Holonic task execution control of multi-mobile-robot systems

Johan PHILIPS

Jury:
Prof. dr. ir. J. Vandewalle, chair
Prof. dr. ir. H. Van Brussel, supervisor
Prof. dr. ir. H. Bruyninckx, co-supervisor
Prof. dr. ir. T. Holvoet
dr. ir. Paul Valckenaers
dr. ir. Erwin Aertbeliën
Prof. dr. Gérard Morel
(Université De Lorraine)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

December 2012

© KU Leuven – Faculty of Engineering
Celestijnenlaan 300B box 2420, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2012/7515/140
ISBN 978-94-6018-606-6

Voorwoord

Het is zover! Na ruim 11 jaar KU Leuven-student te zijn, ben ik eindelijk klaar om mijn studies af te ronden. Toen ik in september 2001 begon aan de opleiding licentiaat in de informatica had ik helemaal niet in gedachte dat ik anno 2012 nog steeds hier aan de KU Leuven zou *zitten*.

Wat eerst leek als uitstel om het échte leven in te gaan, met de extra masteropleiding in artificiële intelligentie, groeide uit tot ongeveer 7 jaar onderzoek aan het departement werktuigkunde.

Tijdens deze periode heb ik veel nieuwe mensen leren kennen en ik zou hier dan ook even de tijd willen nemen om een aantal mensen in het bijzonder te bedanken voor de aangename tijd die we samen gekend hebben.

Ten eerste wil ik mijn verschillende *werkgevers* bedankt bedanken om mij de kans te geven deel uit te maken van de robotica-afdeling. Marnix, om mij aan te werven voor het MAIA-project en mij zo te introduceren in de MLR-groep. Professor Van Brussel (of mag ik nu al Rik zeggen?), bedankt om al uw nuttige feedback, uw ongezouten mening, uw eindeloos geduld en onuitputbare kennis! Herman, bedankt voor uw inzichten tijdens onze vergaderingen, de manier waarop u altijd alles een plaats kon geven in een groter geheel en toch tot in detail mee kon helpen zoeken naar het oplossen van praktische problemen. Paul, bedankt voor de zee aan metaforen waarmee u het onderzoek kon samenvatten en aanprijzen. Ik moet toegeven dat ik soms kwijtraakte in uw redeneringen maar uw ideeën leidde steeds tot nieuwe inzichten in mijn onderzoek.

Ten tweede zou ik mijn *bureaugenoten* hartelijk willen bedanken voor alle discussies, de ene wat filosofischer als de andere, die we gehad hebben tijdens onze tijd in ons bureau. Gerolf, voor zijn nuchtere kijk op heel de academische wereld en expertise in alles wat met informatica in het algemeen en linux in het bijzonder te maken had. Shu, om steeds weer oprecht verbaasd te lijken bij het vertellen van iets nieuws (*Really?!*), zijn leuke kadootjes en souveniers als hij terug kwam van Taiwan en zijn gastvrijheid tijdens ons recent bezoek aan

zijn thuisland. En natuurlijk ook Alexander, voor zijn begeleiding tijdens mijn masterthesis, onze eindeloze maar altijd even interessante gesprekken over alles wat financiële, economische en politieke actualiteit te maken heeft. Ook zijn veelvuldige boekentips ben ik dankbaar. Mijn *Amazon wish list* staat nog vol aanraders.

Naast mijn bureaugenoten, ben ik ook de andere leden van de MLR-groep dankbaar. Eric, bedankt voor de uren en uren (en uren) die we doorgebracht hebben aan de rolstoelen in het robotlab, overdag, 's avonds, 's nachts. Uw kennis in elektronica en mechanica heeft mij talloze keren uit de nood geholpen. Ook de ontspannende momenten tijdens onze *basketball lunch breaks* zal ik zeker herinneren. Dirk, ook bedankt voor de tijd die we tijdens het MAIA-project doorgebracht hebben om *Sharioto* werkende te krijgen. Het *lijnexperiment* op de zolder in de wetenschapsbibliotheek kent geen geheimen meer voor ons. Bedankt ook Erwin voor de nauwe opvolging van mijn doctoraatsonderzoek en Manu, Davy en Xavier om steeds klaar te staan om te helpen met eender welk probleem ik had.

Naast mijn collega's van de MLR-groep, ben ik ook Bart en Jan dankbaar voor hun enorme hulp bij het integreren van de PROSA-implementatie met de robotsoftware. Het laatste jaar, *op de vierde verdieping*, heb ik zoveel tijd in jullie bureau doorgebracht dat ik jullie eigenlijk bij *bureaugenoten* zou moeten rekenen. Bedankt om steeds klaar te staan.

De andere leden van de roboticagroep wil ik ook bedanken voor de aangename momenten in het robotlabo of tijdens oefenzittingen en vergaderingen. Bedankt Koen, Wilm, Nick, Tinne, Steven, Enrico, Marcus, Hans, en iedereen die ik nu vergeet.

Ik ben ook Nele dankbaar om bij elke *klasreünie* die we in de gang hadden mij de vraag te stellen: *Is't al af?*. Jouw directe aanpak hield mij gemotiveerd om het ook effectief af te maken. Wie had dat ooit gedacht, de Grieks-Wiskunde-8 van 2001 is nu een klas *vol* doctors!

Onderzoek aan ons departement zou niet mogelijk zijn zonder de ondersteuning die we krijgen van onze *ATP-ers*, dus bedankt, Jean-Pierre, Bertram, Karin, Lieve, Carine, Regine, Ann, Dirk, Eddy, Ronny en Jan. Jullie stonden steeds klaar om te helpen met al mijn praktische vragen.

Natuurlijk heb ik ook veel steun gekregen bij vrienden en familie en wil ik hier ook een aantal van hen speciaal bedanken.

Gaming buddy (en mede-oprichter van Mambo-Jumbo) Kristof, bedankt voor alle ontspannende momenten de voorbije 11 jaar. Jij staat altijd klaar om mij te helpen met wat dan ook en de uren die we doorgebracht hebben al gamend

zorgde voor de soms noodzakelijke ontspanning en afleiding na de zoveelste tegenslag op het werk.

Training buddy Yuchen, bedankt om de voorbije maanden mee in de powerzaal te gaan trainen. Uw training spirit zorgde op sportief vlak bij mij voor extra motivatie.

Ook bedankt aan iedereen van basketbalclub BBC Haacht en Vipa Leuven voor zowel de ontspanning voor en na en de inspanning tijdens de trainingen en wedstrijden de voorbije jaren.

Al heeft onze relatie niet standgehouden, zou ik Liesbeth toch willen bedanken om mij moed in te spreken en gemotiveerd te houden de laatste jaren. Geert en Ronny, ook jullie bedankt om mij in jullie gezin op te nemen de voorbije jaren.

Tot slot ben ik mijn moeder, mijn vader, bobonne en Fanny dankbaar voor alle goede zorgen, onderdak en financiële steun. Bedankt om mij deze kans te geven, zonder jullie had dit mij nooit gelukt!

Abstract

Robot control software has rapidly evolved the past decade resulting in increasingly complex applications in increasingly dynamic environments. Nevertheless, most existing robot software architectures and frameworks are robocentric. If a resource has to be shared between multiple robots, no separate component or agent is defined to represent that resource. This lack of reflection of reality impedes robustness with respect to unexpected changes or disturbances and limits applicability.

This dissertation presents a paradigm shift towards holonic robotic systems and explicit resource allocation, in which structural decomposition is essential. Consequently, physical entities, required in the application, receive their rightful place in the design, in contrast to the robocentric approach common in current robot software.

In manufacturing control, holonic systems have earned their merit and resulted in more scalable and flexible systems, robust against unexpected disturbances. A pioneer in this field of holonic systems is KU Leuven's Product-Resource-Order-Staff architecture (PROSA) and its short-term forecasting through delegate multi-agent systems (D-MAS). After being successfully applied to numerous manufacturing applications, this thesis investigates the applicability of PROSA and D-MAS to robotics.

The scope of this research is limited to mobile service robotics and a multi-robot navigation scenario is presented to evaluate how well holonic systems fit in service robotics. A potential case study of this scenario is the deployment of a fleet of robotic wheelchairs in a hospital or retirement home, autonomously transporting patients from A to B.

In the first part of the literature overview, concepts behind the holonics paradigm, in general, and PROSA and D-MAS, in specific, are introduced. Also, benefits of adopting structural decomposition and reflection of reality in the domain of robotics are identified.

The second part reviews existing robot software architectures and frameworks, identifying their shortcomings with respect to separation of concerns and applicability. State of the art in multi-robot path planning and coordination is also evaluated keeping the proposed navigation scenario in mind.

After this literature overview, design guidelines to implement holonic robotic applications are provided. The current PROSA implementation, the Holonic Manufacturing Execution System (HMES), and its integration to the existing mobile robot platforms at KU Leuven is also discussed. A set of use cases of holonic systems in service robotics is presented, out of which two are implemented.

Multi-robot simulation experiments evaluate the complexity, scalability and robustness of the HMES. Scalability experiments demonstrate how the HMES copes with a larger set of navigation requests, on the one hand, and a larger environment, on the other hand. Finally, robustness against the unexpected is illustrated by blocking one of the corridors in the environment.

Simulation experiments illustrate the benefits of resource awareness and making each shared resource in the environment a first-class citizen in the software design. Experimental data in simulation show that a coordinated approach with explicit resource allocation achieves better performance in path execution, measured by the defined benchmarks.

The different experiments, in simulation and on the hardware platforms, clearly show the power of structural decomposition and short term forecasts that the HMES offers. The multi-robot path planning is distributed among these resources responsible for the planning, i.e. allocation, in the physical entity they represent. This largely simplifies the multi-robot path planning problem and increases scalability. The short term forecasts take care of unexpected disturbances to the resulting paths.

This thesis provides a first step into applying holonic systems into service robotics and results in multi-robot navigation are promising. The road is paved to future research that further investigates the introduction of holons in other robotics domains such as manipulation or human robot interaction.

Beknopte samenvatting

Robotsoftware heeft een grote evolutie gekend het laatste decennium. Mede hierdoor worden toepassingen steeds complexer en de omgevingen waarin ze uitgevoerd worden meer dynamisch. Desalniettemin missen de meeste huidige robotsoftwarearchitecturen en robotsoftwareraamwerken een correcte weerspiegeling van de realiteit, waardoor ze moeilijker om kunnen gaan met onverwachte veranderingen of storingen waardoor hun toepasbaarheid beperkt blijft.

Deze dissertatie stelt een paradigmawijziging voor waarin holonische robotsystemen en expliciete bronallocatie centraal staan. Structurele decompositie is essentieel in deze aanpak. In tegenstelling tot huidige robotsoftware, waarin de robot centraal staat, krijgen fysische entiteiten, die nodig zijn in de toepassing, hun welverdiende plaats in het ontwerp.

In productieprocescontrole hebben holonische systemen succesvol aangetoond dat ze leiden tot systemen die beter schalen, flexiebeler zijn en robuust zijn bij onverwachte storingen. Een pionieer in dit domein van holonische systemen is de Product-Resource-Order-Staff architectuur (PROSA) en haar korte-termijnvoorspellingen via gedelegeerde multi-agent systemen (D-MAS). Ontwikkeld aan de KU Leuven, deze architectuur is meermaals succesvol toegepast in productietoepassingen. Deze thesis onderzoekt de toepasbaarheid van PROSA en D-MAS in de robotica.

Het toepassingsdomein van deze thesis is beperkt tot mobiele assistieve robotica. Een scenario waar meerdere robots onafhankelijk navigeren, wordt gebruikt om na te gaan hoe goed holonische system in de assistieve robotica passen. Een mogelijke toepassing van dit scenario is een vloot van robotrolstoelen die ingezet worden in een ziekenhuis of rusthuis. Deze robotrolstoelen kunnen autonoom patiënten van A naar B te brengen.

In het eerste deel van het literatuuroverzicht worden de concepten achter het holonische paradigma geïntroduceerd en worden PROSA and D-MAS besproken.

De voordelen van het toepassen van structurele decompositie en weerspiegeling van de realiteit in de robotica worden ook geïdentificeerd.

Het tweede deel bespreekt bestaande robotsoftwarearchitecturen en raamwerken, identificeert hun beperkingen met betrekking tot het scheiden van verantwoordelijkheden en evalueert hun toepasbaarheid. Expertise in padplanning- en coördinatietechnieken voor meerdere robots wordt ook geëvalueerd met het vooropgestelde navigatiescenario in gedachten.

Na dit literatuuroverzicht worden ontwerpregels voorgesteld om holonische robotapplicaties te ontwikkelen. De huidige PROSA-implementatie, het Holonic Manufacturing Execution System (HMES), en haar integratie met bestaande mobiele robotplatformen aan de KU Leuven wordt besproken. Een aantal voorbeelden van holonische systemen in de assistieve robotica worden voorgesteld, waarvan er twee werden uitgewerkt.

De complexiteit, schaleerbaarheid en robustheid van het HMES worden geëvalueerd aan de hand van simulatieëxperimenten met meerdere robots. Schaleerbaarheidexperimenten tonen aan hoe het HMES een toename in het aantal robots enerzijds, en een toename in de grootte van de omgeving anderzijds, aankan. Tenslotte wordt de robustheid ten opzichte van onverwachte storingen geïllustreerd door één van de gangen in de omgeving te blokkeren tijdens de uitvoering.

De simulatieëxperimenten illustreren de voordelen van het in acht nemen van gedeelde bronnen in de omgeving. Daarnaast krijgt ook elke gedeelde bron een eigen plaats tijdens het softwareontwerp. Experimentele data tonen aan dat een gecoördineerde aanpak met expliciete bronallocatie betere prestaties levert in het uitvoeren van een pad en dit gemeten door vooropgestelde maatstaven.

De verschillende experimenten, zowel in simulatie als op de hardwareplatformen, tonen duidelijk de kracht aan van de structurele decompostie en de korte-termijnvoorspellingen, die aangeboden worden door het HMES. De padplanning voor meerdere robots wordt gedistribueerd over bronnen in de omgeving, elk verantwoordelijk voor hun eigen planning, hetzij de allocatie. Hierdoor wordt het padplanningprobleem voor meerdere robots in grote mate vereenvoudigd en neemt de schaleerbaarheid van de aanpak toe. De korte-termijnvoorspellingen nemen onverwachte storingen op de berekende paden voor hun rekening.

Deze thesis biedt een eerste stap aan in het toepassen van holonische systemen in de robotica en de resultaten over de navigatie van meerdere robots zijn veelbelovend. De weg is geëffend voor toekomstig werk, dat de introductie van holons in andere roboticadomeinen, zoals manipulatie en mens-machineïnteractie, verder onderzoekt.

Contents

Abstract v

Contents ix

List of Definitions xix

List of Figures xxvii

List of Tables xxxix

1 Introduction 1

1.1 Acknowledgements 1

1.2 Motivation 2

1.3 Objectives 5

1.4 Problem domain 7

1.4.1 Issues 7

1.5 Contributions 8

1.5.1 Applicability: HMES analysis 8

1.5.2 Robustness through bio-inspired delegate MAS . . . 9

1.5.3 Integration of legacy hardware and software and existing state of the art through incremental compliance . . . 9

1.5.4 Scalability through structural decomposition 9
1.5.5 Experimentation: multi-robot coordination 10
1.5.6 Graphical tools 10
1.5.7 Communication protocol 11
1.6 General overview 11

2 Holonic Systems and Why They are Important for Robotics 15

2.1 Introduction 15
2.2 System complexity levels 17
2.2.1 *S1*: Centralised one-robot system 18
2.2.2 *S2*: Centralised multi-robot coordination 18
2.2.3 *S3*: Decentralised coordination with shared tasks . . 18
2.2.4 *S4*: Decentralised coordination with non-competing tasks 19
2.2.5 *S5*: Decentralised coordination with competing tasks 19
2.3 Terminology in software engineering 20
2.3.1 Component-based software engineering versus object-oriented programming 20
2.3.2 Framework versus architecture 21
2.3.3 Separation of concerns 22
2.3.4 Encapsulation 23
2.3.5 Decomposition 24
2.4 Holons and holonic systems 25
2.4.1 Etymology 25
2.4.2 Definitions 25
2.4.3 Benefits of reflection of reality 26
2.4.4 Previous applications 28
2.4.5 Relation to system complexity levels 28

2.4.6 Relation to robotics & software engineering 28
2.5 Reference architecture PROSA 29
2.5.1 Definition 29
2.5.2 Resource holons and their allocation 30
2.5.3 Product holon 35
2.5.4 Order holon 37
2.5.5 Staff holon 39
2.5.6 Separation of concerns 40
2.5.7 Interaction & data flow 40
2.6 D-MAS as coordination and short-term forecasting mechanism 41
2.6.1 Introduction 41
2.6.2 Exploring ants 43
2.6.3 Intention ants 43
2.7 Generalisation and applicability 44
2.7.1 Manufacturing 45
2.7.2 Open-Air engineering 47
2.7.3 Logistics 48
2.8 Conclusion 50

3 Related Work and State of the Art in Holonic Systems 53
3.1 Introduction 53
3.2 Component-based robot software frameworks 54
3.2.1 Overview 54
3.2.2 Orca 55
3.2.3 RT-Middleware 56
3.2.4 ROS 57
3.2.5 OROCOS 58

3.2.6 Microsoft RDS 59
3.2.7 Player/Stage 61
3.3 Component-based robot software architectures 61
3.3.1 CLARAty 61
3.3.2 LAAS 62
3.3.3 JAUS 64
3.3.4 Mobile-R 64
3.4 Holonic systems 65
3.4.1 Introduction 65
3.4.2 Magenta 67
3.4.3 ADACOR 67
3.4.4 iCoMAS 68
3.4.5 Tactical-AgentFly 2 68
3.5 Multi-robot path planning & coordination 68
3.5.1 Planning approaches 69
3.5.2 Multi-robot coordination 70
3.5.3 Other related approaches 72
3.6 Why PROSA and D-MAS? 73
3.6.1 Complementary to existing approaches 73
3.6.2 Previous applications 74
3.6.3 System complexity 74
3.6.4 Reflection of reality 75
3.6.5 Short-term forecasts 76
3.6.6 Chosen software approach 76
3.7 Conclusion 77

4 Holonic Robot Applications **79**

4.1 Introduction 79
4.2 HMES structure 80
4.2.1 Multi-models 80
4.2.2 HMES-in-the-loop Simulation 81
4.3 Integrating the HMES with existing robot software 83
4.3.1 Level of integration 83
4.3.2 Communication interface 84
4.3.3 Robot world 84
4.4 Applications in general 85
4.5 Use case I: single robot allocation in general 86
4.6 Use case II: obstacle avoidance 89
4.6.1 Goal 89
4.6.2 Resource allocation solution 89
4.6.3 Resource capability solution 92
4.7 Use case III: multi-robot door opening 93
4.7.1 Goal 93
4.7.2 PROSA & D-MAS solution 93
4.8 Use case IV: allocation of external sensors 94
4.8.1 Goal 94
4.8.2 Static environment sensor 94
4.8.3 Sensor redundancy 95
4.8.4 Sensor sharing 96
4.8.5 Looking ahead 96
4.9 Use case V: robot allocation policies 96
4.9.1 Goal 96
4.9.2 D-MAS approach 97
4.9.3 Staff holon approach 97

4.9.4 Human factors in robotic wheelchair applications . . 97
4.10 Use case VI: global and local multi-robot coordination . . . 98
4.10.1 Goal . . . 98
4.10.2 Global multi-robot path planning . . . 98
4.10.3 Local multi-robot path planning . . . 99
4.11 Conclusion . . . 100

5 HMES Analysis **101**
5.1 Introduction . . . 101
5.2 Parameter calibration . . . 102
5.2.1 Exploring rate . . . 102
5.2.2 Refresh rate . . . 106
5.2.3 Evaporation rate . . . 108
5.2.4 Hop limit . . . 109
5.2.5 Order holon selection criteria . . . 109
5.2.6 Conclusion . . . 110
5.3 Computational complexity . . . 110
5.3.1 Parameters . . . 111
5.3.2 Analysis . . . 111
5.3.3 Order holon . . . 112
5.3.4 Resource holon . . . 118
5.3.5 Summary . . . 119
5.4 Scalability . . . 120
5.4.1 Increasing navigation requests . . . 121
5.4.2 Increasing the environment . . . 124
5.5 Robustness against the unexpected . . . 128
5.6 Current implementation limitations . . . 130

5.6.1 Multi-resource allocation 130
5.6.2 Continuous space 131
5.6.3 Safety supervision 132
5.6.4 Execution speed 133
5.7 Conclusion 133

6 Multi-Robot Coordination in Simulation 135
6.1 Introduction 135
6.2 Problem statement 136
6.2.1 Livelock and deadlock 136
6.2.2 Suboptimal path execution 137
6.2.3 Starvation 137
6.3 Experimental setup 138
6.3.1 Environment and scenarios 138
6.3.2 Path Planner 140
6.4 Reactive approach 141
6.4.1 Trajectory controller 142
6.5 PROSA & D-MAS approach 142
6.5.1 Trajectory controller 142
6.5.2 Resource holons 143
6.5.3 Product holons 144
6.5.4 Order holons 145
6.6 Benchmarks 145
6.6.1 Excessive distance 145
6.6.2 Elapsed time 146
6.6.3 Hausdorff distance 146
6.7 Results 146

6.7.1 Without coordination 146
6.7.2 With coordination 151
6.8 Conclusion 154

7 Coordination of Multiple Robotic Wheelchairs 155
7.1 Introduction 155
7.2 Robotic wheelchairs 156
7.2.1 SARA 156
7.2.2 INGA 157
7.3 Autonomous navigation 157
7.3.1 State estimator 158
7.3.2 Path planning 159
7.3.3 Path interpolation 160
7.3.4 Time mapping 162
7.3.5 Trajectory control 163
7.4 Coordination 164
7.5 Integration 164
7.6 Experimental setup 165
7.7 Results 166
7.7.1 Trajectory Control 166
7.7.2 Coordination 167
7.7.3 Discussion 169
7.8 Conclusion 171

8 Conclusion and Future Work 175
8.1 Contributions and objectives revisited 176
8.1.1 Applicability (cf. chapters 2, 4, 5, 6 and 7) 176
8.1.2 Robustness (cf. chapters 5 and 6) 178

8.1.3 Integration (cf. chapters 2, 4 and C) 178
8.1.4 Scalability (cf. chapter 5) 179
8.1.5 Experimentation (cf. chapters 6 and 7) 180
8.1.6 Graphical tools (cf. chapters A and B) 181
8.2 Future work 181
8.2.1 Application specific holons 181
8.2.2 World model extensions 182
8.2.3 Case study 183
8.2.4 Explore robotics domains 183

A Multi-Robot Simulator **185**
A.1 Background 185
A.2 Overview 185
A.3 Collision detection 186
A.4 3D Models 187
A.5 Conclusion 188

B Robot Graphical User Interface **191**
B.1 Motivation 191
B.2 Overview 191
B.3 Conclusion 199

C Communication Protocol **201**
C.1 Motivation 201
C.2 Google protocol buffers 201
C.3 Robot data protocol 202
C.4 Conclusion 209

Bibliography **211**

Curriculum Vitae **223**

List of Publications **225**

List of Definitions

Related to D-MAS

D-MAS This bio-inspired coordination mechanism, called delegate multi-agent system (**D-MAS**) enables holons to receive short-term forecasts with respect to resource allocation and achieve improvement in their task execution through these predictions. The PROSA order holons create (many) lightweight delegate agents to perform some task on their behalf.

Delegate Pattern The **delegate pattern** enables agents, e.g. the PROSA order holons, to *delegate* part of their task to small lightweight agents, referred to as ants. These agents travel through the environment, e.g. the resource holons, execute the requested task, e.g. gather allocation information, and report back to the agent which created them.

Exploring Ant The **exploring ant** virtually traverses the resource graph and searches for a possible solution for the order holon it is linked to. The ant requests those resource holons for information about resource availability and timing without actually allocating the resource. The resulting time is then used as start time for the next resource request. To be able to generate these forecasts, the lightweight agents can make use of the *what-if* functionality of resource holons. As these holons reflect a part of reality, they have to be able to answer what-if questions about the dynamic behaviour of their corresponding reality. Therefore, in a system with D-MAS, every resource holon contains a model that supports virtual execution.

Intention Ant **Intention ants** propagate the order holon's intentions, informing resource holons about future visits (allocations). Their behaviour is similar to that of exploring ants with the exception that

its task is to book the reservation on the resources rather than to request a virtual execution.

Preview Control In contrast to feed forward or feed back control, **preview control** deals with near-future events which are estimated or forecast. In previous research, preview control has mostly been applied in local, fine-grained robot control or shared control with a user. With PROSA and D-MAS this technique is also adopted to increase the robustness in the global planning.

Resource Graph The **resource graph** has resource holons as vertices and the directed edges between those resource holons reflect links between the corresponding entities in the environment. For example, a door is connected to the two rooms or areas it provides access to.

Related to HMES

Evaporation Rate The evaporation rate resembles the pheromone decay rate in ant colonies, i.e. solutions found by exploring ants remain only valid for a limited time interval. The rate defines how many times per seconds the set of solutions is reviewed and old solutions are removed. Evidently, the rate, at which these solutions *evaporate*, should be lower then the exploring and refresh rate.

Exploring Rate The exploring rate controls how many exploring ants are sent per second per order. Consequently, the lower this rate, the longer it takes before an initial solution is found and presented at the order holon. Also, a lower exploring rate results in fewer solutions received by the order holon at a particular time.

HMES The Holonic Manufacturing Execution System (HMES) is the PROSA and D-MAS implementation which was developed at PMA/KU Leuven. This system was adapted so it could be used in the multi-mobile-robot scenario.

Hop Limit The hop limit defines how many hops exploring ants are allowed to make during their search. Every operation virtually executed on a resource counts as one hop. It ensures the exploring ants do not endlessly search for a solution in the research graph. Due to their random choices at junctions, some ants might not find a solution within a desired time frame. If an exploring ant reaches its hop limit, it stops exploring and returns with a failed status.

Refresh Rate The refresh rate determines how many intention ants each order sends out per second. The rate determines how fast a currently

chosen solution is checked and how fast allocations on resource holons are made. Consequently, the system response time to disturbances is directly related to this parameter.

RELEASE TIME A higher release time allows the order holon to send out more exploring ants and selecting a solution before committing to it. This increases the probability of finding a superior solution even if a low E/R ratio is used. Of course, a higher release time also increases the delay between order holon creation and its execution, which in the case of multi-robot navigation not desirable.

Related to Holonics

AGGREGATION The relationship between a holon and one of its higher authorities, i.e. a holon representing a larger whole, is defined as **aggregation**. In this context, aggregation does not solely define a merger between a set of holons but also includes the necessary coordination from the aggregated holon.

COMPLEX ADAPTIVE SYSTEMS Those systems are **complex** due to their diversity and the fact they consist of multiple interconnected elements and they are **adaptive** in the way that they are able to cope with changes and disturbances and are able to learn from past experiences. Examples of complex adaptive systems in other domains include the stock market and social insect colonies.

HOLON Arthur Koestler defines a **holon** as an autonomous, self-reliant unit, which is able to function without instructions from higher authorities. Simultaneously, this holon can be subject to control from one or more of these higher authorities.

Related to Multi-Robot Coordination

DEADLOCK Similar to a livelock, a **deadlock** is caused when two or more entities wait for each other and do not make progression in their task. The difference with respect to livelock is the way these entities *wait*. In a deadlock situation, the progress of one entity depends on the completion of the (partial) task of another entity and vice versa. Therefore they do not change their behaviour but are waiting for each other to finish.

LIVELOCK In a **livelock** two or more entities constantly change their behaviour with regard to one another, none of them progressing in their task. This is caused by the robocentric approach most current robot applications follow. Each robot executes its own task, assuming the

environment is implicitly allocated for its needs and not being aware other users might be present.

Starvation In some multi-robot navigation techniques, it might occur that one or more robots are unable to execute their task because other, usually higher-priority robots, continuously occupy one or more of their resources. This problem is defined in computer science as **starvation**. A livelock can be seen as a special case of starvation, in which all involved entities are in starvation since none of them is able to continue their task.

Related to PROSA

Aggregated Order Holon Several order holons can be combined into an **aggregated order holon**. This holon typically contains a sequence of required orders at a particular time instance. E.g. a customer order can be divided into separate batch orders, and further into work orders.

Aggregated Product Holon Several product holons can also be combined into an **aggregated product holon** representing the combination of those process plans. This way, individual process plans can be efficiently combined but still remain independent from the other process plans in the aggregation. Moreover, an aggregated product holon defines the sequence and conditions in which those product holons are used.

Aggregated Resource Holon Resource holons can be combined into an **aggregated resource holon**, which coordinates these holons. This aggregation is dynamic and resource allocation of an aggregated resource holons implies the allocations of all the holons it represents. Therefore, a design guideline is to only allocate the minimal set of resources required for the application and, thus, also define the aggregated resource holons with that guideline in mind.

Order Holon An **order holon** represents a product *instance* that needs to be produced at a specific time instance. It is responsible to handle the required resource allocations to accomplish this. Therefore, the order holon consults its corresponding product holon to find out what services it needs and searches for the proper resources to accomplish it.

Product Holon A **product holon** encapsulates knowledge of a particular product type; how instances need to be produced and what services or operations are required. These services and operations are offered by resource holons.

PROSA The Product-Resource-Order-Staff Architecture, which was developed at PMA/KU Leuven, is a reference architecture developed in accordance with the holonic manufacturing paradigm. It defines three basic holons: resource, order and product and the optional staff holon.

Resource Holon A **resource holon** corresponds to any entity in the underlying world-of-interest offering services and operations to others, relevant for the application at hand. More concretely, any entity that will be used by other holons for those services, during some period of their lifetime, should be represented by their own resource holon.

Staff Holon In view of the complexity of resource allocation, **staff holons** may assist other holons through advice on particular aspects. However, this advice is not binding and the order and resource holons still take the final decisions.

Related to Software Development

Component-based Software Engineering In **component-based software engineering**, the emphasis is placed on designing separate, stand-alone **components**, which encapsulate a set of services or functionalities. These components can be connected to each other to form a desired application. Typically, the dependence between different components is minimal such that each component is reusable on its own.

Decomposition Decomposition of a software system into manageable subparts is a general principle of software design to cope with large and complex systems. Two main approaches are functional and structural decomposition.

Encapsulation Encapsulation strengthens the interoperability and reusability, and thus system flexibility, by hiding the internal component structure behind a public interface. With encapsulation, the communication interface remains untouched.

Functional Decomposition **Functional decomposition** is common in centralised systems in which each component represents a certain functional requirement. Examples of such components are a scheduler component, a planning component and a resource managing component. Although this approach is suitable for small or virtual systems, it is not scalable and has limited extendibility, whenever there exists an underlying physical system. In the latter case,

functional components become system bottlenecks if throughput increases or the underlying physical system extends.

Multi-Agent System A **multi-agent system** (MAS) consists of interacting independent **intelligent agents**. These agents are autonomous, self-reliant components with a local view of the world and together their interaction results in a decentralised control, able to solve problems which are difficult or impossible to solve using a monolithic system.

Object-oriented Programming In **object-oriented programming**, the software primitives are **classes**, which contain data field members to maintain a state and methods to enable behaviour of their instantiations, namely **objects**.

Reference Architecture A predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artefacts to enable their use. Often, these artefacts are harvested from previous projects.

Separation of concerns Separation of concerns focuses on separating distinct features in distinct software entities without overlapping functionality, thereby decoupling these software entities and thus increasing reusability.

Software application A typical **software application** consists of an instantiation of a software or reference architecture following a certain software framework.

Software Architecture Unlike a software framework, **software architectures** also impose structure between software components and specify their interactions. If the architecture and the responsibilities of its components are clearly defined but implementation of each component is left open to the user, the term **reference architecture** is often used instead of software architecture. A reference architecture provides a generic solution for a certain application domain.

Software Framework A **software framework** defines a common communication and interaction standard for software components. Optionally, it provides a predefined set of reusable components without imposing structure and interaction between components. Thus, it does not prescribe a particular architecture. Software framework users are, or should be, free to arrange components in any desired structure as long as each component is developed by following the imposed communication interface.

STRUCTURAL DECOMPOSITION Contrary to functional decomposition, **structural decomposition** is a more scalable and extendible approach, since each component represents a distinct entity in reality. The preference for such a structural decomposition can also be found in [Parunak 1997] in which Parunak states that agents should be *things*, not *functions*. Components divided based on relevant entities in the physical world resemble also more naturally the reality than functional abstractions. Functionality in such a structural decomposition emerges from the interaction of different agents.

