Flexibel produceren op een productiegrid door het toevoegen van een zelforganiserend en -beschrijvend multi-agentsysteem

Door: Martijn Beek

Studentnummer: 1557514

Eerste examinator: Daniël Telgen

Tweede examinator: Leo van Moergestel

Bedrijfsbegeleider: Erik Puik

Samenvatting

In de praktijk wordt er nu vaak voor elk type product of onderdeel die geproduceerd moet worden een machine ontworpen. Dit is duur, maakt hergebruik moeilijk en maakt het moeilijk om de omvang van de productie snel te veranderen.

Er bestaan onderzoeken die aantonen dat het flexibiliseren van productiemachines kan bijdragen aan het oplossen van deze problemen. De opdrachtgever heeft in dit kader een theoretisch productiesysteem bedacht. Hierin kunnen de verschillende productiemachines, bijvoorbeeld meerdere robots, gezamenlijk werken aan het produceren van producten. De productiemachines, de zogenaamde equiplets, zijn zo goedkoop mogelijk en herconfigureerbaar. Dat wil zeggen dat hardware- en software van verschillende modules van de equiplet vervangen kunnen worden met weinig downtime. Dit heeft ten gevolg dat de functionaliteit en de toestand van het hele productiesysteem voortdurend kan veranderen.

Het is bij een dergelijk oplossing belangrijk dat de software van de losse modules van elkaar gescheiden is en een gedistribueerd karakter heeft. In eerder onderzoek in de afstudeerorganisatie is hier in grote lijnen over nagedacht en is een oplossing gevonden in het gebruiken van zogenaamde agents. Dit zijn op zich zelf staande stukken software die met elkaar samenwerken en sociaal gedrag vertonen. Een dergelijk gedistribueerde softwareoplossing is zeer geschikt voor een gedistribueerd productiesysteem.

In de afstudeerorganisatie was al een globaal ontwerp gemaakt voor een oplossing gebaseerd op agents. Deze is verder uitgewerkt met de praktische haalbaarheid als belangrijkste uitgangspunt. Hierbij is een nieuw overzichtsontwerp gemaakt van het agentsysteem. De ontworpen agents zijn geïmplementeerd samen met andere onderdelen van het systeem, waaronder een oplossing voor een gezamenlijke datastructuur waar alle agents bij kunnen om informatie over producten die gemaakt worden te delen en te bewerken. Dit is het blackboard.

Als het productiegrid dynamisch en modulair moet zijn, moet bij elke wijziging dynamisch worden uitgezocht hoe een wijziging invloed heeft op de functionaliteit. Om dit te kunnen doen is het nodig dat producten kunnen beschrijven hoe ze worden gemaakt en modules kunnen beschrijven welke functionaliteit ze bieden. Om dit te bewerkstelligen is een taal gedefinieerd waarmee het productiegrid zichzelf kan beschrijven.

In een proof-of-concept wordt aangetoond dat de productieplatforms en bijbehorende ideeën van de opdrachtgever praktisch toe te passen zijn. Samen met de softwareoplossing die is ontwikkeld zijn deze productiesystemen in de praktijk inzetbaar om bedrijven flexibeler en daardoor goedkoper kleine tot middelgrote hoeveelheden producten te laten produceren.

Voorwoord

Deze scriptie beschrijft de opdracht zoals ik die heb uitgevoerd bij het Hogeschool Utrecht lectoraat Microsysteem technologie en embedded systems van lector, opdrachtgever en bedrijfsbegeleider Erik Puik. Deze opdracht was een duo opdracht die ik samen met Pascal Muller heb gedaan vanaf februari 2012 tot en met juni 2012.

Ik heb een onderzoeksemester gedaan tijdens de minorruimte naar de scheduling van productagents binnen een multi-agent productiesysteem. Ik was zeer geïnteresseerd in een soort vervolgonderzoek hierop. Het onderzoek dat ik toen heb gedaan is nauw verbonden met de onderzoeken die bij het kenniscentrum van het MST worden uitgevoerd. Sterker nog, de onderzoeken zijn allemaal onderdeel van een groter geheel. Ik was dus erg enthousiast om nog een onderzoek te doen binnen dat kader.

Tijdens deze stage is gezamenlijk gewerkt aan een ontwerp van een systeem dat produceren met flexibele productiesystemen mogelijk moet maken. Dit gebeurde samen met de opdrachtgever en andere binnen zijn kenniskring. Vervolgens hebben we een proof of concept van het ontwerp geïmplementeerd, waarbij we de opdracht hebben uitgesplitst zodat ieder een eigen gedeelte had.

Het is een grote uitdaging gebleken om verschillende input en belangen van promovendi en de opdrachtgever af te wegen en een systeem te ontwerpen waar ieder van hun tevreden mee kan leven. Het is daarbij erg interessant om een middenweg zien te vinden tussen de theoretische ideeën van de promovendi en de praktische input van de opdrachtgever.

Graag wil ik enkele mensen bedanken voor hun inbreng tijdens dit project. Ten eerste bedank ik Erik Puik voor de stageplaats en zijn feedback. Docentbegeleider Daniël Telgen bedank ik voor zijn nuttige tips en input. Tevens bedank ik Leo van Moergestel voor het geven van input en kennis vanuit een andere hoek van het onderzoek. Dit heeft me scherp gehouden. Ik wil graag medeafstudeerder Pascal Muller zeer bedanken voor zijn inzet, harde werk en pientere opmerkingen. Met goede samenwerking bereik je veel meer dan twee individuen. Ten slotte bedank ik iedereen op Pinkpop die daar zo gek is geweest mijn verslag te lezen en me te wijzen op mijn fouten.

Utrecht, 28 mei 2012
Martijn Beek

Inhoudsopgave

Samenvatting ..... **i**
Voorwoord ..... **iii**
1. Inleiding **1**
1.1. Afstudeerorganisatie 1
1.2. Projectorganisatie 2
1.3. Documentopbouw 3
1.4. Achtergrond begrippen 3
1.5. De HUniplacer 8
2. Probleem **9**
2.1. Opdrachtomschrijving 9
2.2. Uitgangssituatie 9
2.3. Probleemstelling, analyse van het probleem 9
2.4. Doelstelling 10
2.5. Onderzoeksvragen 10
2.6. Scope 10
2.7. Randvoorwaarden 11
3. Proces **12**
3.1. Aanpak 12
3.2. Werkwijze en projectactiviteiten 13
3.3. Gebruikte methodieken en tools 14
3.4. Research 15
3.5. Taakverdeling 18
4. Ontwerp **19**
4.1. Het overall-design 19
4.2. De agents 21
4.3. ROS systeem 24
4.4. Dataholders 24
4.5. Data 26
5. Technische Implementatie **28**
5.1. Multi-agentsysteem 29
5.2. Blackboardserver 37
6. Resultaten en analyse **41**
6.1. De opgeleverde resultaten 41
6.2. Analyse van de resultaten 42
7. Conclusies en aanbevelingen **43**
7.1. Beantwoorden onderzoeksvragen 43
7.2. Aanbevelingen en vervolgonderzoek 45
8. Literatuurlijst **46**
Bijlage A - Plan van Aanpak ..... **I**
Bijlage B - Evaluatie eigen functioneren ..... **II**
Bijlage C - TODO-lijst scriptie ..... Fout! Bladwijzer niet gedefinieerd.

1. Inleiding

1.1. Afstudeerorganisatie

Het lectoraat waar de opdracht beschikbaar is, is het lectoraat Microsysteemtechnologie/Embedded Systems van de Hogeschool Utrecht. Deze houdt zich bezig met productverbetering en industrialisatie van microsystemen. [1]

Figuur 1 - De stageplaats

Sinds 2001 is de functie van lector ingevoerd in het Hoger Beroepsonderwijs. Lectoren geven leiding aan een kenniskring van onderzoekers (meestal docenten) die praktijkgericht onderzoek doen. Deze kenniskring wordt lectoraat genoemd. Praktijkgericht onderzoek heeft als doel problemen op te lossen waar professionals in de praktijk tegenaan lopen. Ook bij de afstudeeropdracht is er een koppeling tussen theorie en praktijk.

Het vakgebied van dit lectoraat concentreert zich op systemen met hoge nauwkeurigheden; de productie daarvan behoort tot de meest lastige die er bestaat. Dit kan mogelijk leiden tot productie-uitval, een te lage productieopbrengst, oplopende productiekosten, maar vooral ook projectvertraging op een moment dat investeringen al plaatsgevonden hebben.

Onderzoek van dit lectoraat heeft onder meer betrekking op:

- Optimalisatie van productontwerp en de ontwerpprocedures.
- Modularisering van productieprocessen. Door modules te hergebruiken dalen ontwerpinspanning en de kosten van industrialisatie. Daarnaast verbetert de kwaliteit van de productie.

[1] Delen van dit hoofdstuk zijn geparafraseerd overgenomen van de webpagina op: http://www.technologieeninnovatie.hu.nl/Data/Lectoraten/Microsysteem%20technologie%20Embedded%20Systems.aspx

- Intelligente machines. Door de productiemachines van intelligentie te voorzien, kunnen ze bijvoorbeeld zelf bepalen wat ze produceren, hun productiecapaciteit opschroeven als dat nodig is etc.

Dit lectoraat zet onder de naam HUniversal Production de productiefilosofie van de toekomst neer. Het is gebaseerd op de ontwikkeling rond Agile Manufacturing (AM). AM wil de flexibiliteit en reactietijd bij productievragen verbeteren.

1.1.1. Cultuur

Het kenniscentrum heeft een heel open sfeer. De slagzin van het kenniscentrum is dan ook “Het kenniscentrum is van iedereen”. Er zitten veel verschillende mensen in het kenniscentrum: een aantal lectoren, leraren en stagiaires. Het aantal mensen verschilt dat er zit verschilt vaak; van ongeveer 3 tot 15 personen.

Er worden ook lunchlezingen gehouden in het kenniscentrum. In een dergelijke lunchlezing wordt meestal door iemand van Product Design and Engineering een spreker uitgenodigd om een lezing te houden. Die lezingen waren niet relevant voor deze opdracht.

1.2. Projectorganisatie

Het afstudeerproject is een project wat raakt aan de projecten "HUniplacer" en " Low Cost Vision for Robotics and Mechatronics ". In deze projecten is gewerkt aan respectievelijk een deltarobot en de aansturing hiervan met een camera als sensor. Hier zijn veel organisaties bij betrokken waaronder:

- Fontys
- Avans,
- TU/e
- NHL

In dit huidige project wordt in opdracht van het lectoraat MST een softwareoplossing ontworpen en ontwikkeld om meerdere van zulke machines flexibel in te zetten. De machine wordt verdeeld in losse modules die in elkaar gezet kunnen worden. Op elk moment kunnen modules worden toegevoegd of verwijderd. Ditzelfde geldt voor machines, dit zijn verzamelingen van modules.

Het project heeft een sterk onderzoekskarakter: bij aanvang van het project zijn er wel doelen, maar er is nog geen concreet idee van welke producten zullen worden gemaakt of hoe het project zal verlopen. Er zal een proof-of-concept worden geïmplementeerd van de resultaten van het onderzoek.

Verder zijn er binnen de HU veel studenten en personeelsleden betrokken bij het project waarbij multidisciplinaire samenwerking hoog in het vaandel staat. Zo zijn er studenten van werktuigbouwkunde bezig met het verbeteren van de productiemachines en informatici bezig met het verbeteren van de camera’s van de HUniplacer. Ook zijn er in het verleden veel projectteams geweest die de huidige HUniplacer hebben ontwikkeld.

1.2.1. Betrokken personen

Bij de uitvoering van dit project zijn meerdere personen direct betrokken. Directe betrokkenheid wil zeggen dat ze betrokken zijn bij de uitvoering of beoordeling van het werk of het werk zullen (gaan) gebruiken.

De volgende personen zijn betrokken bij het project:

Tabel 1 Bij het afstudeerproject betrokken personen

Rol	Perso(o)n(en)	Belangrijkste taak
Eerste examinator	Daniël Telgen	- Tussentijdse feedback op resultaten
Bedrijfsbegeleider	Erik Puik	- Bewaken van de business case - Tussentijdse feedback op resultaat
Teamlid	Pascal Muller Martijn Beek	- Ontwerpen - Implementatie proof-of-concept - Promotieactiviteiten, kennis overdragen
Gebruiker	Daniël Telgen Leo van Moergestel Erik Puik	- Tussentijdse feedback geven - Data en software gebruiken voor hun onderzoek - Resultaten publiceren - Bewaken van nut van het project voor hun onderzoek

1.3. Documentopbouw

In het volgende hoofdstuk zal op de opdracht worden ingegaan en zal deze worden geanalyseerd. Vervolgens in het hoofdstuk Proces zal worden ingegaan op welke activiteiten er hebben plaatsgevonden, welke research is gedaan en welke methodes en tools worden gebruikt om dit project mede mogelijk te maken. Hierna komt het ontwerp aan bod dat is gemaakt voor de opdracht en daarna de technische implementatie daarvan. Ten slotte volgen de resultaten en de conclusie.

1.4. Achtergrond begrippen

Voor het goede verloop van de rest van dit verslag zullen er eerst een aantal belangrijke terugkomende begrippen en concepten worden uitgelegd ter verduidelijking van de context van deze opdracht.

1.4.1. Reconfigurable manufacturing

In de afgelopen decennia zijn de mogelijkheden voor automatisering ook doorgedrongen in productieprocessen. Vaak wordt voor een specifiek (type) product een *Dedicated Manufacturing System* (DMS) ontwikkeld. Het ontwikkelen van een dergelijk systeem kost veel tijd en geld, maar is exact gemaakt voor één (type) product en kan dat daardoor goed en goedkoop.

Grote nadelen zijn de kleine flexibiliteit, de hoge ontwikkelkosten en het gebrek aan schaalbaarheid. Als een DMS niets staat te doen zijn de ontwikkelkosten per product relatief hoog. Het is dus noodzakelijk om de capaciteit volledig benutten. Als de marktvraag plots stijgt, kan het DMS niet meer produceren dan de maximale capaciteit en moet er een tweede lijn worden neergezet. Als de marktvraag daalt, al dan niet aan het einde van de levensduur van het product, dan is de DMS ook

onderbezet. Ook is het niet of slecht mogelijk om producten tijdens hun levensduur aan te passen of op maat voor klanten te maken, immers, het is speciaal voor het product op maat ontwikkeld.

Bij *Agile Manufacturing* wordt gekeken naar hoe de problemen met een DMS op te lossen zijn. Dit gaat vooral om het flexibeler maken van productiecapaciteit. Dit wil zeggen dat erover nagedacht moet worden hoe het productieproces zo kan worden ingericht dat snel kan worden gereageerd op een veranderende marktvraag tegen zo laag mogelijke kosten. [1]

Een mogelijke manier om product specifieker te produceren is voor elk soort product een algemene productielijn of machine neer te zetten die veel mogelijkheden biedt. Het zou bijvoorbeeld mogelijk zijn om met een aantal CNC-machines, uitgerust met verschillende gereedschappen, een productielijn in te richten. Dit wordt ook wel een *Flexible Manufacturing System* (FMS) genoemd. [2] Een FMS heeft de mogelijkheid om meerdere soorten producten in kleinere volumes te produceren of producten op maat te leveren. Het nadeel hiervan is dat een algemene oplossing misschien veel meer mogelijkheden biedt dan strikt noodzakelijk is voor de op te leveren producten. Dit brengt extra kosten met zich mee, omdat een machine die meer mogelijkheden biedt in regel duurder is dan een die minder mogelijkheden biedt. Daarnaast kunnen deze machines nooit zo efficiënt werken als een machine die speciaal voor een bepaald product is gemaakt.

De meest flexibele oplossing is een zogenaamd *Reconfigurable Manufacturing System* (RMS). [2] Een RMS bestaat uit herbruikbare en –configureerbare machines. Deze

Koren hanteert de volgende definitie:

> *"Een Reconfigurable Manufacturing System (RMS) is vanaf het begin af ontworpen met om snel in structuur, maar ook in hardware- en softwarecomponenten te wijzigen, met als doel de productiecapaciteit en functionaliteit binnen een famillie onderdelen te wijzigen om snel te reageren op veranderingen in de markt en wetgeving."*

Een RMS is goedkoper doordat deze (deels) hergebruikt kan worden voor nieuwe producten en het systeem makkelijker geschaald kan worden doordat machines of *Reconfigurable Machine Tools* (RMT's) kunnen worden toegevoegd of verwijderd als de marktvraag veranderd. RMT's zijn herconfigureerbare modules die op een machine kunnen worden gezet, bijvoorbeeld extra assen met motoren of een boormachine. Een groot voordeel van RMT's is dat in een productiesysteem tegelijk meerdere soorten producten in meerdere volumes kunnen worden gemaakt.

Tabel 2 Vergelijking DMS/FMS/RMS. RMS combineert voordelen van een DMS met die van een FMS

	DMS	**FMS**	**RMS/RMT**
Structuur van machine	Vast	Vast (CNC)	Aanpasbaar
Initiële kosten[2]	Hoog	Laag	Laag
Kosten per product[3]	Laag	Hoger	Gemiddeld
Schaalbaar	--	+	++
Flexibiliteit	--	++	Zelf te beïnvloeden door ontwerp RMT
Converteerbaarheid[4]	--	++	+

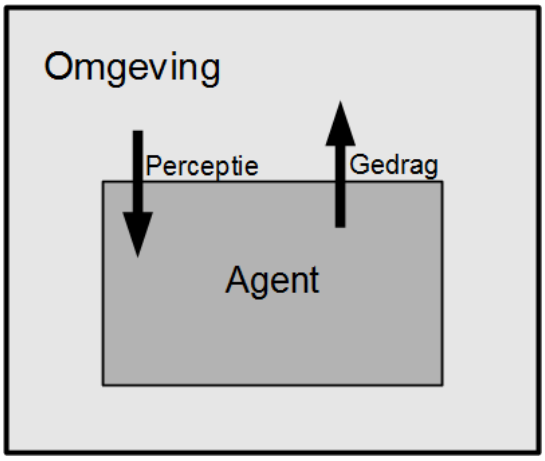
[2] **Lager is beter**.
[3] Bij volledig benutten van de capaciteit die op een bepaald moment beschikbaar is.
[4] Hoe makkelijk is het om de productielijn om te bouwen voor andere producten

1.4.2. Agents en MAS

Agents

Agents zijn software-entiteiten die een bepaalde mate van vrijheid hebben om gedrag te vertonen. Om gedrag te vertonen moeten deze agents in staat zijn om gedrag te uiten in de omgeving en de omgeving te interpreteren.

Figuur 2 - Een agent in zijn omgeving



Een agent is dus feitelijk een intelligente eenheid met in- en outputs waarmee de agent in staat is om de omgeving waar te nemen en acties uit te voeren:

Hoe agents zich gedragen en hoe je individuele agents intelligent gedrag kan laten vertonen, is erg interessant. De definitie van agents die in deze scriptie gehanteerd wordt is de definitie van Wooldridge [3]. Hij hanteert de volgende definitie:

"Een agent is een computersysteem, dat autonoom kan handelen in een bepaalde omgeving om zo de toegewezen doelen te halen."

Multi-agentsystemen

In multi-agentsystemen werken meerdere van deze agents samen om een resultaat te bereiken. Voor agents in een multi-agentsysteem is het vooral van belang dat ze ook in staat zijn om met elkaar te communiceren.

