserialiseerd. Voor elke node wordt er gekeken naar de buren. Als er buren zijn wordt hier een edge voor gemaakt in de vorm van een totaal nieuw Json-object.

De functionaliteiten die hier beschreven staan faciliteren het inladen en visualiseren van de graph. Ook wordt er een backend structuur van objecten aangemaakt waar de simulatie gebruik van maakt voor de analyse.

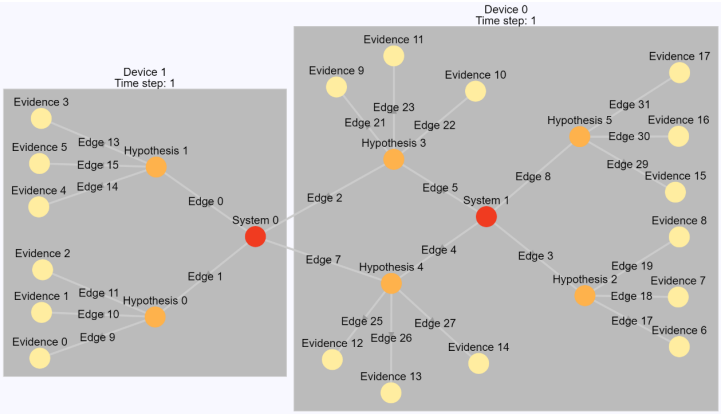
10.3 De graph

Om een goede representatie van de werkelijkheid te krijgen zijn er wat eisen gesteld aan de structuur van de graph, deze eisen zijn te vinden in Tabel 14. Om een visualisatie hiervan te geven kan gekeken worden naar Figuur 10 en Figuur 11. In Figuur 10 is een graph te vinden zoals het in de applicatie ook te vinden kan zijn, het is een realistisch beeld, alhoewel het een erg kleine graph is. Om de graaf beter uit te kunnen leggen is Figuur 11 (Bijlage B) versimpeld. De evidence nodes zijn ter bevordering van de leesbaarheid bijvoorbeeld weggelaten.

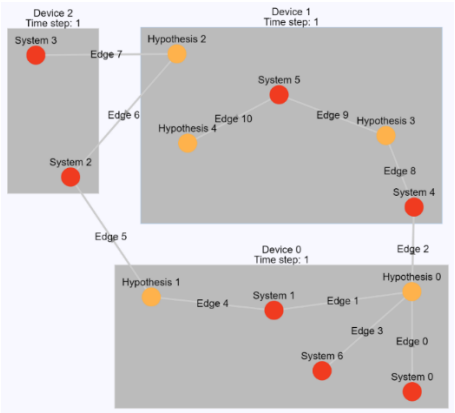
ID	Description	MoSCoW priorisation
GR01	De graph moet nodes hebben met de types System, Hypothesis en Evidence	Must have
GR02	In de graph mogen nodes met het type System alleen een edge hebben naar nodes met het type Hypothesis	Must have
GR03	In de graph mogen nodes met het type Evidence geen edges hebben naar andere nodes.	Must have
GR04	In de graph mogen nodes met het type Hypothesis edges hebben naar nodes met de types System, Hypothesis en Evidence	Must have

Tabel 14, uitgelichte graph requirements van Bijlage F

Zoals aangegeven in de requirements in Tabel 14 zijn er drie soorten nodes: Evidence, Hypothesis en System nodes. De System nodes kunnen alleen een link hebben naar Hypothesis nodes, terwijl Hypothesis nodes links kunnen hebben naar zowel System, andere Hypothesis en Evidence nodes. Het laatste type node, Evidence, heeft geen links, behalve een link vanuit Hypothesis.



Figuur 10, vergrote versie in Bijlage A



Figuur 11, vergrote versie in Bijlage B

10.4 Visualisatie

Om de visualisatie en de werking van het algoritme duidelijk te krijgen kan er gekeken worden naar Figuur 13 en Figuur 12. In deze bijlage is het algoritme en de volgorde van het algoritme afgebeeld. Er wordt aangenomen dat de node Hypothesis 0 de begin node is, waar het algoritme start. Verder wordt er aangenomen dat er geen latency tussen de devices is, ook wordt de timestep buiten beeld gehouden om de uitleg in scope te houden. Als laatste zijn de evidence nodes, zoals eerder, niet aanwezig in de graph.

Het algoritme werkt met een First-In First-Out abstract data type, de Queue. Elke keer als een device een stap in het algoritme mag zetten en er een node in de Queue staat kijkt het algoritme of deze node buren heeft. Als er een buur gevonden wordt, wordt deze toegevoegd aan de Queue en dit gaat zo door, totdat er geen buren meer zijn die nog niet zijn toegevoegd aan de Queue. Ook checked het algoritme of er devices zijn die een node aan het broadcasten zijn. Dit wil zeggen dat er een outgoing edge is die naar een ander device wijst, het device waar de edge van oorsprong vandaan komt vraagt dan aan de andere devices of ze deze node in zich hebben. Op deze manier wordt de graph langzamerhand doorlopen. Met grotere graphs zullen er steeds meer devices tegelijkertijd active zijn, wat de parallelle collaboratie significeert.

In Figuur 12 zijn de draaiboeken van het algoritme te vinden. Deze komen overeen met de nummering van Figuur 13. De tabel geeft per device CPU cycle aan of de device een stap zet in het algoritme, als dat zo is wordt het aangegeven in de kolommen ‘van’ en ‘naar’. De devices kunnen ook idle zijn door een gebrek aan workload, in dat geval zal er N/A staan in de respectievelijke kolom.

Device CPU cycle	Van	Naar
1	Hypothesis 0	System 6
2	Hypothesis 0	System 0
3	Hypothesis 0	System 1
4	Hypothesis 0	(out edge) System 4
5	System 1	Hypothesis 1
6	Hypothesis 1	(out edge) System 2
7	N/A	N/A

Tabel 5, draaiboek device 1

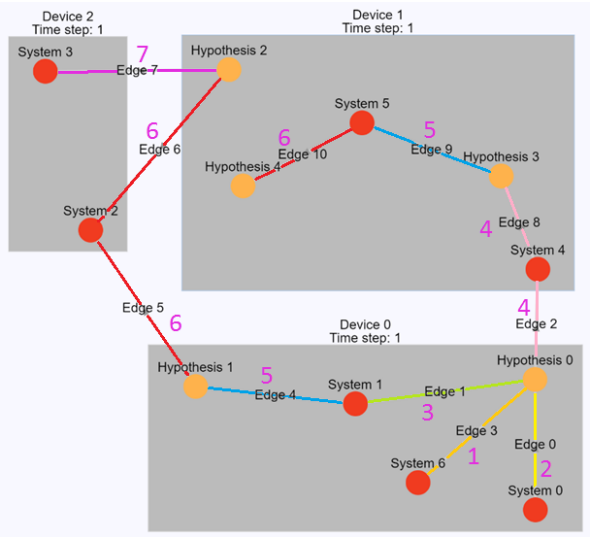
Device CPU cycle	Van	Naar
1	N/A	N/A
2	N/A	N/A
3	N/A	N/A
4	(in edge) System 4	Hypothesis 3
5	Hypothesis 3	System 5
6	System 5	Hypothesis 4
7	(in edge) Hypothesis 2	(out edge) System 3