List of Figures

2.1 Schematic representation of a the system complexity levels as defined by [Bruyninckx 2010]. 17

2.2 Schematic representation of the granularity levels of software entities as defined by [Bruyninckx 2010]. 21

2.3 Schematic representation of a typical software application implementing a particular software architecture and following a particular software framework. 22

2.4 Overview of the five aspects of separation of concerns as defined by [Bruyninckx 2010]. 23

2.5 Overview of the PROSA reference architecture. Each rounded rectangle represents one of the holon types used in PROSA and the arrows between these holons represent their interactions. 30

2.6 Example set of resource holons representing a mobile manipulator. Each rectangle represents a resource holon and the connections indicate aggregations. The granularity level of the resource holon changes from top to bottom. 32

2.7 Example set of resource holons representing a robotic wheelchair. Each rectangle represents a resource holon and the connections indicate aggregations. The granularity level of the resource holon changes from top to bottom. 32

2.8 The aggregation of different sets of resource holons into one comprehensive *universe* holon. This holon ensures proper reflection of reality and serves as a container for all physical entities which are not yet represented by resource holons or do not require this at this moment of time. 34

2.9 Rephrasing of PROSA for a generic application domain. Besides the generalisation in terminology, this schema relates the interactions between the holon types to three task specific questions. *How* a task is executed is communicated between the task and task type holon, deciding *when* the task is executed involves the task and resource holon and determining *what* or which operations can be done results from the interaction between the task type and resource holon. 45

2.10 Open-air engineering processes 47

2.11 An example of a chain conveyor network 48

2.12 Material handling at a typical cross-dock 49

3.1 Schematic overview of communication in Orca between two components running on different operating systems and written in different programming languages. This figure was adopted from [Makarenko et al. 2006]. 55

3.2 Schematic overview of how components are composed in RT-Middleware. From top to bottom it shows first functional and structural components composed into larger structural components which finally are composed into one robot component. In this example a service robot is depicted, consisting of a mobile base, two arms and stereo vision providing recognition. This figure was adopted from [AIST 2012]. 57

3.3 Schematic overview of communication in ROS. Nodes, depicted by ovals, can either publish data on topics, which is then read by other nodes subscribed to those topics, or either perform remote procedure calls through services. This figure was adopted from [Willow Garage 2012]. 58

3.4 Schematic overview of a navigation package in ROS. Nodes are depicted by ovals and the directed arcs indicate a pusblish/subscribe unidirectional communication channel. Adopted from [Quigley et al. 2009]. 58

3.5 Schematic overview of an OROCOS component. It shows the separation of configuration, computation and communication. Adopted from [Intermodalics 2011]. 59

3.6 Data flow in the Visual Programming Language (VPL) of Microsoft's Robotics Developer Studio. Adopted from [Microsoft 2012]. 60

3.7 The CLARAty Architecture. Adopted from [Nesnas et al. 2006]. 62

3.8 The LAAS Architecture. Adopted from [Alami et al. 1998]. 63

3.9 Schematic overview of the JAUS system communication topology. JAUS is a hierarchical architecture consisting from top to bottom out of subsystems, nodes and components. Each node typically has a node manager, which is a component providing universal functionality such as routing of messages within and between nodes. Next to this node manager, the communicator component provides similar functionality between subsystems. Adopted from [JAUS 2007]. 64

3.10 Schematic overview of a heterogeneous multi-robot system using different mobile agent types provided by the Mobile-R architecture. In this example a purely deliberative, purely reactive, hybrid and purely tele-operated robot system are combined in a multi-robot network. The deliberative agents share high level task information, while the reactive agents share behaviour data and the hardware agents share device specific data, all indicated by the dashed lines. Adopted from [Nestinger 2012]. 66

3.11 Deadlocks can occur and are related to which priority scheme is used. Here such a deadlock occurs since the target position of high priority robot R1 leaves low priority robot R2 unable to find a path to its target position. 70

4.1 Example of a Petri net modelling part of a life cycle of an agent, simplified for demonstration purposes. Each transition has an underlying model, which in turn can be a Petri net or an actual implementation model. 81

4.2 This Petri net models the life cycle of an agent in the HMES, from [Saint Germain 2010]. The life submodel is specific to each agent. 82

4.3 This scheme provides an overview of the communication between different *actors* in the multi-robot navigation application. . 85

4.4 This Petri net models the life cycle of the robot world in the HMES. In the life submodel, the handle event model processes incoming events. 86

4.5 This Petri net models the life cycle of the robot world in the HMES. Here, the life submodel contains a model for each possible event. 87

4.6 The mobile robot fleet at PMA/KU Leuven, two autonomous wheelchairs and one mobile manipulator. 87

4.7 Resource holons for a robotic wheelchair. 88

4.8 Example of obstacle avoidance when an obstacle is detected while executing a trajectory. The dotted line indicates the original trajectory calculated by a path planner and a given map of the environment (solid lines). The dashed line represents the actual followed trajectory. The robot controller reactively alters the robot's course to avoid the obstacle and afterwards steers back to the original trajectory. 90

4.9 Example of the obstacle avoidance solution using resource allocation. The checkerboard cells are allocated by the robot during its trajectory. These allocations are temporal and the time interval of each allocation should represent the time when the robot actually occupies this location. 91

4.10 Resource holons in obstacle avoidance task when dividing the environment into individually allocatable grid cells. 91

4.11 Resource holons for a door opening task. 93

4.12 Resource holons for a navigation task in which a robotic wheelchair allocates an external sensor placed in the environment. 95

5.1 Navigation solutions, to which the order holons committed, for a three robot setup in a common office environment. Due to a low exploring rate, the exploring D-MAS failed to explore the solution space completely and only poor solutions are returned (5.1(a) and 5.1(b)). For the navigation request shown in Figure 5.1(c), the exploring D-MAS only has one option to go from start to goal, so it succeeds in returning a satisfactory solution. . . . 104

5.2 Navigation solutions, to which the order holons committed, for a three robot setup in a common office environment. Here an adequate exploring rate, of 1 ant per second, is adopted and the order holons receive a sufficient amount of solutions to choose from. The solution shown in Figure 5.2(b) makes a detour since the order holon's selection policy prefers arrival time over path distance. 105

5.3 The influence of the refresh rate with respect to start time, finish time and duration. On the horizontal axis the ratio between exploring and refresh rate, denoted as E/R ratio, is plotted, while the vertical axis indicates the time in seconds. Although start time is directly proportional to the refresh rate, a performance degradation is shown for low E/R ratios, acknowledging the problem when the order holon has not yet received an adequate amount of solutions from the exploring D-MAS and commits to an inferior solution. 107

5.4 The influence of the refresh rate with respect to start time, finish time and duration, while employing a release time relative to the refresh rate. On the horizontal axis the ratio between exploring and refresh rate, denoted as E/R ratio, is plotted, while the vertical axis indicates the time in seconds. 108

5.5 Cycles in holons which contribute the most to the system's computational complexity. The product holon has no separate cycle, since it is not an *active* object. Rather, it is consulted by the order holon's exploring D-MAS. 112

5.6 Histogram of the number of hops exploring ants made in an environment of 16 rooms and 24 corridors in which the shortest path solution is 5 hops. 113

5.7 Scatterplot of number of hops exploring ants made in an environment of 16 rooms and 24 corridors in which the shortest path solution is 5 hops. 114

5.8 Average number of requests over time on all resources in an environment with 16 rooms, 24 corridors and 16 navigation requests. 114

5.9 Average number of slots per resource in an environment with 16 rooms, 24 corridors and 16 navigation requests. 115

5.10 In this environment with 16 rooms and 24 corridors, 16 executed navigation requests are shown, each executed by another robot. Robot starting positions are indicated by $S1$ to $S16$ and robot goal positions by $G1$ to $G16$. The trajectories plotted here are the solution of one of the simulation runs in this environment. 122

5.11 The number of conflicts in an environment with 16 rooms and 24 corridors with respect to the number of robots is shown here. Only conflicts between optimal paths are taken into account. 122

5.12 Total execution time with respect to number of robots executing in an environment with 16 rooms and 24 corridors. 123

5.13 Throughput, defined as the amount of navigation requests that are executed per second, with respect to number of robots is plotted here. 124

5.14 Throughput with respect to number of robots plotted against the ideal throughput curve, in which all robots would navigate without interference. 125

5.15 The effect of conflicts is denoted here. After increasing linearly in the first 4 sets, the throughput drops due to the introduced conflicts. Nevertheless, it still increases linearly, although parallel to the ideal curve. 125

5.16 Environment with 32 rooms, 52 corridors and 32 robots, each starting in a different room. Each robot navigate to a randomly chosen neighbouring room. Robot starting positions are indicated by $S1$ to $S32$ and robot goal positions by $G1$ to $G32$. The trajectories plotted here are the solution of one of the simulation runs in this environment. 126

5.17 Box plots of navigation durations in seconds in both environments for 8 runs in each environment. The box defines the interquartile range ($IQR = Q3 - Q1$), while the whiskers are positioned at $Q1 - 1.5IQR$ and $Q3 + 1.5IQR$ respectively. The average value is displayed by a $\star$ symbol. 127

5.18 Trajectories are adapted to cope with disturbances, such as a road block. Robot starting positions are indicated by $S1$ and $S2$ and robot goal positions by $G1$ and $G2$. 128

5.19 Effect of an unexpected road block on corridor reservations for one of the robots. In this case, the disturbance is taken care of within 2 refresh intervals. 129

6.1 Example scenario which can lead to livelocks. Robots are represented by rectangular shapes and their desired paths by a solid line. 136

6.2 Example scenario which can lead to suboptimal path execution. Robots are represented by rectangular shapes and their desired paths by a solid line. 138

6.3 Example scenario which leads to starvation in some priority based multi-robot path planning techniques. Robot $R1$ and $R2$ have a higher priority than robot $R3$ and each time block $R3$'s goal. Depending on which targets robot $R1$ and $R2$ receive, robot $R3$ will perpetually be unable to execute its task. A solution to this starvation is to swap priorities from time to time, allow robot $R3$ to plan and execute its path before $R1$ and $R2$ execute theirs. 139

6.4 Path planner combining a breadth first search algorithm with a cost function based on path length and obstacle clearance. The trajectory is constructed from goal to start position by repeatedly updating the costs in the grid cells representing the environment. In this figure, costs are indicated by a greyscale value. White being the lowest and black being the highest cost. After the lowest-cost path is found a list of way points is calculated. These way points are chosen in such a way that in-between way points no obstacles reside. The thick white line indicates the constructed trajectory the robot will execute. 140

6.5 The calculated path is divided into way points, indicated by crosses. These way points are chosen in such a way that in between way points no obstacles reside and a minimum clearance from obstacles is maintained. 141

6.6 The trajectory controller in the reactive approach determines the angle (α) between the robot's current position and the first way point (circles) in the path, calculated every time step by the path planner. The robot rotates in place (ω) until aligned with the way point and moves forward (v) once aligned. 142

6.7 Multi-model of narrow corridor resource holon. After initialisation, the resource waits for operations to execute. Two type of operations are supported, prepare operations and navigate operations. Prepare operations are used to indicate the resource has to get prepared before the operation requested by the order holon can be executed. In these experiments no prepare operations were used. 144

6.8 Executed trajectory of a robot without coordination. The lack of information and intention awareness causes the robot to move down first and then heading back to take the corridor to the right. The original path is indicated by a dashed line, the executed path by a solid line. 147

6.9 Both robots enter the same corridor and fail to reach their target. This results in a livelock, i.e. continuously blocking each other in the corridor without finding a solution. Robots are represented by rectangular shapes and their desired paths by a solid line. 147

6.10 Simulation runs of 2 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted of robot 1 and 2 of a failed ((a)-(b)) and a successful ((c)-(d)) run. The deviations in the successful run are caused by one robot avoiding the other. . 148

6.11 Simulation run of 4 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. In this run all robots successfully reach their target without much excessive travelled distance. . . . 149

6.12 Simulation run of 4 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. In this run robot 1 ((a)) and robot 3 ((c)) both drastically change their trajectory resulting in excessive travelled distance 150

6.13 Executed trajectories of two robots with coordination of a shared corridor. The robots first move to a transit zone at the side of the corridor's entrance and wait for an available slot on the corridor resource. Whenever a robot has successfully allocated the corridor, it can move through it. The original path, indicated by a dashed line, and the executed path, indicated by a solid line, almost completely overlap inside the corridor. 152

6.14 Executed trajectories of two robots with coordination of a shared corridor, using a larger transit zone. This way the deviation due to collision avoidance is almost completely reduced to zero. 152

6.15 Simulation run of 4 robots with coordination through PROSA and D-MAS. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. The use of transit zones is highlighted by the intermediate targets. The robot navigates from transit zone to transit zone until it reaches its target. Although its path is suboptimal in distance, the execution is guaranteed to be smooth since corridors are allocated on beforehand. 153

7.1 The robotic wheelchair SARA has mid wheel drive and is equipped with two Hokuyo URG-04LX laser scanners and has Baumer magnetic encoders on both wheels. The joystick module and encoders are read by a National Instruments NI PCI-6225 data acquisition (DAQ) card. The wheelchair control unit, running Ubuntu 10.04, is a mini-ITX board with an Intel Atom 1.6GHz CPU, 1GB memory, on board Intel GPU and a 160GB hard drive. Interaction with the platform goes through the 100Mbps Ethernet card or the USB WiFI receiver. 156

7.2 The robotic wheelchair INGA has front wheel drive and is also equipped with two Hokuyo URG-04LX laser scanners and Baumer magnetic encoders on both wheels. In addition to its standard joystick connected by serial port, it has a haptic joystick. A National Instruments NI PCIe-6351 is used to read in the encoders. The wheelchair control unit is a ZOTAC Z68-ITX WiFi mini-ITX board with an Intel i5 CPU, 4GB memory, a NVIDIA GeForce GT 430 GPU with 1GB memory and a 160GB OCZ Vertex3 solid state drive. Interaction with the platform goes through the 1Gbps Ethernet card or the on board WiFI card. 158

7.3 Control scheme for autonomous navigation of the robotic wheelchairs. 159

7.4 The set of 262 positions in the calculated path is narrowed down by selecting 11 way points in this example. 160

7.5 Splines connecting each way point to create a smooth interpolation. 162

7.6 Augmented environment, in which the existing first floor robot lab and mechatronics lab are combined with virtual rooms and corridors. The HMES can use this map for hybrid executions, since it sends identical navigation commands to real and virtual robots. 165

7.7 Experimental setup in which two robotic wheelchairs are coordinated to navigate between robotics and mechatronics lab on the first floor at PMA/KU Leuven. 166

7.8 Trajectory followed by robot Inga during tuning process of state-space controller. The desired trajectory, generated using the spline interpolation is depicted by the dashed line, while the actual trajectory followed by the robot, estimated by the state estimator, is depicted by the solid line. 167

7.9 Evolution of angular, parallel en tangent control errors in absolute values during the execution of the trajectory depicted in Figure 7.8. 168

7.10 The environment used during the coordination experiment in which two robotic wheelchairs changed rooms. Access to the mutually exclusive region *NarrowCorridor0* is coordinated by PROSA and D-MAS. 169

7.11 Allocations on resources during the coordination experiment. Robot $R1$ represents robotic wheelchair Inga going from $Room0$ to $Room1$, while robot $R2$ represents robotic wheelchair Sara, going from $Room1$ to $Room0$. Both robots first move to their respective transit zones in their starting rooms. In this experiment, $R2$ is granted access to the mutually exclusive resource $NarrowCorridor0$ first. Immediately after $R2$ exits this resource, $R1$ is able to enter. 170

7.12 3D plot of the trajectories followed by both robotic wheelchairs. The solid lines are the executed trajectories with respect to the time and the dashed lines indicate the corresponding trajectory projected onto the environment. The time dimension shows the robot starting at the left (Inga) waits until the robot at the right (Sara) passes through the area where their trajectories overlap. The fact that the solid trajectories do not intersect indicates no collisions occurred. 171

7.13 Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R2$ has reached its transit zone, while $R1$ is still moving toward its transit zone. 172

7.14 Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R1$ has reached its transit zone, while $R2$ is already toward the narrow corridor. 172

7.15 Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R2$ has reached its goal position, while $R1$ is still waiting in the transit zone. . 173

7.16 Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here both robots have reached their goal position. 173

A.1 MLRSimulator showing an environment with multiple robots. Each robot has its own laser scanner for collision detection. 186

A.2 The intersection visitor calculates collisions between laser scanner rays and the environment. The white spheres indicate detected collisions. 187

A.3 Top view of MLRSimulator. 188

A.4 Third person view of MLRSimulator. 189

A.5 First person view of MLRSimulator. 189

B.1 The main window of the Robot GUI. Different sections are marked from ① to ⑨. 192

B.2 The `Map Generator` window allows the user to specify settings to generate a custom map. 193

B.3 The `Settings` window allows the user to specify connection settings such as the robot's IP address and port number. . . 194

B.4 The `Parameters` window shows the parameters the robot is currently using. 195

B.5 The `Connection Log` window shows messages sent to and received from the robot. 196

List of Tables

3.1 Overview of robot software frameworks, C1 to C5 are the 5 aspects of separation of concerns as described in section 2.3.3. 54

5.1 Recommended values for the HMES parameters for our multi-robot navigation scenario. Rates are inverted and expressed in time intervals, for readability. 110

5.2 Parameters used in complexity analysis. 111

6.1 Benchmarks of runs without coordination. 148

6.2 Benchmarks of runs with coordination. 154

Chapter 1

Introduction

> *When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem. As a result, programming has become one of our most demanding intellectual activities, requiring great clarity of expression and the utmost economy of reasoning. This conclusion has never been refuted; it is, however, regularly denied because of its uncomfortable implications.*
>
> Edsger W. Dijkstra (⋆1930–†2002)
> Visuals for BP's Venture Research Conference, 1986

1.1 Acknowledgements

This work was funded by the Research Council of the KU Leuven, namely within the concerted research action GOA-ACDPS (Autonomic Computing in Decentralised Production Systems) between the Department of Mechanical Engineering, Division PMA and Department of Computer Science, Division Distrinet. Additionally, it was funded by European project BRICS (FP7-231940, Best Practice in Robotics) in which PMA/KU Leuven takes part.

1.2 Motivation

The evolution of robot software

Software development methods in robotics has been rapidly evolving the past decade to catch up with novelties in computer science research. Researchers have become more aware of the necessity of properly designing their robot software application and hereby stimulating non-functional qualities such as reusability and extendibility. The adoption of component-based software engineering and even model-driven development resulted in widespread robot software usable by researchers outside the lab where it was originally conceived.

This growing tendency in robotics to allocate more development time to software design not only has a positive effect on the resulting application in a research perspective but also increased the popularity of the robotics domain as a whole. Properly designed and well documented reusable software components facilitate application development and also lower the learning curve for novice researchers.

Another factor which contributed to the burgeoning robotics research is the development of open source robot software. Sharing robot software with the research community through all the popular media has become the new standard in robotics and by doing so the software is able to improve more rapidly thanks to community testing and debugging, in which a large part of the end users contribute to running tests and solving bugs.

The lack of reflection of reality

Although component-based software engineering has clearly made its way into robotics and several widely used robot software frameworks [B. Gerkey et al. 2003; Bruyninckx et al. 2003; Brooks et al. 2005; Ando et al. 2005] and robot software architectures [Alami et al. 1998; Nesnas et al. 2006] adopt this design style, a clear definition of the responsibilities of the several components is frequently omitted.

Moreover, most robot software architectures adopt a functional decomposition and physical entities required in the application, e.g. robot resources or objects in the environment, have not received their rightful place in the design. All allocation of those entities is implicit and commonly each component interacting with a particular entity has its own representation and maintains its own version of that entity.

Such a *robocentric* approach lacks a proper reflection of reality and hinders the creation of robust robot software with respect to external disturbances and changes in the world-of-interest. This problem becomes more apparent in multi-robot applications in constraint or dynamic environments or single-robot applications in which the robot shares its workspace with humans. Examples are navigation of mobile robots in rooms and corridors or robot manipulators cooperating with humans or executing their operations in a dynamic environment. Actions taken by one robot can have consequences for other present robots or humans. For instance, a mobile robot driving into a narrow corridor might block other *users* of that corridor even if they had entered first.

Not only activities or tasks in a robotic application but also environment entities, such as those described above, should be considered as valuable resources, which require *explicit allocation*, and be treated as first-class citizens in the overall software system. This implies that all resource entities in the environment that two or more robots will have to share, during some period of their lifetime, should get their own software components.

These components should be separate *agents*, hence not reside within the control system of the robots. This is *the* way to assure that an appropriate reflection of the reality is maintained at one single instance.

Moreover, knowledge of entities used in an application should reside at the respective components representing those entities, rather that spread across multiple components. Adopting such a *single source of truth* increases system robustness against disturbances or changes in the environment, since only relevant components should be updated.

Increasingly complex applications require complex systems

Besides the lack of reflection of reality, many existing robot software architectures or frameworks inadvertently limit, by design, the set of applications feasible by systems adopting that particular architecture or framework.

Nevertheless, the increased level of attention robotics has been receiving lately resulted in the desire of increasingly complex applications. E.g., applications in which humans and robots interact, or multiple robots are coordinated in a dynamic environment.

In order to categorise different robotic systems, a taxonomy with respect to

system complexity, based on the S-levels described in [Bruyninckx 2010], is used. Five levels of system complexity, ranging from centralised one-robot systems (level $S1$) to decentralised coordination with competing tasks (level $S5$) are identified and these different levels will be discussed in detail in chapter 2. A transition from one level to a higher, more complex, level always requires a *paradigm shift* or quantum leap concerning system design.

This categorisation of systems is used to motivate the need of reflection of reality since such systems enable applications at the highest complexity level while common robotics approaches currently reside in level three.

Increasingly dynamic environments require adaptive systems

Another focus point in robotics nowadays is the development of applications and robots for more dynamic and realistic environments. Such environments are not only characterised by uncertainty, but also by *unexpected* behaviour or disturbances of either other actors or the environment itself. In order to cope with the unexpected, a robot system should be able to continuously adapt itself.

Therefore, complex adaptive systems are required to cope with such demanding applications and environments.

Those systems are **complex** due to their diversity and the fact they consist of multiple interconnected elements and they are **adaptive** in the way that they are able to cope with changes and disturbances and are able to learn from past experiences. Examples of complex adaptive systems in other domains include the stock market and social insect colonies.

A paradigm shift towards holonic systems

In manufacturing control, the ideas of explicit resource allocation and intention propagation proved to increase scalability and flexibility of the overall system. Moreover, making the environment and activities in this environment first-class citizens in the underlying software system results in a higher decoupling.

This **paradigm shift** toward explicit resource awareness and allocation has led to insights into handling scalability and flexibility issues most other approaches suffer from. Explicitly taking resources into account and treating them as separate entities, facilitates not only resource reusability but also shared resource

control. By sharing resources and task goals or intentions with each other, robots can adjust their behaviour and adapt their task execution to deliver an optimised global task.

These resource allocation, scalability and flexibility issues are more clearly present when several robots with individual capabilities operate in the same environment, implicitly sharing this latter with each other. Besides their own goal, the intention of other robots should also be taken into account, i.e. interference between robots should be minimised.

The Product-Resource-Order-Staff Architecture (PROSA) [Van Brussel et al. 1998] and short-term forecasting mechanism delegate MAS (D-MAS) [Holvoet et al. 2009] were developed at the KU Leuven to provide a solution to these issues in manufacturing control. The implementation of this multi-agent architecture for manufacturing systems, the Holonic Manufacturing Execution System (HMES), has been successfully applied in numerous manufacturing control tasks in the past at several industrial companies [Zamfirescu et al. 2003; Saint Germain et al. 2003; Reitbauer et al. 2005; Kollingbaum et al. 2000; Valckenaers et al. 1997]. PROSA adopts Koestler's idea of holonic systems [Koestler 1968]. Such systems are characterised by their structural decomposition. Each component or *holon* is self-reliant and both a whole and part of a greater whole. Other *holonic systems* (e.g. [Leitão et al. 2005; Skobelev 2011]) have also shown that they are able to cope with external disturbances and changes and provide scalable and flexible solutions.

1.3 Objectives

Rather than trying to solve one particular problem, this dissertation explores the benefits holonic concepts can provide in robotics in general, and multi-robot navigation in specific. The main objective is to **introduce PROSA and D-MAS in robotics**, translating the concepts of this holonic software architecture from manufacturing control to robotics. The applications provided in this dissertation, therefore, function as prototypical examples of this translation.

PROSA consists of three interacting holon types, reflecting an existing concept, object or activity in a real manufacturing system, respectively represented by product, resource and order holon. Optionally, a fourth holon type, the staff holon, can be used to assist the other three types. In PROSA all holons operate at the same level. In manufacturing control such heterarchical systems have shown to be beneficial for both system scalability and system flexibility [Monostori et al. 2006]. Although PROSA was originally developed for manufacturing control, it is a generic software architecture and is applicable to robotics as well.

Moreover, PROSA can be used in robotics at a higher abstraction level coordinating underlying robot components, provided that PROSA's separations of concerns is adopted. This way, state-of-the-art robot software is still usable and can benefit from PROSA's advantages.

This thesis investigates the benefits of using of PROSA in a multi-robot application in which multiple robots have to individually navigate in an environment. The deployment of a fleet of autonomous mobile robots in a dynamic environment, such as a warehouse or hospital, was used as a reference scenario. Although each robot has its own goals, e.g. transporting goods from one point to another, the path planning has to be coordinated at some points to avoid collisions or deadlocks.

In this context, the abstraction level with respect to the world representation at which PROSA and D-MAS are used is a topological environment in which costs are added to the edges. This cost is taken into account to select a proper navigation solution. Robots are treated as navigation tasks in PROSA and D-MAS and abstraction is made from their specific dimensions. Execution of the navigation tasks in done using on-board software on each of the robots.

The objectives can be summarised as follows:

Applicability: investigate how well the holonic concepts fit in the field of robotics. Provide an evaluation by implementing an application scenario.

Robustness: provide guidelines and considerations in the development of robust robot software, which is able to cope with external disturbances and changes in the environment.

Integration: decompose this software in such a way that integration with other existing or new software is feasible and straightforward.

Scalability: provide a solution which scales well with respect to an increase in either the number of robots or either the size of the environment in which these robots operate. This increase should result in a linear or polynomial increase of the system's overall computation time or complexity with respect to the multi-mobile-robot navigation task.

Experimentation: validate above objectives with a clear focus on multi-robot navigation and, more specifically, the coordination of multiple robotic wheelchairs, in which user comfort is key.

This dissertation will provide application scenarios and solutions with respect to above-described concepts, but will not approach this in the context of *system engineering*. Rather, it will focus on the integration and applicability of PROSA

into existing robot software and evaluate robustness and scalability through experimentation.

1.4 Problem domain

During this dissertation the following **multi-robot navigation scenario** was defined: in a hospital or retirement home many patients or inhabitants require a wheelchair to navigate from one location to the other and in most cases they also require assistance during navigation or in mounting and dismounting the chair.

The **goal** of the application is to provide a small set of autonomous mobile robots, i.e. robotic wheelchairs, which are able to navigate autonomously to the user. After the user is assisted into the wheelchair a target location is given to which the robot has to navigate.

The benefit of such an approach is that caretakers are only required to help the user mount or dismount the wheelchair. While navigating, the robot autonomously finds it way and is able to avoid obstacles using its range sensors. If desired, the user is also able to take control of the wheelchair and drive it manually or in a *shared control* mode.

1.4.1 Issues

One of the key criteria for a successful application is to minimise user discomfort. Not only should the autonomous navigation be smooth, but also interference between two or more robotic wheelchairs should be avoided. For instance, if two or more robotic wheelchairs enter a narrow corridor from either side, at least one robot will be forced to retreat.

Moreover, if the two robots keep trying to move forward but at the same time block each other, livelock occurs.

In a **livelock** two or more entities constantly change their behaviour with regard to one another, none of them progressing in their task. This is caused by the robocentric approach most current robot applications follow. Each robot executes its own task, assuming the environment is implicitly allocated for its needs and not being aware other users might be present.

Similar to a livelock, a **deadlock** is caused when two or more entities wait for each other and do not make progression in their task. The difference with respect to livelock is the way these entities *wait*. In a deadlock situation, the

progress of one entity depends on the completion of the (partial) task of another entity and vice versa. Therefore they do not change their behaviour but are waiting for each other to finish.

In some multi-robot navigation techniques, it might occur that one or more robots are unable to execute their task because other, usually higher-priority robots, continuously occupy one or more of their resources. This problem is defined in computer science as **starvation**. A livelock can be seen as a special case of starvation, in which all involved entities are in starvation since none of them is able to continue their task.

Assumptions with respect to the world model are made. Only a partial geometric representation on top of a topological model is used. However, such a world model is sufficient to validate the applicability of PROSA and D-MAS in robotics. In future work, other world models could be used to provide a higher performance.

1.5 Contributions

The main contribution of this thesis is the *translation* of PROSA and D-MAS concepts into multi-robot navigation or, more generally, the introduction of holonic systems to the domain of robotics as described above in section 1.3.

Besides introducing the application scenario used in the thesis, the following paragraphs also list the main contributions to the domain of robotics in general and the research group at PMA/KU Leuven in specific.

1.5.1 Applicability: HMES analysis

The existing PROSA and D-MAS implementation, called HMES, has been analysed for robotics on various aspects. The influence of configurable parameters on system performance is investigated, a complexity analysis has been conducted and scalability and robustness with respect to disturbances was analysed.

Moreover, the possibilities and limitations of the current HMES implementation are discussed with respect to robot applications.

The general applicability of PROSA and D-MAS is discussed in chapter 2, while chapter 5 provides the HMES analysis.

1.5.2 Robustness through bio-inspired delegate MAS

By jointly using D-MAS and PROSA, the coordination mechanism in the robot system evolves from reaction to prediction of disturbances and changes in the environment. Indeed, coupling resource allocation with D-MAS's short-term forecasting power, allows the system to anticipate and avoid bottlenecks or other navigational issues rather than having to deal with them.

This type of control is often referred to as **preview control** which, in contrast to feed forward or feed back control, deals with near-future events which are estimated or forecast. In previous research, preview control has mostly been applied in local, fine-grained robot control or shared control with a user. With PROSA and D-MAS this technique is also adopted to increase the robustness in the global planning.

Chapters 5 and 6 discuss robustness experiments.

1.5.3 Integration of legacy hardware and software and existing state of the art through incremental compliance

Too often robot software is written for one hardware platform and whenever new hardware is used a development step is required before it is usable. The other way around is also common, robot software written for new hardware is usually not compatible with old, legacy, hardware.

By using PROSA in robotics at a higher coordination level, both legacy hardware and existing state of the art can be integrated with newer hardware and software to the degree they are compliant. Three groups can be identified: not compliant, to some extent compliant and fully compliant.

In this thesis, several existing robot wheelchairs were coordinated and used in a joint application showing the integration power offered by PROSA which, in fact, is a result of the structural decomposition PROSA adopts.

Chapter 2 discusses in detail the power of structural decomposed software and chapter 4 and appendix C provide practical implementation details on how heterogeneous software is integrated with each other.

1.5.4 Scalability through structural decomposition

The most apparent influence or contribution is the paradigm shift from functional to structural decomposition of robot software. Introducing **holons** in robotics

is, to our knowledge, a novel approach and adopting a single source of truth has provided a **scalable** solution in multi-robot navigation.

Both scalability with respect to number of coordinated robots and the size of the environment, i.e. the rooms and corridors these robots navigate through, have been investigated and in both cases holonic systems provide a scalable solution.

Experiments investigating the scalability are presented in chapter 5.

1.5.5 Experimentation: multi-robot coordination

To illustrate these contributions and validate the research, two multi-robot navigation experiments have been conducted. A first experiment shows in simulation how bottlenecks and unnecessary travel can be avoided by employing PROSA and its structural decomposition and D-MAS to provide short-term forecasts. The experiment also illustrates that an increase of the number of robots and rooms has no effect on the overall performance. The second experiment shows how D-MAS allows two robotic wheelchairs to propagate their intention to each other and how explicit resource allocation combined with this intention awareness improves the overall multi-robot navigation task.

The results of these experiments are discussed in chapters 6 and 7.

1.5.6 Graphical tools

During this thesis, several graphical user interfaces and tools were developed to enhance feedback of system data to a user or debugger and to speed up experiment configuration. Besides these user interfaces, the existing MLRSimulator [Vanacker 2008] has been redesigned to allow the simulation of multiple robots and increase the feedback of information.

The most elaborated user interface, named Robot GUI, allows remote control of the robotic wheelchair, visualises sensor and algorithm data, and contains a map editor.

Each of these user interfaces has been completely separated from the system code and communicates over network sockets. By doing so, the Robot GUI can run on any system as long as it is connected to the same network as the robot.

1.5.7 Communication protocol

In order to communicate between all the different actors in the multi-robot coordination application, a common communication protocol was designed. The protocol makes use of *protocol buffers* to optimise the encoding of structured data sent *over the wire*.

Both the communication between the robots and PROSA but also the data sent from and to the graphical user interfaces and multi-robot simulator follow this protocol. This way there is no distinction between data coming from a real robot or a virtual robot in the simulator and both can be used in the multi-robot application, even simultaneously.

1.6 General overview

The next chapters are organised as follows:

Chapter 2: Holonic Systems and Why They are Important for Robotics

Since holonic systems have, to our knowledge, not yet been explicitly used in robotic systems, this chapter provides an in-depth overview of the holonic concepts and the existing PROSA reference architecture and its coordination mechanism D-MAS. It provides a the **conceptual basis** for the main objective in this thesis, namely **apply PROSA and D-MAS in robotics**.

It starts by describing the five levels of system complexity, based on the S-levels described in [Bruyninckx 2010], and provides terminology common in software engineering to clarify subtle differences between the variety of software concepts. These definitions are also helpful during chapter 3 in which, among other domains, related work and state of the art in robot software is discussed.

Secondly, it motivates the importance of holons and holonic systems in robotics and also discusses their characteristics and differences to common robot systems. Next, the different PROSA components and D-MAS mechanism are defined and illustrated by some examples. Finally, an overview of previous applications of the PROSA architecture, mainly in manufacturing control, is given.

Also, to introduce this paradigm as efficiently as possible, this chapter does not limit itself to previous work but also includes, to some extent, the translation effort of the different concepts to robotics, one of the main contributions of this research.

Chapter 3: Related Work and State of the Art in Holonic Systems

In this chapter, a comparison with the state of the art is made. We will consecutively deal with (i) relevant **software frameworks and architectures** that underpin the structure of the developed system architecture, (ii) relevant **system architectures**, in particular the multi-agent or holonic approach, (iii) relevant **control architectures**, in particular the D-MAS approach, including path planning and multi-robot coordination approaches.

Chapter 4: Holonic Robot Applications

In this chapter, the PROSA and D-MAS implementations, called HMES, which was used to perform the experiments, are discussed in more detail. How these implementations were integrated with existing robot software and hardware, is also described. Indeed, as will be emphasised throughout the text, PROSA and D-MAS are not meant to replace existing expertise in robotics research. They are meant to be used as a methodology to use existing expertise in a more flexible and adaptable way.

Besides the discussion on the concrete implementation, also some conceptual robot application examples are elaborated to highlight the applicability of the approach in robotics.

Chapter 5: HMES Analysis

The set of configurable parameters of the HMES and their influence on the system performance is analysed through a series of simulation experiments. Moreover, a pragmatic approach of estimating the computational complexity with respect to those parameters is performed. Next to the complexity analysis, scalability and robustness with respect to disturbances are analysed. Finally, this chapter discusses current limitations of the HMES implementation and provides how the HMES could be changed to address these limitations.