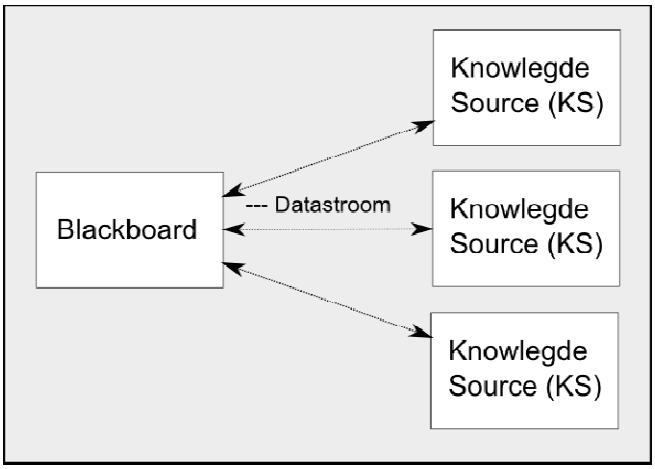
In een multi-agentsysteem is vooral de samenwerking tussen de agents van belang. Het combineren van de acties van meerdere agents kan tot complexe, intelligente systemen leiden, zelfs als het gedrag van een individuele agent relatief simpel is. Het gedrag van het systeem is niet meer te beschrijven met het gedrag van agent. Dit verschijnsel wordt emergentie genoemd. Dat wil zeggen dat het gehele multi-agentsysteem niet meer te beschrijven is door alleen het gedrag van een individuele agent.

Voor iemand die multi-agentsystemen ontwerpt is het dus zowel van belang om te kijken naar het gedrag van individuele agents, maar ook naar welke agents moeten samenwerken en hoe deze samenwerking moet plaatsvinden.

1.4.3. Blackboard

Een blackboardsysteem is een systeem waarbij een gedeelde datastructuur, het blackboard, wordt bijgewerkt door een groep intelligente entiteiten.

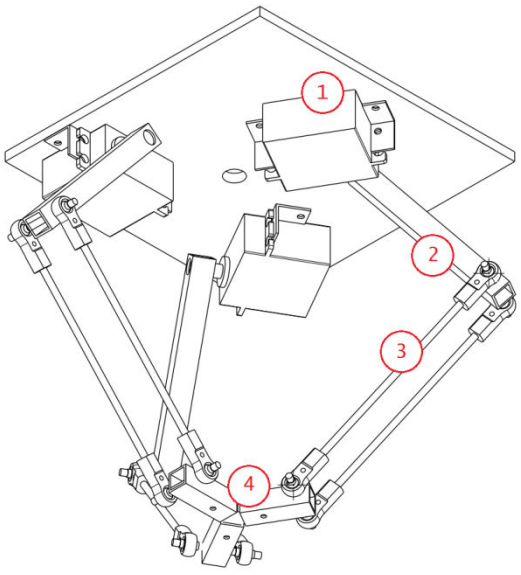
Figuur 3 - Het blackboard en 3 Knowledge Sources



Het blackboard is een database met problemen, gedeeltelijk opgeloste problemen, informatie over het probleem en suggesties [4]. Een initieel probleem wordt op het blackboard geplaatst en de intelligente entiteiten, de zogenaamde Knowledge Sources, zullen elk een stuk van het probleem oplossen en het blackboard met deze oplossing bijwerken, totdat de uiteindelijke oplossing van het probleem is bereikt. Elke entiteit heeft slechts expertise van een deel van het probleem. Er moet dus worden samengewerkt om tot de oplossing te komen.

1.4.4. Deltarobot

e deltarobot



Een deltarobot is een robot die een object in de ruimte kan positioneren of bewegen [5]. Het belangrijkste aspect van het ontwerp is dat gebruik wordt gemaakt van parallellogrammen in de armen. De robot heeft drie vrijheidsgraden (hoogte, breedte, diepte). De robot heeft de volgende belangrijke onderdelen:

1. Motor
2. Heup
3. Arm
4. Effector

De vrijheidsgraden worden gerealiseerd door de effector aan de drie armen te koppelen, vervolgens de armen aan de heupen te vast te maken en die op hun beurt aan de motoren te bevestigen. Belangrijk hierbij is dat dankzij het parallellogramontwerp ongeacht de beweging of de positie van de effector, deze altijd in dezelfde hoek blijft zitten ten opzichte van de basisplaat waaraan de motoren bevestigd zijn.

1.4.5. Equiplet

Een equiplet is een compact en goedkoop productieplatforms dat in het veld opnieuw kan worden geconfigureerd om een groot aantal toepassingen te ondersteunen door het toevoegen van product afhankelijk tools. [6] [7]

Equiplets doen denken aan RMS'en: equiplets zijn herconfigureerbaar en door modules te plaatsen met gereedschappen zijn ze in staat om *productiestappen* uit te voeren. Productiestappen zijn discrete acties die op hoog niveau zijn beschreven. Het uitvoeren van deze stappen in de juiste volgorde leidt tot de productie van een product. Een lijst productiestappen vertoont gelijkenis met de handleiding van een IKEA kast: door het uitvoeren van op hoog niveau beschreven assemblageacties in de juiste volgorde is het mogelijk de kast uit zijn onderdelen te bouwen.

Een equiplet heeft een structuur die het mogelijk maakt modules aan de equiplet vast te schroeven. Tijdens Research&Development-activiteiten kunnen modules worden gebouwd en op kleine schaal worden getest, maar door op meerdere equiplets dezelfde module te zetten kan snel de productie van een nieuw product worden gestart en opgeschaald.

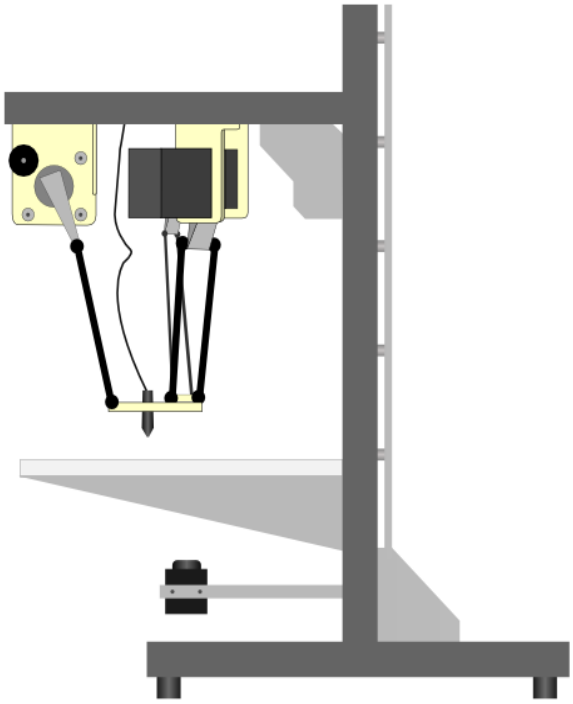
1.5. De HUniplacer

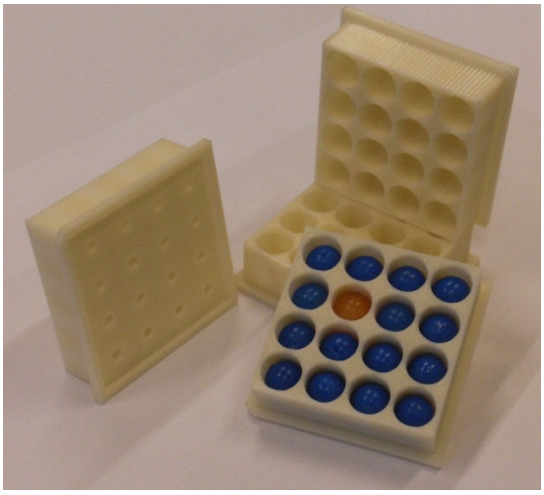
De HUniplacer is de naam van de huidige robot die in het kenniscentrum te vinden is en valt onder de definitie van een equiplet. Deze is in voorgaande projecten in elkaar gezet [8]. De robot bestaat uit drie basisonderdelen.

- Een deltarobot
- Een camera
- Een zuignap

Door de zuignap te plaatsen bij de effector van de deltarobot kan deze worden verplaatst naar elke gewenste coördinaat binnen het bereik van de deltarobot. De zuignap is bedoeld voor het oppakken van objecten. De camera, die onder een glasplaat onder de deltarobot geplaatst is, kan objecten lokaliseren. De enige objecten die hij nu kan herkennen zijn QR-codes.

HUniplacer





De QR-codes zijn geplaatst onder zogenaamde kratjes en in deze kratjes zijn balletjes te vinden. Wat de HUniplacer nu kan is op basis van de QR-code een kratje lokaliseren, er vervolgens een balletje uithalen en die daarna in een ander kratje stoppen op dezelfde plek als waar het balletje in het eerste kratje zat.

Figuur 7 - QR code

2. Probleem

2.1. Opdrachtomschrijving

De hoofdopdracht is het ontwerpen en implementeren van een op agents gebaseerd productiesysteem en vervolgens met een proof of concept aantonen dat deze werkt. Deze proof of concept zal worden aangetoond met het produceren van een product op de HUniplacer in het kenniscentrum. Dit product moet wel binnen de huidige productiemogelijkheden van de deltarobot liggen.

2.2. Uitgangssituatie

Als uitgangssituatie wordt de HUniplacer robot genomen. De resultaten van vorige projecten die daar betrekking op hadden worden tijdens dit project gebruikt als deel van het project. Deze robot zal tevens dienen als demosysteem voor de opdracht.

Voor aanvang van deze opdracht was er al nagedacht over een multi-agentsysteem door de bedrijfsbegeleider en andere betrokkenen. Hier waren al enkele kleine ontwerpen uit voortgekomen. Deze zijn als basis gebruikt voor het ontwerp voor deze opdracht.

2.3. Probleemstelling, analyse van het probleem

In overleg met de opdrachtgever wordt er een ontwerp gemaakt voor een agentgebaseerde softwarearchitectuur die het productiesysteem praktisch moet gaan implementeren. Deze softwarearchitectuur zal de kant voor de aansturing van de deltarobot moeten combineren met een slimme aansturing voor het maken van arbitraire (batches) producten en een GUI waar producten in 3D gemodelleerd kunnen worden. De deltarobot zoals deze er nu staat moet een onderdeel worden van productiesysteem. De deltarobot wordt een equiplet genoemd.

De opdrachtgever heeft enkele eisen aan het op agents gebaseerde systeem, waarvoor een goede oplossing zal moeten bedacht. Twee belangrijke eisen zijn:

- Herconfiguratie van het productiesysteem.
- Flexibel producten en modulen toevoegen aan het systeem .

Om dit systeem te kunnen maken is het noodzakelijk dat eerst de bestaande hard- en software wordt begrepen.

Er zal moeten worden bedacht hoe de herconfiguratie moet plaats vinden, aangezien herconfiguratie grote gevolgen kan hebben voor het productieproces.

Om flexibel modulen en producten aan het systeem toe te kunnen voegen moet het mogelijk zijn om de opbouw van producten abstract te definiëren. In het systeem moeten vervolgens vertaalslagen worden gemaakt van een abstracte beschrijving naar concrete acties in lagere lagen.

Voor deze problemen zal een goede oplossing moeten worden bedacht binnen de te ontwerpen architectuur. Uiteindelijk zal met een proof-of-concept worden aangetoond dat het ontwerp functioneert.

2.4. **Doelstelling**

Tijdens het project zullen de volgende doelstellingen worden behaald:

- Er zal in overleg met de opdrachtgever een agent-gebaseerde architectuur worden ontworpen. Deze is verantwoordelijk voor het aansturen van de robot software en zorgt ervoor dat producten kunnen worden gemaakt. Om dit zo flexibel mogelijk te houden moet worden nagedacht over een abstracte taal om functionaliteiten te beschrijven;
- Het flexibel herconfigureren van de robot zal worden onderzocht. Er moet voor worden gezorgd dat de software van de robot live kan veranderen, maar ook dat het intelligente deel van het systeem hier correct mee omgaat;
- Een implementeren van een proof-of-concept van het ontworpen productiesysteem dat aantoont dat het fabriceren van producten en de herconfiguratie werkt.

2.5. **Onderzoeksvragen**

Voor het project zijn de volgende onderzoeksvragen en bijbehorende deelvragen opgesteld:

- Hoe kan de agent-gebaseerde architectuur worden geïmplementeerd?
    - Kan hiervoor een bestaand ontwerp worden gebruikt of moet er iets nieuws worden ontworpen?
    - Welk agentsysteem kan hier het beste voor worden gebruikt?
    - Hoe moeten producten worden beschreven in een abstracte taal en hoe moet deze taal eruit zien en vertaald worden?
- Hoe kan het herconfigureren van de robot worden geïmplementeerd?
    - Hoe kan dit gedaan worden terwijl het systeem blijft draaien?
    - Welke gevolgen heeft het herconfigureren van een productieplatform en hoe kan dit worden opgelost?

De antwoorden op deze vragen worden in de rest van de verslag gezocht en zullen in de conclusie allemaal specifiek worden beantwoord.

2.6. **Scope**

Het aanpassen en uitbreiden van de aansturing van de hardware van de bestaande robot valt expliciet buiten de scope van dit project. Ook het toevoegen van nieuwe hardware valt niet binnen dit project.

Echter vanwege nieuwe input van buitenaf zijn er toch enkele dagen besteed aan het verbeteren van de deltarobot en de beeldherkenning. Die bevindingen zijn te lezen in hoofdstuk 3.4.3. Overige research.

In verband met de tijd is er voor gekozen om na de ontwerpfase enkel het op agents gebaseerde productiesysteem te implementeren. Het MAST systeem zal niet worden geïmplementeerd in dit project, maar wordt opgeschoven naar vervolgprojecten.

2.7. **Randvoorwaarden**

Om dit project tot een goed einde te brengen zijn de volgende randvoorwaarden opgesteld waaraan voldaan moet worden.

- Kennis van C++ en Java
- Kennis of toegang tot kennis over multi-agentsystemen en computer vision
- Werkruimte in het kenniscentrum bij de deltarobot
- Overdrachtsdocumentatie van vorige projecten
- De bestaande deltarobot en kennis over de deltarobot moeten beschikbaar zijn
- De implementatie van de agents moet technisch haalbaar zijn
- Ervaring met embedded en real-time systemen
- Kennis van Linux
- Kennis van ROS of toegang tot kennis van ROS

3. Proces

3.1. Aanpak

In deze paragraaf staan de hulpmiddelen en afspraken die gemaakt zijn om het project op een professionele manier aan te pakken, waardoor er zoveel mogelijk tijd besteed kan worden aan inhoudelijke en technische zaken.

3.1.1. Git

Voor het beheren van de broncode wordt het versiebeheersysteem Git[5] gebruikt. Hiermee is het eenvoudig om wijzigingen in te zien, oude code terug te halen indien noodzakelijk en gecontroleerd nieuwe code aan het systeem toe te voegen.

3.1.2. Agile Scrum

Om het project in goede banen te leiden wordt er gebruik gemaakt van Agile Scrum. Dit is een flexibele projectmanagementmethode, waarbij de tijd opgedeeld in iteraties met een vaste lengte, zogenaamde sprints. Bij aanvang van een sprint vindt overleg plaats en wordt in overleg met de opdrachtgever bepaald welke onderdelen de meeste prioriteit hebben. Het team stelt taken op en schat de benodigde tijd in. De taken komen op het Scrum bord te staan.

Figuur 8 - Het Scrum bord

Voor een onderzoeksproject waarbij de richting van het project kan veranderen is deze methode zeer geschikt omdat op die veranderingen kan worden ingespeeld.

3.1.3. Communicatie

Bij de aanvang van het onderzoek zijn enkele afspraken gemaakt over de wijze waarop de communicatie tussen de studenten en externe betrokkenen, zoals de docentbegeleider en opdrachtgever, verloopt.

Overeengekomen is dat de opdrachtgever op de hoogte wordt gehouden tijdens wekelijkse scrum meetings. In het geval van ernstige problemen of het dreigen te mislukken van een

[5] Website: http://git-scm.com/

sprint/functionaliteit zal het team ad-hoc contact opnemen. Andersom is het team ook op verzoek van de begeleider beschikbaar voor afspraken.

De communicatie verloopt in eerste instantie via e-mail en kan eventueel worden opgevolgd met een afspraak.

3.1.4. Kwaliteitsbewaking

Binnen het onderzoek zal de kwaliteit bewaakt worden door middel van voortgangsgesprekken met de opdrachtgever, het beoordelen van code van andere projectleden, documenten en het stellen van kwaliteitseisen aan het eindproduct.

Zowel voor alle ontwerpen, documenten en software is terugkoppeling van de opdrachtgever noodzakelijk. Zeker in de beginfase van het project is zijn goedkeuring noodzakelijk ter verificatie van de kwaliteit van het opgeleverde werk.

De kwaliteitsbewaking voor de documenten vindt plaats doordat de teamleden alle uitgaande documenten doorlezen, controleren en corrigeren om de kwaliteit te waarborgen.

De kwaliteitseisen voor de software zullen gebaseerd zijn op de eisen van de opdrachtgever. Voor de kwaliteitswaarborging van de software is het van belang om aan deze eisen te voldoen, tijdens de sprint meetings wordt waar mogelijk de functionaliteit gedemonstreerd. Dit stelt de opdrachtgever in staat in te grijpen en vergroot de kans dat de opdrachtgever tevreden is met het eindresultaat.

3.2. Werkwijze en projectactiviteiten

Het project kan grofweg worden verdeeld in vier hoofdfases; onderzoek, ontwerp en development en testen. Hierna komen nog twee afrondende fases; evaluatie en overdracht. In deze paragraaf wordt er per fase behandeld welke activiteiten zijn uitgevoerd.

3.2.1. Onderzoek

- Inwerken en het huidige systeem onderzoeken;
- Bestaande documentatie lezen en zonodig bijwerken;
- In kaart brengen wensen van opdrachtgever;
- Kennis maken met ROS;
- Onderzoek naar multi-agentsystemen en platformen hiervoor (waarbij uiteindelijk is gekozen voor JADE);
- Onderzoek naar blackboards.
- Onderzoek naar een koppeling tussen ROS en JADE;
- Werken aan en inleveren plan van aanpak.

3.2.2. Ontwerp

- In overleg met de opdrachtgever een architectuur voor herconfiguratie ontwerpen;
- Een architectuur ontwerpen gebaseerd op agent technologie voor het totale systeem in overleg met alle betrokkenen;
- Een abstracte taal definiëren. Dit wordt de taal waarin de producten abstract beschreven kunnen worden door de agents. Deze taal zal verschillende hiërarchische niveaus hebben en zal er dus ook anders uitzien op die niveaus.

3.2.3. Development

- De ontworpen architecturen implementeren.
    - Hierbij horen ook alle specifieke onderdelen van de architecturen, zoals databases, blackboard, scheduling e.d.
- De Abstract Functionality Discription Language uitwerken en implementeren;
- Een demonstrator ontwikkelen

3.2.4. Testen

- Het multi-agentsysteem en de bijbehorende onderdelen grondig testen en de abstracte taal fine-tunen.

3.2.5. Evaluatie

- Onderzoeken of het multi-agentsysteem en bijbehorende geschikt is voor vervolgonderzoeken en heeft voldaan aan de opdracht.
- Verbeterpunten zoeken en opschrijven.

3.2.6. Overdracht

Het is voor vervolgprojecten belangrijk dat er een goede overdrachtsdocumentatie is. Tijdens het project moet er daarom voor gezorgd worden dat alle documentatie op orde is.