Tabel 6, draaiboek device 2

Device CPU cycle	Van	Naar
1	N/A	N/A
2	N/A	N/A
3	N/A	N/A
4	N/A	N/A
5	N/A	N/A
6	(in edge) System 2	(out edge) Hypothesis 2
7	(in edge) System 3	N/A

Tabel 7, draaiboek device 3

Figuur 12, vergrote versie in Bijlage E



Figuur 13, vergrote versie in Bijlage C

10.5 Graph verdeelalgoritme

Om parallelle collaboratie te kunnen bewijzen moeten de graphs opgedeeld kunnen worden over devices. Dit is terug te vinden in de requirements GR06 en GR08, zie Tabel 15. Dit is niet makkelijk om te doen, aangezien er van de grote graph een aantal hypergraphs gemaakt moet worden. Dit wil zeggen dat er in de graph een x aantal kleinere graphs gevonden moet worden. Hierbij moeten zo weinig mogelijk connecties tussen de devices van de graphs zitten, zodat er zo weinig mogelijk netwerkverkeer is, wat het traagste onderdeel is. Deze onderdelen zijn ook terug te vinden in de requirements GR09 en GR10.

ID	Description	MoSCoW priorisation
GR06	Graphs moeten verdeeld kunnen worden in hypergraphs	Must have
GR07	Het aantal hypergraphs waar de graph in verdeeld word moet in te stellen zijn door de gebruiker	Should have
GR08	De hypergraphs moeten worden verdeeld over de beschikbare devices	Must have
GR09	De structuur en verdeling van de nodes in de hypergraphs moet overeenkomen met de structuur van de volledige graph	Could have
GR10	De graph nodes moeten zo gelijkmatig mogelijk over de devices verdeeld worden, met zo min mogelijk edges tussen de devices	Won't have

Tabel 15, uitlichting van requirements van Bijlage F

Omdat het verdelen op deze manier erg lastig is, moest er in verband met de scope gekozen worden om een naïef verdeelalgoritme te maken. Omdat er al een breadth-first-search algoritme in de applicatie was, is dit algoritme genomen als de basis voor het verdeelalgoritme. Breadth-first-search is geen goede keuze voor een verdeelalgoritme, aangezien er geen rekening gehouden wordt met de edges tussen de devices. Dit zorgt ervoor dat de graaf alleen verdeeld is op device niveau. Een ander negatief bijeffect is dat er weinig connecting nodes overblijven voor het laatste device. In Figuur 14 is te zien dat er in device 0 en 1 veel hypotheses zijn, terwijl er geen overblijven voor device 3.

Figuur 14, vergrote versie in Bijlage D

Sectie 3

Evaluatie en terugblik

In deze sectie van het document zullen de keuzes die gemaakt zijn in het project en het proces dat doorlopen is worden geëvalueerd en wordt er een terugblik geworpen op het project. Verder zal eventueel vervolgwerk besproken worden.

11 Evaluatie

11.1 Afwijkingen van afstudeerplan

Er is afgeweken van het afstudeerplan, omdat ik het project in eerste instantie verkeerd geïnterpreteerd had. Het was namelijk niet de bedoeling dat de focus van het project lag op het analyseren van de graph, dit was alleen maar een middel om te bewijzen dat de onderliggende structuur werkt. Het echte doel van het project was namelijk het maken van een benchmarking tool dat kan worden gebruikt om te bewijzen dat het concept parallelle collaboratie werkt. De focus van het project is dus verschoven van analyse naar het proces dat doorlopen wordt om deze analyse te doen. Dit had gevolgen op het project waarbij gedacht kan worden aan vertraging en het herschrijven van een behoorlijk stuk functionaliteit.

11.2 Evaluatie aanpak

Bij deze evaluatie ga ik kijken naar de aanpak die ik heb gehanteerd tijdens de afstudeerperiode. Ik kijk hierbij naar de gehanteerde ontwikkelmethodiek en het proces dat ik doorlopen heb in deze periode.

De gehanteerde ontwikkelmethodiek zoals beschreven in hoofdstuk 6 is over het algemeen goed verlopen. De sprints van Scrum in combinatie met de fase-indeling van RUP zorgde voor een helder proces, hoewel de betekenis van deze indeling verwaterde naar gelang het project vorderde. Er zijn ook milestones opgesteld die een goede mogelijkheid voor reflectie en planning gaven.

Elke twee weken was er een sprint meeting waar de vorige sprint werd besproken, een demo gegeven werd, feedback gegeven werd op de documentatie die ik ingeleverd had en de volgende sprint werd ingedeeld. Hoewel het een stroef begin had, paste ik me aan en deed ik mijn best om de sprint meetings zo goed mogelijk te doen verlopen.

Het stroeve begin van het project kan worden gelinkt aan de complexiteit en de mate van vrijheid die ik in het project kreeg. Om een beter beeld te krijgen van de opdracht moest ik mij meer openstellen en overleg plegen met experts en collega’s. Hoewel ik dit gedaan heb ben ik van mening dat dit een leerpunt voor mij is en dat ik mij beter kan plaatsen in de bedrijfswereld als ik mij nog assertiever opstel.

Ik zette me goed in om de officiële contactmomenten zo goed mogelijk te doen verlopen en ben daarin gegroeid naar mate het project vorderde. Ondanks dat de opdracht complex en uniek was, heb ik door middel van de sprint meetings, overleg met experts op locatie en overleg met collega's de opdracht helder gekregen.

11.3 Evaluatie producten en design choices

Ik ben tevreden over hoe ik het uitzoeken van de benodigde tools heb afgehandeld. Bij bijvoorbeeld het uitzoeken van de library voor het visualiseren van de applicatie heb ik eerst een klein beetje field research gedaan, waaruit een paar libraries kwamen in verschillende talen. Deze libraries heb ik hierna met behulp van kleine prototypes geëvalueerd en ben toen tot de conclusie gekomen dat mijn eerste plan om het in een windows applicatie te maken geen goed idee was. Ik heb hierna mijn project opgesteld met behulp van Asp.NET MVC met het oogpunt op een C# backend. Door dit soort processen te doorlopen ben ik uitgekomen tot een collectie van tools en technieken waar ik achter sta en waarvan ik denk dat ze het project goed faciliteren.

De manier waarop de design decisions zijn gemaakt is goed bevallen. Op basis van de requirements zijn er met behulp van de sprint meetings en overleg met mijn begeleider keuzes gemaakt die goed onderbouwd waren en waarvan de plus- en minpunten overwogen zijn. Er is een aantal keer een beslissing gemaakt die het project op een ander spoor zette, maar dit is gedaan door middel van een weloverwogen keuze.