Chapter 6: Multi-Robot Coordination in Simulation

Simulation experiments with multiple robots, which navigate in a constrained environment, are discussed in this chapter. The experiments were conducted in MLRSimulator, a 3D simulator using OpenSceneGraph which is described in appendix A, and evaluate the performance of PROSA and D-MAS compared to a reactive approach.

Chapter 7: Coordination of Multiple Robotic Wheelchairs

This chapter describes the multi-robot navigation experiment performed with the robot wheelchairs, available in our lab. The experiment evaluates the ability to integrate multiple robotic platforms with PROSA and D-MAS and how coordination and resource allocation of rooms and corridors provides flexible solutions to bottleneck or deadlock situations.

Chapter 8: Conclusion and Future Work

In the conclusion chapter, the postulated objectives in this chapter are evaluated and future work is discussed.

Appendix A: Multi-Robot Simulator

This appendix provides background information on the 3D simulator which was developed at PMA/KU Leuven for robotic wheelchair research. In the context of this thesis, the simulator was changed to accommodate multiple simulated robots which are coordinated by the HMES.

Appendix B: Robot Graphical User Interface

This appendix describes the robot graphical user interface developed to visualise sensor data and algorithm output during execution. Also a map editor was added to define resource holons and their transit zones.

Appendix C: Communication Protocol

Finally, this appendix discusses the communication protocol which all software modules have in common. The HMES interface to the robots, the robot software running on the wheelchairs, the robot graphical user interface and the 3D simulator all use this protocol to communicate with each other. By separating this functionality, switching from simulation to the robotic wheelchair or vice versa is transparent to the HMES.

Chapter 2

Holonic Systems and Why They are Important for Robotics

> *Human beings, viewed as behaving systems, are quite simple. The apparent complexity of our behaviour over time is largely a reflection of the complexity of the environment in which we find ourselves.*

Herbert A. Simon (⋆1916–†2001)
The Sciences of the Artificial – 3rd edition, The MIT Press, 1996

2.1 Introduction

The title of this chapter should not be considered as a form of arrogance towards the reader or the robotics community in general. Rather it stresses the common problem of overspecialisation which causes researchers to focus solely on their own research domain without considering non-mainstream approaches from other fields.

Herbert Simon's insight in the fundamental nature of **complex adaptive systems**, resulted in the definition of holonic systems. This dissertation aims to prove such systems, which are already successful for many years in manufacturing

([Van Brussel et al. 1998], [McFarlane et al. 2000], [Babiceanu et al. 2006], [Monostori et al. 1998]), are equally important for robotics, in which holonics has not been applied before. This chapter functions as a starting point for motivating this claim.

It does so by, firstly, providing a taxonomy of system and coordination complexity, ordered by increasing complexity and applicability. The levels of complexity, which will be described, are used in subsequent chapters to properly evaluate related work and state of the art and provide a clear method to compare approaches and categorise applications.

Secondly, this chapter includes terminology common in software engineering to introduce best practices and to help clarify subtle differences between the variety of software concepts used in this text. These definitions are also helpful during chapter 3 in which, among other domains, related work and state of the art in robot software is discussed.

Next, it introduces the underlying concepts of the reference architecture PROSA and its coordination mechanism D-MAS, in order to sketch previous work in holonic systems, upon which the research presented in this manuscript is built. These concepts employ a system design paradigm, completely different from mainstream robotics applications, but are, nonetheless, able to offer solutions to problems in many of those applications.

Proper justification of applying PROSA and D-MAS in multi-robot applications will be given in subsequent chapters. In this chapter, only the history behind this architecture is provided: Firstly, holons and holonic systems are introduced, which are the core of this paradigm, then the main components of PROSA and D-MAS are explained, and some previous work using this architecture is highlighted.

While this chapter will focus only on holons and holonic systems in general and the PROSA architecture and D-MAS coordination mechanism in particular, related work and state of the art in holonic systems will be addressed in chapter 3.

Although the concepts described here are eventually used in a software implementation, this chapter makes abstraction from specific implementation details while discussing different elements. Proper instantiation and concrete implementation of PROSA and D-MAS in the context of robotics applications will be discussed in chapter 4.

Nevertheless, to illustrate the approach, some examples, focussing mainly on robot navigation, are presented. Robot navigation will be the topic of the remainder of this text. Also, to introduce this paradigm as efficiently as

possible, this chapter does not limit itself to previous work but also includes, to some extent, the translation effort of the different concepts to robotics. In depth discussions and usage of the paradigm will follow in the subsequent chapters.

Finally, throughout this chapter, design guidelines in adopting this paradigm are highlighted and provide a quick overview or summary of the content of the different paragraphs.

2.2 System complexity levels

Bruyninckx defines four distinct levels of system complexity, $S1$ to $S4$, ordered by increasing complexity. Commonly, a number of assumptions with respect to complexity are made during system design. This way the resulting system design only works well in systems with a matching complexity level [Bruyninckx 2010].

The system complexity levels described in the following paragraphs follow the definitions found in [Bruyninckx 2010]. An additional level, $S5$, is defined as a distinction between non-competing tasks (level $S4$) and competing tasks (level $S5$). Figure 2.1 provides a schematic overview of these five levels.

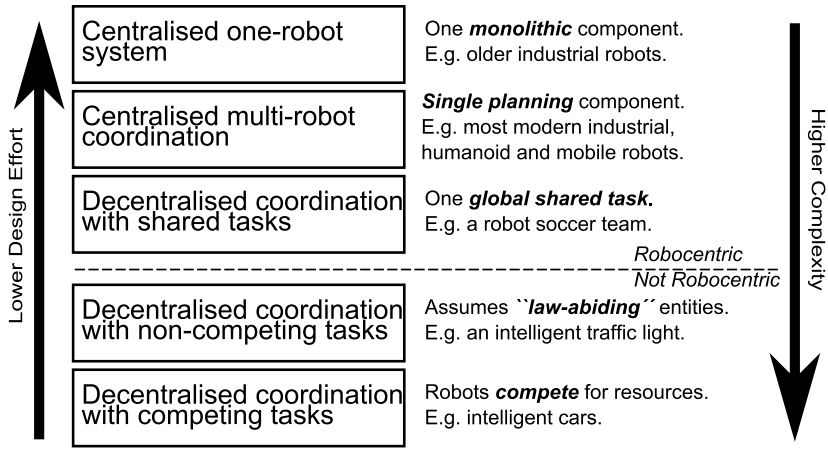


Figure 2.1: Schematic representation of a the system complexity levels as defined by [Bruyninckx 2010].

Systems categorised at the same complexity level provide functionality for a certain subset of thinkable or desirable applications. Higher-level systems support a larger subset of applications. The statement that holonic systems

are important to robotics is motivated by the claim that such systems enable applications at the highest complexity level while common robotics approaches currently reside at level three, although use cases in robotics exist for all five levels.

The following paragraphs describe the five levels and provide example applications at each level. Each step to a next level equals a paradigm shift with respect to design.

2.2.1 *S1*: Centralised one-robot system

Level $S1$ categorises systems in which all coordination is centralised and there is one monolithic component capable of controlling only one robot system and some peripheral equipment. The controller is specifically designed for the specific hardware it controls. Examples are older industrial robots and their control system, which is *locked-in* to specific hardware and functionality and is, by design, difficult to integrate with other systems.

2.2.2 *S2*: Centralised multi-robot coordination

At the second complexity level, $S2$-systems offer multi-robot coordination, but in a centralised way. In these systems all underlying hardware, such as robots and sensors, are coordinated by a single planning component under complete knowledge of the environment.

The most common problem in such systems is scalability, since such a planning component becomes a communication bottleneck and the complexity of the search space usually increases exponentially with respect to the number of robots [Bennewitz et al. 2002].

Another problem is the extendibility. Whenever new robots are added, the coordination component has to be adapted to include routines which are capable of coordinating this new robot type.

2.2.3 *S3*: Decentralised coordination with shared tasks

At this level systems offer a decentralised coordination solution for applications in which there is only one global task shared among several robots.

An example of an $S3$-system is a robot soccer team (see e.g. RoboCup Soccer 2011 [Ruiz-del-Solar et al. 2011]), in which all robots share the same goal, i.e. defeating the opposing team.

The main limitation in these systems is the assumption that all robots share a global task. Applications in which different robots perform different tasks, independently from each other, is not feasible.

Currently, most robot systems are developed with this level of complexity in mind.

2.2.4 *S4*: Decentralised coordination with non-competing tasks

The final two complexity levels break with previous *robocentric* approaches and systems of this complexity allow the coordination of different robots with individual tasks, not having to know about each other.

An *intelligent* traffic light, which governs, for example, the throughput of a junction is a system by which independent entities are coordinated.

The only limitation $S4$ has is that it assumes non-competing tasks. This means all coordinated entities are expected to follow instructions received from local coordination entities, such as a traffic light and do not try to *cheat* the system.

The application scenario described in section 1.4 will be solved using an S4-level system in this dissertation.

Most (holonic) manufacturing applications reside in this level of complexity. Commonly such applications coordinate hundreds of orders of products on a large set of resources, such as machines and tools. These different *agents* in the system run independently from each other.

2.2.5 *S5*: Decentralised coordination with competing tasks

Level $S5$, finally, represents systems in which the independent robots compete for resources and are not obliged to follow coordination instructions. The system needs to be able to cope with much more uncertainty and disturbances than in all previous complexity levels.

E.g., *intelligent* cars fall into this category. Their tasks compete with each other since they want to claim the road they drive on, and have to take into account

that other users of that road will not always follow the prescribed rules, either deliberately or otherwise.

2.3 Terminology in software engineering

In the following paragraphs, key software engineering concepts and best practices are described, with a special focus on component-based software engineering.

2.3.1 Component-based software engineering versus object-oriented programming

The main difference between component-based software engineering and object-oriented programming is the granularity of their respective software primitives. In other words, the approaches are complementary but describe a different level of abstraction.

In **object-oriented programming**, the software primitives are **classes**, which contain data field members to maintain a state and methods to enable behaviour of their instantiations, namely **objects**.

In **component-based software engineering**, the emphasis is placed on designing separate, stand-alone **components**, which encapsulate a set of services or functionalities. These components can be connected to each other to form a desired application. Typically, the dependence between different components is minimal such that each component is reusable on its own.

In [Bruyninckx 2010], also four granularity levels, with respect to software primitives, are defined. In ascending order of abstraction these are data structures ($G1$), classes ($G2$), packages ($G3$) and modules or components ($G4$). A schematic overview of these levels is shown in Figure 2.2.

Object-oriented programming and component-based software engineering are, therefore, not competitors but rather complementary to one another. For instance, components can be implemented in an object-oriented way, where the objects have an equal or finer granularity. It depends largely on the coupling between classes and at what level the components are defined. Since components should be reusable on their own, the coupling or dependence to other components should be minimal. However, components are not obliged to contain object-oriented code.

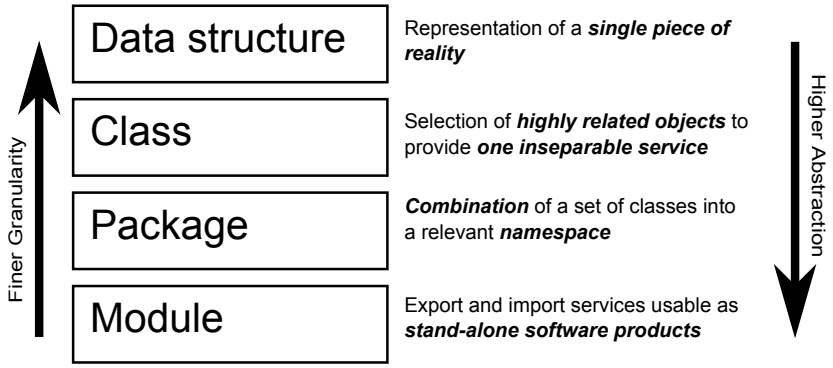


Figure 2.2: Schematic representation of the granularity levels of software entities as defined by [Bruyninckx 2010].

Note that in the remainder of this text mostly the term *component* will be used to indicate a software entity. Nevertheless, this does not limit the concepts described below to only component-based engineering.

2.3.2 Framework versus architecture

Often software framework and software architecture are incorrectly used as synonyms. Although at first, their definition and use might seem similar, in fact they are complementary and both equally important in developing a robot software application. The following paragraphs highlight the differences between software framework and software architecture.

A **software framework** defines a common communication and interaction standard for software components. Optionally, it provides a predefined set of reusable components without imposing structure and interaction between components. Thus, it does not prescribe a particular architecture. Software framework users are, or should be, free to arrange components in any desired structure as long as each component is developed by following the imposed communication interface.

Unlike a software framework, **software architectures** also impose structure between software components and specify their interactions. If the architecture and the responsibilities of its components are clearly defined but implementation of each component is left open to the user, the term **reference architecture** is often used instead of software architecture. A reference architecture provides a generic solution for a certain application domain.

A typical **software application** consists of an instantiation of a software or reference architecture following a certain software framework.

In Figure 2.3, a schematic representation of these definitions is depicted.

For example, a robot control application could implement a sense-plan-act architecture and follow a component-based software framework to separate each of its components. The software architecture defines the functionality each component offers and how the components interact with each other, while the software framework provides the component structure and communication and interaction modalities.

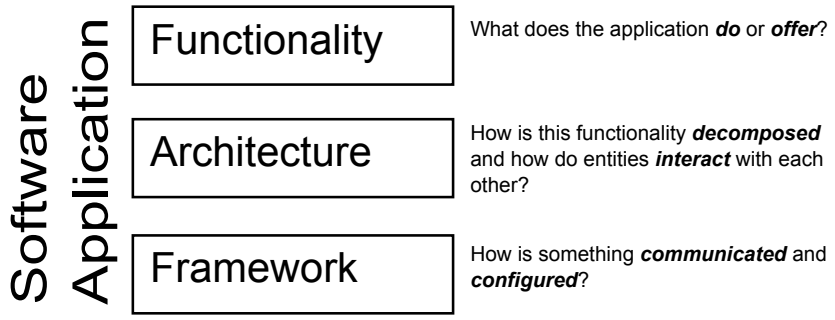


Figure 2.3: Schematic representation of a typical software application implementing a particular software architecture and following a particular software framework.

Thus, a software framework and software architecture are not synonyms nor are they competitors at the same abstraction level. The former defines interfaces while the latter defines structure and interaction. Both are equally required in (robot) software development.

2.3.3 Separation of concerns

Separation of concerns focuses on separating distinct features in distinct software entities without overlapping functionality, thereby decoupling these software entities and thus increasing reusability.

In component-based software these functionalities are separated into independent components, thereby increasing the reusability of each of them and increasing the system flexibility.

Besides defining a set of system complexity and granularity levels, [Bruyninckx 2010] also defines five aspects in separation of concerns which are

> *indispensable* for the analysis and synthesis of complex software systems.

In Figure 2.4 an overview of these *5 Cs* is depicted. From top to bottom: coordination ($C1$), configuration ($C2$), computation ($C3$), communication ($C4$) and perpendicular to the others is composition ($C5$).

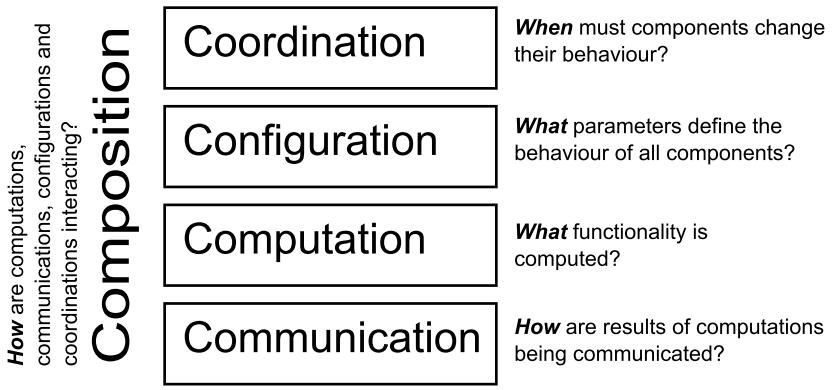


Figure 2.4: Overview of the five aspects of separation of concerns as defined by [Bruyninckx 2010].

For example, a robot controller could be replaced without having to change other components, such as a collision avoidance component. Also, each of these components could be reused separately in another application or another environment without the need to update their environmental representation.

2.3.4 Encapsulation

Encapsulation strengthens the interoperability and reusability, and thus system flexibility, by hiding the internal component structure behind a public interface. With encapsulation, the communication interface remains untouched.

Changing these internal algorithms or representations does not imply changes in other components as long as the interface remains the same, i.e. inputs and outputs of each component are not altered.

For example, consider a scenario in which robotic wheelchairs and a mobile manipulator are coordinated. The internal navigational algorithms of the robotic wheelchairs such as collision avoidance and door opening algorithms of the mobile manipulator are hidden, i.e. encapsulated, behind a public

interface each component offers. The wheelchairs share their intention with a mobile manipulator robot, which opens the door for the wheelchair user. How the manipulator actually opens the door as well as how this door is represented in the environment is not relevant for the robotic wheelchair user. These types of information are encapsulated in the respective software components. Implementations or internal representations may change, as long as the communication interface between components remains the same. Thus, encapsulation strengthens system flexibility.

2.3.5 Decomposition

Decomposition of a software system into manageable sub-parts is a general principle of software design to cope with large and complex systems. Two main approaches are functional and structural decomposition.

Functional decomposition is common in centralised systems in which each component represents a certain functional requirement. Examples of such components are a scheduler component, a planning component and a resource managing component. Although this approach is suitable for small or virtual systems, it is not scalable and has limited extendibility, whenever there exists an underlying physical system. In the latter case, functional components become system bottlenecks if throughput increases or the underlying physical system extends.

Contrary to functional decomposition, **structural decomposition** is a more scalable and extendible approach, since each component represents a distinct entity in reality. The preference for such a structural decomposition can also be found in [Parunak 1997] in which Parunak states that agents should be *things*, not *functions*. Components divided based on relevant entities in the physical world resemble also more naturally the reality than functional abstractions. Functionality in such a structural decomposition emerges from the interaction of different agents.

Structurally decomposed software also is able to adapt better to unexpected changes or disturbances (cf. example in section 2.4.3).

2.4 Holons and holonic systems

2.4.1 Etymology

Simply stated a holon is something that is whole in itself as well as part of a greater whole. Derived from the Greek word ὁλον, the neuter form of ὁ ὁλος, a whole, it was Arthur Koestler who first used the word in The Ghost in the machine [Koestler 1968] to indicate something which is on it own whole as well as a part of a greater whole.

Although the word holon was first used by Koestler, he based this definition on Herbert Simon's insights on complex adaptive systems. Simon observed that complex systems evolve from simple systems more rapidly when there are intermediate stable subsystems. This observation was exemplified by his parable of the two watchmakers [Simon 1968].

2.4.2 Definitions

Arthur Koestler defines a **holon** as

> “An autonomous, self-reliant unit, which is able to function without instructions from higher authorities. Simultaneously, this holon would be subject to control from one or more of these higher authorities.”

This definition highlights that a holon in itself is a stable form which can cope with disturbances and also an intermediate form providing functionality to a larger whole.

Following this definition, a **holonic system**, sometimes called a holarchy, is then defined as a set of holons that, besides functioning as separate wholes and together as a greater whole, also function in coordination with their local environment. The term holarchy merges the words holon and **heterarchy**. The latter was introduced by Warren McCulloch to denote the topology of nervous nets [W. McCulloch 1945; W. S. McCulloch 1988] and hereby demonstrating that the human brain, although it seems orderly, is not hierarchically organised. Unlike a hierarchy, in which entities have either a superordinate or subordinate relationship with each other, a heterarchy defines a network of interconnected entities with *coordinate* relationships, i.e. they exist at the same level.

The relationship between a holon and one of its higher authorities, i.e. a holon representing a larger whole, is defined as **aggregation**. In this context,

aggregation does not solely define a merger between a set of holons but also includes the necessary coordination from the aggregated holon.

2.4.3 Benefits of reflection of reality

It is important to note that, by definition, a holonic system implies a **structural decomposition** of system components, in contrast to the commonly used **functional decomposition**, in which each component represents part of the system functionality. Consequently, knowledge of relevant physical entities is captured by their representative holons, i.e. for each physical entity a *single* holon exists in the system. Naturally, granularity depends on the application at hand and representations of irrelevant entities disappear in aggregated holons.

Such a **single source of truth** assures an appropriate **reflection of reality**. The underlying idea is that reality is inherently coherent and consistent. In order to cope with disturbances or future changes a system should be designed in such a way as to avoid the introduction of future limitations. Properly reflecting reality while designing for a particular application at a particular time will enhance the ability to adjust the system in the future and make it suitable for another application.

Also, a single source of truth results in correct up-to-date information for all interested holons at any time rather than that each holon maintains its own representation of that particular part of reality. In the latter case, inconsistencies and incompatibilities between different representations might get cumbersome and make the application error-prone.

Reflecting reality not only increases system **adaptability**, i.e. the ability of a system to adapt itself efficiently to external changes or changes in the application requirements. It also increases system **flexibility**, i.e. the ability of a system to efficiently cope with uncertainty, and results in an increased **scalability**. Although difficult to define in general [Hill 1990], [Bondi 2000] defines scalability as:

> “The ability to cope with an increasing work load gracefully or the ability to be enlarged to accommodate that growth.”

Since holonic systems are structurally decomposed, the addition of extra physical entities to accommodate the growing work load directly maps to the addition of more holons representing those entities.

Consider the following example as an illustration of the benefits of reflecting reality. In a navigation application two approaches are possible: either provide

the user with a road book, i.e. a sequential list of directions to be taken to navigate from A to B, or supply a map combined with instructions how to read or interpret this map, e.g. by including a legend.

A road book is application-specific and has the advantage that its usage is fairly simple, since it contains preprocessed info describing directions for each crossing. Following directions does not require map reading skills. However, a road book is not robust against disturbances or changes in the world-of-interest. Traffic jams, deviations due to a road block or goal alterations all cause the original road book to be useless, and leave its user helpless or at most with some clue on how to proceed if he manages to get back on track. It is in a sense optimised for one particular task in one particular environment and in case of robot navigation, it might even be optimised for only one particular robot.

The key problem here is that a road book provides a single functional solution. It provides path planning functionality for only one target. Such an approach is suitable for problems for which both the required effort to provide a solution and the uncertainty are low. However, whenever the uncertainty increases, i.e. unforeseen disturbances or changes in the execution environment appear, this solution loses validity and in some cases becomes unusable.

A map, on the other hand, offers a much more robust solution. Provided that the user is able to use a map given its legend, disturbances or changes are solved easily by adapting the currently chosen route to a new one. Also, a map reflects better the underlying reality or environment, whether this would be a town map with houses, roads and points of interest or a building map indicating floors, corridors and offices. Moreover, maps can be composed or aggregated into a larger map or a map with lower or higher detail. Finally, if a map does not correspond to reality and has become outdated, it suffices to look at this particular part in the environment and annotate or change only that part.

One of the reasons why a structurally decomposed solution is more robust lies in its adoption of a single point of truth. Whenever changes occur in the world-of-interest, these changes cause only those components reflecting that part of the underlying reality to adapt. In a functional decomposition all components with a representation of the environment, which in most cases are many, need to update that representation.

Previous example illustrated the usefulness of structural decomposition and reflection of reality with respect to robustness and extendibility. One of the contributions in the research presented in this text will be the application of this concept in robotics and, more specifically, multi-robot navigation.

2.4.4 Previous applications

In the past two decades, these holonic concepts have been successfully adopted, mainly in manufacturing control, and such *heterarchical* systems have shown to be beneficial for both system scalability and system flexibility [Monostori et al. 2006]. At PMA/KU Leuven, the Holonic Manufacturing Execution System (HMES) was developed following the Product-Resource-Order-Staff Architecture (PROSA) [Van Brussel et al. 1998] and using the delegate multi-agent system (D-MAS) coordination mechanism [Holvoet et al. 2009].

In the subsequent paragraphs this HMES will be discussed in further detail, starting with PROSA and the coordination mechanism D-MAS and concluding with a brief overview of past applications. Related holonic systems are presented in chapter 3 and an elaborated analysis of the HMES is provided in chapter 5.

2.4.5 Relation to system complexity levels

Due to the robustness to uncertainty and disturbances it offers, structural decomposition provides a solid basis for designing systems categorised in the $S4$- and $S5$-levels. In fact, making a structural rather than functional decomposition is exactly the required paradigm shift in order to cope with the increased complexity levels of $S4$- and $S5$-systems.

Therefore, adopting the holonic paradigm in robotics is an important prerequisite to reach a higher level of complexity and provides a way to develop applications which can run in realistic dynamic environments.

Much of the previous work in holonic systems dealt with $S4$-systems in manufacturing, while in robotics, most systems belong, due to their design limitations, to the $S3$-level.

2.4.6 Relation to robotics & software engineering

The concept of structurally decomposing an application into holons, representing the relevant entities in reality, maps very well to component-based software engineering in which an application is decomposed into a set of software components.

Since in most recent robot software developments component-based software engineering has played a prominent role, the introduction of holons, PROSA and DMAS into robotics will, at the implementation level, not be cumbersome.

In chapter 3 component-based software engineering is explained in more detail and recent adoptions of this technique in robotics are discussed.

Nevertheless, as will become clear in the remainder of this chapter, the paradigm shift from $S3$-level applications to $S4$-level or even $S5$-level applications is the most difficult step in adopting holonics in robotics.

2.5 Reference architecture PROSA

2.5.1 Definition

The Rational Unified Process [Kruchten 2000] defines a **reference architecture** as:

> “A predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artefacts to enable their use. Often, these artefacts are harvested from previous projects.”

A reference architecture maps functionality onto a system decomposition and defines the interaction between component types. However, it does not enforce constraints on the elaboration of this functionality nor does it yet provide solutions to fulfil this functionality [Wyns 1999; Bass et al. 2003].

It is therefore sometimes referred to as an open-ended or partial solution which defines the building blocks of a system but not a usable system in a particular application domain.

Furthermore, an actual system adopting a particular reference architecture by filling in those components is called an **instantiation** of such reference architecture.

Analogously, in building architecture, the architectural style, such as Gothic architecture or Baroque, is the reference architecture, whereas the actual churches applying this style are the instantiations.

Figure 2.5 shows the main components (holons) of PROSA. This reference architecture is developed in accordance with the holonic manufacturing paradigm (see e.g. [Van Brussel et al. 1998], [McFarlane et al. 2000], [Babiceanu et al. 2006], [Monostori et al. 1998]) and is implemented as a multi-agent system (MAS).

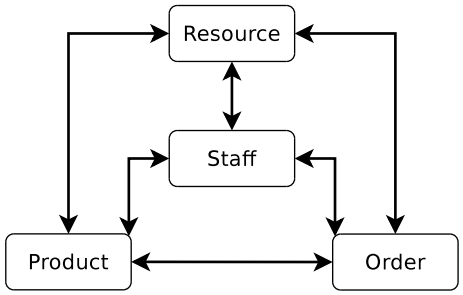


Figure 2.5: Overview of the PROSA reference architecture. Each rounded rectangle represents one of the holon types used in PROSA and the arrows between these holons represent their interactions.

A **multi-agent system** (MAS) consists of interacting independent **intelligent agents**. These agents are autonomous, self-reliant components with a local view of the world and together their interaction results in a decentralised control, able to solve problems which are difficult or impossible to solve using a monolithic system.

Multi-agent systems reflect very well the reality in which different independent agents, such as humans or robots, interact which each other to solve a problem.

A detailed description of PROSA's software architecture can be found in [Verstraete et al. 2008].

The following sections define each of the PROSA-holons and provide examples of their usage in robotics.

2.5.2 Resource holons and their allocation

Guideline 1:

The need to associate a resource holon to an entity in the underlying world-of-interest is based on the entity's relevance to the application.

Definitions

A **resource holon** corresponds to any entity in the underlying world-of-interest offering services and operations to others, relevant for the application at hand. More concretely, any entity that will be used by other holons for those services, during some period of their lifetime, should be represented by their own resource holon.

Moreover, resource holons require **explicit allocation** and deallocation and also maintain specifications and capabilities of the underlying entity. They represent the limitations that other holons have to satisfy, e.g. how many holons are allowed to simultaneously allocate a particular resource holon and during which time intervals.

This explicit allocation of resources is a crucial aspect and differs greatly from most existing robot architectures. The allocation implies the transfer of particular **rights** for a particular time interval toward the **user** of that resource. Depending on the **policy** adopted by the resource, these rights can, for instance, offer read or write access, a subset or complete set of available services, full access, partial access, or conditional access (e.g. possible pre-emption by another holon).

Examples and aggregation

Guideline 2:

Always allocate the minimal set of resources required for the application and allow possible other applications to allocate the rest. Define aggregated resource holons based on those minimal sets of resources.

Which entities are represented by resource holons depends largely on the application. Resource holons can represent parts to be assembled, machines that assemble the parts or even bandwidth on a computer network or processing power of a certain node in such a network. In robotics, these holons represent robotic platforms, their sensors and actuators, individual sensors planted in the environment, but also doors, rooms and corridors.

Resource holons representing sensors and actuators on a robot can be **aggregated** to a resource holon representing that robot as a whole. In Figure 2.6 and Figure 2.7 an example set is given of respectively resources of a mobile

manipulator and resources of a robotic wheelchair. The former consists of a mobile robotic platform with a robot arm on top of it while the latter example represents a robotic wheelchair or any mobile robotic platform with a joystick interface.

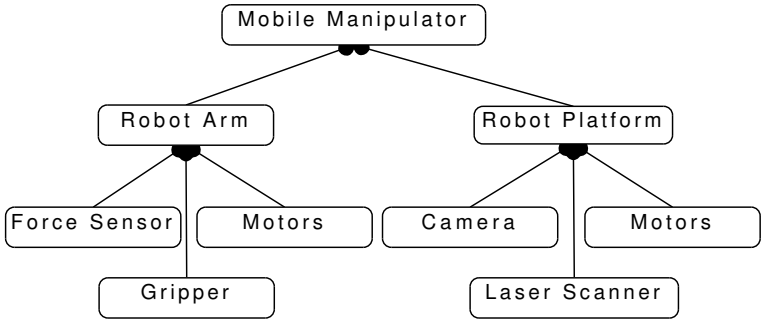


Figure 2.6: Example set of resource holons representing a mobile manipulator. Each rectangle represents a resource holon and the connections indicate aggregations. The granularity level of the resource holon changes from top to bottom.

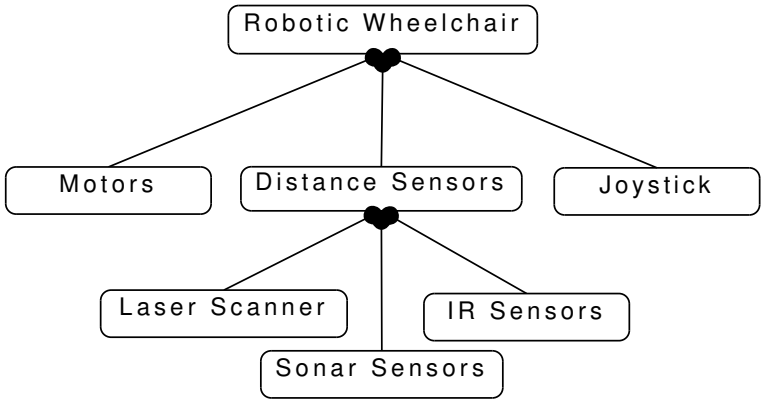


Figure 2.7: Example set of resource holons representing a robotic wheelchair. Each rectangle represents a resource holon and the connections indicate aggregations. The granularity level of the resource holon changes from top to bottom.

The allocation of an aggregated resource holon results in the allocation of all underlying resource holons. The guideline for developers is to always allocate the minimal set of resources required for the application. For instance, an application which requires the motors and laser scanner of a robotic wheelchair

but does not use other sensors or actuators on the platform, should only allocate those two resource holons. Moreover, the allocation should always be limited in time. This ensures that other applications are allowed access to those sensors and actuators of that specific robot whenever they become available.

Note that the definition of aggregated resource holons is also application-specific. For instance, in Figure 2.6 the laser scanner resource holon directly aggregates to the robot platform resource holon while in Figure 2.7 an intermediate distance sensors resource holon is defined as the aggregation of all distance sensors on that particular platform. Since the mobile manipulator has only one type of distance sensor, there is no need to define this extra aggregation. However, if sensors would be added to this platform in the future and such a resource holon would become relevant or even required, it can be easily added. Indeed, the connections or rather relationships between resource holons are dynamic and old aggregations can be replaced by new ones at runtime. This also reflects best the underlying reality in which certain entities appear and disappear, e.g. a robotic wheelchair enters a room at a particular time and later leaves that room again. Such behaviour would not be representable by only static connections.

Both sets of resource holons can, in turn, be aggregated to a resource holon representing the complete reality or *universe*. Including such a holon simply serves as a construct to allow proper extensions to an application conform to the underlying reality. It also serves as a holon containing all other physical entities which are not relevant to the application at this particular moment in time.

This resource holon should not be allocatable, but offer a way to add new entities, to remove or archive existing ones and to provide a method to merge different applications built by different developers or different companies. In essence, it ensures interoperability between all applications adopting this design paradigm.

For example, in Figure 2.8, resource holons of a robotic wheelchair (cfr. Figure 2.7) and a mobile manipulator (cfr. Figure 2.6) are aggregated, together with a holon representing a door and a holon representing a human operator, into a universe resource holon. At this point the rooms leading to the door opening are not represented by resource holons yet and thus reside in the universe holon. However, whenever this becomes necessary, proper room resource holons can be created and placed at the same level as the door resource. In practice, when these rooms are restricted or have a low capacity, e.g. no more than one robot can reside in the room, creating such holons is advised.

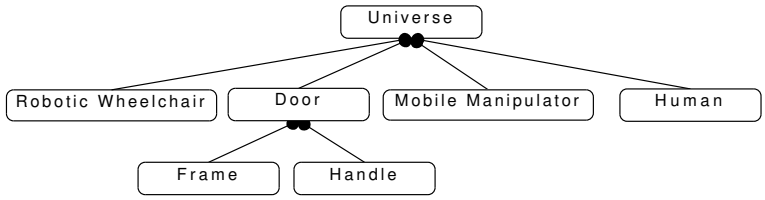


Figure 2.8: The aggregation of different sets of resource holons into one comprehensive *universe* holon. This holon ensures proper reflection of reality and serves as a container for all physical entities which are not yet represented by resource holons or do not require this at this moment of time.