3.3. Gebruikte methodieken en tools

Voor dit project zijn een aantal specifieke methodieken en tools gebruikt. In deze paragraaf wordt uitgelegd welke dat zijn en waarom juist voor deze oplossingen is gekozen.

3.3.1. Agent architectuur

Voor het op een agent architectuur gebaseerde productiesysteem is uiteraard een agent architectuur nodig. Er is onderzoek gedaan om te bepalen welk agent architectuur het beste past bij dit project. Al snel viel de keuze op JADE (Java Agent DEvelopment Framework). Dit is een op Java gebouwd framework voor multi-agentsystemen. Juist omdat dit framework op Java is gebouwd is de verwachting dat hiermee vrij snel een werkend systeem gebouwd kan worden. Ook is dit framework al ruim tien jaar in ontwikkeling, wat dit een erg volwassen platform maakt en ondersteund het framework de FIPA standaard voor agent communicatie. Hiermee kan in een later stadium zelfs nog met andere typen agent frameworks worden gecommuniceerd.

3.3.2. ROS

ROS (Robot OS) wordt gebruikt door de reeds bestaande deltarobot. ROS (Robot Operating System) is een framework met OS achtige features. Het framework bestaat uit zogenaamde ROS nodes die met behulp van services en topics kunnen samenwerken. Het maakt daarbij niet uit of nodes allemaal op dezelfde machine draaien of verspreid zijn over meerdere machines. Doordat de deltarobot al geprogrammeerd is in ROS en deze code ook goed modulair is opgebouwd kan dit worden opgenomen in ons project. ROS wordt geprogrammeerd in C++.

3.3.3. Blackboard

Voor de opslag van product- en productiedata worden blackboards gebruikt. Deze zijn vrij toegankelijk voor iedere agent die er in geïnteresseerd is. Elke agent kan dus data lezen en schrijven

naar elk blackboard. Omwille van de tijd en omdat voor deze opdracht niet het wiel opnieuw uitgevonden moet worden is er besloten om het blackboard niet zelf vanaf niets te implementeren maar te zoeken naar een geschikte al bestaande implementatie. Er is hierdoor gekozen voor Open Blackboard System. Het grote voordeel van deze implementatie van het blackboard is dat hij geschreven is in Java en hierdoor eenvoudig met het multi-agentsysteem geïntegreerd kan worden. Omdat ook het ROS-gedeelte, dat geschreven is in C++, met het blackboard moet kunnen communiceren is ervoor gekozen om het blackboard als server te implementeren.

3.3.4. Data serialisatie

De blackboardserver staat ergens binnen het agent netwerk. Om data op het blackboard op te kunnen slaan zal er data naar de blackboardserver over het netwerk moeten worden verstuurd. Hier zijn een aantal serialisatiemogelijkheden voor. Serialisatie is het omzetten van objecten naar een serie bytes, die vervolgens kunnen worden opgeslagen of worden verstuurd. Als er alleen maar een Java applicatie zou zijn, zou de implementatie van serialisatie in Java een goede eenvoudige oplossing zijn. Dat is echter niet het geval dus dit is geen oplossing. Serialisatie van XML en dat oversturen naar de server is de meest bekende oplossing. Echter er zitten veel nadelen aan. Zo gebruikt het veel ruimte en kan deze oplossing erg performance intensief zijn bij het coderen en decoderen van XML bestanden. Ook het doorzoeken van een XML-boom kan erg complex zijn. Vanwege al deze nadelen is er onderzoek gedaan voor een betere oplossing dan XML. De oplossing die is gevonden is Protocol Buffers. Protocol Buffers is een project van Google. Hierbij kan een van te voren gedefinieerde datastructuur in een Protocol Buffer bestandformaat worden gecompileerd. Hiervan kunnen zowel Java als C++ klassen worden gegenereerd inclusief methodes om de datastructuren te vullen, uit te lezen, te serialiseren en te deserialiseren. Hierdoor ontvangt de server van zowel de Java als de C++ exact dezelfde bytes bij dezelfde aanroep. Dit maakt het eenvoudig om Java en C++ te combineren.

3.4. Research

In deze paragraaf wordt vermeld wat er tijdens dit project is onderzocht.

3.4.1. Blackboard

Er is onderzoek gedaan naar verschillende implementaties van blackboardsystemen. De twee met de meeste waarde voor dit project zijn hier vermeld.

GBBopen

Dit is de meest volwassen en doordachte blackboard implementatie die tijdens het onderzoek naar voren is gekomen. Echter GBBopen is geschreven in Common Lisp en daarom niet echt bruikbaar voor het project. Er is wel zeer veel documentatie te vinden bij dat project, waarin de blackboard architectuur uitgebreid werd behandeld.

Open BBS

Dit is een Java implementatie van een blackboard systeem en een groot framework er om heen. Dat het in Java was geschreven maakt het natuurlijk interessant omdat de agents ook in Java geschreven zijn in JADE. Dit framework is hierdoor uitgebreid onderzocht**.** Open BBS is uitgebracht onder de Apache licentie. Dit heeft als voordeel dat het geen probleem is om de code uit te breiden of aan te passen. Dit bleek later nodig te zijn vanwege een aantal bugs en onvolledigheden in het systeem. De

interne datastructuur van het blackboard ziet er verder goed uit en bleek na een aantal tests ook te gebruiken. Door een gebrek aan documentatie was het framework erg onduidelijk. Het framework bleek hierdoor niet bruikbaar. Omdat dit framework toch niet nodig is, is er voor deze blackboard implementatie gekozen.

Deze blackboard implementatie is getest in combinatie met JADE door meerdere simpele agents iets naar het blackboard toe te schrijven en andere agents dat weer van het blackboard te laten lezen. Het belangrijkste punt dat hier duidelijk wordt is dat het schrijven erg goed presteert, maar het lezen wat langzamer gaat. Dit heeft alles te maken met filtersysteem van het blackboard. Om een bepaald object van het blackboard te lezen moet elk object getoetst worden met een filter waar dat object aan moet voldoen. Dit kan een potentiële bottleneck vormen. Er moet dus tijdens de implementatie van de filters goed rekening gehouden worden met hoe snel een object uit te lezen is op basis van een filter.

3.4.2. Koppeling tussen ROS en Java

Voordat de blackboardserver oplossing er was zijn er andere manieren onderzocht om de C++ ROS kan met Java en JADE te koppelen. De twee potentiële oplossingen die zijn onderzocht zijn JNA en ROS Java.

JNA

Java Native Access is een techniek die gezien kan worden als een soort brug tussen de Java virtual machine, waar Java code wordt uitgevoerd en native code. Native code is code die rechtstreeks als instructies wordt uitgevoerd op de machine waar deze wordt uitgevoerd. Code die uit de programmeertalen C en C++ wordt gecompileerd is ook native code. Met JNA kan ook dat soort code worden uitgevoerd door deze vanuit Java aan te roepen. Het was dus ook mogelijk om een kleine bibliotheek in C te maken vanuit waar ROS code werd aangeroepen. Hiermee was het mogelijk om het programma dat draaide onder Java zich te laten gedragen als ROS node. Het grootte probleem met JNA was dat het een stuk moeilijker bleek om vanuit de native code informatie terug te sturen naar het Java programma, waar JNA eigenlijk niet voor bedoeld is. Om deze reden is ervoor gekozen deze oplossing niet te gebruiken.

ROS Java

Voor ROS bestaat er een project in Java. Deze is mede ontwikkeld door Google voornamelijk om ROS ook op Android devices te laten draaien. Het is een volledige kloon van ROS, maar zou ook gecombineerd kunnen worden met ROS, omdat het protocol tussen de ROS nodes hetzelfde is. Het is een project wat nog zeer in ontwikkeling is, maar het heeft potentie. Het probleem zit hem in de combinatie met JADE. Het bleek onmogelijk om een Java klasse te starten die zowel een JADE agent is als een ROS Java node interface gebruikt. Voor beide is relatief ingewikkeld opstartscript nodig die met elkaar conflicteren. Om deze reden is ook deze oplossing niet bruikbaar.

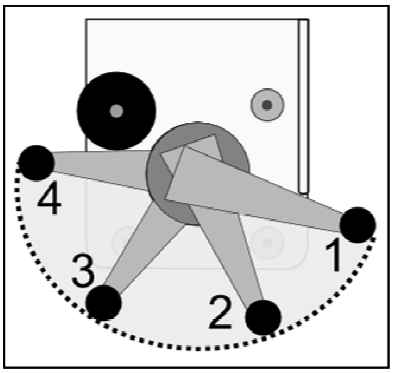
3.4.3. Overige research

Tijdens het project zijn er nog een aantal kleine onderzoeken geweest die indirect gerelateerd waren aan de opdracht. Deze onderzoeken zijn hier vermeld.

Verbeteren beeldherkenning

Tijdens dit project is er geprobeerd de beeldherkenning van de camera te verbeteren. Het doel hierbij was dat er geen lampen als backlight meer nodig zouden zijn om de QR codes te herkennen.

Om dit te kunnen bereiken is contrast stretching geprobeerd. Hierbij wordt de range van pixelkleuren van het camerabeeld opgerekt naar het hele spectrum. Het probleem was dat het licht niet goed verdeeld was onder de glasplaat zodra er geen lampen meer werden gebruikt. Hierdoor konden de QR-codes soms aan de ene kant wel worden uitgelezen, waar nog enigszins achtergrondlicht was en de andere kant waar het vrij donker was niet. Voor een goede permanente oplossing is het belangrijk dat de ranges die gekozen worden voor het contrast stretching dynamische moeten zijn. Voor een goede uitwerking hiervan was tijdens dit project geen tijd dus is de oude situatie weer teruggedraaid.



Figuur 9 - Motorkalibratie, arm beweegt van 1 naar 4

Stappenmotors automatisch kalibreren

Tijdens dit project kwam het vaak voor en voornamelijk in de eerste maanden dat er met de robot een demo gegeven moest worden. Hiervoor moest de robot elke keer worden gekalibreerd. Het kalibratiesysteem wat het vorige projectteam had achtergelaten werkte erg vervelend en duurde lang. Na onderzoek naar stappenmotors te hebben gedaan is er geprobeerd de kalibratie automatisch te laten uitvoeren. Het zou zo zijn dat stappenmotors ongeacht of ze ergens tegenaan komen, bij de deltarobot een object waarvan de exacte hoek ten opzichte van de stappenmotor bekend is, hun huidige positie intern blijven onthouden waardoor de juiste positie gekalibreerd wordt. Echter de stappenmotors die de deltarobot gebruikt slaan als het ware terug als ze ergens tegenaan komen. Hierdoor is het niet mogelijk de motor goed automatisch te kalibreren. Om toch minder tijd kwijt te zijn aan het kalibreren is er voor gekozen om de stappenmotor automatisch te laten draaien totdat de arm bijna tegen het object waarvan de exacte hoek bekend is aan te laten komen. Het laatste gedeelte van de kalibratie gebeurt dan alsnog handmatig.

3.5. **Taakverdeling**

Voor het project is de volgende taakverdeling tot stand gekomen tussen de projectleden.

	Taak

Samen	• Bestaande resultaten verkend
	• Wensen van de opdrachtgever in kaart brengen
	• Ontwerp gemaakt voor het productiesysteeem.
	• Inrichten van de repository
	• Onderzoeken
	  o Vereenvoudigen kalibratie
	  o Verbeteren beeldherkenning
	  o Blackboardsoftware onderzoeken
	  o Koppeling tussen Java en ROS
	• Aanpassen van bestaande ROS software voor communicatie met het agentsysteem

Pascal	• Buildsysteem • Autoconfiguratiesysteem • Ontwerp AFDL • Onderzoeken o Stuctuur repository onderzoeken
Martijn	• Agentsysteem • Blackboardserver • Onderzoek o Agentsoftware onderzoeken

4. Ontwerp

Dit hoofdstuk beschrijft het ontwerp dat is gemaakt van het totale productiesysteem. Hierbij wordt een globaal ontwerp getoond en zullen de agents worden beschreven. Ook zullen de ontwerpkeuzes worden toegelicht en verdedigd. Tevens zullen de ontwerpkeuzes van andere aspecten van het systeem aan bod komen; het blackboard en de data.

4.1. Het overall-design

Deze paragraaf beschrijft een globaal ontwerp van alle agents en ROS-nodes die het productiesysteem zullen aansturen.

Het productiesysteem bestaat uit twee te onderscheiden delen. Er zijn ROS-nodes die met elkaar kunnen communiceren, actuatoren aansturen en sensoren uitlezen; daarnaast is er een overkoepelend slim systeem dat is opgebouwd uit agents die met elkaar communiceren. Het overkoepelende systeem is verantwoordelijk voor het uitvoeren en overzien van de productie van een grid van equiplets, het op ROS gebaseerde deel zorgt voor de hardwareaansturing die de productie mogelijk maakt.

Het gemaakte ontwerp probeert deze twee delen te definiëren en samen te voegen tot één groot coherent productiesysteem.

4.1.1. Theoretisch ontwerp opdrachtgever

In de afstudeerorganisatie is al nagedacht over een softwareoplossing op basis van agents. In [9] wordt dit vrij goed uiteengezet. Dit vormt de grote lijnen waaraan het nieuwe ontwerp moet voldoen. Binnen die geschetste kaders is geprobeerd om een praktische oplossing te implementeren.

Kort samengevat is in dit idee van de opdrachtgever sprake van een productagent en een equipletagent. De productagent probeert het product dat het vertegenwoordigd geproduceerd te krijgen en weet wat er moet gebeuren om dat gedaan te krijgen. De productagent weet alle informatie over het product. Om de productie plaats te laten vinden wordt onderhandeld met de equiple agent. Die kan vertellen welke stappen uitgevoerd kunnen worden. Vervolgens zal de productagent proberen ervoor te zorgen dat productiestappen worden ingepland bij de equiplets. Deze planning zal meteen volledig worden gemaakt (tot het product af is), zodat van tevoren kan worden bepaald of productie al dan niet mogelijk is.

De equipletagent weet welke productiestappen het biedt en hoe de hardware aangestuurd moet worden. Tijdens het uitvoeren van een stap ten behoeve van een product geeft de productagent aan wat er moet gebeuren met welke parameters.

In grote lijnen neemt het nieuwe ontwerp deze ideeën over. Er kleven echter wel enkele bezwaren aan.

De productagent heeft nog veel meer taken en verantwoordelijkheden dan hier worden beschreven en doet bijna alle slimme taken in het systeem. Om het mogelijk te maken functionaliteit goed te kunnen overzien en los te kunnen testen, zeker binnen de beperkte tijd, is het noodzakelijk om een

systeem wat zo complex is, iets pragmatischer te implementeren. Er is voor gekozen om alleen de functionaliteit te implementeren die nodig is voor de casus die opgeleverd zal worden, maar ook om het systeem in meerdere losse agents te splitsen.

Er kleven ook enkele praktische bezwaren aan een productagent die veel kan, mag en weet. De productagent zoals die er is moet altijd weten hoe er met de equiplet dingen uitgevoerd moeten worden. Om te kunnen weten of de modules van een equiplet een bepaalde actie echt kunnen uitvoeren is het noodzakelijk om constraints en randvoorwaarden te snappen. De productagent moet daardoor al veel details weten over de onderliggende hardware. Een van de ontwerpdoelen die zijn opgesteld is dat we willen dat de productagent zo min mogelijk weet over de hardware waarmee de productiestappen worden uitgevoerd. Zo'n afhankelijkheid zou er immers toe kunnen leiden dat de productagent aangepast zou moeten worden als er nieuwe modules worden ontwikkeld. Dat is niet acceptabel, omdat dit ook betekent dat de productagent ingewikkelder wordt als het voor de productie uit verschillende productiegrids zou kunnen gaan kiezen.

Een ander praktisch probleem is de performance, en dan met name de schaalbaarheid, van het systeem. Er zijn meerdere knelpunten. Als de productagent heel veel moet doen betekent dit dat er veel rekenkracht nodig is. Agents zijn makkelijk te distribueren over meerdere systemen, maar één agent is niet zomaar uit te smeren over meerdere systemen. Maar het grootste bezwaar voor de schaalbaarheid is het planningsproces.

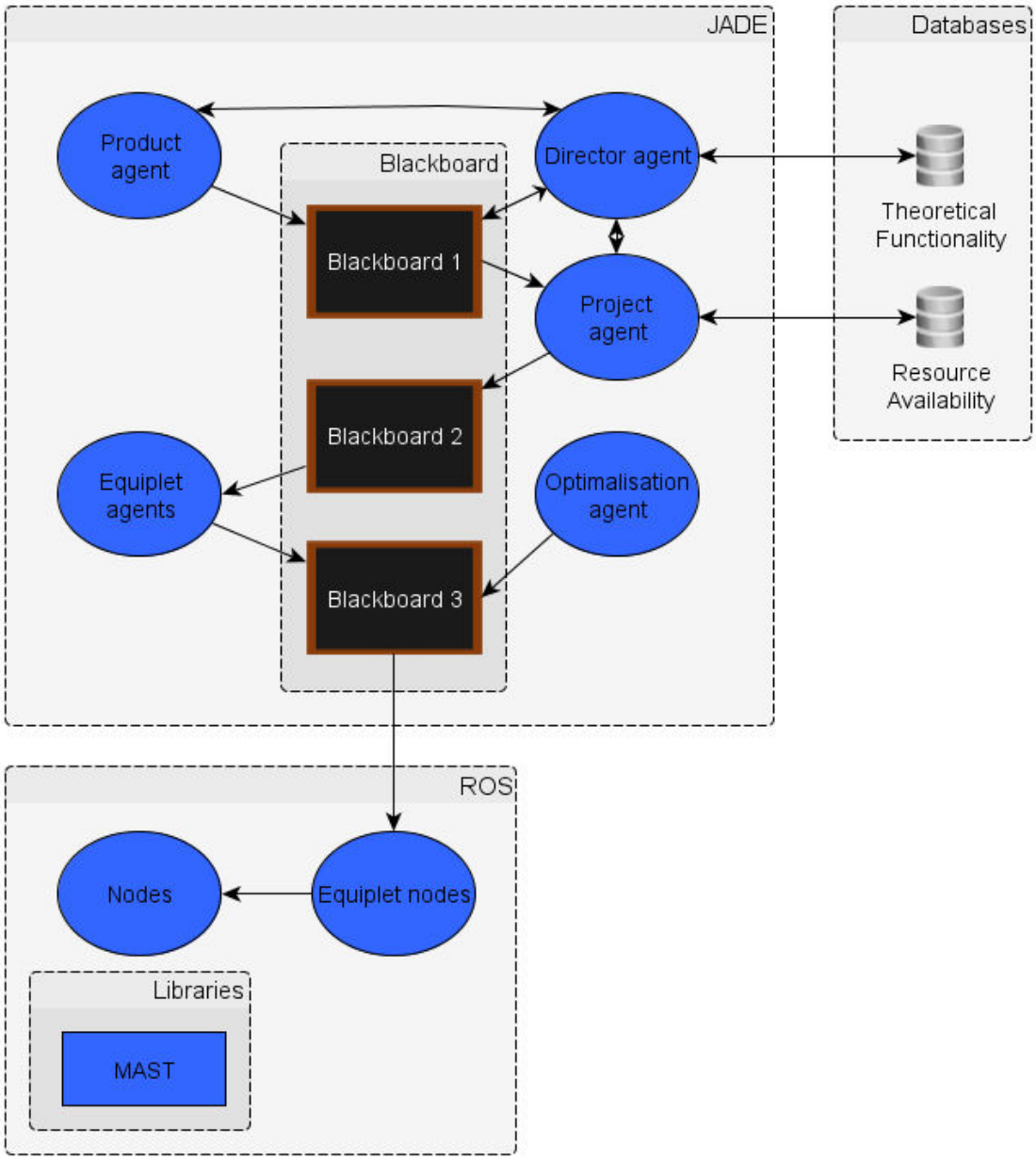
Als een productagent gemaakt moet worden moet de productagent vragen welke equiplets deel van het systeem uitmaken. Vervolgens moet gevraagd worden aan deze equiplets wat ze kunnen. De productagent moet deze beschrijvingen vergelijken met wat het wil. Vervolgens moet voor het plannen van de stappen gekeken worden naar al die equiplets.