Over het product zelf ben ik over het algemeen tevreden, hoewel er minder bereikt is dan ik hoopte. Dit komt gedeeltelijk door de verwarring over de focus en implementatie van het project in de eerdere stages. Ik ben deze verwarring overkomen door goed te overleggen in de sprint retrospectives, maar ben van mening dat ik dit beter had kunnen afhandelen. Als ik bijvoorbeeld meer initiatief had getoond en meer was gaan zitten met de product owner kon ik het project eerder helder krijgen, waardoor er meer gedaan kon zijn. Door deze vertraging is er bijvoorbeeld geen metrieken generatie gedaan, wat ik zelf erg jammer vind. Dit had mijn project veel meer waarde gegeven.

11.4 Verantwoording beroepstaken

In dit hoofdstuk worden de beroepstaken die gekozen zijn in het afstudeerplan verantwoord en wordt er kort uitgelegd wat er gedaan is om deze te halen. Ook de algemene vaardigheden beroepstaken vanuit het afstudeerplan zijn behandeld.

Beroepstaak 1.4: Uitvoeren analyse door definitie van requirements

Ik heb voor requirements verschillende interviews gehouden met de product owner en heb de requirements opgesteld aan de hand van deze gesprekken. Dit is in de sprint retrospectives besproken en aangepast indien nodig. De requirements zijn hierna geprioriteerd met behulp van de MoSCoW methode en op basis van design decisions in de eerdere fasen van het project zijn er requirements toegevoegd. Om het product goed te krijgen is het requirements document levend gehouden en is geëvolueerd tijdens het proces dat doorlopen is.

Beroepstaak 3.1: Ontwerpen Software Architectuur

De front-end van de applicatie is gebouwd met een combinatie van MVC.NET, JQuery, Javascript en Cytoscape.JS. De backend is gebouwd met behulp van C#. De combinatie tussen de front-end en de backend wordt gefaciliteerd door websockets, gebouwd met behulp van SignalR. Om de simulatie goed te laten verlopen wordt het op een aparte thread in de backend gedraaid. Om de applicatie goed aanpasbaar te maken is de graph analyse kant van de applicatie generiek gemaakt, met behulp van abstract classes met wat default behaviour. Op deze manier kan er gemakkelijk een tweede implementatie van de graph analyse functionaliteit ingebouwd worden. Dit is ook gedaan met het graph verdelingsalgoritme, zodat er gemakkelijk in de toekomst uitgebreid kan worden van het naïeve algoritme wat er nu in zit tot een slim algoritme.

Beroepstaak 3.2: Ontwerpen systeemdeel

Er is getracht om met behulp van een aangepaste game loop pattern de simulatie levensecht te doen lijken. Voor de rest is het concept timestep dat in 9.3.1 wordt behandeld toegepast zodat er een andere snelheid gegeven kan worden aan de devices. Er is een abstractieniveau in de applicatie gebouwd tussen de simulatie en het analyse deel zodat de aanpasbaarheid bevorderd wordt.

Beroepstaak 3.3: Bouwen applicatie

Er is (de-)serialisatie functionaliteit gemaakt zodat een JSON file vanuit de front-end kan worden ingeladen en worden geparsed naar de structuur van de backend. Ook is een naïef graph verdeelalgoritme gebouwd, dit zorgt er voor dat een grote graph opgedeeld kan worden in een aantal hypergraphs. Deze hypergraphs kunnen daarna worden verdeeld over de verschillende nodes van het gesimuleerde grid computing netwerk. Voor de rest is er aan de front-end hard gewerkt om een mooie representatie te houden van de graphs en van de vooruitgang van het algoritme.

Beroepstaak 3.5: Uitvoeren van en rapporteren over het testproces

Om de code coverage hoog te houden is er een continuous integration omgeving opgezet waarbij unit tests automatisch voor elke build uitgevoerd worden. Als de tests falen, faalt de build. Hierdoor wordt de kwaliteitsbewaking en code coverage in de gaten gehouden. De unit tests vinden hun oorsprong in de requirements. Het testing framework dat gebruikt wordt is Nunit in combinatie met Opencover en Reportgenerator. Op deze manier kan de code coverage worden berekend en worden gerapporteerd. Deze uitslagen komen op een build monitor te staan bij het koffiezet apparaat, waardoor er een motivatie is om de builds zo goed mogelijk te testen en draaiend te houden.

Beroepstaak AV 2.2: Zelfstandig werken

Een grote focus van Sogyo ligt op zelfstandig werken. Er lopen veel mensen rond met vele gebieden van expertise, maar hier moet zelf om gevraagd worden. Door middel van praten met deze mensen en het constante toezicht houden op de scope van het project is er sprake van een goede mate van zelfstandigheid.

Beroepstaak AV 2.3: Resultaatgericht werken

Er is gewerkt met sprints van twee weken. Aan het eind van deze twee weken wordt er besproken wat er die sprint is gedaan en wordt een demo gegeven van de functionaliteit die gebouwd is. Er wordt hier dan over gediscussieerd en de volgende sprint wordt ingedeeld. Ook wordt er besproken wat voor werk er aan de scriptie is gedaan en wordt er feedback gegeven op het verrichte werk.

Beroepstaak AV 2.5: Creëren en innoveren

Het is een nieuw systeem, dat het concept parallelle collaboratie aanpakt. Dit is nog niet eerder gedaan, hoewel er al wel kennis is van simpele opdrachten sturen naar een grid computing netwerk. Hierdoor moest er creatief denkvermogen ingezet worden en zijn er creatieve oplossingen voor complexe problemen bedacht.

12 Vervolgwerk

Omdat het een complex project is, en een gelimiteerde scope zijn aangehouden is er nog veel vervolgwerk te bedenken. Er kan bijvoorbeeld gebruik worden gemaakt van een threadpool, hierdoor zou een gelimiteerd aantal gebruikers tegelijkertijd met elkaar gebruik kunnen maken van de applicatie, wat op dit moment niet het geval is.

Ook moet er gekeken worden naar het graph verdelingsalgoritme. Het algoritme is op dit punt erg naïef en moet uitgebreid worden zodat er zo weinig mogelijk edges zitten tussen de devices. Als dit gedaan wordt kan er daadwerkelijk gesproken worden over goede hypergraphs die in de devices zitten. Dit wil zeggen dat de oplossing veel effectiever is en veel sneller zal zijn in het oplossen van de graphs. Wel moet rekening gehouden worden met het feit dat het wellicht de parallelliteit aantast van de applicatie. Als er namelijk wordt begonnen bij één node en er is maar één edge tussen de twee devices, zou het wellicht de laatste node zijn die traversed wordt door het analyse algoritme. Als dit gebeurt is er geen sprake meer van parallelliteit.

Een belangrijk onderdeel dat er op dit punt niet inzit is het gebruik van redundantie om het eventuele uitvallen van devices op te vangen. Als er redundantie gemaakt wordt in de devices kan er worden geëvalueerd of de graph inderdaad correct opgelost wordt.

Als laatste wil ik aandragen om metrieken te kunnen genereren en te visualiseren op de front-end. Dit zou een enorme bijdrage leveren aan de bruikbaarheid van het product en kan het goed gebruikt worden om over de simulatie te rapporteren.