Guideline 3:

Include a universe resource holon representing all other entities in the environment which are not yet modelled by other resource holons. It should not be allocatable but ensure interoperability with other applications.

Reflection of reality

Given the definition of a resource holon, these holons clearly reflect the underlying reality and through aggregation any required level of granularity can be achieved. The examples in Figures 2.6 and 2.7 already showed such aggregation.

In some cases, allocating a single sensor of a robot can be more appropriate than allocating the whole robot. For example, if a user or another holon requires information about a particular part of the environment, a robot present in that area could *share* its sensor data. Concretely, the resource holon responsible for that sensor will allow simultaneous allocation to provide this service. In other words, the user of that sensor resource holon receives read rights for a specific time interval. Evidently, not all resource holons are eligible for such multiple allocation at a certain time. Simultaneously allocating an actuator, such as the motors on a mobile platform or a robot arm, should be made impossible. By definition, resource holons can refuse allocation for a requested time slot or service. Sharing a sensor might not be always possible either. Consider a sensor which implies that the robotic platform should stand still, e.g. a camera which needs to take a very precise picture. Following the design guideline with

respect to allocation proposed earlier, allocation of this sensor should only be made possible through an aggregated resource holon containing both the sensor and the motors of the robot platform.

Associating a resource holon to a particular entity is a design decision. If your application, for example, consists of a single robot arm performing a peg-in-hole operation in a reserved workspace without being interfered by others, the benefits of resource allocation are limited and the only resource holon would be the peg which need to be placed in the hole. Representing the robot arm itself by a resource holon might not contribute to this application since the robot arm is not used for anything else but placing pegs in a hole, but would still be good practice and allow your system to support more advanced applications. The resource holon's policy at first could be to basically grant full control to the application and later, when resource allocation becomes more critical, the policy could be updated.

Of course, applications in which multiple robots, users and restricted environments are used, benefit much more from structural decomposition into resource holons and explicit allocation of these resources. Consider the application scenario, in which multiple robotic wheelchairs navigate in a restricted and dynamic environment. Rooms and corridors can then be represented by resource holons to allow proper allocation of the space these mobile robot platforms need. Coordination can then be distributed over these resource holons. Experiments discussed in subsequent chapters show that structural decomposition into resource holons and explicit allocation of these resource holons results in an $S4$-level system and provides a way to robustly solve multi-robot navigation problems, such as deadlocks or bottlenecks.

Guideline 4:

Although not all applications might benefit directly from explicit resource allocation, designing for *the unexpected* increases system flexibility.

2.5.3 Product holon

Definition

A **product holon** encapsulates knowledge of a particular product type; how instances need to be produced and what services or operations are required.

These services and operations are offered by resource holons.

This knowledge is captured in a process plan available to other holons. In its most simplistic form this plan contains a set of sequential steps required to assemble the product type it represents. More advanced plans contain finite state machines [Arbib 1969] or Petri nets [Petri 1966] in which the next step in production is dependent on the current state of involved holons.

Although a product holon contains the process plan, it does not contain information on which specific machines, i.e. resource holons, should be used to perform those operations. Instead, a process plan shows what services or operations, offered by those resource holons, are required. Finding the right resource holons is part of the order holon's task. A product holon also does not contain the state information of the current process plan. In other words, the product holon defines how a product type should be produced but does not maintain the progress of actual products, i.e. product type instantiations, which are produced.

Several product holons can also be combined into an **aggregated product holon** representing the combination of those process plans. This way, individual process plans can be efficiently combined but still remain independent from the other process plans in the aggregation. Moreover, an aggregated product holon defines the sequence and conditions in which those product holons are used.

Also note that a product holon's process plan can change over time, depending on the operations resources offer. For instance, if a new, more efficient resource, i.e. a new robot arm with similar functionality but higher performance, is added, product holons are able to discover its presence and update their plans.

Examples

Guideline 5:

For each task type which requires the allocation of one or more resource holons, a product holon should be defined, containing the task specification and process plan. Separate product holons can be aggregated into an aggregated product holon.

An example of product holon in manufacturing is a vehicle type which needs to be produced in a car factory. The process plan contains knowledge about all required vehicle parts and how they should be manipulated and assembled.

As with resource holons, a product holon might be an aggregation of separate process plans, such as chassis assembly, installation of motor components, inspection of body work and vehicle painting.

In a robotics context, product holons define specifications to achieve a task or goal. The process plan consists of a set of instructions the robot has to follow to execute a certain task. In the multi-robot navigation scenario, a process plan could consist of a sequence of navigation operations to move from one location to the other. The level of detail of these navigation operations depends on desired granularity at which PROSA and its coordination need to be applied. On the one hand, these operations can be very abstract and represent the presence of the robot in a particular room, and on the other hand, they can be fine-grained operations to move the robot at a particular velocity for a particular time interval. In the former case, PROSA's coordination is limited to allocating a set of rooms in subsequent time intervals, through which the robot will move and by doing so, allowing other robots to anticipate on these allocations and update their navigation plan accordingly. In the latter case, PROSA is used at much finer detail to, for instance, avoid obstacles in a particular area. Consequently, the granularity of the product and resource holons has to conform with the desired level of coordination. Indeed, coordinating navigation in such detail requires detailed allocation of part of the room or area.

In other robotics applications, such as manipulation tasks or human-robot interaction, the product holons also represent task specifications, which commonly is described by a finite state machine. To enhance reusability, different states can be represented by separate product holons and then be combined into an aggregated product holon.

In chapter 4 the implementation of these product holons and their specific process plan in robotics applications is discussed in more detail. It will become clear that PROSA's coordination is more appropriate at a global level than at a local robot control level, i.e. the coordination has a coarse-grained detail with respect to space and time.

2.5.4 Order holon

Definition

An **order holon** represents a product *instance* that needs to be produced at a specific time instance. It is responsible to handle the required resource allocations to accomplish this. Therefore, the order holon consults its corresponding product

holon to find out what services it needs and searches for the proper resources to accomplish it.

It's clear that each order holon requires a corresponding product holon of which it receives the process plan. In other words, for each desired task, a task description or specification is required.

Moreover, the order holon is responsible for the actual execution of the process plan and, thus, maintains also the state of the plan it received from its corresponding product holon.

Note that, due to interaction between order and product holon and the data encapsulation within holons, there is no *data model lock-in*. In other words, the representation of the same data, such as a process plan, might be completely different in order and product holons. Therefore, holons are not dependent on each other's internal data model.

Examples

Guideline 6:

For each task that requires a set of resource holons to execute a particular set of services or operations at a specific time instance, an order holon is required. This holon is responsible for searching and allocating a set of available resource holons which offer the required services laid out in the process plan it receives from the product holon.

In the car assembly example, an order holon refers to an actual order entering the system, requesting the assembly of a specific type of car at a specific time. It receives assembly instructions from a corresponding product holon and finds the correct available resource holons to perform these instructions.

In a robotics context and more concretely, multi-robot navigation, an order represents a robot moving from one particular room to another at a given time instance. The relevant product holon provides navigation operations the robot has to perform, e.g. go through a particular room or doorway. The order holon then searches for available resource holons, i.e. rooms and corridors, which allow the robot to move to its destination.

In a human-robot interaction application an order holon could, for example, represent a force following task in which the process plan then specifies that the motors of the robot arm and its force sensor are required resources.

Also order holons can be aggregated into an **aggregated order holon**. Concretely, an order to execute a large task set on a robot can be divided into separate batch orders, and further into individual work orders. For example, a pick and place task of a robot manipulator can be represented by an aggregated order holon. This holon would then consist of a sequence of operations such as moving from a home position to a particular object, opening the gripper to grab the object and moving to a given destination to release the object.

2.5.5 Staff holon

Definition

In view of the complexity of resource allocation, **staff holons** may assist other holons through advice on particular aspects. However, this advice is not binding and the order and resource holons still take the final decisions.

The addition of staff holons introduces a mechanism to overcome the problem that in some situations a traditional heterarchical control architecture is outperformed by a traditional centralised or hierarchical control architecture. In the former all necessary decisions to achieve a certain task are distributed over the distinct holons in the system while in the latter a centralised decision making component steers the other components. In stable and predictable situations, such a centralised system outperforms a distributed solution and vice versa in more uncertain and turbulent situations.

The key difference between staff holons and traditional higher level decision making functionality found in hierarchical systems is the advisory role these holons take on. This way, their decisions are not enforced on to the other holons in an inflexible way, which is one of the main disadvantages of a traditional hierarchical system.

Examples

Guideline 7:

Although optional, a staff holon can provide decision making advice to other holons. This advice should not be enforced but used to structure and enrich the search space.

An example of a staff holon in the multi-robot navigation scenario could be a path planner giving advice on a preferred route the robot could take. This information can then be used by the order holon to optimise its search space and consult resource holons along this route.

Another example would be a staff holon providing advice on which resource holon is most appropriate in a particular situation. In some cases more than one resource offers the particular service and is available at a desired time instance. The order holon needs to choose one of those resources based on some predefined criteria. A staff holon could assist with that decision.

A concrete example of a staff holon in robotic wheelchair navigation and shared control are the estimated user intentions [Demeester 2007; Hüntemann 2011] that can be used to narrow down the search space.

Since PROSA can be used without including staff holons, the experiments and research discussed in the remainder of this text rarely mention these holons. More information on research in which they were used, can be found in e.g. [Van Brussel et al. 1998; Verstraete et al. 2008].

2.5.6 Separation of concerns

It is important to note that PROSA separates two different **concerns**, which have been addressed to some extent in the previous sections.

Firstly, product and resource holons address **technical feasibility**, i.e. making products in a technically correct and validated manner. Product holons are knowledgeable concerning the capabilities and services of resources that are relevant to them, but they ignore resource capacity and availability aspects. The product holons inform the order holons about all technically correct manners to produce their product instance.

Secondly, order and resource holons address the issue of **resource allocation and product flow coordination**. Within their solution space, identified by the product holons in cooperation with the resource holons, order holons arrange resource allocations and the execution of process, storage and transportation services to get their product instance produced.

2.5.7 Interaction & data flow

All three holons interact with each other in some way, and this flow of data between different type of holons is illustrated in following two examples.

The interaction and data flow between different holons in the car assembly example boils down to the following: The order holon representing an order for a specific car at a specific time interacts with the corresponding product holon, i.e. a product holon containing a correct assembly or process plan for the requested vehicle, and receives a set of possible assembly instructions given its current state. Then it consults resource holons, e.g. car parts, assembly lines, robot arms and painting robots, to plan an execution. These resource holons all maintain their own schedule and indicate which time slots are available and what the estimated duration of the requested operation would be. Depending on the availability one of the provided assembly instructions is chosen and the order holon checks the product holon for further instructions. This process is continued until the whole car assembly task is completed.

Concerning the multi-robot navigation example, an order holon represents a request to move an available robot to a particular position or area. This order holon interacts with a product holon containing the knowledge to perform this navigation. This can be a high-level operation a robot resource has to execute or low level navigation operations, as was discussed earlier in section 2.5.3. The order holon then checks the availability of the required resources, i.e. rooms and corridors the robot has to pass through to reach its goal. Depending on the application, the robot itself might also require allocation, and in that case, is also represented by a resource holon. If the order holon has found a resource capable of executing its current operation, it confirms the allocation and reports back to the product holon to request the next set of instructions.

As was already briefly noted above, internal data representation in order and product holons is independent from each other since data is transferred in an interactive manner. In other words, the product holon acts as a *help desk*, rather then providing a user guide to order holon. Therefore there is no data model lock-in between holons in PROSA.

2.6 D-MAS as coordination and short-term forecasting mechanism

2.6.1 Introduction

Imagine a doctor's waiting room in which you do not spend the usual 30 to 45 minutes waiting but are called almost immediately. Or consider rush hour traffic at the end of a working day which actually does justice to its meaning, i.e. rushing home from work instead of burning valuable fuel and time in the usual traffic.

Analogously, in robotics, consider a multi-robot application with a particular number of mobile robots and a larger group of users, desiring a mobile robot with particular capabilities at a certain time instance. The time they have to wait is optimised as is the allocation of robots to users at each time instance and the interference between robots is minimised.

In such highly distributed systems, short-term forecasts with respect to resource allocation can anticipate delays or bottlenecks and solve such a scheduling problem spread over many distinct entities.

This coordination problem is solved in PROSA by the deployment of delegate multi-agent systems (D-MAS) [Holvoet et al. 2006; Holvoet et al. 2009; Valckenaers et al. 2005]. These multi-agent systems adopt the *delegate pattern*.

The **delegate pattern** enables agents, e.g. the PROSA order holons, to *delegate* part of their task to small lightweight agents, referred to as ants. These agents travel through the environment, e.g. the resource holons, execute the requested task, e.g. gather allocation information, and report back to the agent which created them.

In the HMES implementation, order holons use D-MAS to gather resource availability forecasts. Depending on the application, these resources can be doctor appointments, or more specifically an examination room in which the physician you requested will be consulting, road and highways filled with traffic during rush hour or more into scope, robotic wheelchairs required by patients to move from one place to another and the rooms and corridors these robots occupy during such request.

This bio-inspired coordination mechanism, called delegate multi-agent system (**D-MAS**) enables holons to receive short-term forecasts with respect to resource allocation and achieve improvement in their task execution through these predictions. The PROSA order holons create (many) lightweight delegate agents to perform some task on their behalf.

These short-term forecasts are comparable to *preview control*, i.e. control not only based on feedback but also estimations or forecasts of future events. The granularity at which D-MAS is applied depends on the application at hand. The short-term forecasts could improve the overall coordination of multiple robots (coarse-grained) or the individual robot movements (fine-grained).

The order holons make use of two D-MAS types: exploring ants and intention ants. The behaviour of these delegate MAS are based on food foraging of ants. In this context ants travel through resources querying for available allocation slots.

2.6.2 Exploring ants

Their task is to find feasible routings, accounting for resource availability and capacity, capable of fulfilling their task.

The **exploring ant** traverses the resource graph and searches for a possible solution for the order holon it is linked to.

The **resource graph** has resource holons as vertices and the directed edges between those resource holons reflect links between the corresponding entities in the environment. For example, a door is connected to the two rooms or areas it provides access to.

In case of aggregated resource holons, the ant zooms in to the individual resource holons within their aggregation. The ant requests those resource holons for information about resource availability and timing and uses that time as the start time for the next resource request. To be able to generate these forecasts, the lightweight agents can make use of the *what-if* functionality of resource holons. As these holons reflect a part of reality, they have to be able to answer what-if questions about the dynamic behaviour of their corresponding reality. Therefore, in a system with D-MAS, every holon contains a model that supports virtual execution. In the current HMES implementation, multi-models are used, which allow the use of different models with different formalisms and different levels of detail [Fishwick et al. 1992; Verstraete 2009]. After a solution is found the ants notify the order holon, which created them, with the result.

Ants do not multiply themselves on a decision branch, but choose one path based on rules provided by the order holon. The most straightforward rule randomly chooses one of the available paths. Each ant also possesses a predefined life time and hop count to prevent that ants would traverse the resource graph indefinitely.

However, this is a implementation detail and cloning variants exist in which, for instance, the number of ants is constrained by a cloning budget.

2.6.3 Intention ants

Intention ants propagate the order holon's intentions, informing resource holons about future visits (allocations). Their behaviour is similar to that of exploring ants with the exception that its task is to book the reservation on the resources rather than to request a virtual execution.

The resource holons maintain a schedule in which future allocations are registered based on the information from the intention ants. This is accounted for when

exploring ants interact with the resource holon while constructing their candidate routings. The resource's agenda corresponds to a short-term forecast of the load of the resource, while an order's intention is a short-term forecast of the product routing.

The decision on which resource holons are eventually allocated is made by the order holon itself. D-MAS only provides a way to gather information from a distributed set of entities and a way to reserve some of these entities, i.e. make allocations. Those decision rules, encapsulated in the order holon, are application-specific. For instance, in some applications a time optimal solution might be desired, while in other applications space optimal solutions are preferred.

Guideline 8:

The frequency at which order holons send exploring and intention ants, their life time, hop count and rules to use at decision branches, largely depend on the desired application and should be tuned accordingly. Moreover, the order holons decide which resource holons are eventually allocated, based on its optimisation criteria. Chapter 5 analyses these parameters in more detail and provides further guidelines in how to tune them.

2.7 Generalisation and applicability

The developers of PROSA and D-MAS minimised the introduction of constraints on the applicability of its results. Basically, the technology can be applied to applications in which activities are executed on resources subject to technical constraints and virtual execution of these activities is possible and can be much faster than in reality.

Candidate application domains address manufacturing, logistics, traffic and transportation, building construction, road construction, harvesting, power grids, etc. The main challenges are the design and development of the digital mirror image capable of rapid virtual execution and the ability to connect the resulting coordination system to the real-world systems, e.g. capture user intentions by machine learning.

Figure 2.9 shows a *rephrasing* of PROSA to account for a wider and more abstract or generic application domain. Besides the generalisation in terminology, this

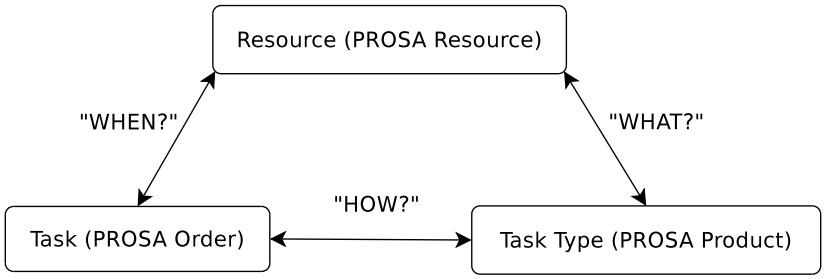


Figure 2.9: Rephrasing of PROSA for a generic application domain. Besides the generalisation in terminology, this schema relates the interactions between the holon types to three task specific questions. *How* a task is executed is communicated between the task and task type holon, deciding *when* the task is executed involves the task and resource holon and determining *what* or which operations can be done results from the interaction between the task type and resource holon.

schema relates the interactions between the holon types to three task specific questions. *How* a task is executed is communicated between the task and task type holon, deciding *when* the task is executed involves the task and resource holon and determining *what* or which operations can be done results from the interaction between the task type and resource holon.

This chapter concludes by demonstrating the general applicability of the HMES in past applications. Besides manufacturing control applications, the overview below discusses several cases in other application domains, such as open-air engineering and logistics. The following is a summary of the work described in Van Belle et al. 2012.

2.7.1 Manufacturing

The HMES is originally developed for manufacturing environments to provide robustness and scalability in an industry with increasing complexity and highly dynamic environments. It offers a distributed, flexible and self-organizing manufacturing control approach and has been applied to several cases, discussed below.

Car paint shop

In this application a large car body paint shop with various manufacturing resources arranged in a complex topology was equipped with the HMES responsible for car body routing through the paint shop and for maintaining the required throughput in the face of disruptions. Since the transport system contained loops, deadlock detection was required. The HMES solved this by a layered decision mechanism in the order holons.

More information on this application can be found in e.g. [Peeters et al. 2001; Valckenaers et al. 2003; Valckenaers et al. 2001; Van Brussel et al. 2000].

Photographic foil facility

A second case study, discussed in more detail in [Saint Germain et al. 2003; Verstraete et al. 2006], dealt with optimising the cutting of large rolls of photographic foil in order to reduce losses of the material as much as possible.

The facility had a flow shop with some flexibility in the routing and storage possibilities and common disturbances included planned and unexpected maintenance. More seriously, some disturbances damaged the photographic foil, resulting in an unusable master roll for the current customer order.

The HMES was responsible for the correct and timely production of customer orders and had to deal with these disturbances, such as recycling damaged parts as much as possible.

Machine tool shop

Another application addressed production organisation by routing containers in an automated storage-and-retrieval system of a job shop. Each container contained a variable number of identical components, travelling together until completion.

Increased throughput, lead-time minimisation and improved labour and resource utilisation were achieved together with an optimised container transporter usage. More information about this application can be found in e.g. [Valckenaers et al. 2005; Verstraete 2009; Zamfirescu et al. 2003].



(a) Open-pit mining



(b) Road construction



(c) Agriculture harvesting

Figure 2.10: Open-air engineering processes

Heat treatment facility

A last manufacturing case study addressed a highly automated multi-plant heat treatment facility, in which routing of metallic parts was optimised. These parts had to receive the correct treatment to obtain the required quality.

An essential part in this application was the possibility to batch parts with compatible process temperature trajectories and environmental conditions. A significant performance increase can be achieved this way. Order holons used D-MAS to discover batching opportunities or, alternatively, to trigger the build-up of such batches.

Scalability was also investigated by coordinating manufacturing and transportation activities within networked production. To cope with this network of factories, order holons had to route their corresponding parts at two levels: the network and factory level. At the network level transportation services between the different factories and heat treatment services were searched.

More details about this application are given in [Saint Germain et al. 2011; Saint Germain et al. 2007].

2.7.2 Open-Air engineering

The HMES has also been applied to coordinate open-air engineering processes such as open-pit mining, road construction and agriculture harvesting (see Figure 2.10). Since in such systems the equipment is considerably expensive, the aim is to optimise performance objectives such as energy consumption through proper resource allocation and scheduling decisions.

In these open-air engineering applications multi-resource allocation is sometimes

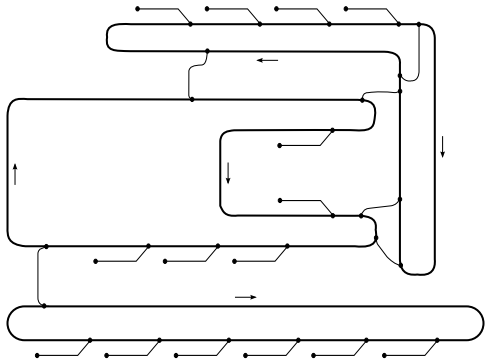


Figure 2.11: An example of a chain conveyor network

required in which an order holon needs multiple services and resources at the same time. For instance, in open-pit mining, to mine a certain area, the service of an excavator is required, together with the service of a dump truck to transfer the excavated product.

For more details, the interested readers are referred to [Ali 2010], [Valckenaers et al. 2011].

2.7.3 Logistics

Finally, two logistics applications are discussed here in which the HMES has been successfully applied.

Chain conveyor system

Distribution centres often use a complex transportation network built out of several chain conveyors, as is depicted in Figure 2.11. The control system has to decide about the routes that products follow and when these products are transported. Moreover, the control system has to deal with uncertainties and disturbances (e.g. defect carts, delays, jammed chains, etc.).

Using the HMES, control of product routes can be made more flexible and dynamic. In contrast to the static approach, by introducing resource holons, maintaining the state of their corresponding conveyor belt and order holon searching for an available route, each product receives an individual route and can react to disturbances such as a jammed chain. The short-term forecasts

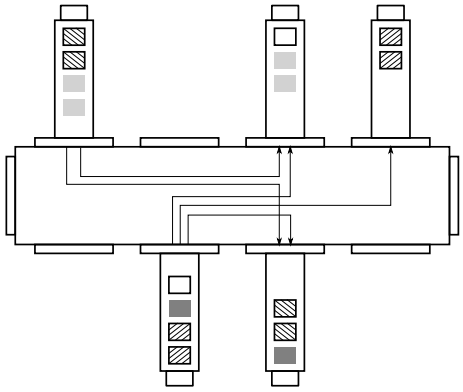


Figure 2.12: Material handling at a typical cross-dock

even allow to anticipate certain disturbances and to find an alternative route. More details can be found in [Van Belle et al. 2009; Van Belle et al. 2011b].

Cross-docking facility

Cross-docking is a logistic strategy in which incoming shipments are (almost) directly transferred to outgoing trailers, with little or no storage in between (see Figure 2.12). If the shipments are temporally stored, this should be only for a short period of a time. An exact limit is difficult to define, but usually 24 hours is assumed to be the limit. Cross-docking can have several advantages: consolidation of shipments, shorter delivery lead times, reduction of costs, etc. However, the organisation of the cross-docking operations is a complex and challenging task, certainly because the arrival and departure times of the inbound and outbound trucks need to be synchronised. Moreover, cross-docks operate in an uncertain and dynamic environment, among others due to a tough competition in the transport and logistics sector and ever-increasing traffic.

The current approaches to control a cross-dock are usually planning approaches, in which the plan is made off-line before the operations start. While in practice several operational decisions have to be taken together, these approaches are usually concerned with just one decision, for instance how will the trucks be assigned to the available dock doors (see e.g. [Tsui et al. 1990; Bartholdi et al. 2000; Yu et al. 2008; McWilliams 2009; Boysen 2010]), at which location will the goods be temporary stored (e.g. [Vis et al. 2008]), how will the vehicle routing be performed (e.g. [Wen et al. 2009]), etc. Moreover, the current approaches usually assume that all necessary information (e.g. the exact content and arrival time

of the incoming trucks) is fixed and known on beforehand. Also, the problems are usually assumed to be static, while the control of a cross-dock is inherently dynamic (trucks arrive early or late, equipment fails, etc.). Consequently, a more robust and dynamic control approach is desirable.

The concepts and principles used in the HMES are applied to control a cross-docking terminal. For every resource in the world-of-interest, there has to be a corresponding resource holon. In the context of cross-docking, this means that for instance all trucks, forklifts and dock doors will have an intelligent resource holon. This intelligent resource contains a model of the dynamic behaviour of the corresponding real-world resource so that what-if questions can be answered. It is also responsible for its own local decisions (e.g. a dock door should decide which truck it handles). All goods that have to be handled in the cross-dock are represented by intelligent products. These intelligent products are responsible for routing their corresponding entities through the cross-docking system. Therefore, they can make use of the available services offered by the intelligent resources, such as loading or unloading, internal transportation, and temporary storage.

Also for this application, multi-resource allocation is an issue. For instance, when goods have to be unloaded from a truck, these goods require the unload service from a forklift (and a driver), while at the same time the truck and a dock door have to be available (these resources also have to be allocated, even if they do not perform an active service in this situation).

In [Van Belle et al. 2010; Van Belle et al. 2011a], more information about this application can be found.

2.8 Conclusion

The aim of this chapter was to present existing research, which primarily covered the manufacturing domain in the past, and the important role it can play the in development of highly complex robotic systems. Indeed, overspecialisation sometimes causes a too narrow focus and interesting developments in other domains are overlooked.

In order to categorise systems with respect to their complexity, five levels of increasing complexity, $S1$ to $S5$, were defined, based on the S-levels introduced in [Bruyninckx 2010]. Each new level introduces a *paradigm shift* or signifies a *quantum leap* compared to its immediate predecessor.

Holons and holonic systems were introduced next and their *structural decomposition* provides the basis to design level $S4$ and $S5$ systems. This chapter

also provided a overview of the Product-Resource-Order-Staff-Architecture (PROSA) and its coordination mechanism delegate multi-agent system (D-MAS), which were developed based on the holonic philosophy.

It also highlighted the importance of reflection of reality in system design to cope with changing environments and disturbances. By giving numerous previous applications of PROSA and D-MAS, it showed the benefits they offer with respect to flexibility and scalability.

Through small examples in each section, PROSA and D-MAS concepts were introduced in the domain of robotics. Furthermore, concrete design guidelines were given to inform the reader how to apply these concepts in the best way for his specific application.

Throughout the remainder of this text, the adoption of holonics in robotics, and more specifically, the application of PROSA and D-MAS to multi-robot navigation is discussed.

First, however, in the next chapter an overview of related work and state of the art in holonic systems, robot software and multi-robot navigation is provided to identify the shortcomings current approaches suffer from and for which of them PROSA and D-MAS are suited to provide solutions.

Chapter 3

Related Work and State of the Art in Holonic Systems

> *Your paradigm is so intrinsic to your mental process that you are hardly aware of its existence, until you try to communicate with someone with a different paradigm.*

Donella Meadows (⋆1941–†2001)
The Global Citizen, Island Press, 1991

3.1 Introduction

This thesis deals with the development of a new paradigm for controlling and coordinating multi-robot systems. In particular, the scenario described in section 1.4 is followed here. Compared to existing systems it exhibits a number of innovative features which make it modular, easily scalable and extendible.

Due to the interdisciplinary nature of this research, related work and state of the art discussed in this chapter are divided into several research domains. The goal of this chapter is to provide a clear overview of past research in these research domains, while still keeping the above scenario in mind.

In this chapter, a comparison with the state of the art is made. We will consecutively deal with (i) relevant **software frameworks and architecture**

that underpin the structure of the developed system architecture, (ii) relevant **system architectures**, in particular the multi-agent or holonic approach, (iii) relevant **control architectures**, in particular the D-MAS approach, including path planning and multi-robot coordination approaches.

3.2 Component-based robot software frameworks

3.2.1 Overview

This section evaluates state of the art robot software frameworks using, on the one hand, the five aspects of separation of concerns discussed in section 2.3.3, and, on the other hand, the constraints or limitations these frameworks might impose with respect to decomposition and software architecture.

Although most frameworks separate one or more concerns, none of them complies with all *5Cs*. Also, some frameworks enforce a particular architecture or decomposition, limiting not only its reusability but also the application domain.

In Table 3.1 an overview is given of the reviewed frameworks. In the following paragraphs each framework will be discussed in more detail.

	Separation of concerns					No constraints towards	
	C1	C2	C3	C4	C5	Decomposition	Architecture
Orca	✓	✓	✗	✓	✓	✓	✓
RT-Middleware	✗	✓	✗	✓	✓	✓	✓
ROS	✗	✓	✓	✓	✗	✓	✓
OROCOS	✗	✓	✓	✓	✗	✓	✓
Microsoft RDS	✓	✓	✗	✗	✓	✗	✗
Player / Stage	✗	✓	✗	✗	✓	✗	✗

Table 3.1: Overview of robot software frameworks, C1 to C5 are the 5 aspects of separation of concerns as described in section 2.3.3.

Reviews on more implementation details such as operating system support, built-in features and graphical tools of robot component frameworks can be

found in [Michal et al. 2011] and [Elkady et al. 2012].

3.2.2 Orca

Orca [Brooks et al. 2005; Makarenko et al. 2007] is an open source framework designed for mobile robotics. Each Orca component is a stand-alone process and interacts with other components through the interfaces provided by Orca. Their main goal is to provide an easy way to develop reusable software components. It aims to provide a component model which facilitates the connection and interaction between components regardless of the host platform and their location. Also, it provides a repository of reusable components for mobile robotics applications.

The design philosophy adopted in Orca is to impose as few constraints as possible. More concretely, no assumptions are made about system architecture, interfaces between components and internal computations.

Communication ($C4$) between components is provided through the Ice middleware [Henning 2004] and clearly separated from internal computations ($C3$). Figure 3.1 shows how two components running on different operating systems and written in different programming languages are able to communicate through this Ice middleware.

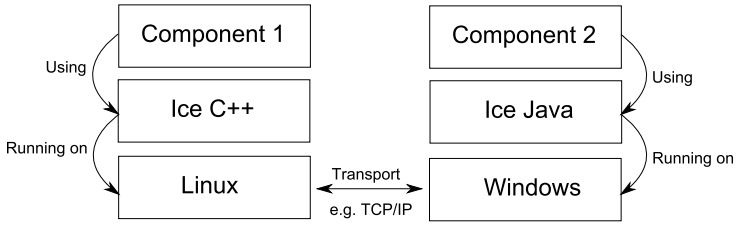


Figure 3.1: Schematic overview of communication in Orca between two components running on different operating systems and written in different programming languages. This figure was adopted from [Makarenko et al. 2006].

Configuration ($C2$) and composition ($C5$) are done through an application server, `IceBox`. A single configuration file defines which components to load with particular configuration details. Component coordination ($C1$) is not explicitly defined but can be added by the component developer and this without being constrained by the framework.

Nevertheless, a typical Orca component mixes computation code such as driver and algorithm implementations with framework dependent code. Such tightly coupled code impedes component reusability.

Furthermore, although Orca was one of the first open source robot frameworks, it seems to have failed to gain the critical mass to attract the attention of modern service robot builders [Bruyninckx 2010].

3.2.3 RT-Middleware

RT-Middleware [Ando et al. 2005] from AIST offers a component framework that supports networked robotic systems. The framework offers the interface for the components and executes the actions defined by the components. They also provide a basic set of reusable components and common robotics libraries as well as a set of development tools. Each component has a state machine defining its activity and communication ports to interact with other components. Open-RTM is the instantiation of RT-Middleware developed at AIST.

Also in this framework, communication ($C4$) between components is separated from other concerns through middleware. There is no constraint on the software decomposition or architecture in Open-RTM. Components can either represent resources, e.g. a sensor or actuator, or functionality, such as a control algorithm or image processing algorithm. An example of a robot running on RT-Components is depicted in Figure 3.2. From top to bottom it shows first functional and structural components composed into larger structural components which finally are composed into one robot component.

Various levels of granularity are supported by Open-RTM. For instance, a mobile platform component could be a composition of several other components. Through introspection components or specialised tools, applications are able to compose components at runtime ($C5$).

Each component has a configuration interface to allow remote calibration of internal properties and parameters, thereby separating the configuration ($C2$) from internal computations. There is, nevertheless, a coupling between computation code and framework specific code ($C3$). Each component has to follow the framework's state machine structure to define its activity.

Coordination ($C1$) is typically done by adding an extra component responsible for the coordination of a set of components. Nevertheless, there is no separate interface to switch a component's behaviour, so coordination has to be done through the same communication channel data is sent by.

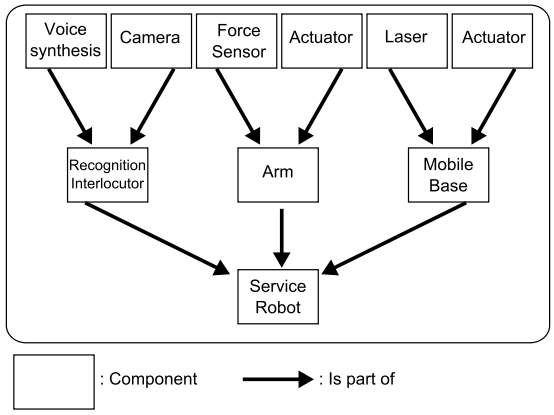


Figure 3.2: Schematic overview of how components are composed in RT-Middleware. From top to bottom it shows first functional and structural components composed into larger structural components which finally are composed into one robot component. In this example a service robot is depicted, consisting of a mobile base, two arms and stereo vision providing recognition. This figure was adopted from [AIST 2012].