Elke productagents moet dit elke keer doen als een product gepland wordt. In een groot productiegrid, zeker als er steeds meer producten gepland moeten worden brengt dit een grote overhead met zich mee.

4.1.2. Het ontwerp

Het globale ontwerp geeft een overzicht van het gehele productiesysteem en voegt het ROS systeem en het multi-agentsysteem samen. In de volgende paragrafen worden de verschillende onderdelen beschreven.

Figuur 10 - Het globale ontwerp



4.2. De agents

4.2.1. Productagent

Het doel van de productagent is ervoor zorgen dat het product die hij representeert wordt geproduceerd. Een productagent heeft nog veel meer taken [10], maar deze worden in dit systeem vertegenwoordigd door andere agents.

Een productagent bevat de abstracte beschrijving van hoe een product geproduceerd moet worden. Deze beschrijving is een taal op hoog niveau van alle productiestappen die nodig zijn om het product

te produceren. Hoe deze beschrijving in elkaar zit en hoe deze is gedefinieerd, is te lezen in de scriptie van Pascal.

Naast het bovenstaande weet de productagent ook extra informatie over het product, bijvoorbeeld de deadline wanneer deze klaar moet zijn.

In dit ontwerp vraagt de productagent aan de directoragent of het productiesysteem de theoretische kennis bevat om al de productiestappen uit te kunnen voeren. De productagent zet hiervoor al zijn productiestappen op het eerste blackboard. Zodra er meerdere directoragents blijken te zijn betekend dat er meerdere productieplatformen aanwezig zijn. De productagent moet dan op basis van een aantal variabelen, zoals de huidige belasting van het productiesysteem kunnen kiezen bij welke directoragent hij zich aanmeldt ervan uitgaande dat het product bij meerdere platformen gemaakt kan worden.

Zodra de productagent van de directoragent te horen krijgt dat het productiesysteem het product theoretisch kan maken zal de directoragent een projectagent toewijzen aan het product en zal de productagent wachten op verdere input van het productiesysteem. Dit betreft in eerste instantie het seintje van een projectagent (zie hoofdstuk 4.2.3 projectagent) of het product daadwerkelijk voor de opgegeven deadline gemaakt kan worden en later eventuele informatie over problemen en ten slotte informatie dat het product voltooid is.

4.2.2. Directoragent

Het doel van de directoragent is het checken of producten theoretisch gemaakt kunnen worden aan de hand van de productiestappen van de producten en modules die in het systeem zouden kunnen zitten. Dit zijn alle modules die bij het systeem aanwezig zijn, maar niet per definitie online of direct beschikbaar hoeven te zijn.

Tevens is het doel van de directoragent het op de hoogte houden van alle andere agents in het systeem over de huidige tijdstap. Dit is een essentiële taak, want project- en equipletagents moeten weten wat de huidige tijdstap is aangezien zij daar hun taken op baseren. Elk equiplet kan op één tijdstap één productiestap uitvoeren en is dan dus bezet door die ene productiestap. Voor dit prototype wordt uitgegaan van een vaste waarde voor een tijdstap.

Om met de afhankelijkheden tussen productiestappen om te kunnen gaan, is het gebruik van tijdstappen noodzakelijk. Het moet immers mogelijk zijn om stappen achter elkaar in te plannen en tussendoor van equiplet te wisselen. Het wordt erg veel werk om de situatie bij te houden en in te plannen als alle acties een andere lengte hebben. Het zou bijvoorbeeld erg moeilijk zijn om te bepalen of een bepaalde stap op een equiplet plaats zal vinden voor of na een andere stap op een andere equiplet. Het invoeren van tijdstappen zorgt ervoor dat tijdstap n altijd wordt gevolgd door tijdstap n+1 op alle equiplets tegelijk. Daarnaast vergemakkelijkt het de implementatie omdat productiestappen in arrays kunnen worden opgeslagen met de huidige tijdstap als index.

Per productiesysteem is er maar één directoragent. Nieuwe productagents melden zich bij de directoragent en publiceren hun productiestappen op het eerste blackboard. De directoragent gaat dan per productiestap controleren aan de hand van de Theoratical Functionality database of de productiestap uitgevoerd zou kunnen worden door het systeem. Zodra dit voor al de stappen geldt, stuurt de directoragent een positief bericht naar de productagent en wijst hij een projectagent toe

aan het product. Als er geen geschikte projectagent in het systeem is dan zal er eentje worden gecreëerd door de directoragent. Als niet alle stappen mogelijk uitgevoerd kunnen worden door het systeem krijgt de productagent een negatief bericht van de directoragent.

4.2.3. Projectagent

Het doel van de projectagent is het zorgen dat het productie wordt uitgevoerd en het product gemaakt wordt namens de productagent.

Een productagent kent een of meerdere equipletagents en weet wat hun functionaliteiten zijn. Deze informatie komt uit de Resource Availability database. Zodra het systeem wordt geherconfigureerd of wanneer er een module in een andere toestand komt zal de database worden aangepast.

Zodra een product theoretisch gemaakt kan worden volgens de directoragent, zal de projectagent aan de hand van de database controleren of het product nu gemaakt kan worden. Dit gebeurd aan de hand van de informatie over beschikbare modules die op dat moment online zijn en of alle onderdelen voor het product aanwezig zijn. Zodra op basis van die informatie blijkt dat het product gemaakt kan worden geeft de projectagent een seintje aan de productagent dat de productie wordt gestart. De projectagent haalt alle productiestappen van het eerste blackboard en gaat ze schedulen over de equiplets door productiestappen aan ze toe te wijzen op het tweede blackboard.

Voor het schedulen is het belangrijk dat de projectagent de huidige tijdstap doorkrijgt van de directoragent. Tijdstappen zijn essentieel voor scheduling, omdat producten in een bepaalde volgorde gemaakt moeten gaan worden en producten tussen verschillende equiplets moeten worden getransporteerd. In het geval dat scheduling niet mogelijk is wordt er gerescheduled. Hierbij communiceert de projectagent met alle andere projectagents in het systeem om al hun geschedulde stappen vrij te geven op het tweede blackboard. De projectagent zal dan al deze stappen van het blackboard halen en een poging doen om aan de hand van een bepaald schedulingsalgoritme [11]alle producten opnieuw te schedulen. Zodra dit lukt, zal de projectagent de stappen in de nieuwe scheduling op het blackboard plaatsen. Zodra de rescheduling ook mislukt, zal de projectagent de oude scheduling weer op het blackboard zetten. Dit betekend wel dat het nieuwe product niet gemaakt kan worden. Er zal dan met de productagent moeten worden gecommuniceerd dat er een probleem is.

4.2.4. Equipletagents

Het doel van de equipletagents is het uitvoeren van productiestappen door middel van het creëren van ‘’werkbriefjes” voor hun ROS equipletnode. Dit kan hij doen door de productiestappen te vertalen. Ook moeten equipletagents ervoor zorgen dat de database bijgewerkt blijft zodra hun onderliggende modules worden geherconfigueerd.

Een equipletagent controleert het tweede blackboard of er nieuwe productiestappen voor hem op zijn gezet. Zodra dit het geval is leest hij deze en zal hij deze gaan vertalen in een aantal werkbriefjes voor zijn ROS modules. Deze werkbriefjes worden op het derde blackboard gezet. Vooral de volgorde van de briefjes is belangrijk, omdat de verschillende modules vaak moeten wachten op input van een vorige module om hun actie uit te voeren.

4.2.5. Optimalizationagent

Het doel van de optimalizationagent is de werkbriefjes van het derde blackboard in een optimale volgorde te zetten voor de onderliggende ROS modules. Hierbij wordt rekening gehouden met werkbriefjes die concurrent kunnen worden uitgevoerd op basis van de informatie op werkbriefje. Deze informatie kan bijvoorbeeld de afhankelijkheid zijn tussen de modules. De werkbriefjes worden uitgevoerd binnen één tijdstap. De optimalizationagent scheduled binnen één tijdstap de werkbriefjes op wat gedefinieerd zou kunnen worden als subtijdstappen.

Deze agent wordt in dit project nog niet geïmplementeerd.

4.3. ROS systeem

Naast het multi-agentsysteem is er ook nog het onderliggende ROS systeem. Deze voert alles uit wat er van boven wordt verteld dat het moet gaan doen. it systeem is onder te verdelen in twee hoofdcomponenten. De equipletnode en modules.

4.3.1. Equipletnode

Er is één equipletnode per equiplet dus ook per equipletagent. Het doel van de equipletnode is werkbriefjes van het blackboard lezen en op basis van wat daar in staat de juiste module aanspreken met de juiste functie en parameters. Bij eventuele output van een module is de equipletnode in staat om deze als input van een andere module te gebruiken.

4.3.2. Modules

Een module is als een werkpaard in het systeem. Voor elke specifiek onderdeel van het systeem is er een module. Bijvoorbeeld een visionmodule die onderdelen kan lokaliseren of een deltarobotmodule die bij input van coördinaten daarheen kan bewegen. Qua uitvoering van taken zit er verder geen intelligentie in een module De intelligentie zit bij de herconfiguratiekant van het systeem. Een module moet zijn status kunnen doorgeven naar de equipletnode zodra deze veranderd. Er zijn al modules geïmplementeerd door voorgaande projectgroepen. De verwachting is dat enkel de interface waarmee de module wordt aangesproken moet worden veranderd.

4.4. Dataholders

Deze gedeeltes van het systeem bevatten data. Op basis hiervan zal het systeem zijn acties uitvoeren. Er zal hier worden ingegaan op hoe de blackboards en de databases in het systeem worden gebruikt.

4.4.1. Blackboards

Er worden in het ontwerp van het systeem drie blackboards gebruikt. Het eerste blackboard wordt gebruikt voor productagents om hun stappen op te publiceren. Het tweede blackboard wordt gebruikt om die productiestappen op equipletagents te schedulen. Voor elke equiplet zegt het per tijdstap wat de equiplet moet gaan doen. Productiestappen kunnen gerescheduled worden. Tijdstappen die in de toekomst liggen kunnen hierdoor worden aangepast. Er kunnen dus veel wijzigingen optreden op dit blackboard. Op het derde blackboard worden taken geplaatst voor de verschillende modules van de equiplet. Hier staat precies op wat die module moet gaan doen, want

de vertaling van de productiestap heeft ervoor al plaatst gevonden. De taken worden geplaatst op subtijdstappen die worden bijgehouden door de equipletnode.

Omdat de blackboards een belangrijke rol spelen tijdens het gehele proces van plannen, maar ook tijdens de productie zouden ze een belangrijke bottleneck in het systeem kunnen vormen. Als het systeem vergroot wordt is het belangrijk om voortdurend te testen of de blackboards geen problemen opleveren. Omdat het blackboard een conceptuele entiteit is, is het mogelijk om in de toekomst het blackboard te vervangen door een oplossing die betere performance biedt.

4.4.2. Databases

In het ontwerp zijn twee databases te vinden. De theoretical functionality database en de resource availability database. Hier wordt vermeld wat hun doel is.

Theoretical Functionality Database

Het doel van deze database is het bijhouden van de productiestappen die theoretisch kunnen worden uitgevoerd met het productiesysteem zodra de juiste machines en onderdelen aanwezig zijn in het systeem.

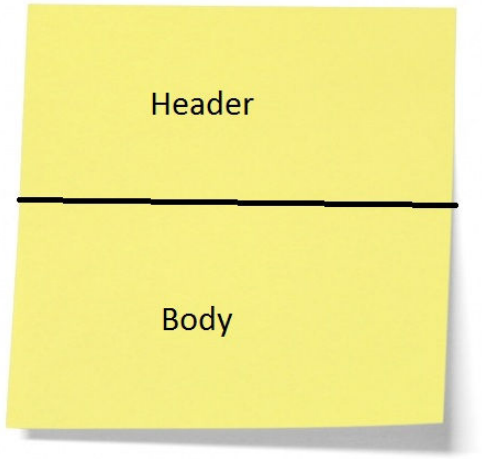
Resource Availability Database

Het doel van deze database is het bijhouden van welke onderdelen van het systeem op dit moment beschikbaar zijn. Dit geldt zowel voor voorraad van productonderdelen als voor de MAST status van de onderliggende modules van equiplets. Deze database bevat een subset van de productiestappen die in de theoretical functionality database zijn opgeslagen. De database wordt vaak bijgewerkt aan de hand van input van het agentsysteem.

4.5. **Data**

In deze paragraaf zal worden ingegaan op de data die wordt opgeslagen en wordt verstuurd in het ontwerp. Als eerste zal worden ingegaan op de data die op het blackboard wordt opgeslagen en vervolgens op hoe een productiestap eruit ziet en hoe deze is gedefinieerd.

Figuur 11 - Een PostIt



4.5.1. Op het blackboard

Data op het blackboard kan worden gezien als een soort Post-it. De ene agent plakt deze op het blackboard en de andere haalt hem er vanaf. De informatie op deze Post-its moet aan de ene kant zo generiek mogelijk zijn opgebouwd zodat ze voor alle drie de blackboards zijn te gebruiken en aan de andere kant specifieke data bevatten voor de taak die op de Post-it staat. De Post-it kan daarom worden opgedeeld in twee gedeeltes. Een header en een body. In de header staat verplichte metadata en de body is vrij in te vullen door de agents.

In deze tabel staan de metadata velden van de header.

Veldnaam	Toelichting
Owner	Hier staat de eigenaar van de Post-it. Dit veld is voor elk van de drie blackboards anders. Eerst staat hier de id van het product. Na de scheduling staat hier de id van de equiplet en op blackboard drie staat hier de id van de module die de taak moet uitvoeren.
Timestep	Dit veld geeft een zekere tijdstap aan. Op blackboard één is dit de tijdstap dat het product het productiesysteem heeft betreden en op blackboard twee en drie op welke tijdstap de Post-it is gescheduled.
Ordinal	Dit geeft het telnummer aan van de productiestap. De hoeveelste stap is het van het product.
Is Processed	Dit veld geeft aan of een Post-it al is verwerkt door de geïnteresseerde. Een Post-it kan simpelweg niet meteen worden verwijderd nadat deze is gelezen, omdat er dan mogelijk vitale informatie voor bijvoorbeeld de scheduling verloren kan gaan. Een Post-it zal door het blackboard worden opgeruimd zodra de huidige tijdstap groter wordt dan het hierboven beschreven Timestep veld. Dan is alle relevantie die deze Post-it had verdwenen.
Productname	Hier staat de verwijzing naar het product. Deze is voor al de blackboards hetzelfde.

4.5.2. Productiestappen

Om het mogelijk te maken om zo abstract mogelijk productiestappen te beschrijven is een taal bedacht. Deze taal is nodig omdat productagents abstract moeten kunnen uitleggen wat ze nodig hebben. Ook de op het systeem beschikbare functionaliteit moet abstract worden gedefinieerd. Het is noodzakelijk dat deze beschrijvingen redelijk abstract zijn omdat we het systeem willen kunnen uitbreiden met modules die niet van tevoren voorzien zijn.

In de gewenste situatie weet een nieuwe module wat hij kan en stuurt zijn abstracte beschrijving naar het multi-agentsysteem. Als de software die de productagents maakt die productiestap nodig heeft, wordt de stap op een soortgelijke wijze gedefinieerd en wordt gevraagd of het product gemaakt kan worden. Zo min mogelijk agents en ROS nodes hoeven te begrijpen hoe de hardware hoeft worden aangestuurd voor een bepaalde productiestap.

De *Abstract Functionality Description Language* (AFDL) die is gedefinieerd is een XML-formaat dat stappen en bijbehorende constraints kan beschrijven.

Aanvankelijk was het idee om te kiezen voor een natuurlijke taal die stappen beschrijft, maar daar is vanaf gezien. Het parsen van een natuurlijke taal is ingewikkeld en veel werk, maar het had ook nadelen voor de doelen die er zijn. Als een natuurlijke taal geparsed moet worden dan zijn stukken die niet begrepen worden een probleem; je weet immers niet of het veilig genegeerd kan worden of niet. Bij XML is aan tags een attribuut toe te voegen die aangeeft of het noodzakelijk is om de inhoud van die tag te begrijpen. Als de taal uitgebreid moeten kunnen worden met zo min mogelijk impact voor het gehele systeem, dan is XML een betere keuze.

Een ander groot probleem is dat natuurlijke taal heel erg moeilijk formeel te beschrijven is, bij XML zijn hier oplossingen voor zoals XML Schema (XSD) en RELAX NG. Met deze XML schema's kan de XML formeel beschreven worden. Het is dan mogelijk een de leverancier van bijvoorbeeld modules voor het systeem deze definitie te geven, waarmee gegarandeerd kan worden dat ze correcte XML genereren die door ons systeem geparsed kan worden. Het ontwerp en de uitwerking van de AFDL is te vinden in het verslag van mijn medeafstudeerder [12].

5. Technische Implementatie

In dit hoofdstuk zal de implementatie van het agentsysteem worden toegelicht en zal er in de technische details worden gedoken van het project. Hierbij wordt er alleen naar de implementatie gekeken van die gedeeltes van het systeem die ik (Martijn Beek) heb opgeleverd. Dit zijn het agentsysteem en de blackboardserver. De andere gedeeltes zijn opgeleverd door Pascal Muller. Dit zijn de databases, de taal voor het beschrijven van de productiestappen en de vertaling van een productiestap in verschillende taken voor het ROS systeem.