References

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002, November). SETI@home: An experiment in Public-Resource Computing. *COMMUNICATIONS OF THE ACM*, pp. 56-61.

Arun, G. (2016, December 25). *REST vs WebSocket Comparison and Benchmarks*. Retrieved from Arun Upta: http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/

Baker, M., & Buyya, R. (1999). Cluster Computing: The Commodity Supercomputer. In R. Buyya, K. Cooper, R. Jones, A. Poggi, S. Srirama, & N. Horspool, *Software: Practice and Experience* (pp. 551-576). John Wiley & Sons Ltd.

Collaris, R.-A., & Dekker, E. (2016, December). *Rup op Maat <naslagsite>.* Retrieved from Rup op Maat: http://www.rupopmaat.nl/naslagsite2011/index.html?content=fasen/

Foster, I. (2004, April). Father of the Grid. (A. M. Braverman, Interviewer)

GFLewis. (2016, 12 3). *Wikimedia Commons.* Retrieved from Wikimedia: https://en.wikipedia.org/wiki/OpenUP#/media/File:Openup-basic_lifecycle.jpg

Gustafsson, B. (2008). OpenUP - The Best of Two Worlds. *Methods & Tools, Practical knowledge for the software developer, tester and project manager, 16*(1), 21-32.

IBM. (2002-2003). IBM Solutions Grid for Business PartnersHelping IBM Business Partners to Grid-enable applications for the next phase of e-business on demand. United States of America.

Kruchten, P. (2004). *The Rational Unified Process: An Introduction.* Boston, MA: Addison-Wesley.

Lakeworks. (2016, 11 30). *Wikimedia Commons.* Retrieved from Wikimedia: https://upload.wikimedia.org/wikipedia/commons/5/58/Scrum_process.svg

Lemke, T. (2013, May 8). *NSA Breaks Ground on Massive Computing Center*. Retrieved from Patch.com: http://patch.com/maryland/odenton/nsa-breaks-ground-on-massive-computing-center

Oles, O. K. (2016, September 23). Retrieved from Twitter: https://twitter.com/olesovhcom/status/779297257199964160

Paul. (2014, August 25). *The History of the Multi Core Processor - News - BurnWorld*. Retrieved from BurnWorld.com: www.burnworld.com/the-history-of-the-multi-core-processor/

Smith, P., & Kemp, P. (n.d.). *Wikimedia Commons.* Retrieved from Wikimedia: https://commons.wikimedia.org/wiki/File:Waterfall_model.svg

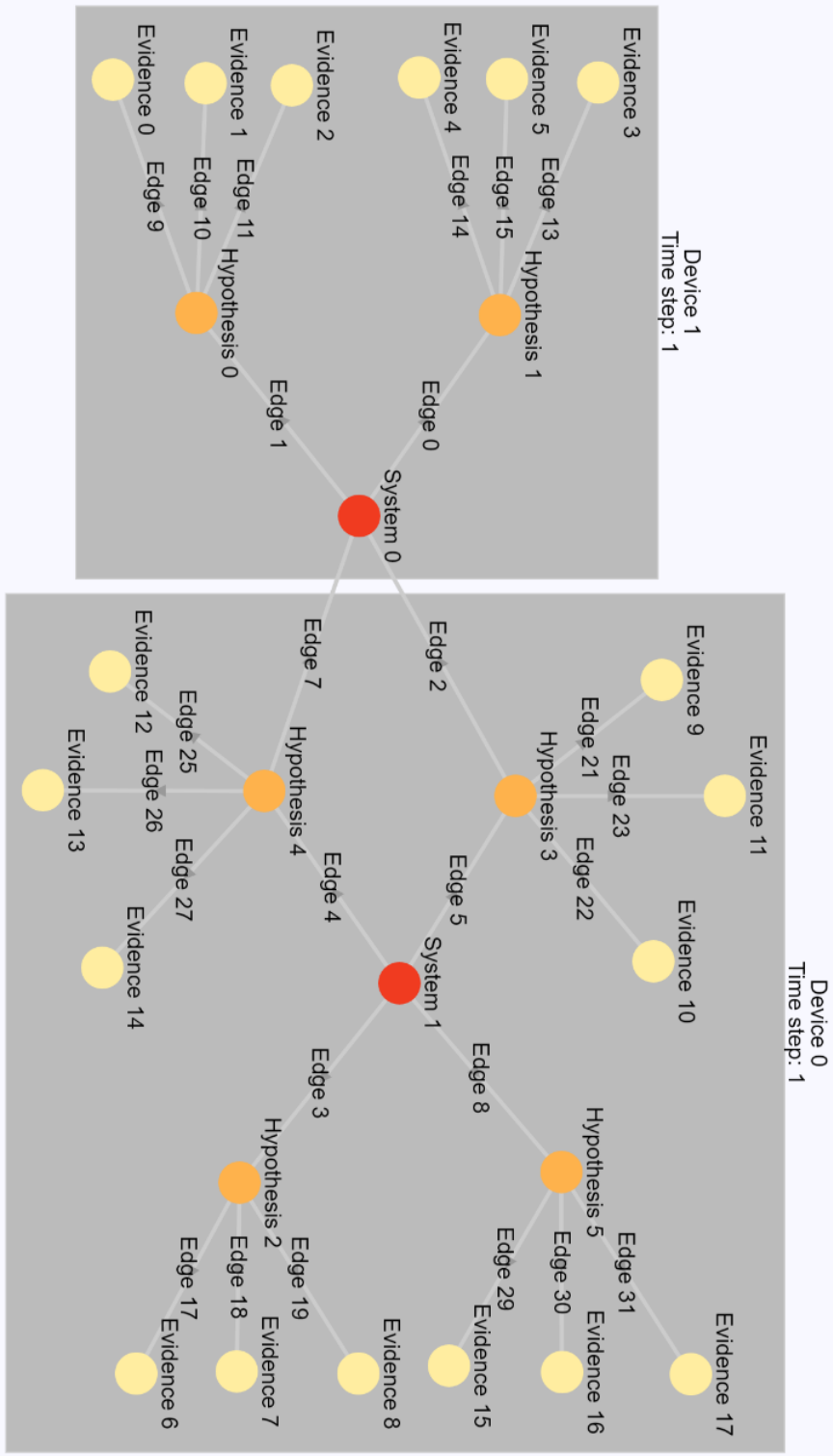
Toffler, A. (1991). *Powershift: Knowledge, Wealth, and Violence at the Edge of the 21st Century.* London, United Kingdom: Bantam.

Wang, F. Z., & Udoh, E. (2009). *Handbook of Research on Grid Technologies and Utility Computing: Concepts for Managing Large-Scale Applications.* Hershey, Pennsylvania: Information Science Reference.