3.2.4 ROS

Robot Operating System / Robot Open Source (ROS) [Quigley et al. 2009], originally developed at Stanford but now maintained by Willow Garage, is an open source robot software framework offering libraries and tools to facilitate the creation of robot applications.

One of its main goals is to provide an architecture independent platform with several communication mechanisms. This way ROS is suitable to be used in all kinds of robot software architectures. Also, no constraints with respect to decomposition are imposed. Both purely functional or structural components are feasible. Nevertheless, the focus is on offering functionality rather than reflecting reality.

In ROS, both communication ($C4$) and configuration ($C2$) are separated from computation ($C3$). Computation is divided into processes, called **nodes**, which offer topics to provide publish-and-subscribe communication and services for remote procedure calls. This concept is depicted in Figure 3.3 and decouples production of information from its consumption, since nodes publish their messages to topics and other nodes subscribe to certain topics to receive the messages. Configuration parameters for all nodes are managed by a parameter server.

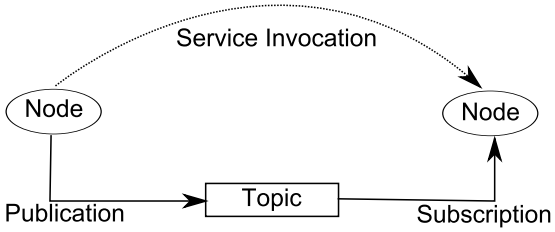


Figure 3.3: Schematic overview of communication in ROS. Nodes, depicted by ovals, can either publish data on topics, which is then read by other nodes subscribed to those topics, or either perform remote procedure calls through services. This figure was adopted from [Willow Garage 2012].

Connections between nodes are managed by a master node and inferred by the publish and subscribe calls during initialisation. A set of nodes providing related functionality can be composed into a **package** and a collection of related packages is bundled in a **stack**. This composition ($C5$) is done at the file system level, though, and no interface is provided to perform compositions at runtime. An example of a ROS package for navigation is shown in Figure 3.4.

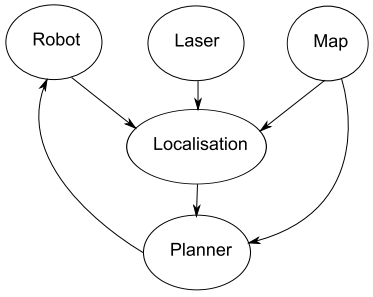


Figure 3.4: Schematic overview of a navigation package in ROS. Nodes are depicted by ovals and the directed arcs indicate a pusblish/subscribe unidirectional communication channel. Adopted from [Quigley et al. 2009].

Finally, coordination ($C1$) between nodes is implicit through messages between those nodes and no separate interface is provided.

3.2.5 OROCOS

OROCOS [Bruyninckx et al. 2003] is an open source framework with a focus on realtime control of robots and machine tools. It also offers libraries for Bayesian

filtering components and kinematics and dynamics components. Moreover, OROCOS makes no assumptions regarding architecture or decomposition.

Control components of this framework provide several interfaces for configuring components and connecting to and communicating with other components. Besides these interfaces it allows to write finite state machines (FSMs) to define the application specific logic.

The structure of an OROCOS component is shown in Figure 3.5, highlighting the separation of configuration ($C2$), computation ($C3$) and communication ($C4$).

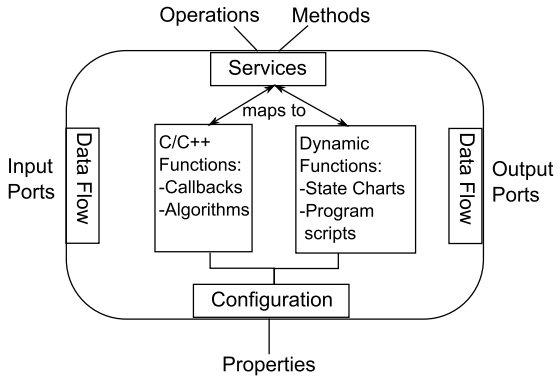


Figure 3.5: Schematic overview of an OROCOS component. It shows the separation of configuration, computation and communication. Adopted from [Intermodalics 2011].

Composition ($C5$) is possible by combining finite state machines in which each state of the top FSM is, in itself a state machine. However, this is not very practical at component level. Though not explicitly part of the framework interface, coordination ($C1$) can be provided through these state machines. For example, iTaSC, a framework for **i**nstantaneous **Ta**sk **S**pecification using **C**onstraints, generates motions by specifying constraints in geometric, dynamic or sensor space between the robots and their environment [De Schutter et al. 2007] and is implemented using OROCOS. In iTaSC, the configuration and coordination of a set of tasks, called a **skill**, is achieved using FSMs.

3.2.6 Microsoft RDS

Microsoft's Robotics Developer Studio (RDS) [Microsoft 2008] is a platform which offers tools and components to create robotics applications. The platform

can be divided into four distinct features: the Concurrency and Coordination Runtime (CCR), Decentralized Software Services (DSS), a Visual Programming Language (VPL) and a Visual Simulation Environment (VSE).

CCR provides support for applications with loosely coupled software components, which can be developed independently without any assumptions with respect to runtime environment and other components. Communication ($C4$) between such components is done by message passing and coordination ($C1$) is possible through the coordination primitives, CCR's API offers.

DSS is built on top of CCR and offers state-oriented services, exposed as resources. Applications are compositions ($C5$) of services regardless where these services are running.

VPL offers an application development environment to graphically create, connect and configure ($C2$) components together. An example data flow, consisting of a connected sequence of activities, is shown in Figure 3.6.

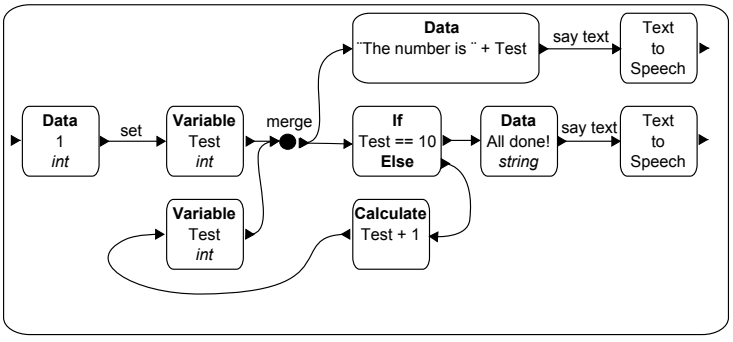


Figure 3.6: Data flow in the Visual Programming Language (VPL) of Microsoft's Robotics Developer Studio. Adopted from [Microsoft 2012].

VPE consists of a simulation engine, responsible for rendering entities, a physics engine and entities representing hardware and physical objects in the simulation world.

Although Microsoft's RDS offers a large set of tools to develop robotics application both for real platforms and simulation, developers are forced to follow Microsoft's service oriented architecture and computation ($C3$) is not clearly separated from middleware code. This limits the reusability of RDS components and the ability to integrate computational code with other frameworks or existing applications.

3.2.7 Player/Stage

Player/Stage [B. Gerkey et al. 2003; Collett et al. 2005] has been widely used in the past by the robotics research community. Although not explicitly designed as a component based framework, it employs many component-based ideas [Makarenko et al. 2006].

Player is a client/server interface which allows to control the robot's sensors and actuators through a simple interface. Various robot hardware is already supported in the current version and the developer is able to create own components for his specific hardware. Nevertheless, this *delineation* of server space from client space implicitly imposes a certain architecture. Inter-client communication is not defined and developers have to add this functionality themselves by porting the client code to server space.

Moreover, there is no clear separation between coordination ($C1$), computational code ($C3$), and communication middleware ($C4$). Configuration ($C2$) of the application is done through configuration files and separated from other code. Composition ($C5$) of several components, called *devices* in Player, is done by specifying their device-device connections in the configuration file.

Stage acts as a simulation environment to test developed client software. All developed client software for Player can be used in Stage as if it would be connecting to the real robots.

3.3 Component-based robot software architectures

Many robot architectures in the past are structured in three layers [Arkin et al. 1997; Gat et al. 1997; Hexmoor et al. 1993]. A deliberative planning layer is separated from a reactive layer by an intermediate executive layer. Functionality is then decomposed over these layers. Also more recent architectures adopt this layered approach or an adaptation on it. A few of these architectures are described in the following paragraphs.

3.3.1 CLARAty

In CLARAty [Nesnas et al. 2006] the decisional and executive layers are merged into a single layer which communicates with a lower functional layer. It distinguishes itself from other architectures by allowing access to robotic functionality at different levels of abstraction. For instance, a move command could be directed at a motor, mobile robot, or a whole team of robots.

This two-layer architecture focuses also on flexibility with respect to robotic systems to be able to cope with differences in physical capabilities, sensor configuration and hardware control architecture. The decision layer defines activities in a declarative way and at runtime a feasible plan is constructed based on system and mission constraints found in those activities. The functional layer employs multi-level abstraction to be able to offer platform independent functionality at a higher level and swappable platform specific algorithms at a lower level.

In Figure 3.7 the CLARAty architecture is depicted and applied to an rover exploration example.

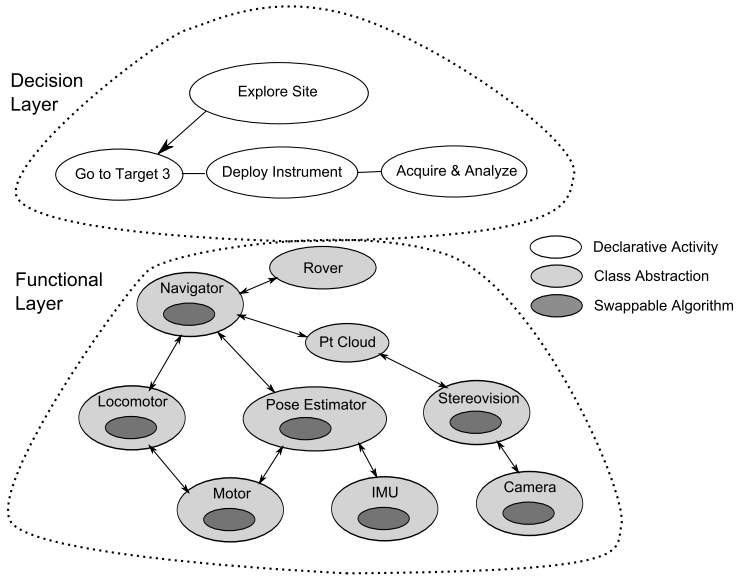


Figure 3.7: The CLARAty Architecture. Adopted from [Nesnas et al. 2006].

3.3.2 LAAS

The LAAS architecture [Alami et al. 1998; Py et al. 2004; Bensalem et al. 2009] has been designed for mobile robotics and tested on several mobile platforms. Its main goals are robustness with respect to control, extensibility regarding new functionality and new tasks and autonomy and adaptability of the robots according to perceived environment.

The architecture is decomposed into three levels with different temporal constraints: a decisional level to produce task plans and supervision, an execution control level checking requests sent from the decision level and maintaining resource usage and a functional level containing basic robot action and perception capabilities. The functional level communicates with the actual physical system, such as sensors and actuators, through the logical system, which makes abstraction from the specific hardware. Figure 3.8 shows the LAAS architecture.

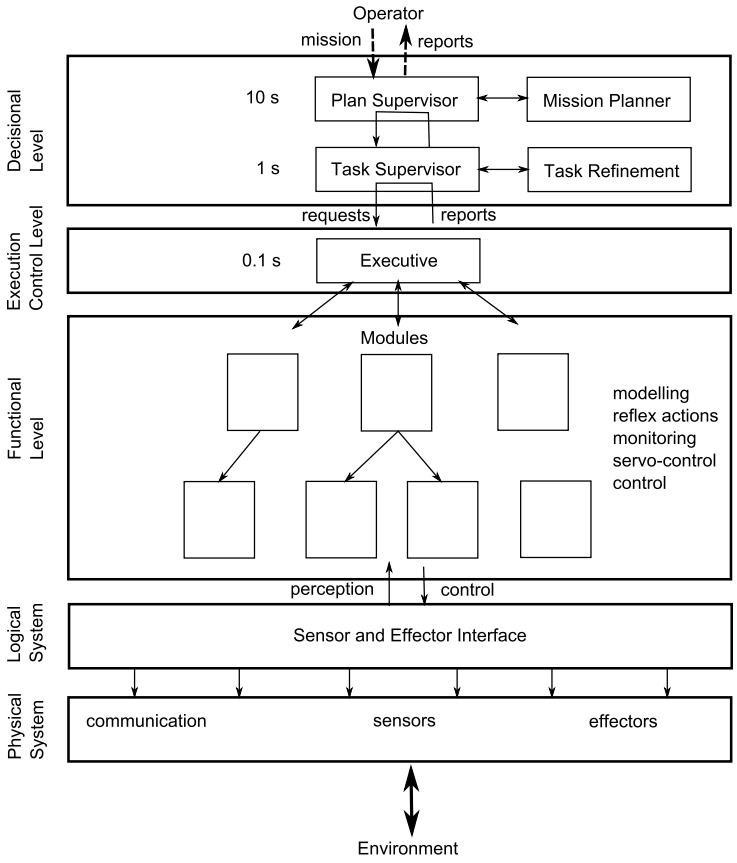


Figure 3.8: The LAAS Architecture. Adopted from [Alami et al. 1998].

3.3.3 JAUS

The Joint Architecture for Unmanned Systems (JAUS) [JAUS 2007] was initially developed by the United States Department of Defense and aims at developing an open architecture with standardised interfaces for unmanned systems. It defines an hierarchical system built up out of subsystems, nodes and components.

A subsystem represents a physical entity in the network, e.g. an unmanned vehicle or operator control unit. Subsystems are divided into nodes, which represent any computing device. These nodes contain a set of components, which typically represent applications offering services.

In Figure 3.9 the JAUS system communication topology is shown in which this hierarchy is also visible. Each node typically has a node manager, which is a component providing universal functionality such as routing of messages within and between nodes. Next to this node manager, the communicator component provides similar functionality between subsystems.

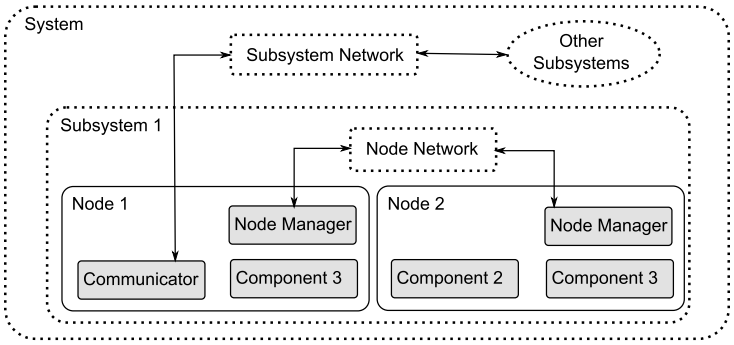


Figure 3.9: Schematic overview of the JAUS system communication topology. JAUS is a hierarchical architecture consisting from top to bottom out of subsystems, nodes and components. Each node typically has a node manager, which is a component providing universal functionality such as routing of messages within and between nodes. Next to this node manager, the communicator component provides similar functionality between subsystems. Adopted from [JAUS 2007].

3.3.4 Mobile-R

Mobile-R [Nestinger et al. 2011] is a reconfigurable platform for deployment of multi-robot systems. It supports both mobile and stationary robot systems for

both physical and virtual robots. These systems are represented by multi-agent systems, with *mobile agents*.

A mobile agent can be deployed at runtime to any robot system in the multi-robot network and is able to transport itself from one system in the network to another. This way data produced by resources in such a network of systems can be shared. For example, a camera system is shared with robots by sending a mobile agent containing code on how to use the camera over to the camera system and after execution sending this agent back to the requesting robot.

Mobile-R identifies several types of mobile agents to accommodate for various robot architectures, such as purely reactive, purely deliberative or hybrid architectures. A deliberative agent represents the deliberative or executive layer of the robot system, a reactive agent represents a low-level reactive robot controller, a hardware agent can represent a sensor or actuator or communication hardware, a task agent provides task allocation and sharing, a behaviour agent is similar to a task agent but strictly deal with the reactive layer, a device agent allow hardware reconfigurations by installing or removing hardware agents, and a service discovery agent keeps track of the services a robot is able to provide other robots in the network [Nestinger et al. 2011].

All agents types can be combined which allows the developer to create a particular robot architecture suited for his application. Figure 3.10 shows a heterogeneous multi-robot system using different mobile agent types. In this example a purely deliberative, purely reactive, hybrid and purely tele-operated robot system are combined in a multi-robot network. The deliberative agents share high level task information, while the reactive agents share behaviour data and the hardware agents share device specific data, all indicated by the dashed lines.

3.4 Holonic systems

3.4.1 Introduction

As introduced in chapter 2, holonic systems are defined as a set of holons that besides functioning as separate wholes and together as a greater whole also function in coordination with their local environment.

Although software engineering in robotics has received increased interest lately, **resource allocation** seems to have been neglected. Some frameworks are open enough to support resource allocation but do not provide it yet. It is left to the developer to include this functionality. Most reviewed robot architectures

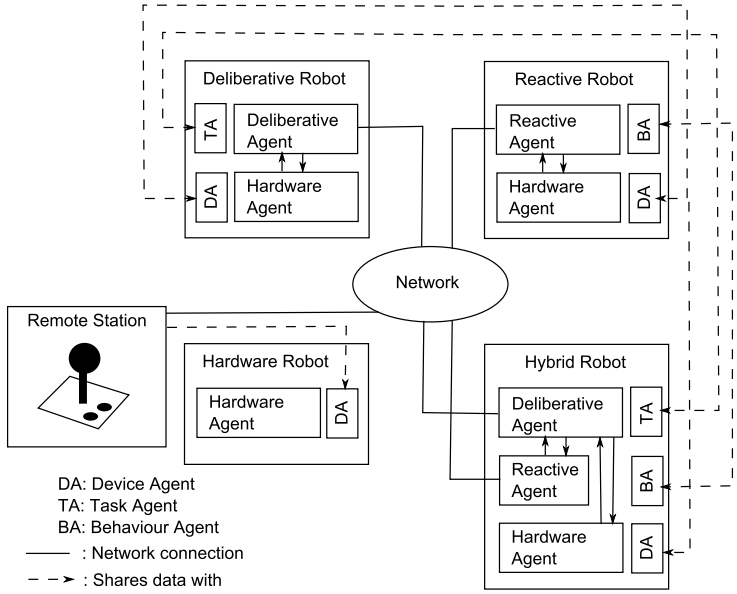


Figure 3.10: Schematic overview of a heterogeneous multi-robot system using different mobile agent types provided by the Mobile-R architecture. In this example a purely deliberative, purely reactive, hybrid and purely tele-operated robot system are combined in a multi-robot network. The deliberative agents share high level task information, while the reactive agents share behaviour data and the hardware agents share device specific data, all indicated by the dashed lines. Adopted from [Nestinger 2012].

have no notion of resource allocation and assume during task execution that all necessary robot resources are implicitly allocated and usable. This is in contrast to most holonic systems for manufacturing which do take resources and their allocation into account.

Also, most robot software architectures have a hierarchical design. Previous work in manufacturing control systems has shown this limits scalability and flexibility in case of unforeseen changes in requirements [Monostori et al. 2006].

Hierarchical systems are more difficult to modify and incorporate unforeseen changes, due to their static and deterministic nature. Moreover, the reactivity or resulting response time to disturbances is lower, since the detection of this disturbance has to be fed back to the appropriate level in the hierarchy [Bongaerts et al. 2000].

Holonic systems have proved to be successful in manufacturing (cf. also the applicability of PROSA in chapter 2). Therefore, in this section, state of the art in holonic **manufacturing** systems is discussed.

3.4.2 Magenta

In [Skobelev 2011], an overview of resource allocation and scheduling applications in logistics is provided. It discusses numerous logistics applications, such as truck scheduling, car rental scheduling and taxi scheduling, the company Magenta has deployed and tested using their architecture with the same name.

Their approach is bio-inspired and based on Demand-Resource Networks and self-organisation of agents in a virtual market. Demand-Resource Networks are created between agents demanding a certain service and agents offering that service. These agents negotiate with each other on a virtual market. Whenever a solution is found, relevant links between the agents are made. The links are not permanent however. The agents continue to search for an optimal solution and if, for instance, their current configuration fails, they are able to break the links and reorganise.

This approach can cope with unexpected situations. Whenever a disturbance occurs, the affected agents will query the market for new solutions and a new equilibrium will be established. Since agents are allowed to change the goals of other agents, a chain of events could occur.

Furthermore, this approach is based on finding an equilibrium between demand agents and resource agents in real-time and uses no short-term forecast. The applicability of this virtual market approach is limited and functions best in pure allocation problems. The logistics applications described in [Skobelev2011] approximate such allocation problems well.

3.4.3 ADACOR

The Adaptive Holonic Control Architecture for Distributed Manufacturing Systems (ADACOR) [Leitão et al. 2005; Leitão et al. 2006] is a holonic control architecture in which 4 distinctive holon classes can be identified: product holon, task holon, operational holon and supervisor holon. This architecture is based on PROSA and, thus, the behaviour and role in the system of the different holons resemble well those of respectively the product, order, resource and staff holon in PROSA.

The system succeeds in reconfiguring itself by dynamic holon reaggregation, in which holons aggregate to new larger holons at runtime. Self-organisation is achieved by the adoption of a bio-inspired pheromone spreading mechanism. Pheromone intensity is proportional to the need for holons to reconfigure and adapt to the changes or forecasted disturbances. An increase of pheromone near a holon triggers self-adaptation of its behaviour and propagation, in decreasing intensity, of these pheromones to its neighbours.

3.4.4 iCoMAS

The hierarchical multi-agent system iCoMAS [Sprenger et al. 2011] is developed to solve vehicle routing problems for cooperative transportation planning in food industry. It combines decision-making agents in one layer and staff agents, based on staff holons in PROSA, in another layer. A blackboard is used by the agents to share information.

3.4.5 Tactical-AgentFly 2

In [Vokřínek et al. 2011] Tactical-AgentFly 2 (TAF2) is described for multi-agent control of unmanned aerial vehicle (UAV) operations. TAF2 is an experimental multi-agent simulation system in which a group of agents is supplied with a ISTAR task (Intelligence, Surveillance, Target Acquisition and Reconnaissance). Three agent types are identified: task agents solving problem preprocessing and generating tasks for other agents; allocation agents decomposing tasks and delegating these subtasks to resource agents; resource agents performing individual resource planning.

3.5 Multi-robot path planning & coordination

The application scenario described in section 1.4 requires both multi-robot path planning and coordination to ensure that robotic wheelchairs navigate safely from one location to another and that mutual interference is minimal. Moreover, the adopted approach should be scalable with respect to the number of robots and environment complexity. The approach should be able to cope with uncertainty and unexpected disturbances.

In the following paragraphs, the state of the art in multi-robot planning and coordination is discussed with the application scenario of section 1.4 and its requirements in mind.

3.5.1 Planning approaches

In such a multi-robot navigation scenario, a multi-robot path planning approach is typically used to ensure no collisions or livelocks occur. Multi-robot path planning can be split up in two categories [Bennewitz et al. 2002]: *centralised* and *decoupled* planning.

Centralised

In centralised multi-robot path planning, configuration spaces of individual robots are combined to one configuration space. The size of such a combined space increases exponentially with the number of robots. Therefore, most practical implementations use heuristics to narrow down the search space.

Decoupled

Decoupled multi-robot path planning calculates first the individual paths of each robot and then tries to combine them resolving conflicts. Commonly, priorities are assigned to robots and paths for the robots are planned accordingly. Lower priority robots take into account the already planned paths of higher priority robots. Although this solution is computationally efficient, it does not provide an optimal solution. Deadlocks can occur and are related to which priority scheme is used. For example, such a deadlock occurs when the path of a high priority robot blocks the path of a low priority robot. Figure 3.11 shows a simple environment and priority scheme resulting in such a deadlock.

Another commonly used decoupled planning technique is the path coordination method in which each robot is constrained to one, fixed-path coordination, or more, fixed-roadmap coordination, paths. Some literature on multi-robot path planning provides scalable solutions, e.g. [Peasgood et al. 2008; van den Berg et al. 2009].

A more recent approach uses subdimensional expansion in which not the full configuration space but a low dimensional subspace is considered [Wagner et al. 2011]. It assumes the set of robots are loosely coupled, i.e. planning in a joint configuration space in not necessary. Indeed, in most cases, the robots are spread around in the environment. Planning is then first done for each individual robot and the dimensionality of search space is locally augmented if collisions occur.

Nevertheless, these solutions usually utilise a central planner with complete knowledge of the environment and all robots moving in it. Such a *closed world assumption* limits system extendibility. Instead, decomposing the planning over

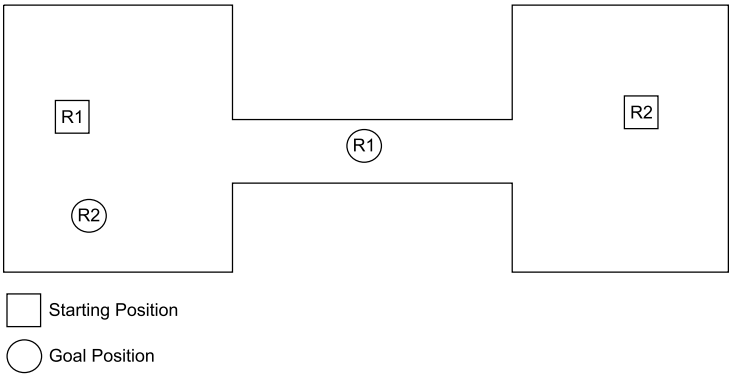


Figure 3.11: Deadlocks can occur and are related to which priority scheme is used. Here such a deadlock occurs since the target position of high priority robot R1 leaves low priority robot R2 unable to find a path to its target position.

the entities in the environment (e.g. rooms, corridors) results in a scalable solution in which only relevant local knowledge (e.g. which robot will pass in this room at what time) is kept at the relevant entities.

3.5.2 Multi-robot coordination

Also coordination is generally divided in two categories based on *where* this coordination is being done: **centralised** at one entity (robot, computer,...) or **distributed** among several agents.

Centralised

In a centralised approach the multiple robots are coordinated by a central computer or one of the robots is assigned as team leader. All degrees of freedom are then combined to offer an optimal solution to the specified task.

[Brummit et al. 1998] introduced the Generalized Robotic Autonomous Mobile Mission Planning System (GRAMMPS) which is an example of a centralised multi-robot coordination approach. It is a two-layer architecture, of which the top layer runs on a single robot, the coordinator of the team, and the second layer runs on all robots.

Although good at responding to uncertainty and dynamic environments, this centralised system suffers from most other problems a centralised coordination

approach suffers from. Most problems are similar to problems in a centralised path planning approach:

Firstly, this approach becomes computationally difficult as complexity is exponential with respect to the number of robots involved.

Secondly, it assumes that all sensor and state information of all robots can be collected by the single coordinator and that this information does not change during planning. In unknown or dynamic environments, where communication is limited, these assumptions are unrealistic.

Thirdly, if the leader or coordinator malfunctions, a new leader must be available or the entire task fails.

Distributed

Distributed coordination addresses the above described problems centralised approaches suffer from. Each robot acts on locally available information. Coordination is distributed among all robots and, for instance, a robot can coordinate with other robots nearby to solve a sub-task it cannot solve alone.

Typically, computational complexity is alleviated since each robot plans its own activities. Moreover, communication is also limited and robots are able to respond to unknown or dynamic environments better.

The downside of this approach is the inability to provide optimal solutions for the global task. Also, the approach works best if problems can be decomposed into independent sub-problems.

In past research, most distributed coordination approaches use auction and market techniques to divide tasks among robots, e.g. [B. P. Gerkey et al. 2002] and [Dias et al. 2000].

In [Dias et al. 2000], rules of free market economy are adopted to coordinate multiple robots. In a free market, individuals are free to exchange goods and services and enter into contracts as they see fit. A robot receives revenue by accomplishing a global team task but also by offering a particular service to another robot. For instance, one robot might require services from another robot to accomplish a certain sub-task. The price a robot pays for a service tends toward the intersection of the supply and demand curves for that service.

In [B. P. Gerkey et al. 2002] an auction-based task allocation system, MURDOCH, is described. A new task is announced by an auctioneer agent, containing all the required task details, and this announcement is sent only to those subjects which represent the resources required to execute the task.

This way, only robots currently capable of performing the task will receive the announcement. Each robot then calculates how appropriate it is to perform the requested task and this score is used as a bid and determines which robot will win the auction.

3.5.3 Other related approaches

Multi-agent systems

Although multi-agent systems are often used for multi-robot applications, the research mostly focusses on task allocation problems or on emergent behaviour in large groups of small robots, called *swarms*.

Task allocation problems are often addressed using auction-based techniques in which agents bid for particular tasks and the winner of the auction is assigned that task. The criteria determining the agent's bid are most of the time application specific. More on this topic can be found in [Dias et al. 2006; B. P. Gerkey et al. 2002; Zhang et al. 2010].

Swarm robotics systems are characterised by a very large set of small robots performing jointly a task. Each robot is represented by an agent with a set of basic rules to follow and how to react to changes in the environment or in the behaviour of its neighbours. These rules and behaviours are usually bio-inspired such as the ant colony optimisation [Dorigo et al. 2004]. Out of these local actions of individual agents a global behaviour of the group emerges. More information can be found on the SWARM-BOTS [Mondada et al. 2002; Dorigo 2001] and Swarmanoid [Dorigo 2006] project websites.

Intelligent transportation systems

Also research in intelligent transportation systems provides solutions usable in multi-robot path planning. E.g. in [Hausknecht et al. 2011] autonomous intersection management is proposed which employs a grid based collision detection and accepts or rejects reservation requests based on spatial and temporal constraints. However, no start to finish allocations of the roads and intersections is possible and no short term forecasts on these reservations are provided, in contrast to the PROSA and D-MAS system.

In [Olmi et al. 2011] a control strategy for traffic coordination of Automated Guided Vehicles (AGVs) is proposed. The AGVs follow predefined paths in a dynamic environment and a mapping is defined between a set of motion constraints and the configuration space of the fleet. Although motion

actions are determined in polynomial time using a heuristic algorithm, the centralised planning and coordination component assumed full knowledge of the environment and the AGVs and their movements are limited by predefined paths.

3.6 Why PROSA and D-MAS?

With a plethora of existing robot frameworks and architectures and coordination approaches, the question rises why it would be beneficial or worth while to investigate the applicability of PROSA and D-MAS in robotics. More specifically, why adopt PROSA and D-MAS techniques in the coordination of multi-robot navigation tasks.

Chapter 2 already provided, in depth, motivations why holonic systems in general and PROSA in specific are important to robotics. The following paragraphs will shortly describe specific considerations which convinced the author, and hopefully also some of the readers, the adoption of PROSA and D-MAS in robotics is justified.

3.6.1 Complementary to existing approaches

It is extremely important to note that the paradigm presented in this dissertation has not the intention to replace existing state of the art. No, it offers an infrastructure in which state of the art of different robotics research domains can be integrated and reused.

Moreover, both on the software level as the hardware level integration and reuse are possible. E.g., robots running legacy software can be taken into account and thus integrated with robots employing the PROSA and D-MAS paradigm.

How this integration is done will be shown in the chapters discussing simulation and real-world experiments, namely chapter 6 and chapter 7.

The main reason why the HMES or another PROSA and D-MAS implementation is complementary to existing state of the art is their level of operation or level of abstraction.

Indeed, the granularity level at which to adopt PROSA concepts influences the reusability of previous developments. In other words, the fine-grained holons are less likely to be reused in a different applications domain than coarse-grained holons.

For example, in a coarse grained approach a robotic wheelchair with obstacle avoidance software is represented as one resource holon capable of safe navigation and used as is, while in a fine grained approach the obstacle avoidance software is reimplemented by for instance dividing the environment in small resource holons and allocating these holons required to navigate from one location to the other.

As will become clear, in most cases the first approach is preferred, since this makes most use of existing research.

3.6.2 Previous applications

As was shown in chapter 2, the Holonic Manufacturing Execution System (HMES), the PROSA implementation developed at PMA/KU Leuven, has been used in a variety of applications ranging from manufacturing control to open-air engineering and logistics (cfr. section 2.7).

Recently, the HMES was used for the coordination of harvesters and tractors on large agricultural fields [Ali 2010].

Both this last application and other previous successful applications encouraged the idea of adopting PROSA in robotics. In order to investigate if the PROSA concepts, which have a clear manufacturing control background, could be *translated* into the robotics domain, a multi-robot navigation application was chosen. Nevertheless, the general applicability of PROSA in other robotics applications was kept in mind.

The coordination of multi-robot navigation resembles, to some extent, the harvester application. Of course, there are a number of key differences. First and foremost, the environment in which the robotic wheelchairs operate is completely different from the fields on which the harvesters and tractors drive. Moreover, in the robotics scenario, both autonomous control and shared control between the robot and its user are desired, while in the harvester application, the harvesters and tractors were manually driven and users received their instruction through a monitor.

3.6.3 System complexity

In section 2.2 five levels of system complexity were introduced, based on the complexity levels defined in [Bruyninckx 2010]. The PROSA and D-MAS system solving the multi-robot navigation scenario investigated in this dissertation is categorised as of a $S4$-level complexity.

In contrast, most existing robot systems provide $S3$-level complexity, mainly due to their functional design and lack of reflection of reality.

In this dissertation a *paradigm shift* towards $S4$-level systems is proposed by introducing holonics in the robotics research domain. This paradigm shift is also one of main motivation points to choose PROSA and D-MAS above existing research in the field of multi-robot coordination.

Besides the proof of concept nature of this research, this dissertation provides a starting point for a novel research focus in robotics in which reflection of reality and resource allocation are the major differences with respect to existing robot systems.

3.6.4 Reflection of reality

The review of existing robot software architectures identified a lack of reflection of reality, since most robot software architectures adopt a functional decomposition.