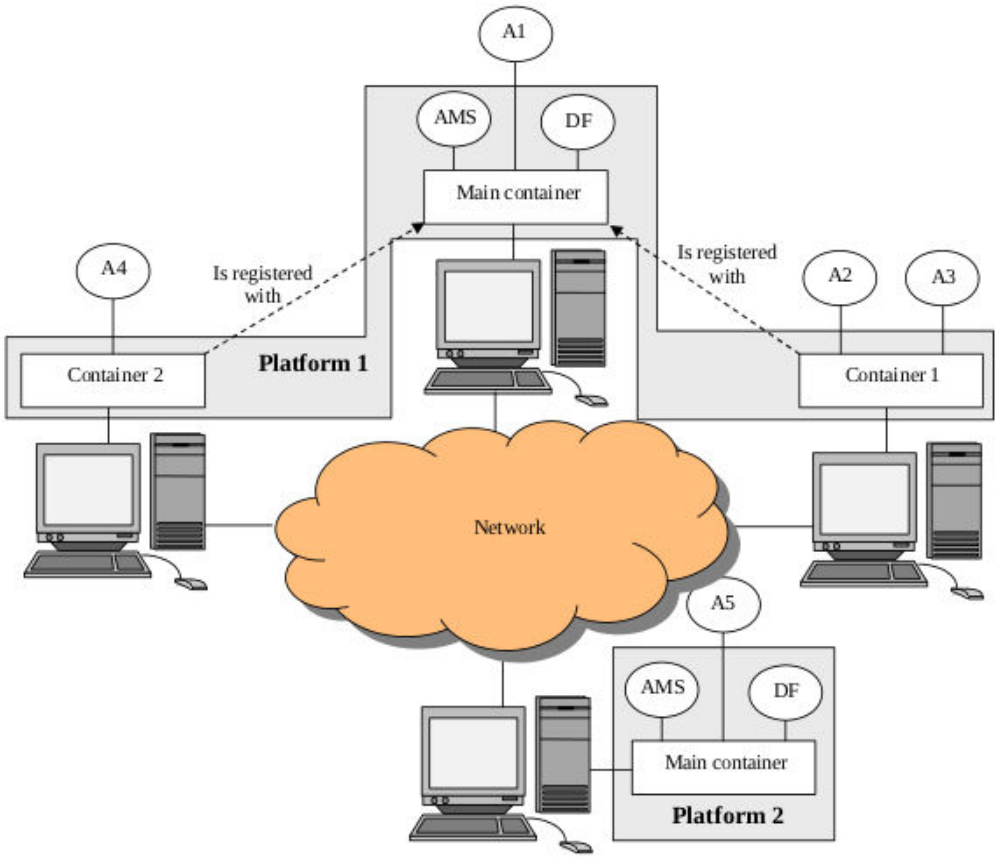
De paragrafen over het multi-agentsysteem en de blackboardserver worden ingeleid met een algemeen gedeelte over de werking van de onderliggende technieken resp. 5.1.1 t/m 5.1.3 en 5.2.1. Vervolgens wordt er in hoofdstuk 5.1 per agent uitgelegd wat zijn concrete taken zijn in de implementatie. Deze taken formuleren de concrete verantwoordelijkheden van een agent gebaseerd op de doelen die zij volgens het ontwerp (zie hoofdstuk 4.2) willen bereiken.

5.1. Multi-agentsysteem

In deze paragraaf wordt uitgelegd hoe het multi-agentsysteem werkt in JADE. Hiervoor zal eerst worden uitgelegd hoe het JADE framework eruit ziet en hoe een JADE agent eruit ziet. Vervolgens een paragraaf over communicatie tussen agents en ten slotte zullen de implementaties per agent worden behandeld.

5.1.1. JADE framework

Figuur 12 - JADE Framework



Een JADE framework bestaat uit één of meerdere platforms. Een platform bestaat op zichzelf weer uit één of meerdere containers. Per container draait er altijd één JADE runtime environment. De containers kunnen allemaal één of meerdere agents bevatten zodra hij draait op een bepaalde host. De container die altijd aanwezig is, is de hoofdcontainer. De andere containers moeten zich registreren bij de hoofdcontainer om deel uit te kunnen maken van het platform. Zodra een JADE platform wordt opgestart worden er twee belangrijke agents in de hoofdcontainer opgestart. Dit zijn de AMS en de DF agent. (Zie figuur)

AMS

De AMS (Agent Management System) is de hoofdcontroller van het agent platform. Deze bevat een service die ervoor zorgt dat elke agent die zich aanmeldt bij het systeem een unieke naam krijgt en deze agent fungeert als autoriteit binnen het platform. Zo kan hij agents creëren en doden als dat aan hem wordt gevraagd.

DF

De DF (Directory Facilitator) zorgt voor een Yellow Pages (Gouden Gids). Agents kunnen zich bij de DF aanmelden met het bericht dat ze een bepaalde service kunnen aanbieden. Als andere agents die bepaalde service nodig hebben kunnen ze bij de DF vragen om welke agents dat precies gaat. Zij ontvangen dan de ID van de agent die de service aanbiedt en kunnen vervolgens dan die service gaan gebruiken.

5.1.2. JADE agents

Een JADE agent bestaat uit een paar standaardcomponenten. Hieronder een voorbeeldklasse op basis waarvan de componenten worden uitgelegd.

Voorbeeld van een agent in JADE

```
public class MyAgent extends Agent
{
        public class MyBehaviour extends Behaviour
        {
                public void action(){
                }
                public boolean done()   {
                }
        }

        public void setup(){
                addBehaviour(new MyBehaviour);
        }

        public void takeDown() {
        }
}
```

Setup

Een JADE agent heeft altijd een setup methode. Zodra een agent wordt gestart wordt de code die daar in staat eenmalig uitgevoerd. Hierin kunnen o.a. behaviours worden toegevoegd aan de agent en services op de DF worden aangemeld.

Behaviour

Een behaviour representeert een taak die uitgevoerd kan worden door de agent en is geïmplementeerd door een object van het type Behaviour. Elke behaviour heeft twee methoden. In de action methode staat de taak die de behaviour moet uitvoeren en in de done methode wordt een boolean teruggegeven of de behaviour al dan niet voltooid is. De done methode wordt aangeroepen zodra de action afgerond is. Behaviours kunnen ook worden toegevoegd aan de agent vanuit andere behaviours. Zodra een agent meer dan één behaviour heeft worden ze gescheduled. Deze scheduling is niet pre-emptive maar coöperatief. Dit betekent dat zodra een behaviour CPU-tijd krijgt, de action methode wordt aangeroepen en dat deze volledig wordt uitgevoerd voordat de volgende behaviour wordt uitgevoerd. Dit principe heeft een aantal voordelen:

- Elke agent gebruikt slechts één Java thread.
- Betere performance binnen een agent, omdat van behaviour wisselen veel sneller is dan van Java thread wisselen.

- Er zijn geen synchronisatieproblemen tussen behaviours die dezelfde resources gebruiken.

Belangrijk hierbij is juist omdat alle behaviours in één thread draaien er zo weinig mogelijk code in een behaviour moet komen te staan. Zeker bij behaviours die constant worden aangeroepen.

Een behaviour wordt zolang de done methode 'false' teruggeeft opnieuw gescheduled. Zodra hij 'true' teruggeeft is de behaviour klaar en verdwijnt het behaviourobject.

Er zijn een aantal voorgedefinieerde standaard behaviours:

- One shot Behaviour: Dit type behaviour wordt maar één keer uitgevoerd. Onder water geeft de done methode altijd true terug.
- Cyclic Behaviour: Dit type behaviour wordt altijd weer uitgevoerd en is nooit klaar. Onder water geeft de done methode altijd false terug.
- Ticker Behaviour: Deze behaviour wordt iteratief uitgevoerd met een opgegeven tijd dat er wordt gewacht totdat hij opnieuw wordt uitgevoerd.
- Waker Behaviour: Bij dit type behaviour moet ook een tijd worden opgegeven. Zodra de tijd verloopt wordt de behaviour eenmalig uitgevoerd en is daarna klaar.

Bij normale behaviours zijn de done methodes leeg en kan door de programmeur worden bepaald hoe vaak de behaviours moeten worden uitgevoerd. Dit wordt bijvoorbeeld gedaan om grote behaviours op te delen in kleinere zodat andere behaviours ondertussen ook CPU-tijd krijgen of vanwege een complex communicatieprotocol waarbij er messages heen en weer verstuurd worden. De behaviour ziet er dan uit als onderstaand voorbeeld.

Voorbeeld van normale behaviour

```
public class MyBehaviour extends Behaviour {
	private int step = 0;
	public void action() {
		switch (step) {
		case 0:
			// Do something
			step++;
			break;
		case 1:
			// Do something
			step++;
			break;
		case 2:
			// Do something
			step++;
			break;
		}
	}
	public boolean done() {
		return step == 3;
	}
}
```

Bij dit voorbeeld wordt bij de eerste keer dat de behaviour wordt gescheduled de code bij case 0 uitgevoerd. De done methode geeft dan false terug en de behaviour wordt opnieuw gescheduled.

Zodra de behaviour dan opnieuw wordt uitgevoerd wordt de code onder case 1 uitgevoerd enzovoorts, totdat de step teller op 3 staat. Dan is de behaviour klaar en wordt afgesloten.

Take down

In deze methode komt een agent zodra ergens een kill commando wordt aangeroepen voor deze agent. Dit kan van buitenaf zijn via de AMS of vanuit de agent zelf door een eigen behaviour. Een van de belangrijkste zaken die in deze methode komt is zorgen dat de agent uit de DF wordt gehaald. Hierna verdwijnt de agent uit de container en wordt de thread gestopt.

5.1.3. Agent communicatie

Behalve communicatie via het blackboard volgens het ontwerp zullen agents ook via een andere manier moeten communiceren, omdat enkel met productiestappen communiceren niet voldoende is. Voor zaken als de synchronisatie van de tijdstappen en het aanmelden van productagents bij directoragents wordt normale JADE communicatie toegepast. Communicatie tussen agents verloopt asynchroon met behulp van ACL messages. Deze messages zijn gedefinieerd in de FIPA standaard voor agents. [13]

De belangrijkste velden van een ACL message zijn:

- Performative: Dit veld geeft het type van het bericht weer.
- Sender: De zender
- Receiver: De ontvanger
- Content: De informatie die wordt verstuurd.

De overige velden van een ACL message dienen voor het nog verder specificeren van het bericht.

Iedere agent heeft een soort brievenbus geïmplementeerd, een message queue. Zodra een agent een message krijgt, krijgt hij een seintje van de queue. Wat daar verder mee gedaan moet worden moet worden geïmplementeerd. Constant checken of de queue een seintje heeft gestuurd is geen optie aangezien dat onnodig veel performanceverlies oplevert. JADE heeft hiervoor een ‘blocked waiting’ methode.

Voorbeeld van behaviour die een message ontvangt

```
public void action() {
        ACLMessage msg = myAgent.receive();
        if (msg != null) {
                //do something
        }
        block();
}
```

De behaviour die berichten moet ontvangen kan worden geblokkeerd. Zodra er een bericht binnenkomt worden alle geblokkeerde behaviours opnieuw gescheduled, zodat het bericht verwerkt kan worden. Als dat gebeurd is kan de behaviour weer geblokkeerd worden.

Templates

Om complexere communicatie tussen agents mogelijk te maken waarbij verschillende soorten berichten heen en weer moeten worden verstuurd met verschillende behaviours is de bovenstaande code niet afdoende. Elke behaviour wil slechts één bepaald soort bericht hebben op een bepaald

moment. Er moet dus voor worden gezorgd dat de berichten uniek(er) worden. Dit kan worden gedaan met behulp van templates. Met een template kunnen een aantal condities worden opgegeven waar de message aan moet voldoen. De condities kunnen betrekking hebben op vrijwel alle velden die gespecificeerd zijn in de ACL message standaard.

Voorbeeld van behaviour die een message ontvangt met een voorwaarde

```
public void action() {
	MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
	ACLMessage msg = myAgent.receive(mt);
	if (msg != null) {
		// Do something
	}
	block();
}
```

Alleen zodra de template overeenkomt met de ontvangen message zal de action methode verder worden uitgevoerd.

5.1.4. Productagent & Directoragent

Uitgaande van het ontwerp heeft de directoragent de volgende concrete taken:

- Aanmelding van een productagent ontvangen en op basis van de informatie van het blackboard en de database bepalen of het productiesysteem het product gaat maken.
  - Zo ja? Dan moet hier een projectagent voor gevonden worden.
- Bijhouden van de huidige tijdstap en deze naar agents communiceren die hier belang bij hebben.

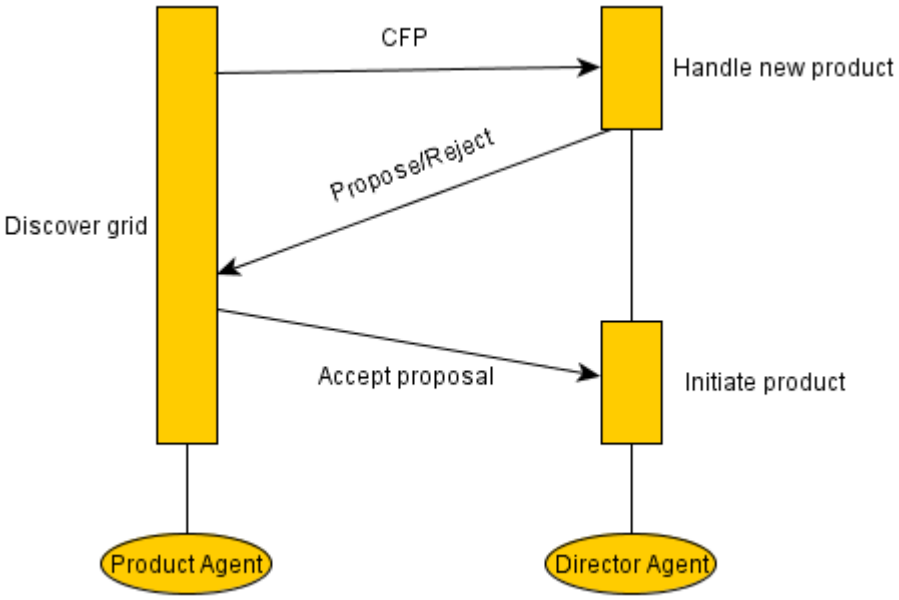
De productagent doet het volgende:

- Zich aanmelden bij directoragents op het JADE platform.
- Zijn productiestappen op het eerste blackboard van elke directoragent zetten.

Aanmelding van een productagent

Zodra een directoragent wordt gestart wordt hij aangemeld bij de DF als zijnde een directoragent met een directoragent service. Hierdoor kan een productagent bij de DF alle directoragents binnen het platform opzoeken om deze te benaderen.

Figuur 13 - Communicatiemechanisme tussen product agent en director agent



Zodra een productagent wordt gecreëerd wordt de Discover grid behaviour gestart. Deze vraagt aan de DF alle agents die de directoragent service aanbieden. Aan deze directoragents wordt een CFP (Call For Proposal) bericht gestuurd en de productiestappen worden op het eerste blackboard gezet. Hiermee vraagt de productagent de directoragent een voorstel te doen. Er wordt met behulp van een template met servicetype “Director” gezocht in de DF.

Deze agents worden opgenomen als ontvanger voor het CFP bericht. Verder wordt er nog een productobject meegestuurd. Hierin staan meta-informatie over het product o.a. de ID van het product om de productiestappen te kunnen vinden op het blackboard en de deadline van het product. De directoragent ontvangt de CFP in de Handle new product behaviour. De directoragent controleert daar aan de hand van de database of de productiestappen theoretisch uitgevoerd kunnen worden. Als dit niet kan wordt een Reject bericht teruggestuurd. Als dit wel zo is wordt een Propose bericht teruggestuurd met een getal. Dit getal is een score. In de uitwerking nu is deze 1. Hier kan later echter informatie over bijvoorbeeld de huidige productieload en andere variabelen zoals productiekosten bij op worden geteld. Hoe hoger de score hoe gunstiger het voor de productagent is om het product bij dit productieplatform te laten maken. De productagent verstuurt een Accept proposal bericht naar de directoragent met de hoogste score. Dit wordt opgevangen in de Initiate product behaviour. Vanaf hier wordt een behaviour gestart die de projectagent kiest bij dit product. (zie projectagent).

Er kunnen natuurlijk meerdere productagents zich tegelijkertijd aanmelden bij directoragents. Een belangrijke regel die wordt toegevoegd in het CFP bericht is de msg.setReplyWith(). In dit veld komt een nagenoeg unieke waarde te staan. Deze wordt weer teruggestuurd met het Proposal/Reject bericht van de directoragent. In de messagetemplate van het ontvangen van het Proposal/Reject bericht kan deze waarde worden opgenomen, zodat gegarandeerd wordt dat alleen Proposals en Rejects die bedoeld zijn voor deze productagent worden ontvangen.

Tijdstapsynchronisatie

Een belangrijke taak van de directoragent is het bijhouden van de huidige tijdstap waar het productieplatform zich in bevindt en zodra deze wijzigt dit naar project- en equipletagents communiceren. Dit is een vitale handeling, want zodra equiplets desynchroniseren zal de scheduling totaal mislukken. De lengte van een tijdstap is op dit moment een vaste waarde en wordt aangegeven door middel van een Ticker behaviour met een bepaalde intervaltijd. Hierin wordt dan een Inform message verstuurd met de huidige tijd als content. De lengte van de tijdstap is erg interessant. Deze een vaste waarde geven kan problemen opleveren, want wat gebeurd er zodra het uitvoeren van een productiestap de tijdstap overschrijdt. Een mogelijke oplossing hiervoor is dat elke equiplet een bericht stuurt dat hij klaar is met zijn werk voor de huidige tijdstap. Zodra alle berichten binnen zijn wordt dan de tijdstap verhoogd. Dit kan worden geïmplementeerd in een vervolgproject als de communicatie naar boven wordt onderzocht en gebouwd. Zeker omdat het ontvangen van de tijdstap zeer tijdgebonden is, is het belangrijk dat de andere behaviours bij ontvangende agents erg klein zijn om het bericht snel na versturing te ontvangen. Mocht in de toekomst blijken dat dit niet afdoende is, moet er overwogen worden om deze behaviours in aparte threads te draaien.

5.1.5. Projectagent

Het doel van de projectagent is het zorgen dat het productie wordt uitgevoerd en het product gemaakt wordt namens de productagent. De projectagent wordt hiervoor gestart door de directoragent.

Een projectagent heeft de volgende taken:

- Zodra een product is toegewezen controleren of in de availabilty database of het product nu gemaakt kan worden.
    - Zo ja? de stappen van het eerste blackboard halen en deze op het tweede blackboard schedulen over zijn equiplets.

Product toewijzing

Een projectagent wordt gemaakt door de directoragent. Er zijn hierbij twee principes waar rekening mee is gehouden.

- Zodra voor elk product één projectagent wordt aangemaakt kan dat performance problemen opleveren, omdat er dan teveel agents in het systeem ontstaan en daarbij ook te veel threads.
- Als er maar één projectagent is voor alle producten kan dan ook performance problemen opleveren omdat alles verwerkt moet worden in één thread.

Als oplossing is hier een middenweg voor gevonden. Zolang er nog geen projectagents zijn in het systeem wordt de eerste aangemaakt op basis van het product dat klaar staat op blackboard één. Hierbij wordt ook de meta-informatie over het product meegestuurd en opgeslagen binnen de projectagent. Van het product worden alle productiestappen die op het blackboard staan ingelezen door deze nieuwe projectagent en opgeslagen in een lijst. Deze lijst wordt gefilterd totdat er een lijst ontstaat van unieke productiestappen. De projectagent zal deze lijst van unieke productiestappen als zijn eigendom beschouwen. Echter meerdere projectagents kunnen wel dezelfde equiplet in hun lijst hebben opgenomen. Op basis van deze lijst stappen en met behulp van de availabilty database worden hier equiplets bij gezocht die deze stappen kunnen uitvoeren. Als dit gelukt is wordt het

product gescheduled. Zodra er vervolgens een nieuw product klaar staat op blackboard één, zal de directoragent via hetzelfde communicatieprotocol als beschreven staat in 7.1.4 alle projectagents in het systeem vragen of het nieuwe product met een subset van hun lijst van productiestappen te maken is. Zodra een projectagent zegt dat dit kan, wordt het product aan die projectagent toegewezen. Zodra geen projectagent een proposal stuurt, creëert de directoragent weer een nieuwe projectagent.