Wikipedia. (n.d.). *Supercomputer - Wikipedia*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Supercomputer

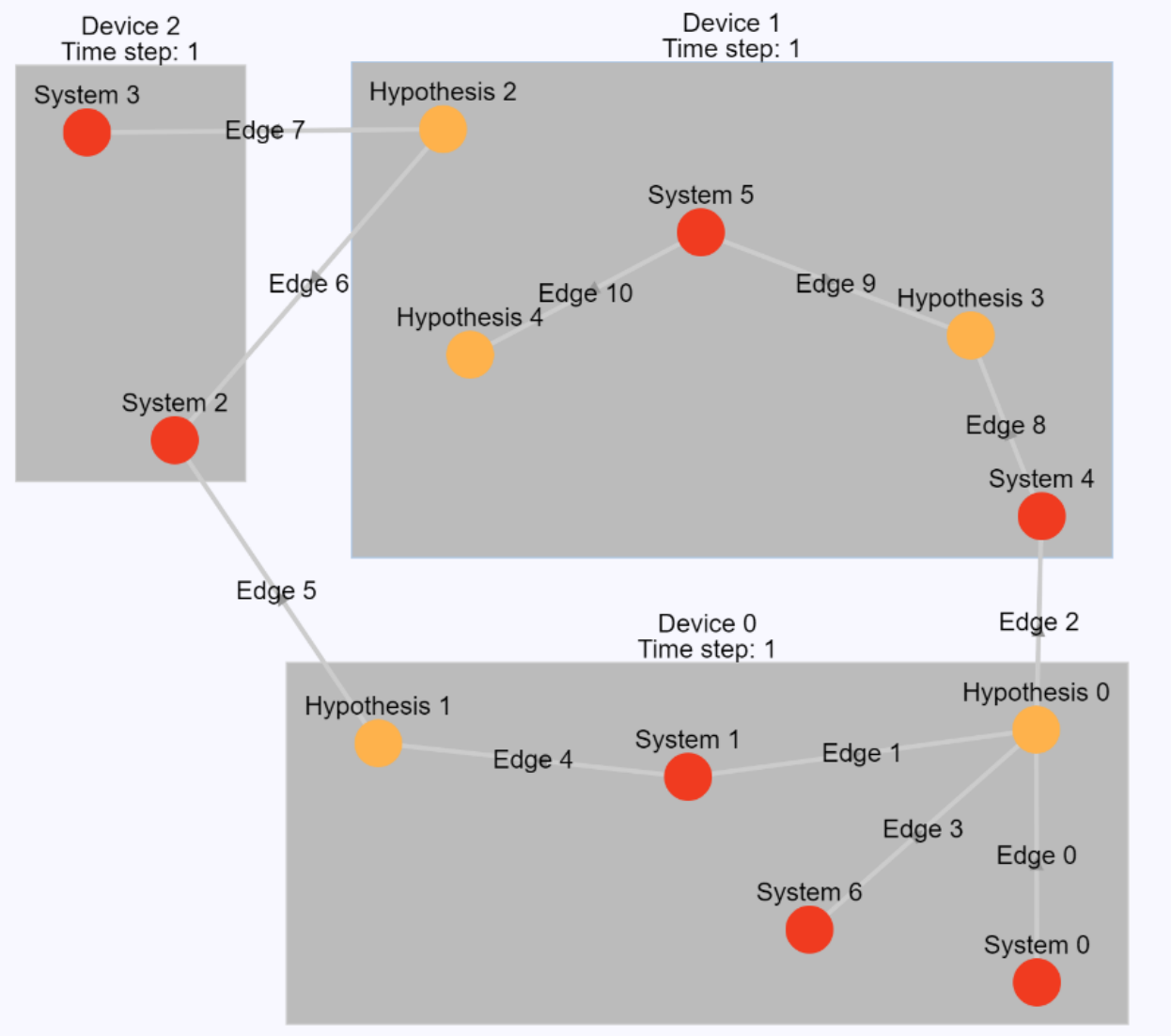
BIJLAGEN

Bijlage A) Realistische, kleine grap



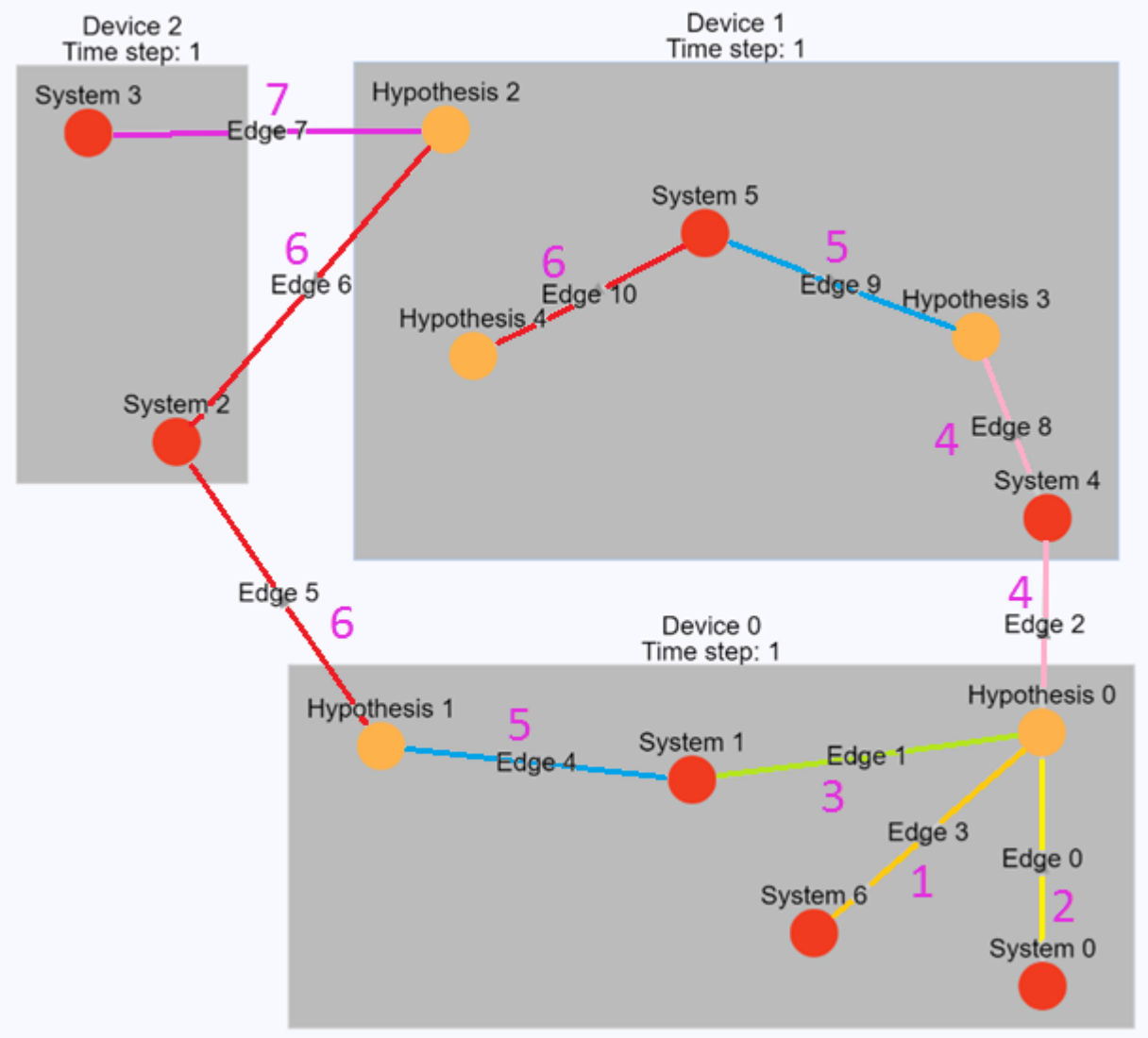
Bijlage A, realistische, kleine graph

Bijlage B) Onrealistische representatie gebruikt voor uitleg



Bijlage B, onrealistische representatie gebruikt voor uitleg

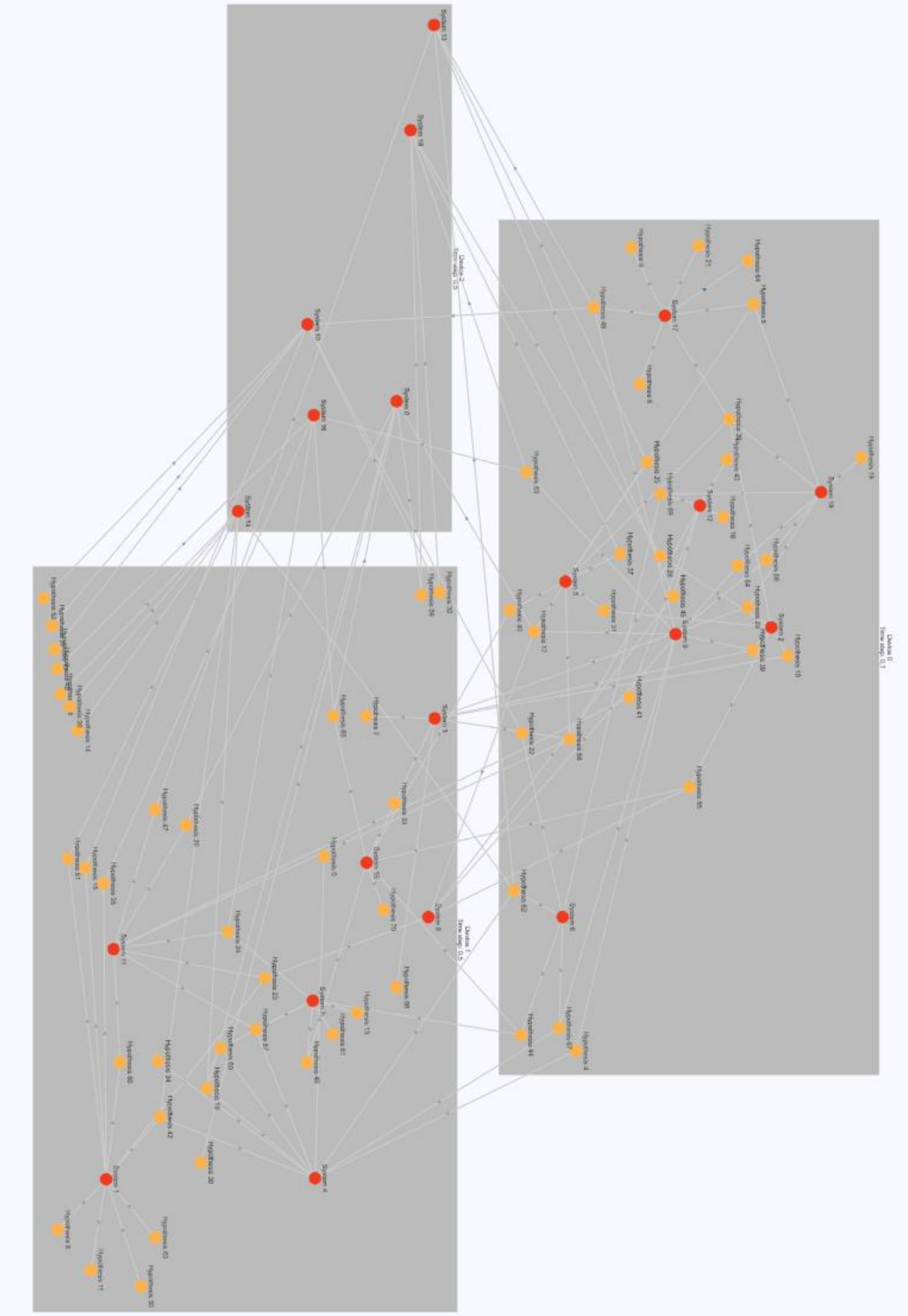
Bijlage C) Visuele representatie van parallellisatie



Bijlage C, visualisatie van graph

BIJLAGEN

Bijlage D) Visualisatie van probleem met naïef verdeelalgoritme



Bijlage D, visualisatie van het probleem met een naïef verdeelalgoritme

BIJLAGEN

Bijlage E) Tabellen uitleg visualisatie

Bijlage E, draaiboeken van devices

Device CPU cycle	Van	Naar
1	Hypothesis 0	System 6
2	Hypothesis 0	System 0
3	Hypothesis 0	System 1
4	Hypothesis 0	(out edge) System 4
5	System 1	Hypothesis 1
6	Hypothesis 1	(out edge) System 2
7	N/A	N/A

Tabel 16, draaiboek device 1

Device CPU cycle	Van	Naar
1	N/A	N/A
2	N/A	N/A
3	N/A	N/A