The physical entities required in the application, e.g. robot resources or objects in the environment, are assumed available and no allocation is performed. Moreover, each component interacting with a particular entity has its own representation and maintains its own version of that entity.

As discussed earlier, see e.g. sections 1.2 and 2.4.3, such a *robocentric* approach has difficulties with external disturbances and changes in the world-of-interest. Especially in multi-robot applications and dynamic environments this problem becomes clear. For instance, in our application scenario, a mobile robot driving into a narrow corridor might block other *users* of that corridor even if they had entered first.

Reflecting reality is one of the key features of the PROSA architecture. Combined with short-term forecasting, offered by D-MAS, this has led in previous manufacturing applications to an increase in overall system robustness with respect to unexpected changes and disturbances.

Another benefit of reflection of reality and consequently the structural decomposition of software components is scalability. In the multi-robot navigation scenario this can be defined with respect to the number of robots in the system as well as the size of the environment, e.g. the number of rooms and corridors.

Typically, in multi-robot navigation or trajectory planning the computational complexity in time grows exponentially with the number of robots, since the dimensionality of the search problem increases. Many techniques exist already

with lower computational complexity but less optimal solutions or increased deadlock chances, as was discussed in section 3.5.

In PROSA this multi-robot planning problem is split up and divided among all the environment resources, which each are responsible for their own part in reality. Adding a robot would then only influence the planning required in the relevant rooms and corridors this robot is travelling.

As mentioned before, the intention is not to replace the existing state of the art research in trajectory planning. Rather these state of the art techniques can be used locally in each environment resource, thus simplifying the global planning problem.

3.6.5 Short-term forecasts

Finally, the use of D-MAS in PROSA offers short-term forecasts by querying resources on their allocations in the near future. These predictions are useful to anticipate on upcoming problems rather than react to them when it is too late.

For example, when two or more robots enter the same narrow corridor from opposite sides, this will lead to a situation in which at least one robot will have to abandon its current trajectory and return from where it came.

By continuously refreshing intentions and checking resource allocations, in this case resources are rooms and corridors, the D-MAS are able to predict the example situation, mentioned above, and adapt accordingly beforehand.

3.6.6 Chosen software approach

Given the above motivation, the decision was made to use the existing PROSA implementation in manufacturing control, i.e. the Holonic Manufacturing Execution System or HMES, in conjunction with the existing robot control software developed for the robotic wheelchairs in the KU Leuven/PMA lab.

Choosing not to start anew and develop a robot framework taking all previously indicated problems, current frameworks suffer from, into account, allows to focus more on the conceptual mapping of the PROSA architecture from manufacturing to robotics. Also, reusing the existing robot control software allows to integrate the state of the art research on robot wheelchair shared control into multi-robot coordination experiments.

3.7 Conclusion

This chapter provided an overview of related work and state of the art research in different research domains touched in this dissertations.

Firstly, common software terminology was defined to clarify the variety of research in robot software architectures and robot software frameworks.

Next, existing holonic systems, which mainly focus on manufacturing control, were reviewed since the aim of this dissertation is to introduce holonics into robotics research.

In the light of the proposed application scenario, an overview of the state of the art in multi-robot path planning and coordination is given and drawbacks of current approaches are identified.

Finally, keeping this related work and state-of-the-art research in mind, the choice of PROSA and D-MAS to solve the multi-robot navigation scenario was motivated.

Summary:

This dissertation proposes a paradigm shift towards $S4$-level and $S5$-level systems. Holonic systems, in general, and PROSA and D-MAS, in particular, provide valid means to solve the multi-robot navigation scenario in a scalable and robust manner. The key differences with existing approaches in robotics are *reflection of reality* and *explicit resource allocation*.

Chapter 4

Holonic Robot Applications

> *There are two ways of constructing a software design: One way is to make it so simple that there are* ***obviously*** *no deficiencies, and the other way is to make it so complicated that there are no* ***obvious*** *deficiencies. The first method is far more difficult. It demands the same skill, devotion, insight, and even inspiration as the discovery of the simple physical laws which underlie the complex phenomena of nature.*

Sir Charles Antony Richard Hoare (⋆1934–)
The Emperor's Old Clothes, Turing Award Lecture, 1980

Parts of this chapter have been published in:

J. Philips et al. [2011]. "PROSA and Delegate MAS in Robotics". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V. Marík et al. Vol. 6867. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 195–204

4.1 Introduction

While chapter 2 contains an introduction to holonic systems in general and an overview of PROSA and D-MAS in specific, this chapter will describe the Holonic Manufacturing Execution System (HMES) in more detail. This PROSA

and D-MAS implementation was used in the multi-robot navigation experiments. The chapter will focus on the changes that were made to support multi-robot applications.

Next to the HMES structure and its integration with robot software, concrete examples of robotics applications are elaborated following the (holonic) guidelines set forth in chapter 2. The experiments both in simulation and in the lab using the robotic wheelchairs show one of these examples, namely the application scenario introduced in section 1.4. The results of these experiments are discussed in chapters 6 and 7 respectively.

4.2 HMES structure

In chapter 2, previous applications using the HMES developed at PMA/KU Leuven already showed the benefits of this holonic system. Here we will highlight the structure and some implementation details of the HMES, focusing on the components that were used to implement the multi-robot navigation scenario. The notations and figures in subsequent paragraphs follow those used in [Saint Germain 2010] and [Verstraete 2009], which provide a complete overview of the HMES.

In general, the HMES maps the holonic concepts from PROSA onto a multi-agent system and uses the D-MAS pattern to coordinate the intentions of these PROSA holons.

4.2.1 Multi-models

To model the different components in the HMES and the relevant part of the world-of-interest, **Petri net controlled multi-models** are used. A Petri net, see e.g. [Petri 1966] and [Petri et al. 2008], consists of several transitions, places and connections, usually denoted respectively by bars, circles and directed arcs. Each transition represents a discrete event that may occur, places signify conditions and connections describe which places are pre- and/or post-conditions for which transitions. Places may contain zero or more *tokens*, which a transition requires to *fire*. If all input arcs of a transition contain their required amount of tokens, the transition fires and all tokens are removed. The transition then places tokens at the end of all its output arcs.

In Figure 4.1 a Petri net is shown which models, simplified here, an event cycle of a particular agent. After initialisation, the agent waits for incoming events, and, whenever an event arrives, handles this event.

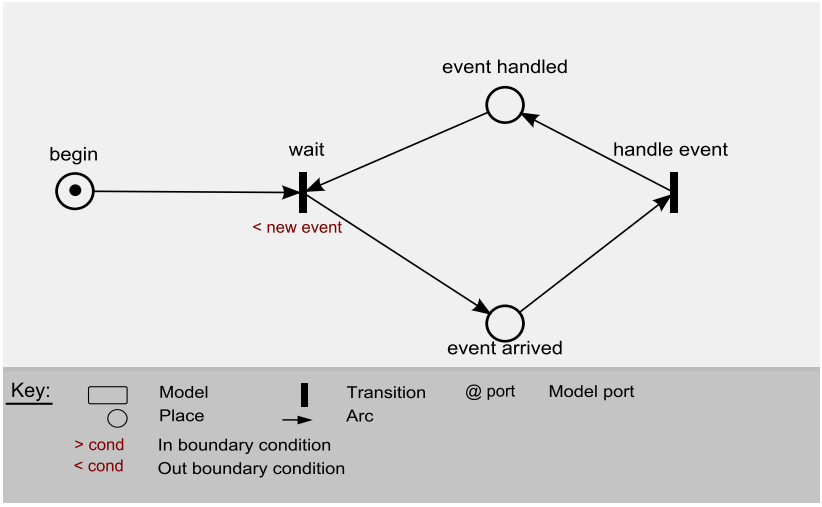


Figure 4.1: Example of a Petri net modelling part of a life cycle of an agent, simplified for demonstration purposes. Each transition has an underlying model, which in turn can be a Petri net or an actual implementation model.

This example already shows a Petri net augmented to support the multi-model formalism adopted in the HMES. Each transition can have **boundary conditions** either at its input or output. The model, represented by this transition, is active if all in-boundary conditions are satisfied and finished if all out boundary conditions are met. The model can be on its own a Petri net or an actual implementation model.

In the HMES, each agent is represented by at least one Petri net consisting of a set of transitions, places and directed arcs. Its transitions are either Java models or Petri nets. In Figure 4.2 the life cycle of an agent is shown and modelled by a multi-model Petri net. The upper part presents the main model, while the lower part denotes the submodel for the `run`-transition, which contains a life cycle submodel, specific to each agent.

4.2.2 HMES-in-the-loop Simulation

In order to test the HMES, a simulation environment was developed which allows software-in-the-loop simulation. More concretely, the HMES controller running in simulation is the same one as the one being used in real applications.

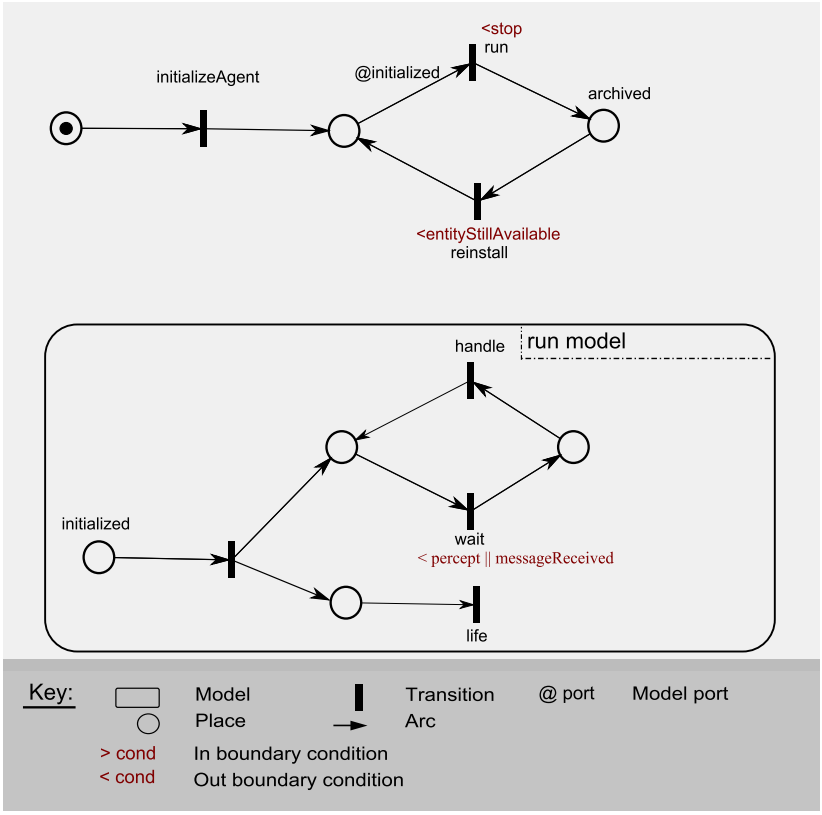


Figure 4.2: This Petri net models the life cycle of an agent in the HMES, from [Saint Germain 2010]. The life submodel is specific to each agent.

During simulation the HMES communicates with an emulator simulating the multi-models representing the world-of-interest.

Running the actual HMES controller *in-the-loop* during simulation requires an hybrid emulator able to switch between a discrete time space and a continuous time space.

Imagine that each discrete step in emulation represents 3 minutes in reality and takes 3ms to execute. If a control algorithm would schedule these 3 minutes but needs at least 15ms to execute, it would not be able to execute completely, since the discrete step is too short.

Therefore, synchronisation between discrete-event mode and real-time mode is required in order to run simulations with a controller in the loop.

To resolve this problem, the emulator maintains a list of running control algorithms at a particular time. Whenever there is one or more control algorithms performing executions emulation maintains a continuous-time space, else it switches to a discrete-event mode.

4.3 Integrating the HMES with existing robot software

4.3.1 Level of integration

The adoption of holonic systems in robotics can, per definition, be done at different granularity levels. As already indicated in section 2.4.3 and guideline 1 of chapter 2, the desired level depends mostly on the application.

For example, in a multi-robot navigation application, it might be sufficient to allocate each robot as a single resource instead of defining resources which represent parts of that robot. More important in such application is the representation of the environment by resources.

The level of integration of the HMES with the already existing software on our robotic platforms corresponds to the granularity level defined by the application. Indeed, integrating a robotic platform and its on-board algorithms as a single holon is less *invasive* than defining holons for all relevant parts of that robot.

Such a *single-holon approach* has some advantages with respect to reusability of existing robotics expertise. It only requires a communication interface between the HMES and the robot software and only minor changes, if any, to previous developments are required in order to be reused in the multi-robot application. This way the coordination by the HMES is transparent to the software running on each of the robots.

The disadvantage of such a coarse-grained representation is that individual robot parts, such as sensors or actuators, cannot be allocated separately. This shows, evidently, that the applicability of the holonic approach is limited by the level of integration in the implementation. Nevertheless, it also shows holonic systems are well suited for *partial integration* due to their structural decomposition.

4.3.2 Communication interface

In this thesis, the development of a communication interface was chosen to keep both the holonic implementation, developed in JAVA, and the existing robot software on the robotic wheelchair, developed in C^{++}, clearly separated. This keeps the amount of changes to the robot software to a minimum. Communication was set up using TCP/IP sockets and protocol buffers were used to wrap the messages and send the data in an efficient way. This communication protocol is further discussed in appendix C.

At the robot software side, the same communication interface was used to send data to and receive commands from the graphical user interfaces (cf. appendix B). At the HMES side, an emulation model of the *robot world*, i.e. a model containing state information of the environment and the available robots in that environment, was developed. This model links the other HMES models and PROSA and D-MAS software with the communication interface.

In Figure 4.3 an overview is depicted on how different *actors* in a multi-robot application utilise this communication interface. In this example, two robots are connected to the HMES through the communication interface. Each robot has a separate communication *layer* in which events from the controllers, actuators and sensors are gathered and wrapped in message types specified by a common communication protocol. These messages are then sent over the TCP/IP socket to all connected clients. The HMES is augmented with a similar communication interface which makes connections to the robots and receives the relevant messages. These messages are then converted to events the robot world component interprets. The robot world, in turn, sends specific events to the HMES based on the data received from the robots. Analogously, this communication channel is also used to send navigation commands from the HMES back to the robots.

More detailed information on the actual implementation can be found in appendix C. The remainder of this section will focus on the robot world model.

4.3.3 Robot world

Basically, the multi-model representing the robot world is similar to the typical life cycle of an agent in the HMES. In Figure 4.4 this multi-model is depicted. Firstly, an initialisation is done in the `startRobotWorld`-transition which loads a map file containing information on the environment, such as relevant rooms and corridors and their geometric properties. This file is also used to determine which resource holons need to be created. Next, the robot world model enters

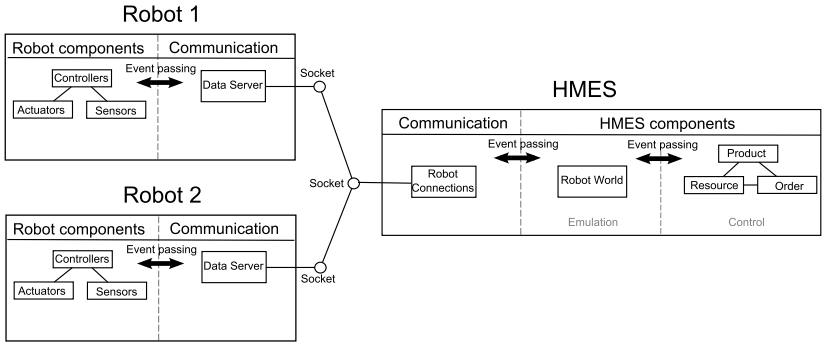


Figure 4.3: This scheme provides an overview of the communication between different *actors* in the multi-robot navigation application.

a loop waiting for events both coming from the HMES and connected robots (through the communication interface).

In Figure 4.5 a more elaborated version of a multi-model for the robot world is shown. Instead of creating only one underlying model to handle events, a specific model is created for each individual event. The advantage is the underlying models become less complex and the Petri net becomes more expressive, showing which events are supported. Nevertheless, each new event requires a change to this model.

Following section discusses the use of PROSA & D-MAS in single and multi-robot applications and provides some starting points to implementations. Since not all described applications have been developed during this thesis, this section also provides pointers to future work in holonic robotics.

4.4 Applications in general

The holonic concepts and the PROSA & D-MAS implementation are usable for applications designed for a single robot, e.g. its control system or obstacle avoidance on a mobile robot, and applications for multiple robots, e.g. a fleet of mobile robots such as depicted in Figure 4.6.

As mentioned before, this depends mostly on the desired task and availability of existing robot software.

In the following sections, several single and multi-robot applications are discussed. The use of PROSA and D-MAS in these applications is explained and evaluated.

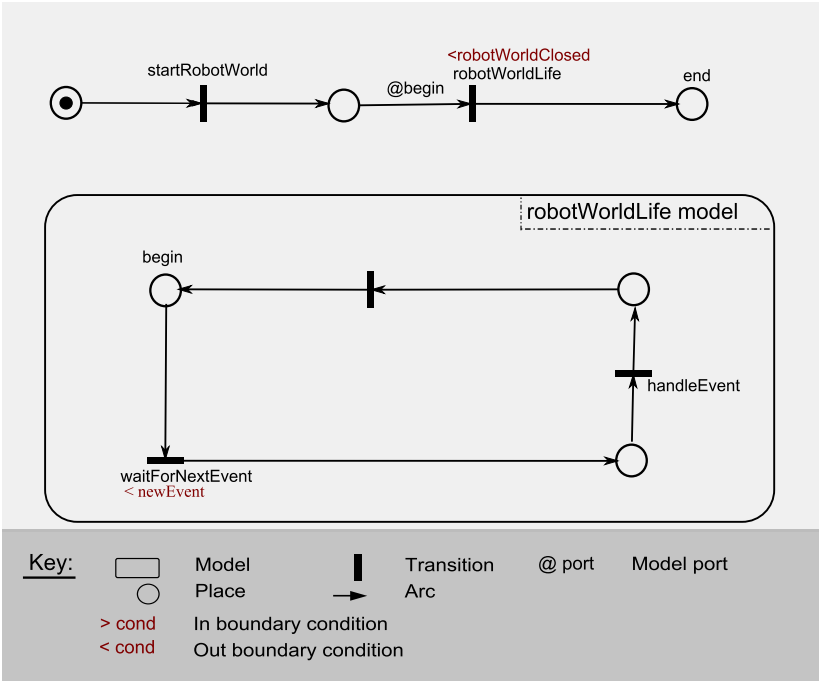


Figure 4.4: This Petri net models the life cycle of the robot world in the HMES. In the life submodel, the handle event model processes incoming events.

This chapter concludes by motivating why which of these applications have been implemented and tested either in a simulation environment or on the robotic wheelchairs depicted in Figure 4.6.

4.5 Use case I: single robot allocation in general

At the level of a single robot, the main components constitute resource holons. Figure 4.7 shows resource holons inside a mobile robot such as the robotic wheelchairs depicted in Figure 4.6. The level of detail is task dependent, since only resources with an active role need to be represented. If e.g. CPU optimisation is required, one of the resources would be the CPU. By doing so, order holons can explicitly allocate CPU time, e.g. to guarantee real-time performance.

The `Robotic Wheelchair` resource has knowledge about its dimensions, its

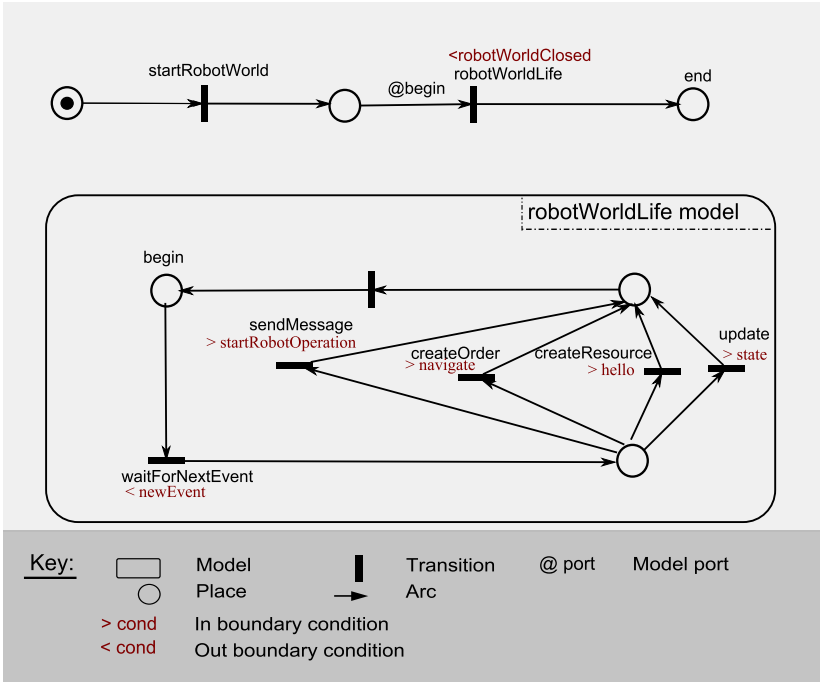


Figure 4.5: This Petri net models the life cycle of the robot world in the HMES. Here, the life submodel contains a model for each possible event.



(a) SARA (b) LAURA (c) LiAS

Figure 4.6: The mobile robot fleet at PMA/KU Leuven, two autonomous wheelchairs and one mobile manipulator.

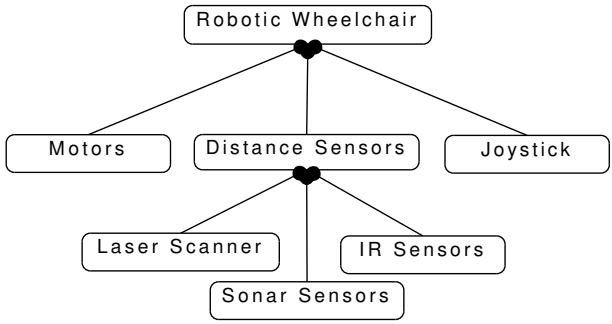


Figure 4.7: Resource holons for a robotic wheelchair.

overall minimal and maximal speed and capabilities such as obstacle avoidance and navigating from one point to another. It consists of `Motors` controlling translational and rotational velocity, `Distance Sensors` scanning for obstacles and a `Joystick` representing the user's input. Typically, several distance sensors are mounted on a mobile robot. In this case, a `Laser Scanner`, `Sonar Sensors` and `IR Sensors` are represented by resource holons. These sensors are used to detect obstacles in the environment and form the input to, for instance, an obstacle avoidance algorithm.

Product holons and order holons can be defined at the single-robot level addressing primitive actions such as moving the robot in a given direction while keeping a safe distance (as measured by the available sensors) from obstacles. PROSA implementations enable to realise such a task as the composition of two more primitive tasks: move to a target position and avoid obstacles. Moreover, a PROSA implementation makes it possible to allocate these embedded resources to tasks from outside the single robot. For example, one robot's laser sensor may be used by another robot. In a way, this provides a bridge between PROSA and behaviour based robot systems. In behaviour based systems each primitive task is represented by a behaviour and the execution of a tasks follows by combining a set of behaviours in a particular way. More information on a behaviour based approach in mobile robotics can be found in [Huang 2011].

Also, in the case of a single robot, it might be opportune to allocate only a subset of its resources. This way multiple order holons can execute their operations simultaneously. Moreover, since resource allocations are temporal, resource sharing between different applications running on the robot is also possible. The difference with classical approaches here is that all the knowledge about the resource, be it a sensor, actuator or a whole platform, is captured in *one* (aggregated) resource holon. As highlighted before, this **single point of truth** maintains a consistent reflection of reality, known by all other actors

through interaction.

4.6 Use case II: obstacle avoidance

4.6.1 Goal

In mobile robotics, the goal of obstacle avoidance is generally to navigate from one location to another while avoiding collisions with obstacles in the environment. These obstacles can be structural or static, such as walls or doors in the environment, but also dynamic such as other robots or humans moving in the same environment.

Often this problem is solved in the control algorithm of the robot in a reactive manner, in contrast to (global) path planning, in which a pre-computed obstacle-free path is followed by the robot. In Figure 4.8, a basic example is depicted of obstacle avoidance when an obstacle is detected while executing a trajectory. The dotted line indicates the original trajectory calculated by a path planner and a given map of the environment (solid lines). The dashed line represents the actually followed trajectory. The robot controller reactively alters the robot's course to avoid the obstacle and afterwards steers back to the original trajectory.

4.6.2 Resource allocation solution

To some extent, the obstacle avoidance problem can be interpreted as a resource allocation problem in which the environment is divided into a set of grid cells and the desired trajectory is represented by the allocation of a sequence of these grid cells. The resource holons representing the environment are, thus, divided into grid cell resource holons which can be individually allocated. The mobile robot is allowed to move along its desired path if it allocates the required sequence of grid cells at the relevant time intervals.

Another robot moving in the same environment will not be able to allocate the same space at the same time and this way collisions between both robots are avoided.

In Figure 4.9 an example is depicted of the obstacle avoidance solution using resource allocation. The checkerboard cells are allocated by the robot during its trajectory. These allocations are temporal and the time interval of each allocation should represent the time when the robot actually occupies this location.

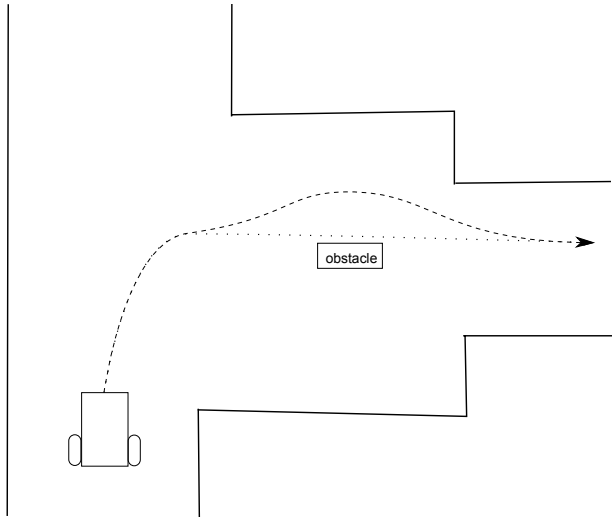


Figure 4.8: Example of obstacle avoidance when an obstacle is detected while executing a trajectory. The dotted line indicates the original trajectory calculated by a path planner and a given map of the environment (solid lines). The dashed line represents the actual followed trajectory. The robot controller reactively alters the robot's course to avoid the obstacle and afterwards steers back to the original trajectory.

Figure 4.10 shows the relevant resources in this use case. The `Wheelchair` resource might be further decomposed as depicted in Figure 4.7.

This solution, however, assumes that all actors in the environment adopt this resource allocation. It does not take into account humans or other robots moving around without allocating their space on beforehand. Another problem might be the difference between allocation time and actual execution time of the trajectory. If the model, used to estimate the time a robot needs to move to its target, is not adequate, robots might deviate from this schedule.

The short-term forecasts provided by D-MAS are able to detect such deviations or, if sensor data are linked to the environment resources, dynamic obstacles can be taken into account. Nevertheless, how fast the robot is able to avoid obstacles depends on how fast a change in the environment is detected. If the robot only relies on its own sensors, forecasting is very limited and there might be not enough time to complete a full exploration and allocation cycle using D-MAS.

If, however, each room is equipped with its own sensors to keep its occupancy

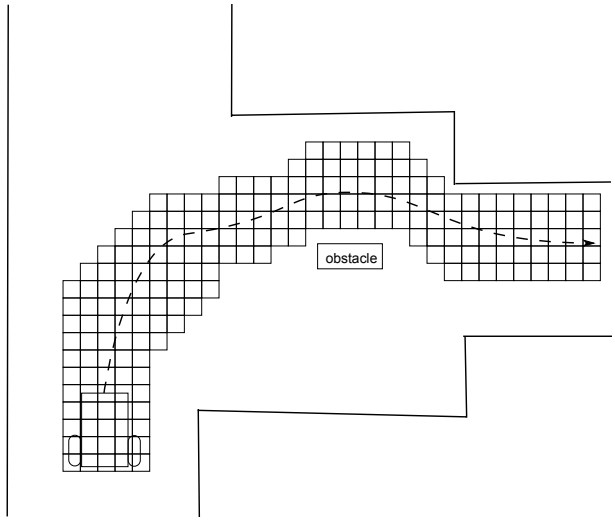


Figure 4.9: Example of the obstacle avoidance solution using resource allocation. The checkerboard cells are allocated by the robot during its trajectory. These allocations are temporal and the time interval of each allocation should represent the time when the robot actually occupies this location.

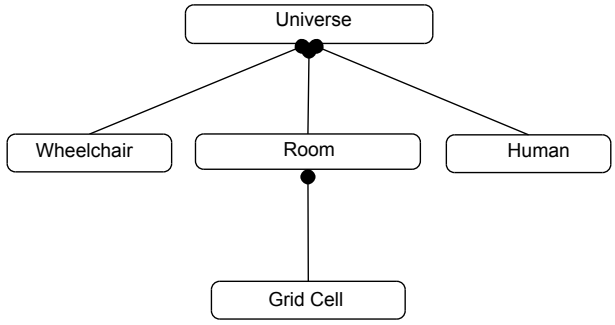


Figure 4.10: Resource holons in obstacle avoidance task when dividing the environment into individually allocatable grid cells.

up to date, changes which are in conflict with a robot's trajectory can be propagated in time, through the exploring D-MAS, and alternative solutions can be selected.

Since the above-described solution highly depends on restricting assumptions and, even if the assumptions are met, a highly dynamic environment would generate an large amount of allocations and changes, solving obstacle avoidance by *pure* resource allocation is not recommended, at least not at such a fine granularity level. A more appropriate solution is to combine global planning in PROSA/D-MAS with local obstacle avoidance in the robot control algorithm.

4.6.3 Resource capability solution

In this case, the ability to (locally) avoid obstacles is attributed to the resource holon representing the robot. The product holon now specifies that the resource should, besides navigation capabilities, also have obstacle avoidance capabilities. For example, if such a product holon is used, an allocated robot would not only activate its motors to navigate but would also use its sensors to detect and avoid obstacles.

Therefore, product holons can be defined for navigation with or without obstacle avoidance, depending on the desired application. Also, specifying this capability in the process plan ensures that only those robots, which are able to perform obstacle avoidance, are allocated.

The resource capability solution would use PROSA & D-MAS to allocate the rooms and corridors needed to navigate to the destination and allocate a robot with an obstacle avoidance algorithm on board for collision-free execution of the path. This way, global conflicts in narrow corridors or at intersections are dealt with by PROSA & D-MAS coordination, while local conflicts, e.g. within a room or corridor, are taken care of by the robot controllers.

This implies that if in a particular environment resource the conflicts cannot be solved by obstacle avoidance, it should be modelled in PROSA and solved by the higher-level coordination PROSA and D-MAS offer.

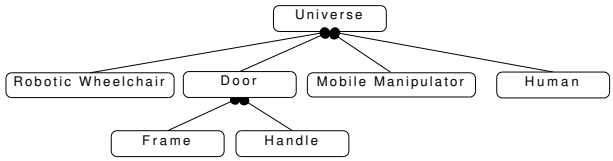


Figure 4.11: Resource holons for a door opening task.

4.7 Use case III: multi-robot door opening

4.7.1 Goal

The more interesting applications, benefiting more directly from the available technology of the holonic manufacturing execution systems domain, **include the environment of the robots explicitly** and have resource holons representing the resources therein.

The goal in the multi-robot door opening task is to have one or more (mobile) robots capable of opening a door and other robots required to navigate from one room to another through a door opening. Ideally, if more than one robot has to move through the door within a short time interval, the door should be opened only once.

4.7.2 PROSA & D-MAS solution

Figure 4.11 shows the resource holons for a door opening scenario/task.

The `Universe` resource allows future additions of resource holons to the application. Here, the `Universe` is divided into resources belonging to the mobile manipulator executing the task, task-relevant resources such as the `Door`, its `Handle` and its `Frame`, `Robotic Wheelchair` resources and the `Human` operator whose control is shared with the wheelchair.

This explicit resource representation and resource allocation in the robot's environment facilitates coordination. E.g., when two wheelchairs arrive at the same time at the door, the explicit allocation of the door resource ensures that the mobile robots are able to exit one by one and to avoid deadlock. Assume that both wheelchairs are driving towards to the door opening and issue allocation requests to the `Door` Resource. Since these requests are handled sequentially, the first robot requesting allocation at a particular time, will receive that allocation slot, while the other robot will receive a slot behind the previous slot. If both

wheelchairs reach the door within an acceptable time delay, the door should only be opened and closed once. The door holons will handle this in a manner analogous to a machine holon in a factory managing its change-over and set-ups.

When order holons handling the wheelchairs apply the D-MAS mechanism, the resources in the environment will be informed about future allocations. Therefore, the door opening coordination is not limited to handling the current situation. The order holons will observe the predicted future states of the resources in the environment and are able to account for these predictions. E.g. when an order holon sees that it will have to wait for other wheelchairs traversing the door in the opposite direction, it may adapt its trajectory in space and/or time, in order to reduce power consumption, robot component wear or have its occupant wait in a more agreeable or useful location (e.g. in front of a television or at a battery recharging station).

4.8 Use case IV: allocation of external sensors

4.8.1 Goal

This use case illustrates how a structurally decomposed software system with explicit resource allocation handles unexpected events such as sensor failure. Also it illustrates how resource aggregation is not limited to how resources are physically connected. E.g., a robot system without distance sensors might allocate a laser scanner of another robot in the room to perform its task.

4.8.2 Static environment sensor

In this particular scenario, the rooms in the environment are equipped with their own sensors. E.g, with a laser scanner in a top corner of a room, an occupancy map of this room can be constructed to provide the resource holon representing this room with up-to-date information.

The resource graph of this case is shown in Figure 4.12 and is very similar to the ones shown before. The only difference is the additional `Sensor` holon, which is part of the `Room` holon.

In order to use the external sensor to navigate in the room, for instance, the process plan of the navigation product holon contains the sensor allocation, next to any necessary room or corridor allocations. Evidently, not any sensor can be allocated, and the process plan would also indicate to allocate the sensor connected to the relevant room the robot wants to navigate in.