Scheduling

Productiestappen worden op het tweede blackboard gescheduled. Dit betekend dat productiestappen over de equiplets worden verdeeld en in een tijdstap worden geplaatst waarbij de volgorde van de productiestappen gewaarborgd blijft. Voor de scheduling is een stappenplan ontwikkeld welke is geïmplementeerd:

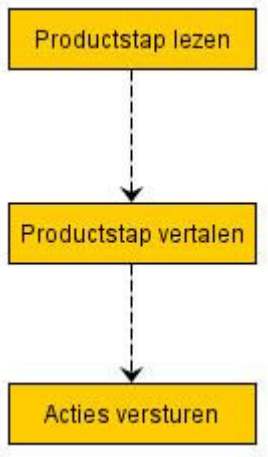
1. De projectagent moet weten welke tijdstappen per equiplet nog beschikbaar zijn binnen de deadline van een product. Hiervoor worden alle Post-its die op het tweede blackboard staan met een tijdstap kleiner dan de deadline van het nieuwe product gelezen.
2. Er wordt geprobeerd het nieuwe product te schedulen in de beschikbare tijdstappen binnen de deadline.
3. Werkt dit niet dan wordt er een back-up gemaakt van alle Post-its zoals ze nu zijn.
4. Al de producten van de projectagent moeten nu opnieuw gescheduled worden. Dit wordt gedaan via een bepaald schedulingsalgoritme. De projectagent heeft nu alleen nog EDF (Earliest Deadline First) geïmplementeerd. Hierbij wordt het product met de deadline die het eerst zal komen als eerst opnieuw gescheduled.
5. Kan het nieuwe product nu nog niet gescheduled worden dan zit het productieplatform te vol en kan het product niet gemaakt worden.

In de huidige implementatie kan een projectagent alleen zijn eigen toegewezen product schedulen. In de toekomst moeten projectagents kunnen onderhandelen over de scheduling zodra er gerescheduled moet worden.

5.1.6. Equipletagent

De equipletagent heeft de volgende concrete taken:

- Productiestappen van het tweede blackboard lezen.
- deze vertalen naar concrete acties voor zijn onderliggende modules
- acties versturen naar het derde blackboard.



Figuur 14 - Schematische weergave van stappen van een equiplet

Zodra de equipletagent van de directoragent het bericht ontvangt dat er een nieuwe tijdstap is ingegaan, haalt hij de productiestap van het tweede blackboard af die hij op die tijdstap moet gaan uitvoeren. Deze productiestap wordt vertaald [12]. Wat er van de productiestap overblijft, is een aantal acties, die allemaal als aparte Post-its op het derde blackboard worden gezet. De ROS equipletnode zal deze meteen lezen en beginnen met de eerste actie. De tijd die het kost om een stap van het tweede blackboard te lezen, te vertalen en op het derde blackboard zetten is verwaarloosbaar ten opzichte van de tijd dat het kost om de stap daadwerkelijk uit te voeren. Zodra de equiplet dit van te voren

zou doen wordt het risico gelopen dat de stappen op het tweede blackboard opnieuw worden gescheduled. Bijkomend voordeel van deze manier is ook dat het ROS gedeelte van het systeem hierdoor niet van de huidige tijdstap op de hoogte moet zijn. Deze voert nu enkel het werk uit dat op het derde blackboard verschijnt.

5.2. Blackboardserver

In deze paragraaf wordt uitgelegd hoe de blackboardserver werkt en de communicatie tussen deze server en het agentsysteem. De datastructuur die wordt overgestuurd is geserialiseerd met Protocol Buffers. Eerst zal er worden ingegaan op hoe een Protocol Buffer message wordt gedefinieerd, vervolgens komen de messages aan bod voor die voor dit project zijn gebruikt. Als laatste wordt er dieper ingegaan op hoe de blackboardserver werkt en wordt er uitgelegd hoe de filtering werkt op het blackboard.

5.2.1. Protocol buffer message

Protocol Buffer messages (vanaf nu messages) zijn te vergelijken met XML alleen een stuk compacter. De messages worden aangemaakt in een .proto bestand. Van dit bestand kan er met de Proto compiler voor zowel Java als C++ klassen worden gegenereerd. Deze klassen kunnen vervolgens worden gebruikt om message objecten te maken, uit te lezen of te (de)serialiseren. Hoe een message eruit ziet zal worden uitgelegd aan de hand van de message implementatie voor de Post-its (zie hoofdstuk 6.5 voor de betekenis van de velden).

Protocol Buffer message van een Post-it

```
message PostIt
{
	optional int64 timestep = 1;

	optional string owner = 2;

	optional bool isProcessed = 3;

	optional int32 ordinal = 4;

	optional string productRef = 5;

	required string payload = 6;
}
```

Zoals te zien is bij deze message bestaat elk veld uit vier componenten.

- Modifier: Deze kan 3 waardes hebben: required, optional of repeated
  - Required: Dit veld moet worden ingevuld bij de creatie van een object.
  - Optional: Dit veld mag leeg worden gelaten.
  - Repeated: Van dit veld kunnen er meer zijn, als een array.
- Type: Dit kan een aantal voorgedefinieerde waardes zijn zoals int32 of een string, maar kan ook een ander type message zijn.
- Naam van het veld.

- Tag: Dit is het unieke nummer dat achter het veld staat. Deze dient als ID zodra de message gecodeerd is.

Verder is het mogelijk om messages binnen messages te beschrijven. Details over hoe een message wordt gecodeerd zijn te vinden in de Protocol Buffer documentatie [14].

5.2.2. Post-It Box

Het is niet efficiënt om voor elke Post-it een aparte verbinding met de blackboardserver te maken en de Post-it dan over te sturen. Hiervoor is de Post-it Box geïntroduceerd. Dit is als het ware een doos waar meerdere Post-its in kunnen worden opgeslagen om vervolgens het geheel in een keer naar het blackboard te sturen. Om de interface van de blackboardserver zo eenvoudig mogelijk te houden is er voor gekozen om naast de Post-its extra meta informatie mee te sturen met de Post-It Box. Ook informatie voor filtering wordt meegenomen. Alle communicatie met de blackboardserver gaat dus via Post-it boxen, zowel van het blackboard lezen als schrijven. Als er van het blackboard gelezen moet worden zal er een lege box naar het blackboard worden gestuurd met bepaalde filterinformatie, de server zal dan de box vullen met Post-its en weer terugsturen. Bij het schrijven naar het blackboard wordt een door de agents gevulde Post-it box verstuurd naar de blackboardserver. Hieronder is de message voor een Post-it box te zien en een tabel die velden beschrijft.

Protocol Buffer message van een Post-it box

```
message PostItBox
{
	required bool isWrite = 1;

	required string zone = 2;

	optional string sender = 3;

	message Filter
	{
		required string filtername = 1;

		repeated string agentname = 2;

		optional int64 currentTimestep = 3;

		optional int64 deadline = 4;

	}
	optional Filter filter = 4;

	repeated PostIt postIts = 5;
}
```

Veldnaam	Toelichting
isWrite	Dit veld geeft aan of de clientagent wil lezen van of schrijven naar het blackboard.
Zone	Dit geeft aan naar welk blackboard geschreven moet worden. De drie blackboards hebben in de implementatie de namen “BB1”, “BB2 en “BB3” meegekregen.
Sender	Dit is de naam van de afzendende agent.
Filter	Dit is het filter object wat wordt meegegeven bij een leesactie richting het blackboard. (Zie hoofdstuk 7.2.3 over hoe de filtering wordt toegepast)
PostIts	Dit is het veld waarin de Post-its worden verstuurd.

5.2.3. Werking van de blackboardserver

De blackboardserver is een multithreaded server. Voor elke verbinding met de server draait er een nieuwe thread die een Post-it box kan verwerken. De server draait ergens in het netwerk. De server kan gevonden worden met behulp van de Autoconfiguratie server. Dit is een server die bepaalde waardes multicast over het netwerk [12]. In die waardes zijn onder andere het IP-adres en de poort van de server opgenomen. Onderwater heeft de blackboardserver een Open BBS blackboard object. De drie blackboards in het systeem zitten onderwater allemaal in datzelfde object. Open BBS heeft een implementatie om het blackboard op te delen in meerdere gedeeltes, zogenaamde zones. Elk van de drie blackboards wordt gerepresenteerd door een dergelijk zone en op basis van de informatie van de Post-it box komen de Post-its in de juiste zone terecht.

Zodra een Post-it box binnenkomt in de serverthread, wordt er eerst op basis van de zonenaam een blackboardaccess interface aangemaakt. Met dat object kunnen lees/schrijf/verwijder acties uitgevoerd worden op die zone. Vervolgens wordt er bekeken of het een lees- of schrijfbox is. Voor het schrijven van Post-its naar het blackboard worden de Post-its uit de box gelezen en één voor één naar het blackboard toe geschreven via het access object. Moeten er Post-its van het blackboard worden gelezen dan zal er gekeken worden naar welke filter moet worden toegepast. Op basis van dat filter worden alle Post-its op het blackboard gefilterd en de gefilterde Post-its teruggestuurd.

Filters

Op basis van de informatie over de filter in de Post-it box wordt bepaald welke filter wordt toegepast. Het Open BBS systeem heeft een interface-klasse voor filters. Hierin zit een select methode die elk object van het blackboard opvraagt. In de implementatie van de filter klasse moet de conditie worden opgegeven waaraan de objecten op het blackboard aan moeten voldoen om door de filter heen te komen. Bij Open BBS zijn een aantal standaard filters meegeleverd. Deze zijn echter niet afdoende voor dit project. Er zijn drie filters gemaakt die allemaal een specifiek doel vervullen.

- Read filter: Dit is de standaard leesfitler. Op basis van de eigenaar van de Post-It wordt er gefilterd. Dit wordt bijvoorbeeld gebruikt als een equiplet zijn Post-its van het blackboard wil halen.
- Availabilty filter: Deze filter wordt gebruikt vlak voor de scheduling. Hierbij worden alle Post-Its gefiltert op het blackboard van de opgegeven equiplets en met een tijdstap van voor de opgegeven deadline. Deze worden allemaal teruggestuurd naar de geïnteresseerde projectagent.

- Reschedule filter: Deze filter wordt toegepast voor de rescheduling. Het geeft alle Post-its terug van opgegeven producten. Deze Post-its worden dan als back up bewaard bij de projectagent.

Verder controleert de blackboardserver zelf ook nog op Post-its die verlopen zijn. Waarbij de huidige tijd later is dan de tijd op de Post-it. Deze Post-its worden dan verwijderd.

6. Resultaten en analyse

In dit hoofdstuk word een opsomming gegeven van alle producten die tijdens de afstudeerstage zijn opgeleverd. Niet al deze producten waren oorspronkelijk gepland maar bleken pas tijdens het project wel of niet noodzakelijk. Dit valt te wijten aan het onderzoekende karakter van de afstudeeropdracht. De oorspronkelijke lijst is opgenomen in het Plan van Aanpak. (**Fout! Verwijzingsbron niet gevonden.**)

6.1. De opgeleverde resultaten

In deze paragraaf zijn de resultaten beschreven die zijn opgeleverd tijdens dit project.

De volgende resultaten zijn behaald tijdens dit project:

- Er is een overliggende agentgebaseerde architectuur opgeleverd die één of meerdere equiplets kan aansturen om zo productiestappen uit te voeren. Hiervoor is eerst een ontwerp gemaakt van het systeem en welke agents daarbij nodig zijn. Vervolgens is dit ontwerp geïmplementeerd in JADE. Hierbij is rekening gehouden met de communicatie tussen de agents, de scheduling van productiestappen en de tijdsynchronisatie.
- Er is een server opgeleverd die dient als blackboard. Hierbij kan er gecommuniceerd worden tussen de server en de agents voor het versturen en ophalen van werkbriefjes, zogenaamde Post-its.
- Er is een abstracte taal om productiestappen te beschrijven. Deze taal is gedefinieerd in XML. Op basis van deze taal kunnen equipletagents een vertaalslag doen naar concrete acties voor verschillende onderliggende modules.
- Een configuratieserver die via multicast een aantal configuratieregels rondstuurt over het netwerk.
- Een buildsysteem om het gehele systeem in een keer te bouwen.

De volgende punten zijn nog niet opgeleverd, maar zullen worden opgeleverd voor de eindpresentatie van het project:

- De bestaande software voor de HUniplacer equiplet aanpassen ten behoeve van de agentgebaseerde architectuur.
- Een proof-of-concept demonstrator die het multi-agentsysteem laat werken met de HUniplacer.

De demonstrator zal vermoedelijk op de volgende manier werken:

1. Met externe software ontwikkeld bij een ander project worden er in een virtuele omgeving in een 4 x 4 grid balletjes met een bepaalde kleur geplaatst.
2. Op basis hiervan wordt een XML document gegenereerd welke beschrijft hoe de balletjes zijn geplaatst.
3. Op basis van dit XML document wordt een projectagent gestart, die ook door een ander project is ontwikkeld, waarbij het XML document wordt vertaald naar productiestappen.
4. De projectagent meldt zich bij dit agentsysteem.

5. Het multi-agentsysteem zal de productiestappen uitvoeren. Er zullen vier kratjes op de HUniplacer equiplet liggen. Drie kratjes met ieder een andere kleur balletjes en een vierde die precies zo gevuld moet worden als is aangegeven in de omgeving bij punt één.

6.2. Analyse van de resultaten

Op de eerste plaats is het jammer dat op het moment van schrijven van dit document de huidige software van de HUniplacer nog niet is aangepast waardoor we nog geen concrete demonstrator hebben. Hierdoor is het ook niet mogelijk om deze zaken te beschrijven in het verslag. Voor de presentatie van dit project zullen deze punten wel afgerond zijn, waardoor het volledige systeem dan werkend toonbaar is.

6.2.1. Multi-agentsysteem

Het multi-agentsysteem werkt en de performance is voldoende voor de acties die het systeem moet doen. Dit betekend dat voor de huidige lengte van een productiestap, die qua tijd in de orde van secondes nodig heeft om uitgevoerd te worden, het systeem werkt. Zodra productiestappen potentieel sneller uitgevoerd zouden kunnen worden, doordat de equiplets steeds geavanceerder worden, zou het systeem mogelijk tegen een bottleneck aan lopen vanwege de overhead en netwerkcommunicatie die de agents nodig hebben om voldoende te werken. Een ander probleem waar het systeem tegenaan loopt is dat het ontwerp en implementatie niet goed aansluiten bij het onderzoek van Leo van Moergestel over productagents. Vooral de verantwoordelijkheden van de projectagent in dit systeem worden daar door de productagent vervuld. Nu al kan met zekerheid gezegd worden dat dit systeem niet goed op zijn productagent aansluit. Er zal in de toekomst moeten worden nagedacht over hoe de productagent van Leo op een praktijk gerichte manier in het systeem kan worden geïntegreerd.

6.2.2. Blackboardserver

De blackboardserver werkt in de huidige omgeving. Het data-driven principe komt hier goed naar voren door de server zijn acties uit te laten voeren op basis van wat hij van de agents binnen krijgt. Qua performance is het blackboard intern niet de bottleneck. De communicatie met de agents is dat wel zodra er veel informatie moet worden verstuurd. Protocol Buffers helpt zeker om de berichten compact te houden. Op de meest kritische punten in het systeem die tijdafhankelijk zijn zoals de communicatie van de equipletagent naar de equipletnode is er wel zo weinig mogelijk dataverkeer met het blackboard. Verbeterpunten zitten in de filtering van Post-its. Deze zijn op dit moment nog niet voldoende uitgewerkt en niet goed uitbreidbaar. Een mogelijke verbetering is het filteren doen op basis van lijst met filterregels waarin veldnamen kunnen worden opgenomen, waar een waarde in kan staan en waar een expressie in staat, bijvoorbeeld groter dan of is gelijk aan. Hiermee wordt het filtersysteem nog flexibeler en kunnen de aparte filters zo als die er nu zijn verwijderd worden.

7. Conclusies en aanbevelingen

Dit hoofdstuk geeft de conclusies van dit project. Het beantwoord de onderzoeksvragen en geeft aanbevelingen waar in het vervolg op gelet moet worden. Ten slotte worden er mogelijke vervolgonderzoeken vermeld en aanbevelingen gedaan.

7.1. Beantwoorden onderzoeksvragen

In de uitwerking van het ontwerp en de implementatie worden de onderzoeksvragen beantwoord. Hier is een samenvatting van opgesteld.

7.1.1. Hoe kan de agentgebaseerde architectuur worden geïmplementeerd?

Kan hiervoor een bestaand ontwerp worden gebruikt of moet er iets nieuws worden ontworpen?

Het ontwerp voor het multi-agentsysteem is gebaseerd op eerder werk binnen de afstudeerorganisatie. [9] [7] Omwille van efficiëntie en andere praktische bezwaren voldeed dit niet. Een systeem waarin iedere productagent met iedere equiplet moet onderhandelen is niet schaalbaar genoeg.

Waar mogelijk is er wel voor gekozen de rol van de agents te baseren op eerdere ontwerpen omdat in grote lijnen de bestaande ideeën erg goed zijn.

Welk agentsysteem kan hier het beste voor worden gebruikt?

Er is gekozen voor het JADE agentsysteem. Dit is het meest volwassen agentplatform voor Java en wordt ook rechtstreeks in Java geprogrammeerd. Een andere optie was 2APL. 2APL is gebouwd op JADE, maar implementeert een andere structuur voor agents. 2APL heeft een betere manier om het gedrag van agents formeel op te schrijven. Hier was echter niet voor gekozen om meerdere redenen. Ten eerste speelde de tijd een factor Ten tweede is het niet mogelijk om agents te creëren terwijl het platform draait. Dit is een vitale reden om niet voor 2APL te kiezen.

Een groot voordeel van JADE is dat het communiceert volgens de FIPA standaard. Deze standaard wordt door veel agentplatforms geboden en definieert het communicatieprotocol tussen agent. Dankzij deze standaard kunnen agents uit verschillende platformen met elkaar communiceren in een multi-agentsysteem.

Hoe moeten producten worden beschreven in een abstracte taal en hoe moet deze taal eruit zien en vertaald worden?

Aanvankelijk was het idee om te kiezen voor een natuurlijke taal die stappen beschrijft, maar daar is vanaf gezien. Het parsen van een natuurlijke taal is ingewikkeld en veel werk, maar het had ook nadelen. Een groot probleem is dat natuurlijke taal heel erg moeilijk formeel te beschrijven is.

Hierdoor is gekozen voor een taal die gedefinieerd is in een XML-formaat die stappen en bijbehorende constraints kan beschrijven.

Bij XML is aan tags een attribuut toe te voegen die aangeeft of het noodzakelijk is om de inhoud van die tag te begrijpen. Als de taal uitgebreid moeten kunnen worden met zo min mogelijk impact voor het gehele systeem, dan is XML een betere keuze.

7.1.2. Hoe kan het herconfigureren van de robot worden geïmplementeerd?

Het herconfigureren van het systeem bestaat uit een tweetal soorten problemen. Aan de ene kant is er een probleem hoe dit met de hardware wordt gedaan: hoe weet de hardware dat er een module op een equiplet is ingeplugd of verwijderd? Ook moet er over worden nagedacht hoe ervoor gezorgd kan worden dat modules fysiek aan elkaar gekoppeld kunnen worden.

Dit deel van het probleem is niet binnen de tijd van het project uitgezocht en zal tijdens toekomstig onderzoek moeten worden uitgewerkt.

Een tweede probleem is de vraag hoe de software zich aanpast aan de veranderende hardware. Dit probleem is wel opgelost.

Hoe kan dit gedaan worden terwijl het systeem blijft draaien?