4	(in edge) System 4	Hypothesis 3
5	Hypothesis 3	System 5
6	System 5	Hypothesis 4
7	(in edge) Hypothesis 2	(out edge) System 3

Tabel 17, draaiboek device 2

Device CPU cycle	Van	Naar
1	N/A	N/A
2	N/A	N/A
3	N/A	N/A
4	N/A	N/A
5	N/A	N/A
6	(in edge) System 2	(out edge) Hypothesis 2
7	(in edge) System 3	N/A

Tabel 18, draaiboek device 3

BIJLAGEN

Bijlage F) **Bijlage F, Requirements document**

Graph Analyse Op Mobile Grid Computing Netwerk

Tom van Amstel

20 november 2016

Informatica – Haagse Hogeschool, locatie Zoetermeer

Eerste examinator: Dhr. R. Loke

Tweede examinator: Dhr. H. Al-Ers

BIJLAGEN

Functional requirements

Business requirements

ID	Description
BR01	Er moet een product gemaakt worden dat als benchmark gebruikt kan worden om het concept ‘parallelle collaboratie’ te bewijzen
BR02	Het product moet voldoende instelbaar zijn
BR03	Het product moet andere implementaties van analyse algoritmes kunnen ondersteunen
BR04	Het product moet zo veel mogelijk op de realiteit lijken, zodat er een goede vergelijking getrokken kan worden met de realiteit