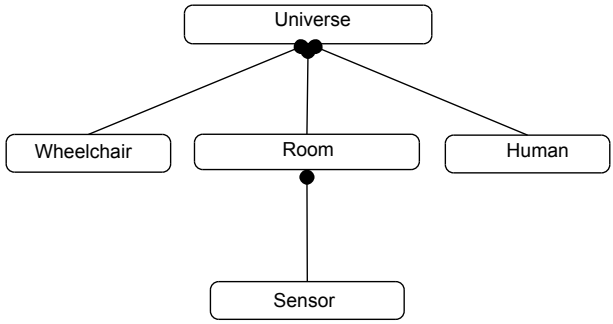


Figure 4.12: Resource holons for a navigation task in which a robotic wheelchair allocates an external sensor placed in the environment.

Furthermore, the process plan can easily be extended to allow navigation between different rooms by indicating which sensor holon to allocate at which time instant.

The exploring D-MAS would in this application provide information about when a desired sensor will be available, while the intention D-MAS then uses this schedule to allocate other necessary resources, such as rooms and corridors. Eventually, in case of a robotic wheelchair, the user might be informed when it is possible to start the navigation.

4.8.3 Sensor redundancy

The power of D-MAS and explicit resource allocation is its robustness versus unexpected disturbances. Consider a mobile robot navigating from one room to another using its own laser scanner to scan the environment and avoiding obstacles. Assume that, due to some unexpected problem, the sensor fails to produce any valid scans. For instance, in robotic wheelchairs, the sensor might be occluded by the user's clothing.

The above described PROSA & D-MAS solution, in which the process plan contains the explicit allocation of a sensor, would solve this problem by allocating, if available, another sensor nearby, after the D-MAS notices the currently allocated sensor failed. Of course, the robot control software needs to be capable of interpreting sensor data from remote sensors, e.g. by applying the correct coordinate frame transformations.

Having redundant sensors available increases reliability and can be useful during sensor malfunction. It also enables sensor reboots or *online* software upgrades

without having to suspend the robot's task execution.

4.8.4 Sensor sharing

Depending on the type of sensor, it is also possible to share this sensor between different order holons. For instance, a laser scanner providing distance measurements of the complete room can be shared among different robots, while a controllable camera pointing at specific location will only be shared if both users require data from the same location.

To allow sensor sharing, the sensor resource holon could model a set of allocation policies. For instance, *read-only* allocations would always be possible, while *read-write* allocations, i.e. allocations allowing to move the sensor into a desired direction, should be mutually exclusive.

4.8.5 Looking ahead

Since the allocation of external sensors is not limited to where these sensors are placed, it could also be used to look ahead and query availability or occupancy in a target room. This way, a better end position of the robot can be determined based on the remote sensor data. For instance, in a class room or meeting room an available desk could be identified and already be reserved for the robotic wheelchair user.

Again, up-to-date information provided by the D-MAS, allows the order holon to change its intention if necessary. This look-ahead information, eventually, results in a changed trajectory executed by the robot.

4.9 Use case V: robot allocation policies

4.9.1 Goal

This use case illustrates how D-MAS or a staff holon might determine which robot is allocated for which user, based on some pre-defined criteria or policy. For example, different users might require different types of robots, such as wheelchairs with specific joysticks or mounting equipment. Other policies could be to always choose the closest robot or have priorities among users or their requests.

4.9.2 D-MAS approach

One way to match robots and users, based on user preferences or requirements, is to define specialisations of a common robot resource holon for each user specific or adapted robotic wheelchair and specify the selection criteria or policy in the order holon.

The exploring D-MAS return, as usual, suggestions of resource holon allocations which provide required capabilities, e.g. navigation. The order holon then selects one of these solutions which matches the policy requested by the user.

Another solution is to define specific operations or capabilities for each specialisation and create multiple product holons. In this case, the D-MAS would already select only those resource holons matching the user preference.

Time or distance based policies, e.g. users requesting the closest wheelchair or the one which can arrive the fastest, are usually taken care of in the selection procedure in the order holon. The order holon can rank the solutions provided by the exploring D-MAS, based on time or distance to reach their target, and select the best one by sending out intention ants to allocate the relevant resource holons.

4.9.3 Staff holon approach

Alternatively, these policies and knowledge on all robots in the environment can be centralised in a staff holon. This staff holon would keep track of where each robot is, what its capabilities are, which specialised equipment it might have and when it is available for allocation.

At each user request, this staff holon can provide advice to the order holon on which robot resource holon seems to best fit this request. As was discussed in section 2.5.5, a staff holon only provides advise. The final allocation decision in still made by the order holon. This way, robustness and system optimisation are decoupled to prevent poor performance in case of disturbances. For example, if the staff holon's choice has a flat tire, the D-MAS and order holon are still able to adequately react and choose another robot.

4.9.4 Human factors in robotic wheelchair applications

Whenever humans are involved, policies, such as those described above, become extremely important. For example, depending on the mood or the intention of the user, a time optimal solution might be preferred over a trajectory with

the shortest travelled distance. Note that in multi-robot path planning, these two are not necessarily the same. Indeed, the order holon might have a choice between a solution in which the robot has to wait a few minutes before taking a corridor, or a solution in which the robot takes an alternative longer route but arrives at its target earlier.

In environments with many rooms and corridors, such as office buildings or hospitals, the number of possible paths from start to target is large. One user might request a wheelchair to go to the restroom and in that case prefers the fastest solution, while another user, sitting in a recreation room in front of the television, would prefer that the robotic wheelchair will arrive a few minutes later instead of taking a detour.

Taking care of these human factors is not straightforward, since each human is different and in each situation another solution might be preferred. This thesis will, besides this short discussion, not focus on these differences and during the experiments only one policy, shortest time, has been adopted.

4.10 Use case VI: global and local multi-robot coordination

4.10.1 Goal

When a mobile robot navigates from one room to another, it not only requires a global plan, indicating which sequence of rooms and corridors it will take to reach its target, but also a local plan, used to avoid conflicting target positions between robots within one room.

This use case discusses how PROSA and D-MAS can coordinate multiple mobile robots for both global and local path planning. Also some issues, typically for robotic wheelchairs or other mobile robots interacting with humans, will be discussed.

4.10.2 Global multi-robot path planning

This application has been the focal point of this text and a PROSA & D-MAS solution has been discussed throughout previous chapters, see e.g. sections 2.5.2, 2.5.3 and 2.5.4 for a detailed discussion.

In short, resource holons represent rooms and corridors in the environment of which some have a limited capacity with respect to the number of robots. In the product holon, a sequence of operations on rooms in the environment is defined. The order holon receives solutions containing available rooms from its exploring D-MAS and allocates these rooms using its intention D-MAS.

4.10.3 Local multi-robot path planning

One problem not handled by the global planning is where each robot is positioned within a room or corridor and how this might interfere with other robots. For example, a large room might have the capacity to hold five robots, but if one of those robots is positioned in front of this room's only entrance or exit, other robots are unable to enter or leave the room.

To incorporate this local within-one-room path planning problem, the scheduler of the resource holon reflecting a particular room could be augmented with a multi-robot path planner solving local issues. Besides time slots also target positions would be taken into account to determine which robot best enters the room first.

This local path planner can also take care of where robots in transit should position themselves within the room to minimise the interference with others. For example, if a robot moves from room A to B and passes a set of corridors and other rooms, the allocations made by the order holon might not guarantee this navigation is performed in one movement. Depending on other robots in those rooms, the robot might require to pause in one or more intermediate room. Also, the location or transit zone, where the robot pauses, is depending on how many robots are already in that particular room.

4.11 Conclusion

Summary:

This chapter first showed how the HMES, which was developed for manufacturing control, is used in robotics and how communication between existing robot software and the HMES is done.
Also, various holonic robot applications were discussed to illustrate how PROSA and D-MAS can be used in robot applications and highlight their diverse applicability.
In the context of this thesis, out of the use cases formulated in this chapter, the multi-robot path planning scenario was implemented and tested both in simulation and on the robotic wheelchairs available at PMA/KU Leuven.

In the following chapter, simulation experiments analysing the performance of the HMES with respect to scalability and robustness are presented. Also a computational complexity analysis is provided and the limitations of the current implementation are discussed.

The chapters following these analyses present experiments and results of the implemented use cases on multi-robot path planning.

Chapter 5

HMES Analysis

> *As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.*
>
> David Parnas (⋆1941–)

5.1 Introduction

In this chapter, the HMES performance in multi-robot navigation tasks is analysed. Firstly, configurable parameters and their influence on system performance are identified. Nextly, a complexity analysis in term of these parameters is conducted. Then, scalability with respect to number of robots and environment size is investigated. Next, robustness against disturbances is analysed. Finally, the current implementation limitations are highlighted and possible solutions are provided.

In order to analyse the above mentioned characteristics, simulation experiments were used. In simulation, the number of robots and size of the environment is easily changed and a high number of runs can be conducted to increase statistical relevance.

However, only robots and the environment were simulated. The HMES software interacting with this simulation remained unchanged, with the exception of its parameters being tuned for specific simulations. The coordination software used

for actual robots and simulated robots is the same since the communication interface, described in 4.3.2, separates the HMES from the robot control software.

5.2 Parameter calibration

The HMES defines a set of tunable parameters. Most of these parameters influence the use of exploring and intention D-MAS by order holons. As was pointed out in chapter 2, and more specifically in guideline 8, parameter values are application dependent. Moreover, selecting erroneous values can result in undesired behaviour and impede the performance of the HMES. In this section, these parameters and their influence will be discussed. Simulation experiments will also show the effect of changing one of these parameters.

In addition to the D-MAS parameters, how an order holon chooses one solution over another and when it decides to change a solution it previously committed to, can also be configured in the HMES and, again, depends on the application. Changing these selection criteria can influence the system performance and nervousness. The selection criteria, chosen for the multi-robot navigation scenario, will also be discussed in this section.

5.2.1 Exploring rate

The exploring rate controls how many exploring ants are sent per second per order. Consequently, the lower this rate, the longer it takes before an initial solution is found and presented at the order holon. Also, a lower exploring rate results in fewer solutions received by the order holon at a particular time.

In section 2.6, the exploring and intention ants were defined as lightweight agents which virtually traverse the resource graph and either search for a possible solution (exploring) or perform allocation on specific resource holons (intention) for the order holon their linked to.

Since the order holon will commit to one of the solutions presented by the exploring ants, configuring the exploring rate badly can result in two issues:

Firstly, the order holon will not have received any solutions yet and the order is idling. In the case of robot navigation, this results in a prolonged waiting time between the navigation request and the actual start of the navigation.

Secondly, and even worse, the order holon receives a few solutions but since the exploring ants randomly navigate the resource graph, these solutions might not

be ideal. In the navigation example, this can cause large detours and inferior navigation performance.

Figures 5.1 and 5.2 show solutions for three navigation requests in a common office environment, respectively using a low and high exploring rate. The solid line indicates the path the robot executed and the crosses are transition points where the robot waited before moving through a narrow corridor.

Due to a low exploring rate, the exploring D-MAS failed to explore the solution space sufficiently and only poor solutions are returned (Figure 5.1(a) and 5.1(b)). For the navigation request shown in Figure 5.1(c), the exploring D-MAS only has one option to go from start to goal, so it succeeds in returning a satisfactory solution.

In Figure 5.2 an adequate exploring rate, of 1 ant per second, is chosen and the order holons receive a sufficient amount of solutions to choose from. The solution shown in Figure 5.2(b) makes a detour since the order holon's selection policy prefers earlier arrival time over shorter path distance.

In the multi-robot navigation scenario set forth in this text, a recommended minimum for the exploring rate is 1 ant per second. This is due to the real-time character of the application, i.e. user requests should be handled in real-time and are not known beforehand. This means solutions should be searched for as fast as possible.

The robotic wheelchairs in our scenario move at maximum 1 m/s. In that case, sending out 1 *explorer* each second, means that during navigation for each meter the robot moves, at least one new exploring ant is sent to detect disturbances.

Since the HMES is only used here to coordinate the allocation of corridors and small rooms, and not local obstacle avoidance, the proposed exploring rate is high enough. However, if processing power allows it, a higher rate is desirable, since it results in more solutions to choose from. On average, in the experiments discussed in this chapter, an exploring ant required 100ms to find a solution, so sending 10 exploring ants per second is still feasible.

The quality of the solutions and percentage of failed ants also depends on how the environment, i.e. the resource graph, is constructed. The number of possible paths increases exponentially with the number of edges in this graph. Therefore, to ensure the exploring ants are able to return valid solutions, adding additional functionality such as loop detection are recommended.

Moreover, resource holons could provide signs indicating the recommended, shortest path for other resource holons, similarly to the colour coded routes in a hospital. This information could be used by the exploring ant as a weight in their selection process. This way the probability an exploring ant will follow

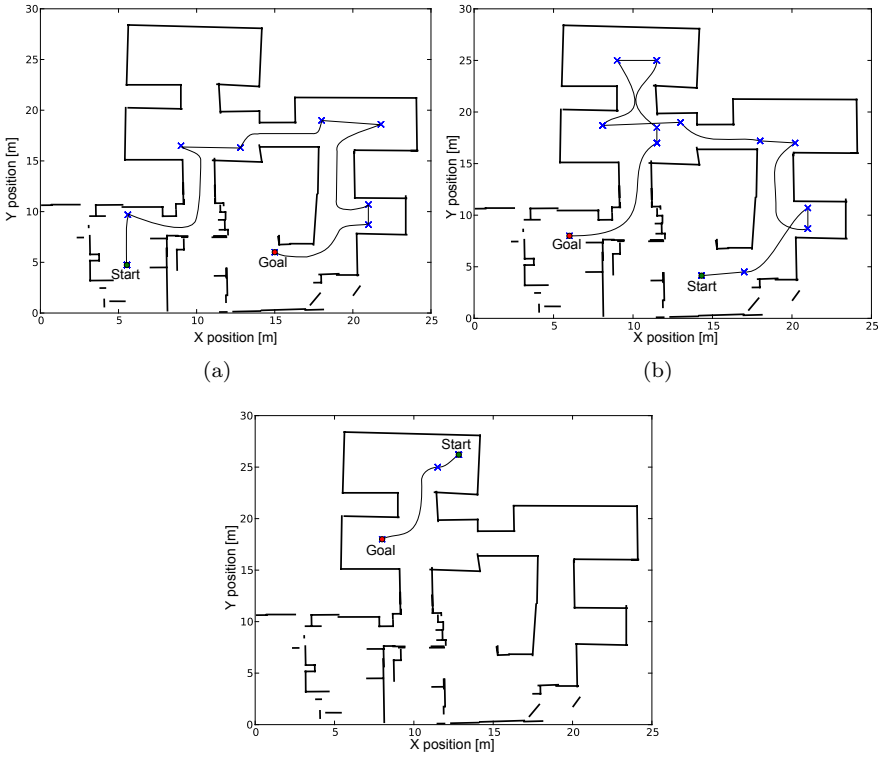


Figure 5.1: Navigation solutions, to which the order holons committed, for a three robot setup in a common office environment. Due to a low exploring rate, the exploring D-MAS failed to explore the solution space completely and only poor solutions are returned (5.1(a) and 5.1(b)). For the navigation request shown in Figure 5.1(c), the exploring D-MAS only has one option to go from start to goal, so it succeeds in returning a satisfactory solution.

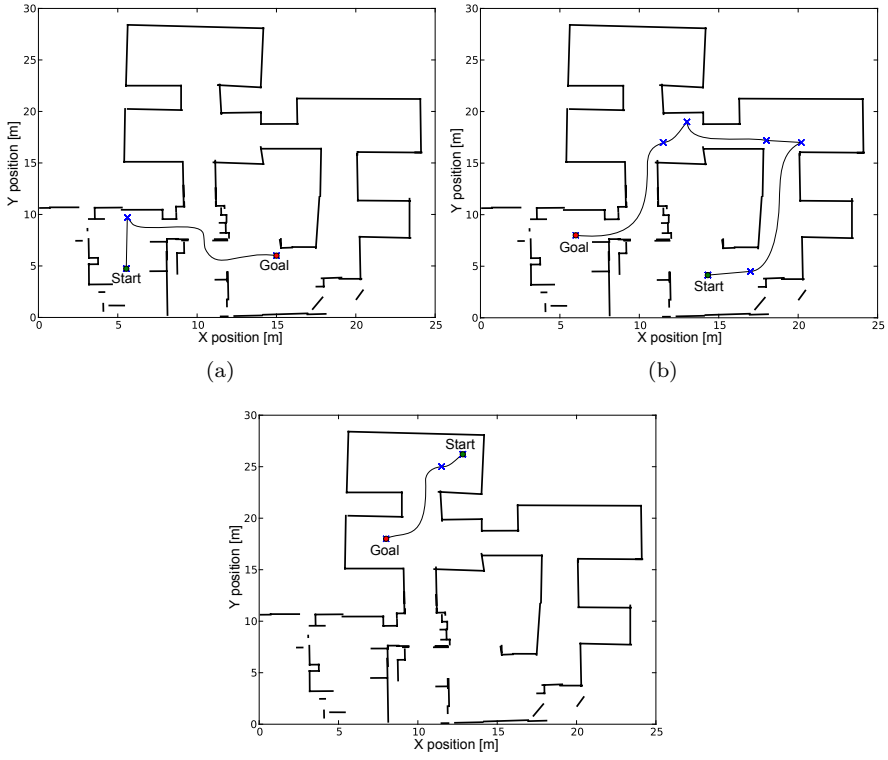


Figure 5.2: Navigation solutions, to which the order holons committed, for a three robot setup in a common office environment. Here an adequate exploring rate, of 1 ant per second, is adopted and the order holons receive a sufficient amount of solutions to choose from. The solution shown in Figure 5.2(b) makes a detour since the order holon's selection policy prefers arrival time over path distance.

the shortest path increases while still other paths are explored to remain robust against disturbances. In the current implementation only basic loop detection is applied by decreasing the probability of the already explored path.

More information on the search space covered by ants and the trade off between exploration and exploitation can be found in [Verstraete 2009].

5.2.2 Refresh rate

The refresh rate determines how many intention ants each order sends out per second. The rate determines how fast a currently chosen solution is checked and how fast allocations on resource holons are made. Consequently, the system response time to disturbances is directly related to this parameter.

Also, in order to ensure sufficient solutions are returned by the exploring D-MAS before committing to a solution or changing an intention, the refresh rate should always be lower than the exploring rate. Nevertheless, a too low refresh rate results in a too rigid system, unable to adapt to unexpected situations in a timely manner.

Evidently, sending intention ants at a low rate also influences how long it will take to commit to a solution, since the solutions an order holon received by exploring D-MAS will be checked at this same rate. In the navigation scenario this results in a delay between the navigation request and its execution.

Figure 5.3 shows the influence of the refresh rate to the same three navigation requests as described above. On the horizontal axis the ratio between exploring and refresh rate, denoted as E/R ratio, is plotted, while the vertical axis indicates the time in seconds. The different graphs show the start time, finish time and duration of the order holons in relation to the refresh rate.

It shows the start time is directly proportional to the refresh rate, since a solution is only committed to after a first refresh is performed. Therefore, the E/R ratio should not be set too high, to avoid undesirable delays between a user request and the actual start of its execution.

The finish time and duration graphs also indicate a performance degradation for low E/R ratios, acknowledging the problem when the order holon has not yet received an adequate amount of solutions from the exploring D-MAS and commits to an inferior solution.

This problem is largely due to the randomness of the exploring ants with respect to junctions in the environment and the order holon's selection and switching criteria. The performance for low E/R ratios can be slightly increased by

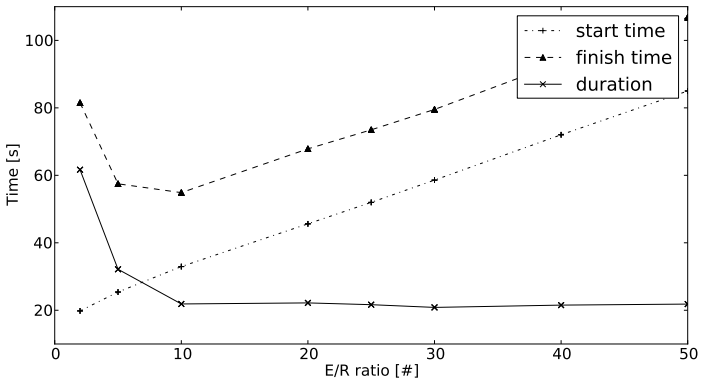


Figure 5.3: The influence of the refresh rate with respect to start time, finish time and duration. On the horizontal axis the ratio between exploring and refresh rate, denoted as E/R ratio, is plotted, while the vertical axis indicates the time in seconds. Although start time is directly proportional to the refresh rate, a performance degradation is shown for low E/R ratios, acknowledging the problem when the order holon has not yet received an adequate amount of solutions from the exploring D-MAS and commits to an inferior solution.

incorporating a **release time**, which determines at what time, relative to the order's creation time, the order is allowed to start.

A higher release time allows the order holon to send out more exploring ants and selecting a solution before committing to it. This increases the probability of finding a superior solution even if a low E/R ratio is used. Of course, a higher release time also increases the delay between order holon creation and its execution, which in the case of multi-robot navigation not desirable.

In this application scenario the user demands a robot at a particular time and also expects it to start navigating as soon as possible. Hence, the order arrival times, i.e. the navigation requests, are not known in advance. This is in contrast with a common manufacturing application in which the order arrival times are usually known well in advance.

Figure 5.4 shows the result of employing a release time proportional to the refresh rate. Here the release time is set at three times the refresh rate, to allow the order holon to change, if necessary, three times its solution before committing to it. Start time, finish time and duration are plotted with respect to the E/R ratio. It shows that for lower ratios from 2 to 10, the duration of executing a navigation request now practically remains constant, in contrast to

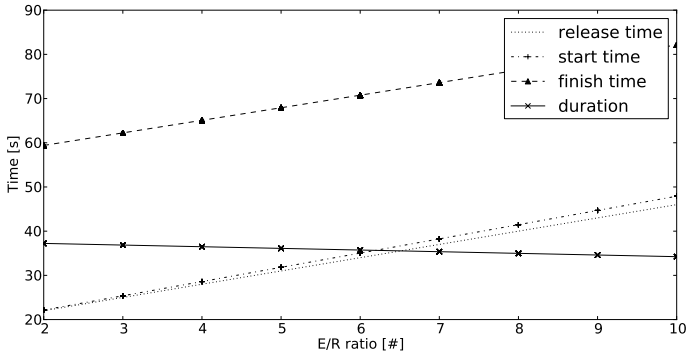


Figure 5.4: The influence of the refresh rate with respect to start time, finish time and duration, while employing a release time relative to the refresh rate. On the horizontal axis the ratio between exploring and refresh rate, denoted as E/R ratio, is plotted, while the vertical axis indicates the time in seconds.

the results presented in Figure 5.3.

In our application scenario, the refresh rate is set to 5 seconds. With an exploring rate of 1 second this implies that every refresh cycle five solutions could be evaluated, depending on how long it takes for the exploring ants to return with a solution.

5.2.3 Evaporation rate

The evaporation rate resembles the pheromone decay rate in ant colonies, i.e. solutions found by exploring ants remain only valid for a limited time interval. The rate defines how many times per seconds the set of solutions is reviewed and old solutions are removed. Evidently, the rate, at which these solutions *evaporate*, should be lower then the exploring and refresh rate.

Nevertheless, a too low evaporation rate, results in keeping too old solutions, which might have become invalid due to the dynamic nature of the environment. Although such invalid solutions will be detected when refreshing these solutions, adopting an adequate evaporation rate acts as a filter, removing these solutions automatically. This way no unnecessary refreshes are required.

In our application scenario, the evaporation rate is set at once every 15 seconds. This way, solutions are evaluated in at most three refresh cycles. Moreover,

changes or disturbances in the environment appearing within 15 seconds from each other are only reflected by new exploring ants.

5.2.4 Hop limit

The hop limit defines how many hops exploring ants are allowed to make during their search. Every operation virtually executed on a resource counts as one hop. It ensures the exploring ants do not endlessly search for a solution in the research graph. Due to their random choices at junctions, some ants might not find a solution within a desired time frame. If an exploring ant reaches its hop limit, it stops exploring and returns with a failed status.

Choosing a low hop limit decreases computation time but might also make it impossible to find solutions. In the navigation scenario, an appropriate lower bound for the hop limit is the longest desirable path. This longest path corresponds to the diameter of the resource graph. The diameter of a graph is defined as the longest shortest path between any two vertices in that graph. To calculate this graph diameter several algorithms can be used, for example by first calculating the shortest path between each 2 vertices and then taking the maximum [Harary 1994].

In more advanced navigation requests, for instance a set of target locations to visit, the solution space increases and, thus, the hop limit should be increased as well. Also, in that case, loops might be allowed.

5.2.5 Order holon selection criteria

Besides the configurable parameters of the HMES, the selection criteria or policies of the order holons also influence how well the system performs and how fast valid solutions are found.

Firstly, the order holon has to define a set of criteria to evaluate if a solution is worth committing to or not. These criteria allow the order holon to choose the best fit from a set of solutions, provided by the exploring ants.

In the navigation scenario two obvious criteria are time and distance. E.g. a solution arriving at an earlier time or a solution being shorter can be preferred. In our application scenario these choices are not always straightforward, as was already discussed in more detail in section 4.9.

Secondly, the order holon also has to determine when to switch to a better solution. Since switching too much results in unwanted nervous behaviour, some

Parameter	Value	Remarks
Exploring [s]	≤ 1	Maximum robot velocity of 1 m/s
Refresh [s]	5	Able to choose between 5 solutions
Release Time [s]	15	Set at 3 times refresh rate
Evaporation [s]	15	Remove solutions after 3 refresh cycles
Hop Limit [#]	15	Depends on environment size

Table 5.1: Recommended values for the HMES parameters for our multi-robot navigation scenario. Rates are inverted and expressed in time intervals, for readability.

randomness can be incorporated in this switching behaviour. For example, if a better solution is found, the order holon will have a 60% chance to actually commit to that new solution.

This way, not all order holons will change their solutions simultaneously in case of a disturbance, such as a blocked corridor.

More information on how the selection policy of order holons affects the system nervousness can be found in [Hadeli et al. 2006].

5.2.6 Conclusion

In Table 5.1 an overview of the above-described parameters is given with recommended values in our application scenario. Rates are inverted and expressed in time intervals, for readability.

5.3 Computational complexity

While in section 5.2 the effects of certain tunable HMES parameters was demonstrated, this section will take a more detailed look at the computational complexity of the current HMES implementation. As a result, an estimate of the system response time in function of a set of parameters will be presented.

Typically in computational complexity analysis of algorithms, their response time is computed with respect to the input size N. Although, a distinction can be made between best case, average case and worst case complexity, but commonly worst case or upper bounds are calculated. To denote these upper bounds, the *big O* notation is used, hiding constant factors and lower terms. For

Parameter	Description
Exp [#]	Average # Exploring Ants per Order
Ref [#]	Average # Refreshing Ants per Order
Hop [#]	Hop Limit
Sol [#]	# of Solutions
$Slot$ [#]	Average # of Slots
Res [#]	# of Resources
Ord [#]	# of Orders
Ops [#]	Average # of Operations
Req [#]	Average # of Requests
Rec [#]	Average Recipe Size
Con [#]	Average # of Connection Points

Table 5.2: Parameters used in complexity analysis.

example, a binary search algorithm on a sorted list has a worst case complexity of $\mathcal{O}(log_2(N))$, since it cuts the list in halve until it reaches the desired element.

In the case of the HMES, there is no single input size but a set of parameters which determine how well the system responds. Besides the number of orders, also the number resources, the complexity of the process plans and all the tunable parameters contribute to the system performance.

Due to the complex nature of the HMES and its internal interactions, a practical approach was chosen to determine its computational complexity, as opposed to providing a mathematical proof. Also, for some parameters empirical estimates will be provided to illustrates the difference between worst case and average case.

5.3.1 Parameters

In Table 5.2 the set of parameters and their abbreviations, used to determine the complexity, is given. While some parameters have already been defined in previous sections, others will be explained in the relevant subsequent paragraphs.

5.3.2 Analysis

Figure 5.5 shows which cycles in which holons contribute the most to the system's computational complexity. The product holon has no separate cycle,

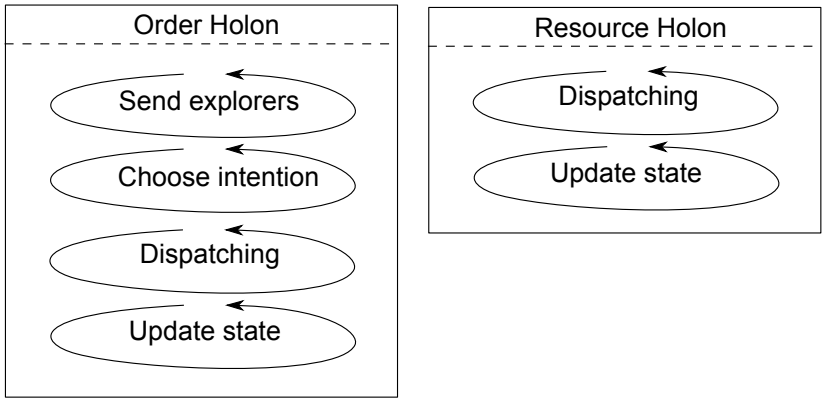


Figure 5.5: Cycles in holons which contribute the most to the system's computational complexity. The product holon has no separate cycle, since it is not an *active* object. Rather, it is consulted by the order holon's exploring D-MAS.

since it is not an *active* object. Rather, it is consulted by the order holon's exploring D-MAS.

In the subsequent paragraphs, each of these cycles will be analysed in more detail to determine their computational complexity with respect to the above mentioned system parameters.

5.3.3 Order holon

In the order holon several cycles with distinct periodicities are maintained simultaneously. Exploring ants are sent at a faster rate than intention ants. The dispatching cycle represents the communication towards the real world or emulation, while the update state cycle represents the processing of events received from the real world or emulation.

Send explorers

Each exploring ant performs in the worst case Hop number of hops, in which each hop consists of consulting the product holon for options, choosing an option and allocating a resource. However, the hop limit Hop is an overestimate and in this case an empirical estimate is more appropriate. The analysis of the exploring ants sent out during the scalability experiments, presented in

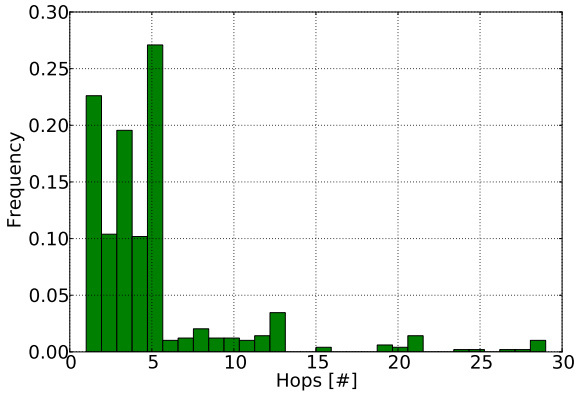


Figure 5.6: Histogram of the number of hops exploring ants made in an environment of 16 rooms and 24 corridors in which the shortest path solution is 5 hops.

section 5.4 is depicted in Figure 5.6 and Figure 5.7. This histogram and scatter plot show an average hop count of 4.781 in an environment of 16 rooms and 24 corridors in which the shortest path solution was 5 hops.

Exploring ants are continuously sent out, also after a part of a solution is already executed. Therefore, many ants find a solution in less than 5 hops.

Consulting the product holon has a complexity of $\mathcal{O}(Rec + Con)$, since the product holon has to determine the next operation according to the process plan and check for each connection point if this next operation is possible.

Choosing an option out of the proposed next steps is constant ($\mathcal{O}(1)$) and, thus, can be discarded in the overall calculation.

Allocating a resource is a complex process and will be discussed in a separate paragraph, in which its complexity will be estimated as $\mathcal{O}(Req \cdot Slot)$ on average and in the worst case $\approx \mathcal{O}(Hop^2)$. This worst-case situation only occurs if an exploring ant with a maximum length solution would allocate each of its operations on the same resource.

Figure 5.8 and Figure 5.9 show empirical estimates for Req and $Slot$ respectively. These averages are calculated using an environment with 16 rooms, 24 corridors and 16 navigation requests, in which the robots move from one room to one of their neighbouring rooms. These data clearly show hop limit Hop is an upper bound.

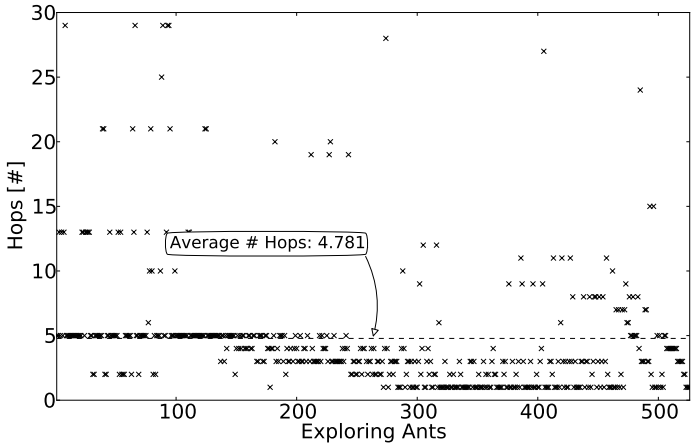


Figure 5.7: Scatterplot of number of hops exploring ants made in an environment of 16 rooms and 24 corridors in which the shortest path solution is 5 hops.

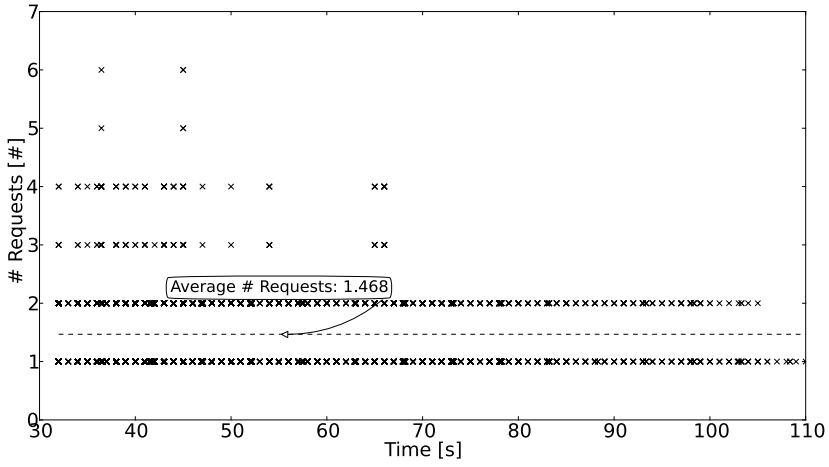


Figure 5.8: Average number of requests over time on all resources in an environment with 16 rooms, 24 corridors and 16 navigation requests.