Doordat het gehele productiegrid is verdeeld in equiplets, hebben wijzigingen aan een equiplet geen gevolgen. Terwijl een equiplet wordt geherconfigureerd kunnen productiestappen op andere equiplets gewoon uitgevoerd worden.

Welke gevolgen heeft het herconfigureren van een productieplatform en hoe kan dit worden opgelost?

Als de hardware verandert is het mogelijk stukken software te starten die ervoor zorgen dat de hardware wordt aangestuurd en het productiesysteem weet welke functionaliteit het grid als geheel biedt. Vervolgens kunnen producten ook op de nieuwe modules worden gemaakt.

Als optimalisatie is het mogelijk om alle producten die al gepland waren opnieuw te verdelen over de toegevoegde equiplet.

Als modules verwijderd worden is het probleem lastiger. Alle producten die gepland waren kunnen dan misschien niet meer gemaakt worden, zelfs na een poging om opnieuw te plannen. Dit kan betekenen dat er halve producten weggegooid moeten worden. De optimalisation agent, die buiten de scope van het project valt, zou operatoren van het grid advies kunnen geven over de bezetting van de verschillende soorten equiplets zodat een goed geïnformeerde keuze kan worden gemaakt over te verwijderen of bij te voegen equiplets. Bij een praktisch productiegrid zou er in dat soort gevallen een keuze moeten worden gemaakt door de productagent over wat er met zijn product moet gebeuren. Zo zou het afronden van een product kunnen worden uitgesteld of volledig kunnen worden geannuleerd.

7.1.3. Conclusie

De antwoorden op de deelvragen laten zien dat een productiesysteem, zoals deze door eerder werk van de opdrachtgever in grote lijnen is geschetst, ook praktisch mogelijk is. De meeste praktische bezwaren kunnen namelijk worden weggenomen. Bepaalde vraagstukken vallen helaas buiten de scope van het project, maar de resultaten laten zien dat verwacht kan worden dat ook deze kunnen worden opgelost.

De proof-of-concept toont de werking van het productiesysteem en laat zien dat het idee in de praktijk werkt.

7.2. Aanbevelingen en vervolgonderzoek

In de rest van de scriptie zijn veel aandachtspunten gegeven voornamelijk op technisch niveau. In deze paragraaf worden er aanbevelingen gedaan voor vervolgonderzoeken.

Een suggestie voor werktuigbouwkundig vervolgonderzoek is het uitzoeken hoe modules op elkaar moeten worden gezet. Bepaalde mechanismes zullen fysiek aan elkaar moeten worden verbonden. Interessant hierbij is het onderzoek naar hoe de hardware weet dat er een module is ingeplugd.

De optimalisation agent die is gedefinieerd moet in toekomstig onderzoek uitgewerkt worden. Deze zou werkorders op het derde blackboard verder kunnen optimaliseren. Ook zou door het analyseren van de werkorders op het derde blackboard de operatoren van het grid advies(rapporten) kunnen geven over de bezetting van de verschillende soorten equiplets zodat een goed geïnformeerde keuze kan worden gemaakt over te verwijderen of bij te voegen equiplets.

Het zou interessant zijn om producten van een tekening die op de computer is gemaakt direct te produceren. In een vervolgonderzoek zou kunnen worden uitgezocht hoe een tekening van een product, bijvoorbeeld gemaakt met CAD-software, om te zetten is in een productagent die het product kan maken, al dan niet door aan de hand van de tekenening een AFDL-document te genereren. Er is een database met theoretische functionaliteit en eigenschappen van onderdelen, deze zou gekoppeld kunnen worden aan intelligente ontwerptools waardoor producten snel kunnen ontworpen die ook gelijk kunnen worden geproduceerd door het grid.

Een ander interessant onderwerp om te onderzoeken is zelflerende productie. Als bij een eindinspectie blijkt dat het product niet de gewenste kwaliteit heeft zou automatisch kunnen worden uitgezocht welke modulen daarbij betrokken zijn geweest. De ROS-nodes die deze aansturen zouden kunnen leren van de fouten die gemaakt zijn. Het is een goede vraag hoe fouten gedetecteerd en hersteld kunnen worden. Een interessant studentenproject zou het toevoegen van een visuele eindinspectie kunnen zijn, waarbij camerabeelden worden gebruikt om te verifiëren of een product correct is gemaakt.

Het in elkaar zetten van een equiplet en de juiste modules monteren zou ook kunnen worden gezien als een verzameling productiestappen. Ditzelfde telt voor het uit elkaar halen of herconfigureren van modules. In de verre toekomst zou het mogelijk zijn om equiplets op te nemen die het productiegrid dynamisch kunnen herconfigureren. Als het grid onder grote druk staat zou een productagent gestart kunnen worden die een equiplet met de benodigde modules uitrust om zo de druk te verlichten. Andersom zouden modules geautomatiseerd uit elkaar gehaald kunnen worden en in het magazijn worden gezet. Dit is echter een ver toekomstbeeld.

Om vormen frezen en 3D-printen beter mogelijk te maken is het van belang om ondersteuning voor het definiëren van geometrische vormen. Als er informatie is over geometrische vormen van de onderdelen en over de productiestappen moet het mogelijk zijn om een 3D-model af te leiden uit de AFDL-beschrijving van een product.

8. Literatuurlijst

[1] A. Gunasekaran, Agile manufacturing: the 21st century competitive strategy, Elsevier science, 2001.

[2] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy en H. v. Brussel, „Reconfigurable Manufacturing Systems," in *CIRP Annals Manufacturing Technology*, 1999.

[3] M. Wooldridge, An Introduction to MultiAgent Systems, Second Edition, Wiley, 2009.

[4] H. Nii, „The blackboard model of problem solving and the evolution of blackboard architectures," *AI magazine,* pp. 38-53, 1986.

[5] R. Clavel, „Device for displacing and positioning an element in space". EP Patent 0.250.470, juli 1991.

[6] E. Puik en L. v. Moergestel, „Agile multi-parallel micro manufacturing using a grid of equiplets," in *IPAS 2010 proceedings*, 2010.

[7] D. Telgen, L. v. Moergestel, E. Puik en J. Meyer, „Agile Manufacturing Possibilities with Agent Technology," 2012.

[8] Z. Mouris, „Vision for Mechatronics & Robotics," 2012.

[9] L. v. Moergestel, *Multi-agent-based production scheduling,* 2011.

[10] L. Moergestel, E. Puik, D. Telgen en J. Meyer, „Decentralized Autonomous-Agent-Based Infrastructure for Agile Multiparallel Manufacturing," in *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, 2011.

[11] P. Muller, M. Beek, R. d. Kok en L. Stolwijk, „Onderzoek verbeteren van Multi-Agent Based Production Scheduling," 2012.

[12] P. Muller, „Flexibel produceren op een productiegrid door het toevoegen van een zelforganiserend en -beschrijvend multi-agentsysteem," 2012.

[13] FIPA, „FIPA ACL Message Structure Specification," [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html. [Geopend 20 5 2012].

[14] Google-Developers, „Developer Guide - Protocol Buffers," [Online]. Available: https://developers.google.com/protocol-buffers/docs/overview. [Geopend 20 5 2012].

Bijlage A - Plan van Aanpak

Plan van Aanpak

Afstudeeropdracht Reconfigurable Agile Manufacturing

Pascal Muller – 1548051
Martijn Beek - 1557514
Technische Informatica

Versie 1.0: 14 maart 2012

Gegevens betrokkenen

Gegevens afstudeerders

Pascal Muller
pascalmuller@gmail.com / pascal.muller@student.hu.nl
06 - 53 90 14 51

Martijn Beek
martijn.beek@student.hu.nl / beek.martijn@gmail.com
06 - 30 18 97 63

Docentbegeleider

Daniël Telgen
daniel.telgen@hu.nl
06 – 26 11 46 89

Gegevens afstudeerorganisatie

KENNISCENTRUM
TECHNOLOGIE & INNOVATIE
HOGESCHOOL
UTRECHT

Lectoraat Microsysteemtechnologie / Embedded Systems

Oudenoord 700
Postbus 182
3500 AD Utrecht

088 - 481 86 81

Bedrijfsbegeleider

Erik Puik
erik.puik@hu.nl
06 - 515 540 41

Inhoudsopgave

1. Inleiding 1
2. Afstudeerorganisatie 1
3. Methoden en technieken 2
  3.1. Multi-agent systeem 2
  3.2. ROS 3
  3.3. Git 3
  3.4. Agile Scrum 3
4. De opdracht 3
  4.1. Probleemomschrijving 3
  4.2. Doelstelling 4
  4.3. Onderzoeksvraag 4
  4.4. Op te leveren producten 4
  4.5. Projectgrenzen 5
5. Planning 5
6. Kwaliteitsbewaking 7
7. De projectorganisatie 7
  7.1. Betrokkenen 7
  7.2. Communicatie 7
  7.3. Documentatie 7
8. Risico's 8
9. Literatuur 8
Bijlage A. Gantt-chart planning I

1. Inleiding

Dit document is het plan van aanpak voor het afstudeerproject van Pascal Muller en Martijn Beek bij het Lectoraat Microsysteemtechnologie van de Hogeschool Utrecht. Hier lopen verschillende projecten rond flexibele productieprocessen.

Wij hebben het afstudeervoorstel voor dit project afzonderlijk ingeleverd. In overleg met de docentbegeleider is besloten om een gezamenlijk plan van aanpak op te leveren, maar toch ieder een eigen scriptie op te leveren.

Voor aanvang van dit project werd er door een groep van acht student-stagairs gewerkt aan een project. Zij hebben het systeem ontwikkeld waar nu verder aan wordt gewerkt. Het huidige systeem maakt gebruik van een deltarobot die aangestuurd wordt met behulp van computer vision algoritmes. Deze draaien op een Linux platform met daarop Robot Operating System (ROS). Sommige objecten kunnen worden herkend door de machine en de locatie daarvan kan worden doorgegeven aan het regelsysteem van de deltarobot.

De opdrachtgever heeft in het kader van onderzoek in het kenniscentrum een theoretisch productiesysteem bedacht. Hierin kunnen de verschillende productiemachines, bijvoorbeeld meerdere deltarobots, gezamenlijk werken aan het produceren van producten. De productiemachines, de zogenaamde equiplets [2], zijn zo goedkoop mogelijk en herconfigureerbaar. Dat wil zeggen dat hardware- en software van verschillende modulen van de equiplet vervangen kan worden met weinig downtime. Dit heeft ten gevolg dat de functionaliteit en de toestand van het hele productiesysteem voortdurend kan veranderen.

Voor dit theoretische productiesysteem wordt er een agent-gebaseerde architectuur [5] ontworpen. Het is hierbij belangrijk dat de ROS-gebaseerde architectuur voor de equiplets en hun hardware wordt aangestuurd door deze agent-gebaseerde architectuur. Agents zijn software-entiteiten die een bepaalde mate van vrijheid hebben om gedrag te vertonen. Om gedrag te vertonen moeten deze agents in staat zijn om gedrag te uiten in de omgeving en de omgeving te interpreteren [4].

De werking van de ontworpen architectuur zal worden aangetoond met een proof of concept.

Dit document beschrijft hoe het afstudeerproject zal worden aangepakt. De probleemstelling en doelstellingen van het afstudeerproject worden beschreven en er wordt aangegeven welke beroepsproducten er worden opgeleverd om deze te behalen.

2. Afstudeerorganisatie

Sinds 2001 is de functie van lector ingevoerd in het Hoger Beroepsonderwijs. Lectoren geven leiding aan een kenniskring van onderzoekers (meestal docenten) die praktijkgericht onderzoek doen. Deze kenniskring wordt lectoraat genoemd. Praktijkgericht onderzoek heeft als doel problemen op te lossen waar professionals in de praktijk tegenaan lopen. Ook bij de afstudeeropdracht is er een koppeling tussen theorie en praktijk.

Het lectoraat waar de opdracht beschikbaar is, is het lectoraat Microsysteemtechnologie/Embedded Systems van de Hogeschool Utrecht. Deze houdt zich bezig met productverbetering en industrialisatie van microsystemen. [1]

Dit vakgebied concentreert zich op systemen met hoge nauwkeurigheden; de productie daarvan behoort tot de meest lastige die er bestaat. Dit kan mogelijk leiden tot productie-uitval, een te lage productieopbrengst, oplopende productiekosten, maar vooral ook projectvertraging op een moment dat investeringen al plaatsgevonden hebben.

Onderzoek van dit lectoraat heeft onder meer betrekking op:

- Optimalisatie van productontwerp en de ontwerpprocedures.
- Modularisering van productieprocessen. Door modules te hergebruiken dalen ontwerpinspanning en de kosten van industrialisatie. Daarnaast verbetert de kwaliteit van de productie.

Dit lectoraat zet onder de naam HUniversal Production de productiefilosofie van de toekomst neer. Het is gebaseerd op de ontwikkeling rond Agile Manufacturing (AM). AM wil de flexibiliteit en reactietijd bij productievragen verbeteren. [1]

De opdrachtgever denkt dat deze filosofie breder inzetbaar is dan alleen productievraagstukken.

Bij de onderzoeksprojecten van dit lectoraat waren steeds bedrijven, studenten en tutoren betrokken en werd een actueel probleem van een bedrijf geadresseerd.

3. Methoden en technieken

In dit hoofdstuk worden de gebruikte methoden en technieken beschreven. De software voor de agents zal geschreven worden in Java, alle andere software in C++. Enkele methoden, tools en technieken spelen een belangrijke rol in het project en zullen hieronder kort worden behandeld.

3.1. Multi-agent systeem

In multi-agentsystemen werken meerdere agents [4] samen om gezamenlijk een resultaat te bereiken [3]. Voor agents in een multi-agentsysteem is het vooral van belang dat ze ook in staat zijn om met elkaar te communiceren.

3.1.1. Jade agents

JADE (Java Agent DEvelopment Framework)[2] is een software framework voor multi-agent systems gebouwd in Java. Het is hiermee mogelijk om agents te programmeren in Java, waarbij er al functionaliteit aanwezig is voor onder meer gedrag en communicatie. Er zijn ook andere agent frameworks, maar tijdens het vooronderzoek is gekozen voor JADE omdat dit al jarenlang in ontwikkeling is en een van de meest volwassen platforms is. JADE gebruik van de FIPA-standaard voor communicatie tussen agents.

[1] Delen van dit hoofdstuk zijn geparafraseerd overgenomen van de webpagina op: http://www.technologieeninnovatie.hu.nl/Data/Lectoraten/Microsysteem%20technologie%20Embedded%20Systems.aspx

[2] Website: http://jade.tilab.com/

3.2. ROS
ROS (Robot Operating System)[3] is een modulair opgebouwd platform gericht op software voor robots. Het biedt tools en bibliotheken om het schrijven van (gedistribueerde) software voor de aansturing van robots te vergemakkelijken. De modulaire opbouw maakt het eenvoudig om softwaremodules te hergebruiken, nieuwe softwaremodules toe te voegen aan een robot en problemen te traceren. ROS stelt de programmeur in staat om zich vooral te richten op de functionaliteit.

3.3. Git
Voor het beheren van de broncode wordt het versiebeheersysteem Git[4] gebruikt. Hiermee is het eenvoudig om wijzigingen in te zien, oude code terug te halen indien noodzakelijk en gecontroleerd nieuwe code aan het systeem toe te voegen.

3.4. Agile Scrum
Om het project in goede banen te leiden maken we gebruik van Agile Scrum. Dit is een flexibele projectmanagementmethode, waarbij de tijd opgedeeld in iteraties met een vaste lengte, zogenaamde sprints. Bij aanvang van een sprint vindt overleg plaats en wordt in overleg met de opdrachtgever bepaald welke onderdelen de meeste prioriteit hebben. Het team stelt taken op en schat de benodigde tijd in. Voor een onderzoeksproject waarbij de richting van het project kan veranderen is deze methode zeer geschikt omdat op die veranderingen kan worden ingespeeld.

4. De opdracht

4.1. Probleemomschrijving
Bij het vorige project is gewerkt aan de hardwareaansturing van de deltarobot waarbij de camerabeelden als sensor worden gebruikt. Deze resultaten worden tijdens dit project gebruikt als deel van het project.

Om de kennis over de resultaten op peil te houden is het noodzakelijk ons te verdiepen in de bestaande hard- en software.

De robot zal als equiplet een onderdeel worden van een groter productiesysteem. In overleg met de opdrachtgever(s) zal een ontwerp worden gemaakt voor een agent-gebaseerde softwarearchitectuur die zo'n productiesysteem praktisch moet gaan implementeren. Deze softwarearchitectuur zal de kant voor de aansturing van de deltarobot (met ROS) combineren met een slimme aansturing voor het maken van arbitraire (batches) producten.

De opdrachtgever heeft enkele eisen aan het systeem, waarvoor een goede oplossing zal moeten bedacht. Twee belangrijke eisen zijn:

- Herconfiguratie van het productiesysteem moet mogelijk zijn.
- Flexibel producten en modulen toevoegen aan het systeem moet mogelijk zijn.

Er zal moeten worden bedacht hoe de herconfiguratie moet plaats vinden, aangezien herconfiguratie grote gevolgen kan hebben voor het productieproces.

[3] Website: http://www.ros.org/
[4] Website: http://git-scm.com/

Om flexibel modulen en producten aan het systeem toe te kunnen voegen moet het mogelijk zijn om de opbouw van producten abstract te definiëren. In het systeem moeten vervolgens vertaalslagen worden gemaakt van een abstracte beschrijving naar concrete acties in lagere lagen.

Voor deze uitdagende problemen zal een goede oplossing moeten worden bedacht binnen de te ontwerpen architectuur. Uiteindelijk zal met een proof-of-concept worden aangetoond dat het ontwerp functioneert.

De ontwerpfase is gezamenlijk, maar de betrokken personen zullen ieder verantwoordelijk zijn voor een eigen stuk van het proof-of-concept.

4.2. Doelstelling

Tijdens het project zullen de volgende doelstellingen worden bereikt:

- Er zal in overleg met de opdrachtgever een agent-gebaseerde architectuur worden ontworpen. Deze is verantwoordelijk voor het aansturen van de ROS-gebaseerde robot en zorgt ervoor dat producten kunnen worden gemaakt. Om dit zo flexibel mogelijk te houden moet worden nagedacht over een abstracte taal om functionaliteiten te beschrijven;
- Het flexibel herconfigureren van de robot zal worden onderzocht. Er moet voor worden gezorgd dat het ROS-systeem verandert, maar ook dat het intelligente deel van het systeem hier correct mee omgaat;
- Een implementeren van een proof-of-concept van het ontworpen productiesysteem dat aantoont dat het fabriceren van producten en de herconfiguratie werkt.

4.3. Onderzoeksvraag

Voor het project zijn de volgende onderzoeksvragen en bijbehorende deelvragen opgesteld:

- Hoe kan de agent-gebaseerde architectuur worden geïmplementeerd?
    - Kan hiervoor een bestaand ontwerp worden gebruikt of moet er iets nieuws worden ontworpen?
    - Welk agentsysteem kan hier het beste voor worden gebruikt?
    - Hoe moeten producten worden beschreven in een abstracte taal en hoe moet deze taal eruit zien en vertaald worden?
- Hoe kan het herconfigureren van de robot worden geïmplementeerd?
    - Hoe kan dit gedaan worden terwijl het systeem blijft draaien?
    - Welke gevolgen heeft het herconfigureren van een productiesysteem of een productieplatform en hoe kan ervoor worden gezorgd dat dit kan worden opgelost?