Universele requirements

ID	Description	MoSCoW priorisation
UR01	Een grid computing network moet gesimuleerd worden	Must have
UR02	Nodes van een grid computing network (genoemd devices) moeten gesimuleerd worden	Must have
UR03	De devices van de simulatie moeten samen kunnen werken	Must have
UR04	De devices van de simulatie moeten een graph kunnen doorzoeken door middel van een breadth-first-search algoritme	Must have
UR05	Het breadth-first-search algoritme moet per device stap voor stap uitgevoerd kunnen worden	Must have
UR06	De graph moet gevisualiseerd worden	Must have
UR07	De simulatie moet gevisualiseerd worden	Must have
UR08	Er moeten metrieken worden gegenereerd van de simulatie gebaseerd op de instellingen	Should have
UR09	De metrieken moeten gevisualiseerd worden	Could have

BIJLAGEN

Simulatie requirements

ID	Description	MoSCoW priorisation
SR01	Packet loss moet controleerbaar zijn	Could have
SR02	Latency moet controleerbaar zijn	Must have
SR03	Individuele device snelheid moet controleerbaar zijn	Must have
SR04	Devices moeten kunnen uitvallen	Won't have
SR05	De gebruiker moet de hoeveelheid devices kunnen instellen	Must have
SR06	Individuele device snelheid moet ingesteld kunnen worden	Must have
SR07	De gebruiker moet de snelheid van de visualisatie kunnen instellen	Must have
SR08	Latency moet door de gebruiker ingesteld kunnen worden op de simulatie	Should have
SR09	De implementatie van een graph analyse algoritme moet makkelijk uit te breiden zijn.	Should have
SR10	Elk device in de simulatie moet een graph analyse algoritme bevatten	Must have

Graph requirements

ID	Description	MoSCoW priorisation
GR01	De graph moet nodes hebben met de types System, Hypothesis en Evidence	Must have
GR02	In de graph mogen nodes met het type System alleen een edge hebben naar nodes met het type Hypothesis	Must have
GR03	In de graph mogen nodes met het type Evidence geen edges hebben naar andere nodes.	Must have
GR04	In de graph mogen nodes met het type Hypothesis edges hebben naar nodes met de types System, Hypothesis en Evidence	Must have
GR05	Graphs moeten gegenereerd kunnen worden	Could have
GR06	Graphs moeten verdeeld kunnen worden in hypergraphs	Must have
GR07	Het aantal hypergraphs waar de graph in verdeeld word moet in te stellen zijn door de gebruiker	Should have
GR08	De hypergraphs moeten worden verdeeld over de beschikbare devices	Must have
GR09	De structuur en verdeling van de nodes in de hypergraphs moet	Could have

	overeenkomen met de structuur van de volledige graph	
GR10	De graph nodes moeten zo gelijkmatig mogelijk over de devices verdeeld worden, met zo min mogelijk edges tussen de devices	Won't have

Non-functional requirements

ID	Description	MoSCoW priorisation
NR01	Er moet redundantie worden toegevoegd zodat uitgevallen devices opgevangen kunnen worden	Won't have
NR02	De implementatie van het analyse-algoritme moet vervangbaar zijn met een alternatieve implementatie	Could have
NR03	De methode van verdeling van de graph moet vervangbaar zijn met een alternatieve implementatie	Could have
NR04	Meerdere gebruikers kunnen tegelijkertijd gebruik maken van de simulatie	Won't have

Bijlage G) Tests

Hieronder zijn de unit tests te vinden die gemaakt zijn voor het project. Dit is gedaan naar aanleiding van de requirements. Sommige tests bevatten code van een helper class, hierbij gaat het om de methoden: Helper.TestRootObjects(), GenerateRootEdge(), GenerateRootNode() en de interne NodeComparer() klasse. De code hiervan is aan het eind van deze bijlage te vinden.

Serializer graph cast test – Links between nodes

Bij deze test wordt er een lijst van rootobjects gemaakt en naar een graph gecast. Er wordt een tweede lijst aangemaakt met de verwachte nodes en deze nodes worden hierna gelinkt. Hierdoor ontstaat er een correcte 'expectedlist', waarna gekeken wordt of de lijsten gelijk zijn. Omdat de lijsten circulair doorloopbaar zijn is er een aparte klasse geschreven die de lijsten vergelijkt: NodeComparer.

```
[Test()]
public void CastToGraphTest()
{
    List<RootObject> rootObjects = Helper.TestRootObjects();
    Graph graph = new Graph();

    graph.CastToGraph(rootObjects);

    List<Node> actualList = graph.Nodes;
    List<Node> expectedList = new List<Node>
    {
        new Node() {data = new Data() {id = "System 0", Type = Node.NodeType.System}, marked = false},
        new Node() {data = new Data() {id = "System 1", Type = Node.NodeType.System}, marked = false},
        new Node() {data = new Data() {id = "Hypothesis 0", Type = Node.NodeType.Hypothesis}, marked = false},
        new Node() {data = new Data() {id = "Hypothesis 1", Type = Node.NodeType.Hypothesis}, marked = false},
        new Node() {data = new Data() {id = "Hypothesis 2", Type = Node.NodeType.Hypothesis}, marked = false}
    };

    for (int i = 0; i < 9; i++)
    {
        expectedList.Add(new Node()
        {
            data = new Data()
            {
                id = "Evidence " + i, Type = Node.NodeType.Evidence
            }, marked = false
        });
    }

    linkNodes(expectedList);
    Console.WriteLine(expectedList);
    CollectionAssert.AreEqual(expectedList, actualList, new NodeComparer());
}
```

Serializer graph cast test – Count of nodes

Om zeker te weten dat de array niet te klein of te groot is, maar precies de grootte die hij moet zijn is er de GraphCounterTest() opgezet.

```
[Test()]
public void GraphCounterTest()
{
    List<RootObject> rootObjects = Helper.TestRootObjects();
    Graph graph = new Graph();

    Tuple<int, int> expectedCount = Tuple.Create(14,13);
    Tuple<int, int> actualCount = graph.CastToGraph(rootObjects);
    Assert.AreEqual(expectedCount, actualCount);
}
```

Serializer graph cast test – Edge addition test

Bij deze test wordt er gekeken of de requirements van de graph nageleefd worden. Hierbij moet gedacht worden aan de restricties die per node type worden beschreven, bijvoorbeeld: Systems mogen alleen een edge hebben naar hypotheses.

```
[Test()]
public void AddEdgeTest()
{
    List<Node> actualList = new List<Node>();
    List<Node> expectedList = new List<Node>();

    Graph graph = new Graph();
    Node s1Node = new Node { data = new Data() { id = "System", Type = Node.NodeType.System } };
    Node h1Node = new Node { data = new Data() { id = "Hypothesis", Type = Node.NodeType.Hypothesis } };
    Node e1Node = new Node { data = new Data() { id = "Evidence", Type = Node.NodeType.Evidence } };

    Node s2Node = new Node { data = new Data() { id = "System", Type = Node.NodeType.System } };
    Node h2Node = new Node { data = new Data() { id = "Hypothesis", Type = Node.NodeType.Hypothesis } };
    Node e2Node = new Node { data = new Data() { id = "Evidence", Type = Node.NodeType.Evidence } };

    // Try to add all possible edge possibilities (even wrong ones)
    // "SH" stands for System -> Hypothesis, "HE" for Hypothesis -> Evidence etc.
    graph.AddEdge("SH", s1Node, h1Node); graph.AddEdge("SH", s1Node, h1Node);
    graph.AddEdge("HS", h1Node, s1Node); graph.AddEdge("HS", h1Node, s1Node);
    graph.AddEdge("HE", h1Node, e1Node); graph.AddEdge("HE", h1Node, e1Node);
    graph.AddEdge("EH", e1Node, h1Node); graph.AddEdge("EH", e1Node, h1Node);
    graph.AddEdge("SE", s1Node, e1Node); graph.AddEdge("SE", s1Node, e1Node);
    graph.AddEdge("ES", e1Node, s1Node); graph.AddEdge("ES", e1Node, s1Node);
    graph.AddEdge("SS", s1Node, s1Node); graph.AddEdge("SS", s1Node, s1Node);
    graph.AddEdge("HH", h1Node, h1Node); graph.AddEdge("HH", h1Node, h1Node);
    graph.AddEdge("EE", e1Node, e1Node); graph.AddEdge("EE", e1Node, e1Node);

    // Expected edges
    s2Node.Neighbours.Add("SH", h2Node);
    h2Node.Neighbours.Add("SH", s2Node);
    h2Node.Neighbours.Add("HE", e2Node);
    e2Node.Neighbours.Add("HE", h2Node);
    h2Node.Neighbours.Add("HH", h2Node);

    actualList.Add(s1Node);
    actualList.Add(h1Node);
    actualList.Add(e1Node);

    expectedList.Add(s2Node);
    expectedList.Add(h2Node);
    expectedList.Add(e2Node);

    CollectionAssert.AreEqual(expectedList, actualList, new NodeComparer());
}
```

BIJLAGEN

Bfs ordering test

In deze test wordt er gekeken naar de ordening van de nodes als gevolg van het analyse algoritme. Deze volgorde moet overeenkomen met de handmatig opgestelde volgorde. Indien het niet overeenkomt zal de test falen, omdat er een verkeerde volgorde en dus geen breadth-first-search gedaan wordt.

```
[Test()]
public void BfsOrderTest()
{
    DeviceBase device = new Device(1);
    Graph testGraph = Helper.TestGraph();
    List<string> expectedList = new List<string>()
    {
        "SH00", "SH01", "SH10", "HE00", "HE01", "HE02",
        "HE13", "HE14", "HE15", "HE26", "HE27", "HE28"
    };

    device.InQueue.Enqueue(testGraph.Nodes.Find(i => i.data.id == "System 0"));

    List<string> outEdges = new List<string>();
    List<string> actualList = new List<string>();

    for (int i = 0; i < 12; i++)
    {
        actualList.AddRange(device.ResolveStep(ref outEdges));
    }
    CollectionAssert.AreEqual(expectedList, actualList);
}
```

BIJLAGEN

Helper classes used in tests

Hier zijn de helper classes te vinden. Hier worden de lijsten handmatig opgezet waarover getest gaat worden. Het is verdeelt over de verschillende functionaliteiten, zodat er geen conflict komt tussen de tests.

```
public static List<RootObject> TestRootObjects()
{
    List<RootObject> rootObjects = new List<RootObject>();
    rootObjects.Add(GenerateRootNode("System 0", Node.NodeType.System));
    rootObjects.Add(GenerateRootNode("System 1", Node.NodeType.System));
    rootObjects.Add(GenerateRootNode("Hypothesis 0", Node.NodeType.Hypothesis));
    rootObjects.Add(GenerateRootNode("Hypothesis 1", Node.NodeType.Hypothesis));
    rootObjects.Add(GenerateRootNode("Hypothesis 2", Node.NodeType.Hypothesis));

    for (int i = 0; i < 9; i++)
    {
        rootObjects.Add(GenerateRootNode("Evidence " + i, Node.NodeType.Evidence));
    }

    rootObjects.Add(GenerateRootEdge("SH00", "System 0", "Hypothesis 0"));
    rootObjects.Add(GenerateRootEdge("SH01", "System 0", "Hypothesis 1"));
    rootObjects.Add(GenerateRootEdge("SH10", "System 1", "Hypothesis 0"));
    rootObjects.Add(GenerateRootEdge("SH12", "System 1", "Hypothesis 2"));
    rootObjects.Add(GenerateRootEdge("HE00", "Hypothesis 0", "Evidence 0"));
    rootObjects.Add(GenerateRootEdge("HE01", "Hypothesis 0", "Evidence 1"));
    rootObjects.Add(GenerateRootEdge("HE02", "Hypothesis 0", "Evidence 2"));
    rootObjects.Add(GenerateRootEdge("HE13", "Hypothesis 1", "Evidence 3"));
    rootObjects.Add(GenerateRootEdge("HE14", "Hypothesis 1", "Evidence 4"));
    rootObjects.Add(GenerateRootEdge("HE15", "Hypothesis 1", "Evidence 5"));
    rootObjects.Add(GenerateRootEdge("HE26", "Hypothesis 2", "Evidence 6"));
    rootObjects.Add(GenerateRootEdge("HE27", "Hypothesis 2", "Evidence 7"));
    rootObjects.Add(GenerateRootEdge("HE28", "Hypothesis 2", "Evidence 8"));
    return rootObjects;
}
```

```
private static RootObject GenerateRootEdge(string id, string source, string target)
{
    return new RootObject()
    {
        data = new Data(){id = id, source = source, target = target},
        position = new RootObject.Position(){},
        group = "edges",
        removed = false,
        selected = false,
        selectable = true,
        locked = false,
        grabbable = true,
        classes = ""
    };
}

private static RootObject GenerateRootNode(string id, Node.NodeType type)
{
    return new RootObject()
    {
        data = new Data(){ id = id, Type = type },
        position = new RootObject.Position(){ x = 0, y = 0 },
        group = "nodes",
        removed = false,
        selected = false,
        locked = false,
        grabbable = true,
        classes = ""
    };
}
```

BIJLAGEN

```
internal class NodeComparer : Comparer<Node>
{
    public override int Compare(Node node, Node otherNode)
    {
        int error;
        if (node == null || otherNode == null) return -1;
        if ((error = node.data.Type.CompareTo(otherNode.data.Type)) != 0) return error;
        if ((error = string.Compare(node.data.id, otherNode.data.id, StringComparison.Ordinal)) != 0) return error;
        if ((error = node.marked.CompareTo(otherNode.marked)) != 0) return error;
        if (node.Neighbours.Count != otherNode.Neighbours.Count) return -1;

        foreach (KeyValuePair<string, Node> neighbour in node.Neighbours)
        {
            var foo = otherNode.Neighbours[neighbour.Key];
            if (neighbour.Value.data.id != otherNode.Neighbours[neighbour.Key].data.id
                || neighbour.Value.data.Type != otherNode.Neighbours[neighbour.Key].data.Type
                || neighbour.Value.Neighbours.Count != otherNode.Neighbours[neighbour.Key].Neighbours.Count)
                return -1;
        }
        return 0;
    }
}
```