4.4. Op te leveren producten

De op te leveren producten zijn te onderscheiden in twee categorieën:

- Afstudeerproducten documenteren het proces van het afstudeertraject;
- Beroepsproducten worden opgeleverd ten behoeve van de afstudeerorganisatie.

Beroepsproducten:

- Een overliggende agent-gebaseerde architectuur die één of meerdere equiplets kan aansturen om zo productiestappen uit te voeren:
    - Ontwerp;

    - Wijzigingen aan de ROS-gebaseerde architectuur om dat mogelijk te maken;
    - Ontwerp van benodigde databases;
    - Implementatie van een prototype.
- Een architectuur die flexibele herconfiguratie mogelijk maakt:
    - Ontwerp;
    - Implementatie van een prototype.
- Een abstracte taal om productiestappen te beschrijven. Deze taal zal verschillende hiërarchische niveaus hebben en zal er dus ook anders uitzien op die niveaus.
    - Ontwerp;
    - Implementatie.
- Nieuwe ROS-nodes met herconfiguratie interfaces ten behoeve van de agent-architectuur met interfaces voor herconfiguratie;
- GUI voor diagnostische gegevens en herconfiguratie;
- Technische documentatie.

Belangrijkste afstudeerproducten:

- Plan van aanpak;
- Scriptie.

4.5. Projectgrenzen

Alle op te leveren producten, zoals opgenomen in het voorgaande hoofdstuk, vallen binnen de grenzen van dit project.

Het aanpassen en uitbreiden van de aansturing van de hardware van de bestaande robot valt expliciet buiten de scope van het project.

5. Planning

Voor het afstudeerproject zijn van het begin tot het inleveren van de scriptie ongeveer honderd werkdagen beschikbaar. Dit hoofdstuk beschrijft een globale planningvoor de invulling van deze tijd.

Door de gekozen softwareontwikkelmethode (hoofdstuk 3. Methoden en technieken), maar ook door het onderzoekende karakter van het project, is het niet mogelijk om een gedetailleerde planning van het project te maken. Er kan wel goed worden ingespeeld op de veranderende inzichten van de opdrachtgever.

Bij aanvang van elke sprint zal het aantal uren voor een taak worden geschat.

Tijdens het afstudeertraject zal ieder een eigen richting kiezen, maar dat zal pas gebeuren als de ontwerpfase is afgerond. De ontwerpfase van het systeem wordt met alle betrokkenen uitgevoerd. In de fase daarna zal in overleg met de opdrachtgever een eigen richting uitgekozen worden en ieder zal de verantwoordelijkheid voor een eigen stuk dragen.

Onderstaande planning laat duidelijk de fasering van het project zien en bevat de deadlines voor deze fasen.

Deze planning is ook opgenomen als Gantt-grafiek in **bijlage A**.

Dag 1-20: Vooronderzoek

- Inwerken en het huidige systeem onderzoeken;
- Bestaande documentatie lezen en bijwerken;
- In kaart brengen wensen van opdrachtgever;
- Kennis maken met ROS;
- Onderzoek naar multi-agentsystemen en platformen hiervoor (waarbij uiteindelijk is gekozen voor JADE);
- Onderzoek naar een koppeling tussen ROS en JADE;
- Werken aan en inleveren plan van aanpak.

Dag 21-40: Ontwerpen

- In overleg met de opdrachtgever een architectuur voor herconfiguratie ontwerpen;
- Een architectuur ontwerpen gebaseerd op agent technologie voor het totale systeem in overleg met alle betrokkenen;
- Een abstracte taal definiëren. Dit wordt de taal waarin de producten abstract beschreven kunnen worden door de agents. Deze taal zal verschillende hiërarchische niveaus hebben en zal er dus ook anders uitzien op die niveaus.

Dag 41-75: Development proof of concept

- De ontworpen architecturen implementeren, testen en opleveren;
    - Hierbij horen ook alle specifieke onderdelen van de architecturen; databases, blackboard e.d.
- De AFD Language uitwerken en implementeren;
- Scriptie iteratief schrijven;
    - Halverwege deze periode wordt er een voorlopige versie opgeleverd die samen met de docentbegeleider wordt besproken;
- Alles documenteren gedurende de werkzaamheden;
- Evaluatie proof of concept.

Dag 76-80: Scriptie afronden

- Scriptie afronden en inleveren.

Dag 81-100: Afronden en nazorg

- Laatste open eindjes wegwerken;
- Overdrachtsdocumentatie schrijven;
- Presentatie voorbereiden.

Het einde van de fasen ‘Voorbereiden’ en ‘Scriptie afronden’ zijn gekozen met in acht name van de deadlines voor het inleveren van het plan van aanpak en de scriptie.

6. Kwaliteitsbewaking

Binnen het onderzoek zal de kwaliteit bewaakt worden door middel van voortgangsgesprekken met de opdrachtgever, het beoordelen van elkaars code en documenten en het stellen van kwaliteitseisen aan het eindproduct.

Zowel voor alle ontwerpen, documenten en software is terugkoppeling van de opdrachtgever noodzakelijk. Zeker in de beginfase van het project is zijn goedkeuring noodzakelijk ter verificatie van de kwaliteit van het opgeleverde werk.

De kwaliteitsbewaking voor de documenten vindt plaats doordat de teamleden alle uitgaande documenten doorlezen, controleren en corrigeren om de kwaliteit te waarborgen.

De kwaliteitseisen voor de software zullen gebaseerd zijn op de eisen van de opdrachtgever. In het ontwerp voor de softwarearchitectuur zijn ook functionele eisen opgenomen, onder meer in de vorm van use cases. Voor de kwaliteitswaarborging van de software is het van belang om aan deze eisen te voldoen, tijdens de sprint meetings wordt waar mogelijk de functionaliteit gedemonstreerd. Dit stelt de opdrachtgever in staat in te grijpen en vergroot de kans dat de opdrachtgever tevreden is met het eindresultaat.

7. De projectorganisatie

7.1. Betrokkenen

Alle betrokkenen staan vermeld op de eerste pagina van het binnenwerk van dit document.

7.2. Communicatie

Bij de aanvang van het onderzoek zijn enkele afspraken gemaakt over de wijze waarop de communicatie tussen de studenten en externe betrokkenen, zoals de docentbegeleider en opdrachtgever, verloopt.

Overeengekomen is dat de opdrachtgever op de hoogte wordt gehouden tijdens wekelijkse scrum meetings. In het geval van ernstige problemen of het dreigen te mislukken van een sprint/functionaliteit zal het team ad-hoc contact opnemen. Andersom is het team ook op verzoek van de begeleider beschikbaar voor afspraken.

De communicatie verloopt in eerste instantie via e-mail en kan worden opgevolgd met een afspraak.

7.3. Documentatie

De teamleden zijn verantwoordelijk voor het documenteren, en bijwerken van bestaande documentatie gedurende het project. Zij zijn ook verantwoordelijk voor het beschrijven van hun (technische) keuzes in de scriptie die aan het einde van het project wordt opgeleverd.

8. Risico's

In de oriënterende fase hebben wij een inventarisatie van de risico's gemaakt die zouden kunnen optreden. In de volgende tabel staan deze vermeld, met de bijbehorende prioriteit. Daarnaast vermeldt de tabel ook tegenmaatregelen die kunnen worden ingezet in het geval dat het betreffende risico optreedt.

Beschrijving risico	Kans	Impact	Prioriteit	Maatregel
Uitval door ziekte	0,1	2	0,2	Taken overdragen aan andere projectleden
Technische complicaties	0,15	3	0,45	Alternatief gebruiken/Technische ondersteuning zoeken.
Uitloop planning	0,1	2	0,2	Extra tijd erin steken, ruimer plannen
Te weinig kennis vooraf	0,1	3	0,3	Vooraf goede kennis opdoen via bijvoorbeeld internet en/of boeken.
Ontwerpfase loopt uit door besluiteloosheid of ontevredenheid van bepaalde partijen	0,3	3	0,9	Vasthouden aan de deadline voor het einde van de ontwerpfase.
Product voldoet niet aan de eisen	0,2	4	0,8	We kunnen dit verminderen door wekelijks contact te houden met de begeleider.

9. Literatuur

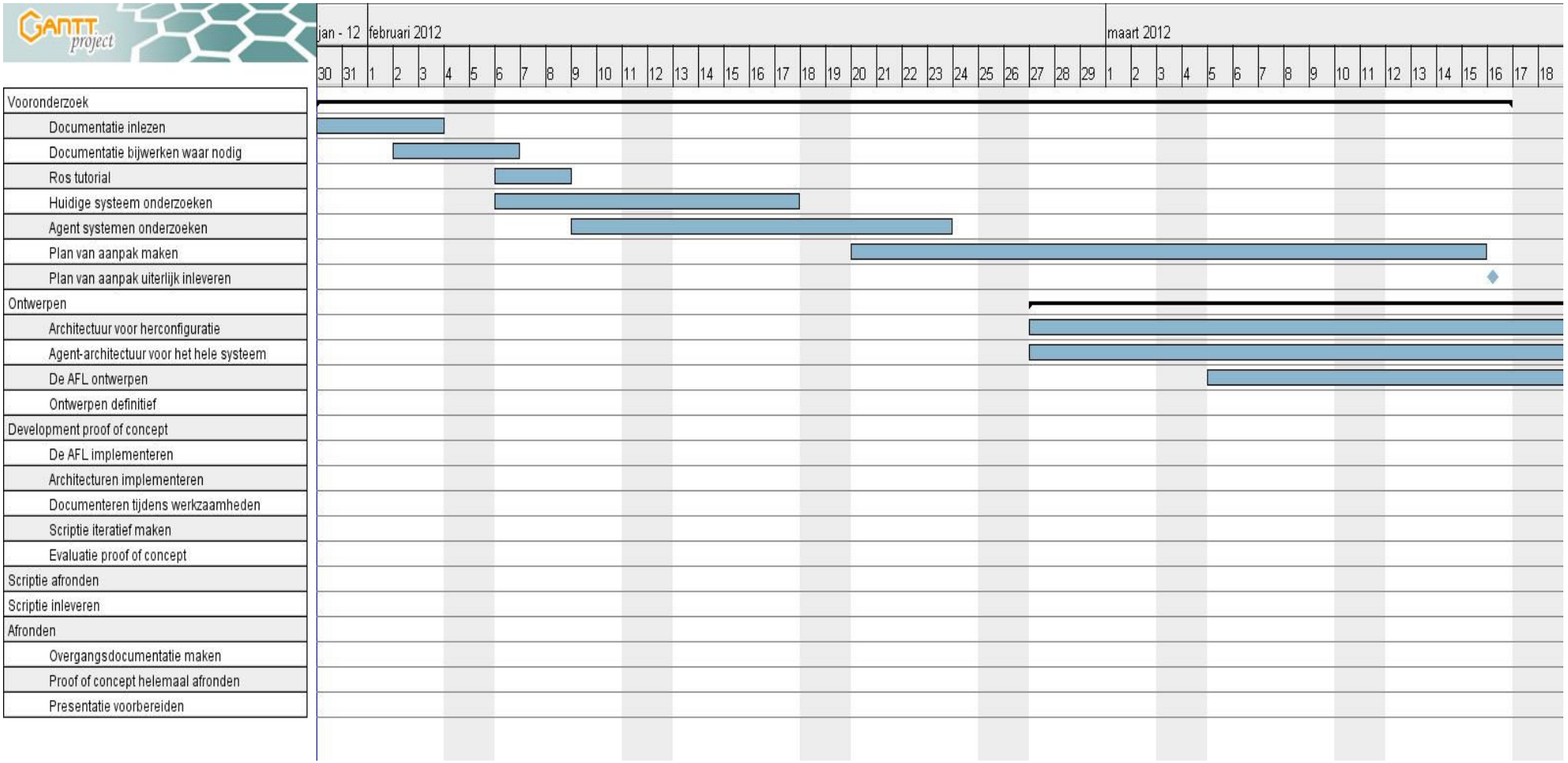
[1] A.Gunasekaran, "Agile manufacturing: the 21stcentury competitive strategy", 2001

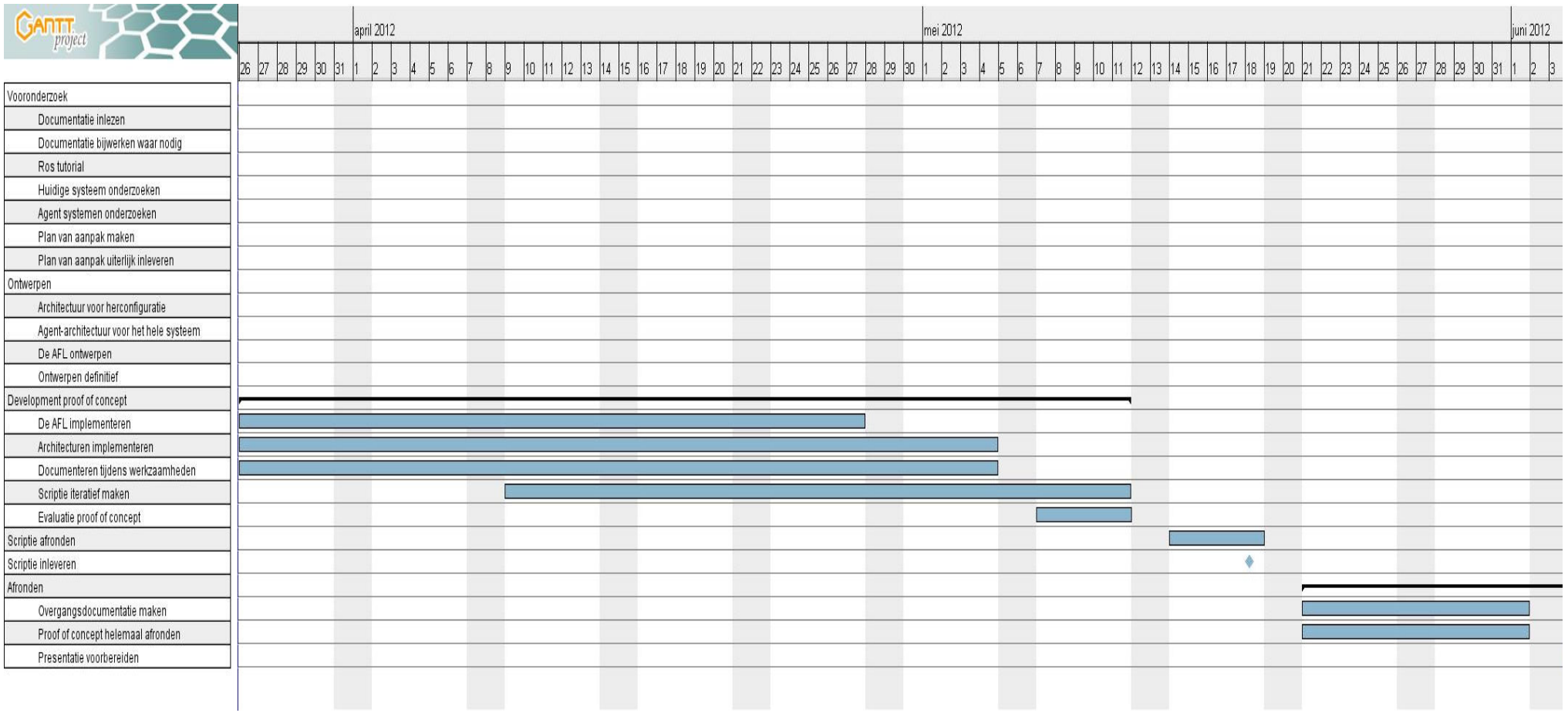
[2] E. Puik and L.J.M. van Moergestel, "Agile multi-parallel micro manufacturing using a grid of Equiplets", IPAS 2010 proceedings, p. 271-282, 2010

[3] M. Wooldridge, An Introduction to Multi Agent Systems, Second Edition, Wiley, 2009

[4] M. Wooldridge and N. Jennings. Intelligent Agents: Theory and practice. The knowledge Engineering Review, p. 115-152, 1995

Bijlage A. Gantt-chart planning




GANTT project
april 2012
mei 2012
juni 2012
26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3
Vooronderzoek
Documentatie inlezen
Documentatie bijwerken waar nodig
Ros tutorial
Huidige systeem onderzoeken
Agent systemen onderzoeken
Plan van aanpak maken
Plan van aanpak uiterlijk inleveren
Ontwerpen
Architectuur voor herconfiguratie
Agent-architectuur voor het hele systeem
De AFL ontwerpen
Ontwerpen definitief
Development proof of concept
De AFL implementeren
Architecturen implementeren
Documenteren tijdens werkzaamheden
Scriptie iteratief maken
Evaluatie proof of concept
Scriptie afronden
Scriptie inleveren
Afronden
Overgangsdocumentatie maken
Proof of concept helemaal afronden
Presentatie voorbereiden

Bijlage B - Evaluatie eigen functioneren

Dit is de verplichte bijlage die mijn evaluatie geeft over het afstudeerproject. Eerst zal ik het proces evalueren en vervolgens het tot nu toe opgeleverde product.

In de eerste oriënterende weken van het project waar we veel onderzoek deden, waren we nog vrij vaak afhankelijk van het vorige projectteam bij het werkend krijgen van de robot. Met alleen de documentatie erbij was het niet gelukt. Ik heb gemerkt dat het goed bijhouden van technische projectdocumentatie tijdens het project essentieel is voor een goede overdracht. Bij achteraf documenteren zijn veel zaken vanzelfsprekend geworden en worden ze niet beschreven.

In de eerste weken heb ik naar mijn mening goed kunnen werken en me goed kunnen verdiepen in het project en de achtergronden. Maar daarna tijdens de ontwerpfase ging het niet snel genoeg meer. De ontwerpfase liep teveel vertraging op door gebrek aan vergaderingen en andere bijeenkomsten waarbij denk ik alle direct betrokkenen aanwezig zouden moeten zijn voor een snel verloop. Hierdoor zijn we ook veel input vanuit een andere hoek misgelopen. We hebben in deze fase te veel gewacht op goedkeuring dan wel input van anderen en dat is de productiviteit toen niet ten goede gekomen.

Mede door de lange ontwerpfase is er te weinig tijd besteed aan de implementatie van het ontwerp. Hierdoor zijn ook niet alle producten opgeleverd voor het schrijven van deze scriptie en dat is vervelend. Ook heb ik het gevoel dat de implementatie van de opgeleverde producten nog veel beter kan als er nog een aantal weken de tijd voor wordt genomen.

Ondanks mijn goede voornemens is het toch niet gebeurd dat ik de scriptie vanaf het begin van het project goed heb bijgehouden. Hierdoor is deze pas in de laatste weken tot stand gekomen. Het werk dat dit heeft opgeleverd is me toch erg tegengevallen. Waar je in een normaal projectteam een gedeelte van het werk kunt maken, moet nu alles zelf worden geschreven en dat heb ik sterk onderschat.

Ik heb ook de projecttijd onderschat. Ik had verwacht veel meer te kunnen doen, echter als je alles tot in detail wilt onderzoeken gaat dit erg langzaam. Dit is echt een proces wat je zelf moet ondervinden voordat je gelooft wat anderen zeggen.

Al met al heb ik veel geleerd en goede ontwerpen kunnen maken. Verder heb ik veel initiatief kunnen tonen. Van de fouten die ik heb gemaakt heb ik geleerd en dat ga ik meenemen in mijn verdere loopbaan.