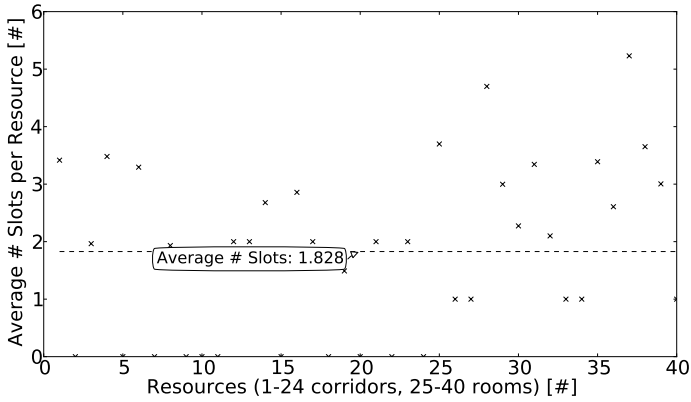


Figure 5.9: Average number of slots per resource in an environment with 16 rooms, 24 corridors and 16 navigation requests.

Thus, the worst-case complexity of sending an exploring ant is:

$$\begin{aligned} \mathcal{O}(\text{Send explorer, worst case}) &\approx \\ \mathcal{O}(Hop \cdot (Rec + Con + Hop^2) + Hop \cdot Exp) &\approx \\ \mathcal{O}(Hop^3) & \end{aligned} \tag{5.1}$$

The hop limit is used here as an upper bound for the average hop count by an ant and the average requests and average slots on a resource.

The average complexity of sending an explorer is:

$$\mathcal{O}(\text{Send explorer, avg. case}) \approx \mathcal{O}(Hop \cdot Req \cdot Slot) \tag{5.2}$$

If an exploring ant fails, it has to backtrack to undo its allocation. In the worst case, Exp ants have to backtrack over Hop resources, resulting in $\mathcal{O}(Hop \cdot Exp)$

Choose intention

At a lower frequency than the exploring rate, the order holon evaluates its set of received solutions and commits to the most appropriate one based on its selection criteria.

In this cycle first the current solution is refreshed, i.e. an ant is sent out to check if the current solution is still valid. Sending out a refreshing ant has a similar complexity as sending out an exploring ant. However, the product holon does not need to be consulted in this case. Since the allocations are still made for at most Hop resources, the resulting complexity remains $\approx \mathcal{O}(Hop \cdot Req \cdot Slot)$ on average and $\mathcal{O}(Hop^3)$ in the worst case.

Next, among the set of solutions the best alternative is chosen. The solution set size Sol is the determining factor here. This size is the ratio between the exploring rate and evaporation rate. For instance, if every 100ms an exploring ant is sent out and solutions evaporate after 5 seconds, the average amount of solutions $Sol = \frac{5s}{0.1s} = 50$.

After a best alternative has been chosen, this solution is refreshed and if still valid an intention ant is sent out to commit the solution. Sending out an intention ant has the same complexity as sending a refreshing ant.

Thus, the worst case complexity estimate of the choose intention cycle is the sum of the above estimates:

$$\begin{aligned} \mathcal{O}(\text{Choose intention, worst case}) &\approx \\ \mathcal{O}(Hop^3 + Sol + Hop^3 + Hop^3) &\approx \\ \mathcal{O}(Hop^3) & \end{aligned} \tag{5.3}$$

The average complexity is:

$$\mathcal{O}(\text{Choose intention, avg. case}) \approx \mathcal{O}(Hop \cdot Req \cdot Slot) \tag{5.4}$$

Dispatching

For each operation the order holon sends an event to the real world or emulation to start the execution. Processing this event is constant ($\mathcal{O}(1)$). Before it sends out this event, it refreshes the solution one last time.

The worst case complexity of the dispatching cycle then becomes:

$$\mathcal{O}(\text{Dispatching, worst case}) \approx \mathcal{O}(Hop^3) \tag{5.5}$$

In the average case, this yields again:

$$\mathcal{O}(\text{Dispatching, avg. case}) \approx \mathcal{O}(Hop \cdot Req \cdot Slot) \tag{5.6}$$

The amount of events sent to the real world or emulation depends on the number of operations Ops.

Update state

The order holon updates its state each time it receives an event from the real world or emulation. In this implementation, each time an operation starts and stops an event is sent. Processing a state update is constant and the amount of events is again dependent on the number of operations Ops.

$$\mathcal{O}(\text{Update state}) \quad \approx \quad \mathcal{O}(1) \tag{5.7}$$

In worst case the average number of operations is equal to the hop limit Hop.

Total

Combining all the order execution cycles with their respective frequencies yields:

$$\begin{aligned}
\mathcal{O}(Orders, worstcase) &\approx \\
\mathcal{O}(Ord \cdot (Exp \cdot (\text{Eq.}(5.1)) + Ref \cdot (\text{Eq.}(5.3)) + & \\
Ops \cdot (\text{Eq.}(5.5)) + Ops \cdot (\text{Eq.}(5.7)))) &\approx \\
\mathcal{O}(Ord \cdot (Exp \cdot Hop^3 + Ref \cdot Hop^3 + Ops \cdot Hop^3 + Ops \cdot 1)) &\approx \\
\mathcal{O}(Ord \cdot Hop^3 \cdot (Exp + Ref + Ops)) & \qquad (5.8) \\
\mathcal{O}(Orders, avg.case) &\approx \\
\mathcal{O}(Ord \cdot (Exp \cdot (\text{Eq.}(5.2)) + Ref \cdot (\text{Eq.}(5.4)) + & \\
Ops \cdot (\text{Eq.}(5.6)) + Ops \cdot (\text{Eq.}(5.7)))) &\approx \\
\mathcal{O}(Ord \cdot Hop \cdot Req \cdot Slot \cdot (Exp + Ref + Ops)) & \qquad (5.9)
\end{aligned}$$

5.3.4 Resource holon

Each resource holon has its own dispatching and update state cycle and also handles the allocation requests from the ants sent out by the order holon.

Dispatching

The dispatching cycle in the resource holon in negligible, since it only sends events at a fixed rate to the real world or emulation.

Update state

Every event the resource holon receives from the real world or emulation has an influence on its current schedule. Thus, the complexity of processing a state update is $\approx \mathcal{O}(Slot)$.

Similar to the order holon, the amount of received events is dependent on the number of operations Ops and in the worst case Hop.

Resource allocation

Each resource has a scheduler, which processes the resource allocation requests it receives from the D-MAS. For each request it evaporates old slots, consults its cache or schedules a new request and restores the previous state in case of virtual execution.

In the evaporation step, the scheduler reviews all slots and removes those slots which are older than the time window defined by the evaporation rate. The resulting complexity is $\mathcal{O}(Slot)$

Next, it consults the cache to check if the request has been scheduled before, so it can be updated. Browsing and comparing in a list of previous requests is $\mathcal{O}(Req)$ since a hash map is used, which has constant lookup time. In the worst case one order visits the same resource Hop times and the complexity becomes $\mathcal{O}(Hop)$

If the cache returns no result, it checks all slots for an already existing request from this order and removes it. Since a binary tree is used, removing this request is $\mathcal{O}(log_2(Slot))$.

Adding the new request has a complexity of $\mathcal{O}(Req \cdot Slot)$, since all slots are reviewed for each request. Finally, restoring the previous state requires $\mathcal{O}(Slot)$, since a copy of all slots is restored.

The overall computational complexity of resource allocation is the sum of above estimates. The addition of a new request is the determining factor.

$$\mathcal{O}(\text{Resource allocation, worst case}) \approx \mathcal{O}(Hop^2) \tag{5.10}$$

$$\mathcal{O}(\text{Resource allocation, avg. case}) \approx \mathcal{O}(Req \cdot Slot) \tag{5.11}$$

Total

Only the dispatching and update state cycle are counted here, since the allocation is already taken into account in the order cycles. Therefore, the overall complexity becomes:

$$\mathcal{O}(\text{Resources, worst case}) \approx \mathcal{O}(Res \cdot Hop^2) \tag{5.12}$$

$$\mathcal{O}(\text{Resources, avg. case}) \approx \mathcal{O}(Res \cdot Slot \cdot Ops) \tag{5.13}$$

5.3.5 Summary

Combining both the order holon and resource holon cycles yields the following estimates of the total computational complexity:

$$\begin{aligned} \mathcal{O}(\text{HMES, worst case}) &\approx \\ \text{Eq.}(5.8) + \text{Eq.}(5.12) &\approx \\ \mathcal{O}(Ord \cdot Hop^3 \cdot (Exp + Ref + Ops) + Res \cdot Hop^2) & \end{aligned} \tag{5.14}$$

$$\begin{aligned} \mathcal{O}(\text{HMES, avg. case}) &\approx \\ \text{Eq.}(5.9) + \text{Eq.}(5.13) &\approx \\ \mathcal{O}(Ord \cdot Hop \cdot Req \cdot Slot \cdot (Exp + Ref + Ops) + Res \cdot Slot \cdot Ops) & \end{aligned} \tag{5.15}$$

These calculations show the order holon contribute far more to the complexity than the resource holon and, in worst case, the hop limit Hop is the determining

factor. Note that, as was stated before, this complexity analysis concerns the current implementation of the HMES and not PROSA and D-MAS in general.

In order to look at the complexity of PROSA and D-MAS in general, abstraction has to be made from implementation specifics. Each virtual exploration or allocation step made by respectively exploring or intention ants is then denoted as a *tick* in the system. The actual computational complexity of this tick depends on how the resource graph traversal and resource scheduling are implemented. Also the resource state updates have an implementation specific part, which will be denoted as *update*.

The overall complexity then becomes:

$$\mathcal{O}(\text{PROSA\&DMAS}) \approx$$
$$\mathcal{O}(Ord \cdot Hop \cdot tick \cdot (Exp + Ref + Ops) + Res \cdot Ops \cdot update) \tag{5.16}$$

Optimisations are possible in both the order and resource cycles and above analysis indicates that optimising the resource scheduler could improve the overall performance.

However, there is always a trade off to be made between optimisations specific to an application and the general applicability of the system. Also, it is not guaranteed that an optimised scheduler with respect to computational complexity will deliver similar results with respect to the application.

5.4 Scalability

In this section the scalability of the PROSA and D-MAS approach is illustrated by analysing how the HMES performs when the environment or the number of robots and navigation requests increases.

It should be noted that, unlike a classical computer system, the multi-robot navigation scenario also has **physical limitations**. For example, using a small environment with a high number of robots is bound to results in poor system performance, since the limitations in reality, i.e. the small environment, causes more conflicts between the robots.

In order to evaluate scalability, system performance will be measured by its total execution time and throughput. Total execution time is defined as the time between the start and finish of a set of navigation requests, while throughput

incorporates the amount of navigation requests and is defined as how many navigation requests are handled per second.

Although these two measures provide an indication on how well the system can handle concurrent requests, the total execution time is also highly dependent on the length of the path the robot has to execute. A longer path results in a longer execution time. Therefore, the navigation requests are chosen in this experiment in such a way that their optimal path lengths are the same. In other words, when executed without interference or conflicts, each navigation request would result in a similar path length.

This way, longer path lengths are due to either conflicts which the HMES has to solve or, in lesser degree, the random behaviour of the exploring ants returning suboptimal solutions. As was discussed in section 5.2, a proper calibration of the HMES parameters is required to minimise this effect.

5.4.1 Increasing navigation requests

In this experiment a fixed environment of 16 rooms and 24 corridors is used. Each room starts with 1 robot and a navigation request represents a move to a randomly chosen neighbouring room. The amount of navigation requests is then increased from 1 to 16, with each new navigation request associated to a new robot.

In Figure 5.10, the environment is depicted, together with the 16 navigation requests, each executed by another robot. Robot starting positions are indicated by $S1$ to $S16$ and robot goal positions by $G1$ to $G16$

Note that some goal positions are coinciding, since in this simulation abstraction has been made of how to solve collisions in rooms. The corridors, however, can only hold one robot at the same time.

Figure 5.11 shows how the number of conflicts increases with respect to the number robots executing their navigation requests. A conflict is only counted if two robot's optimal paths share a corridor. Of course, if one robot has to make a detour due to one conflict, other conflicts might occur on that detour as well.

In Figure 5.12 the total execution time with respect to the number of robots is plotted. The correlation between number of conflicts and total execution time is clearly present between 1 and 8 navigation requests. In the first 4 sets, the robots execute their navigation task in parallel without interfering each other and total execution time remains constant. In the next 4 sets, 2 robots have a conflicting corridor while the other 6 are conflict free. The total execution time increases slightly due to this one conflict but remains constant over the 4 sets.

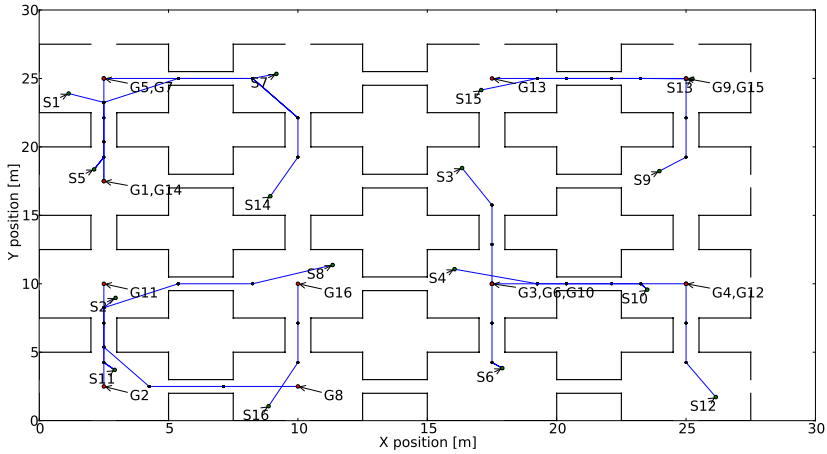


Figure 5.10: In this environment with 16 rooms and 24 corridors, 16 executed navigation requests are shown, each executed by another robot. Robot starting positions are indicated by $S1$ to $S16$ and robot goal positions by $G1$ to $G16$. The trajectories plotted here are the solution of one of the simulation runs in this environment.

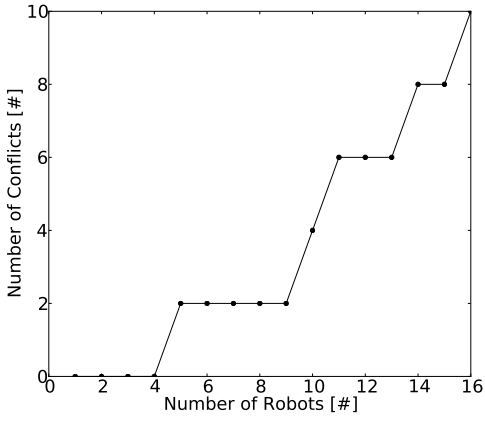


Figure 5.11: The number of conflicts in an environment with 16 rooms and 24 corridors with respect to the number of robots is shown here. Only conflicts between optimal paths are taken into account.

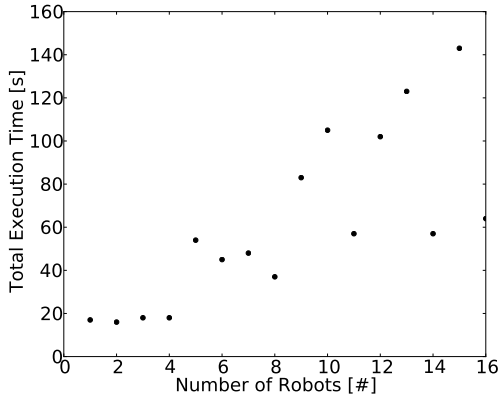


Figure 5.12: Total execution time with respect to number of robots executing in an environment with 16 rooms and 24 corridors.

Beyond 8 robots not only the initial conflicts with respect to the optimal path increase but also possible conflicts that arise due to random choices made by exploring ants increase. Depending on the committed solution, a robot might have to navigate longer or have longer idle interval in between. Due to the emergent behaviour of the delegate MAS, the correlation to the number of conflicts is less prominent.

Another way to evaluate system performance is its throughput, defined here as how many navigation requests are executed per second. Figure 5.13 shows the throughput with respect to the number of robots.

At first sight there does not seem to be a relation between the system's throughput and the amount of robots. However, if these values are plotted against the ideal throughput, i.e. if all navigation requests were executed in parallel without interference, a similar correlation with the number of conflicts as with the total execution time can be identified. Figure 5.14 and Figure 5.15 show this correlation.

In Figure 5.14 the throughput increases linearly for the first four sets, since all navigation requests are executed completely in parallel, while in Figure 5.15 the effect of the conflicts is shown for the first 8 sets. The throughput still increases linearly although it has an offset to the ideal curve, denoted here as *conflict penalty*.

It can be concluded that the system handles an increase of the number robots in the same environment well. The throughput never degrades below the nominal

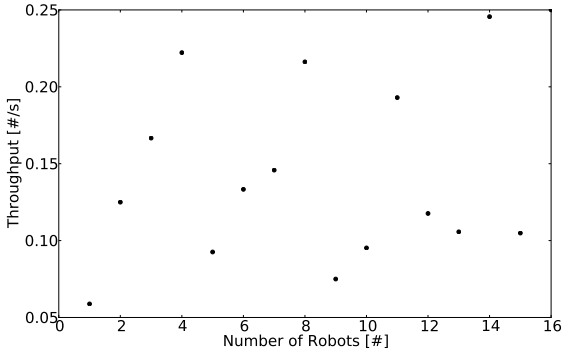


Figure 5.13: Throughput, defined as the amount of navigation requests that are executed per second, with respect to number of robots is plotted here.

level of one navigation requests. Physical limitations of the environment when the number of robots becomes high inevitably results in a decrease of throughput. The higher the number of conflicts, the more throughput deviates from the ideal throughput curve, in which all tasks would be executed in parallel without interference.

Nevertheless, this experiment highlights the power of the parallelism resulting from the structural decomposition of PROSA and the delegate multi-agent systems. As long as the navigation tasks do not interfere with each other, they can be executed fully in parallel. E.g. allocation of a set of corridors and rooms in the north west of the environment has no influence to all other navigation tasks in other regions, since all scheduling is distributed among the respective resources.

5.4.2 Increasing the environment

In this experiment the environment will be increased while the *traffic density* will be kept constant. It is assumed that most traffic is very local, i.e. between a small set of rooms and corridors and remains local even though the environment increases.

For example, in an office building most traffic is on the same floor or even in the same corridor while only a small percentage of people have to traverse the whole building. One of the reasons is the redundancy of frequently visited rooms such as restrooms, elevators or staircases.

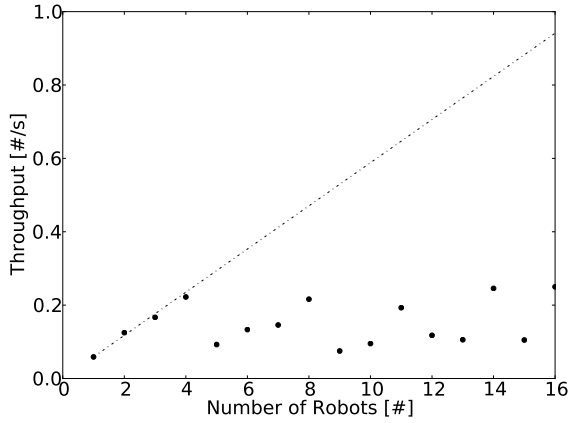


Figure 5.14: Throughput with respect to number of robots plotted against the ideal throughput curve, in which all robots would navigate without interference.

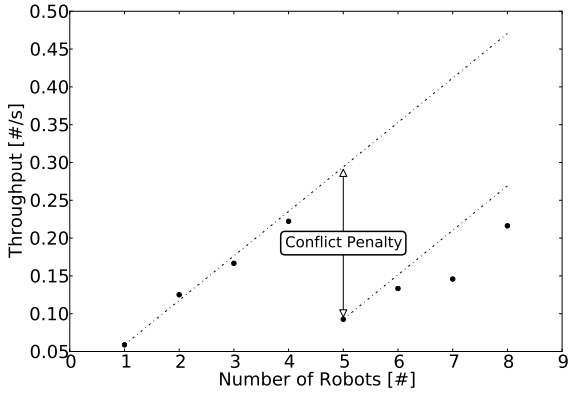


Figure 5.15: The effect of conflicts is denoted here. After increasing linearly in the first 4 sets, the throughput drops due to the introduced conflicts. Nevertheless, it still increases linearly, although parallel to the ideal curve.

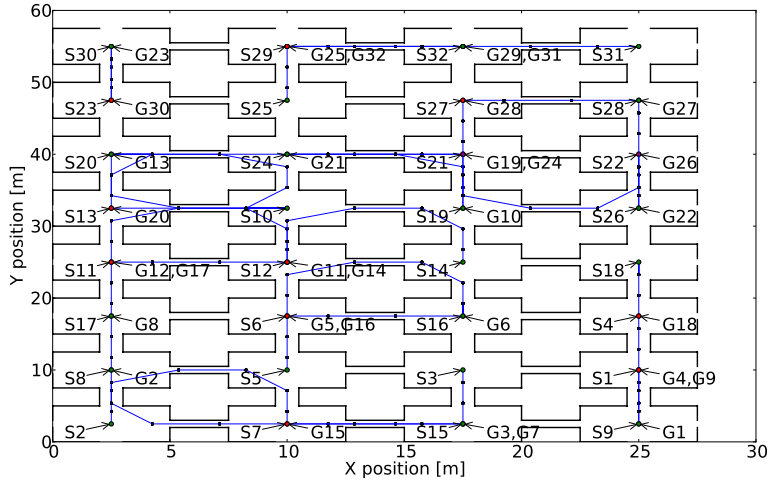


Figure 5.16: Environment with 32 rooms, 52 corridors and 32 robots, each starting in a different room. Each robot navigate to a randomly chosen neighbouring room. Robot starting positions are indicated by $S1$ to $S32$ and robot goal positions by $G1$ to $G32$. The trajectories plotted here are the solution of one of the simulation runs in this environment.

In order to assess the influence of a larger environment on the system performance, the type of navigation tasks and the relative traffic intensity will be kept constant.

Two environments will be compared in these experiments. The first environment has 16 rooms and 24 corridors, where 16 robots navigate around in the same topology and configuration as shown in Figure 5.10. The second environment has 32 rooms and 52 corridors and 32 robots. For example, the first could represent one floor of a building, while the second represents two floors connected by elevators or staircases. Figure 5.16 shows the topology and configuration of the second environment.

In each room 1 robot navigates to a randomly chosen neighbouring room. All navigation requests are created simultaneously. The traffic intensity is defined as the number of conflicting or shared corridors with respect to the total number of navigation requests. A corridor is conflicting or shared if two robots would allocate it when they would execute their optimal trajectory. In this case, this corresponds to how many times two robots swap rooms with respect to the total number of navigation requests.

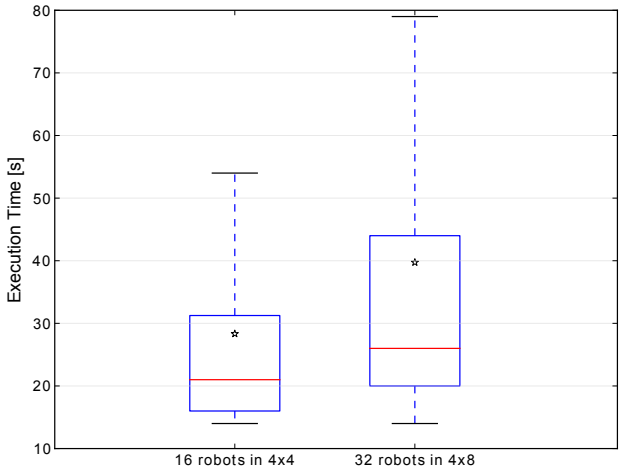


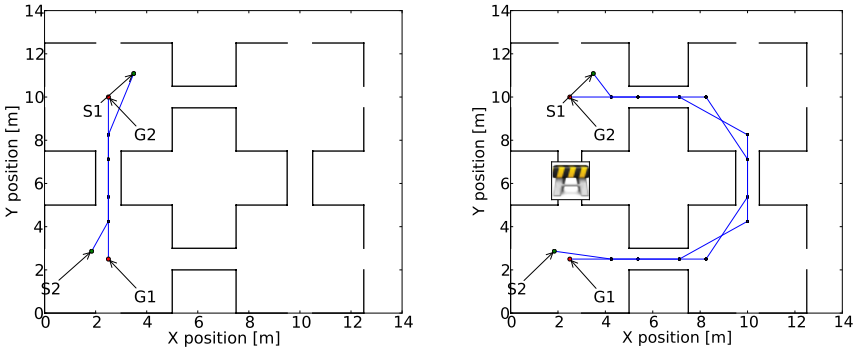
Figure 5.17: Box plots of navigation durations in seconds in both environments for 8 runs in each environment. The box defines the interquartile range ($IQR = Q3 - Q1$), while the whiskers are positioned at $Q1 - 1.5IQR$ and $Q3 + 1.5IQR$ respectively. The average value is displayed by a $\star$ symbol.

In both environments 8 runs are performed and the execution times are compared. The execution time is defined as the difference between the creation time of the navigation requests and the navigation finish time. In Figure 5.17 two box plots are depicted, showing the differences between the two environments. The box defines the interquartile range ($IQR = Q3 - Q1$), while the whiskers are positioned at $Q1 - 1.5IQR$ and $Q3 + 1.5IQR$ respectively. The average value is displayed by a $\star$ symbol.

These box plots show that the majority of the navigation requests in the larger environment are executed with a similar duration. The median and average values, $21.0s$ and $28.34s$ for the small environment, and $26.0s$ and $39.73s$ for the large environment do not differ that much.

However, a Wilcoxon rank sum test rejects the null hypothesis that two sets of measurements are drawn from the same distribution with a z-score of -4.82 and p-value $1.43e-06$. This indicates the HMES performance decreases when increasing the environment, though this decrease is rather small.

One of the reasons of this inferior performance is the exponential increase of possible paths the ants are able to take when the environment increases.



(a) Road free (b) Road blocked

Figure 5.18: Trajectories are adapted to cope with disturbances, such as a road block. Robot starting positions are indicated by $S1$ and $S2$ and robot goal positions by $G1$ and $G2$.

Especially in these test environments, where each room is connected to three or four corridors, with the exception of the corner rooms.

The introduction of guidance markers, such as colour code routes, could improve the time exploring ant require to find a valid solution. Nevertheless, the parallel execution of the order holons and their D-MAS and the distributed planning among the resource holons show that also in larger environments the performance is still acceptable.

5.5 Robustness against the unexpected

To asses the robustness, an setup of 4 rooms and 4 corridors is used, in which 2 robots have to swap rooms. During execution, the corridor connecting the two rooms, where the robots start from, will become unavailable. This disturbance represents, for instance, a road block at a certain time with undetermined end time.

In Figure 5.18, the effect of this road block with respect to executed trajectories is depicted. In Figure 5.18(a), the corridor is available and both robots choose this route over the longer detour. Note that, since these robots share the corridor, one robot waits for the other one before entering. In Figure 5.18(b), however, the corridor is blocked and both robots choose for the detour.

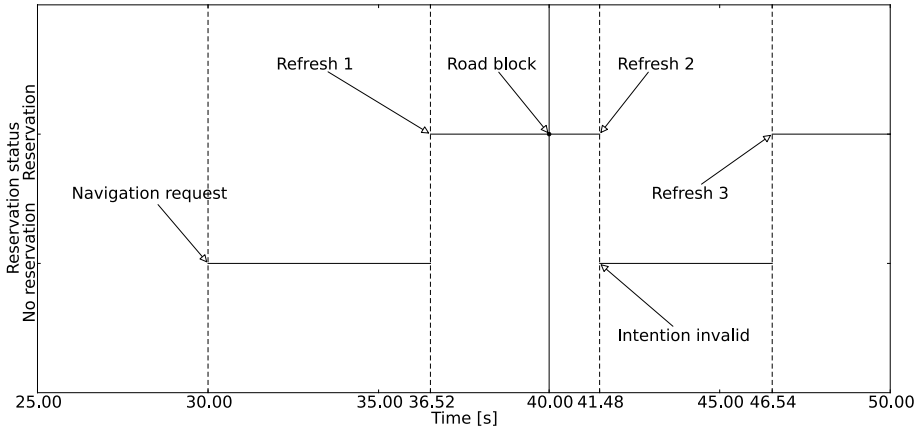


Figure 5.19: Effect of an unexpected road block on corridor reservations for one of the robots. In this case, the disturbance is taken care of within 2 refresh intervals.

Figure 5.19 shows the effect of this unexpected road block on reservations for one of the robots. The navigation request is sent at $t = 30$. At this time, the robot starts sending out exploring ants. The time between two successive evaluations of received solutions is determined by the order holon's refresh rate, which was set here at once every 5 seconds. At $t = 36.52$ the order holon refreshes for the first time, i.e. it chooses the best solution, that time, with respect to its selection criteria.

The corridor becomes unavailable at $t = 40$ and remains blocked indefinitely. The exploring ants, which are sent out every second, detect this disturbance and during the next refresh, at time $t = 41.48$, the reservation is cancelled. Since by that time no suitable alternative was found yet, it takes until the third refresh interval, at time $t = 46.54$, to make a new reservation and take the detour.

The recovery time, i.e. the time between the start of the road block and the selection of a detour, depends on when the disturbance occurs and how fast the order holon explores and refreshes. It also depends on how many disturbances might occur on the alternative paths and how many alternatives are possible. However, this experiment shows that the continuous use of D-MAS by the order holons allows to cope with unexpected disturbances.

5.6 Current implementation limitations

This section discusses the limitations of the current HMES implementation with respect to robot applications. Also, possible solutions to overcome these limitations are proposed.

5.6.1 Multi-resource allocation

Multi-resource allocation can be defined as a situation in which at a particular time instant the order holon is required to do two operations on two different resource holons. The execution of two operations simultaneously is not problematic. The challenge lies in finding and allocating those resources.

Applications, which require multi-resource allocation, typically have resources moving through the environment. For instance, if part of a robot, moving through rooms and corridors, also needs to be allocated, both the robot resource and the room resource require allocation at the same time.

The challenge here is that not only time, i.e. during which time intervals do which resources need to be allocated, but also space, i.e. where do these (robot) resources have to be, becomes a factor in the exploration of solutions. Indeed, the allocation of a robot resource for a particular time interval is not sufficient. This robot also has to follow a particular path in the environment.

Sharing a sensor between two robots is an example of multi-resource allocation. Allocation of this sensor is not necessarily the problem, but finding a sensor at the right time and pointing in the right direction is more difficult.

The current implementation does not support multi-resource allocation, but a possible solution in provided in the following paragraphs.

First of all, the order holon has to be notified that multiple resources required at the same time instant. This information is typically provided by the product holon. This product holon either provides a complete set of possible combinations of resources with the desired capabilities or only supplies the order holon with the desired capabilities. In the latter case, the exploring ants of the order holon have to find the relevant resource holons.

More challenging is finding the right set of resources to allocate. The exploring ants have to be able make reservations on multiple resources at the same time and, thus, be able to evaluate if the schedule of both resources fits in the global solution. One of the determining factors here is the duration of each operation on each resource. In the current implementation, exploring ants

virtually execute an operation on the model of the resource holon to discover how long this operation would take on that particular resource. Since there are now multiple resources with multiple operations, these models should be executed simultaneously. Moreover, if one resource would influence another, the relevant state changes between those resources should be taken into account.

The current implementation offers this functionality to some extent, since the emulation of the real world works in a similar way. The extension from executing one model on one entity to a complete microsimulation of a set of entities could be implemented the same way the emulation is currently implemented. The difference here is that exploring ants reason on the near future and knowing which data or state information at which virtual time instant is required where is difficult. This functionality is currently only supported for individual resources in their scheduler.

Another possible solution to these challenges is the introduction of application specific staff holons. These holons could maintain a list of which robot will be where at which time and provide the exploring ants with this information.

Currently, another researcher at PMA/KU Leuven regarding the use of the HMES in logistics is investigating this problem of multi-resource allocation.

5.6.2 Continuous space

The current implementation maintains a resource graph in which resources are vertices and connections between resources are edges.

Although these connections can have coordinates representing physical locations in reality, such a connection location is represented by a single point. In the multi-robot navigation scenario, connections between resources typically represent doorways or openings leading from one room or corridor to another and, thus, span of a particular area in space.

Currently, at resource connection level, this continuous space cannot be represented. Nevertheless, for the application scenario in this thesis the introduction of a continuous environment in which the robots operate was required. The choice was made to let PROSA and D-MAS handle global coarse-grained path planning and let the robots use their on board obstacle avoidance algorithms to allow local deviations from a proposed path.

This also shows how PROSA can be integrated with existing robot software and the multi-robot path planning problem was even further reduced by distributing the path planning among the rooms and corridors in the environment, which were represented in PROSA as resource holons and required explicit allocation.

Although this structural decomposition simplified the planning problem, another challenge was how to make a transition from one room to the other without blocking the doorway for other robots. Transition zones were defined near the doorways where robots could temporarily reside. Whenever a robot, previously occupying a narrow corridor, left, the robot which was stationary in the transition zone could continue its path.

However, the resource holons could be largely improved by integrating a local multi-robot path planner. Since each room would have its own planner, the computational complexity would remain low and this way the resource holons could specify a free spot to any robot requesting to pass through. The local multi-robot path planning would ensure the provided positions are free at the time to robot would reside there. Also, the resource holon could manage how many robots are still allowed in its room or corridor by taking the size of the robots into account.

Moreover, rules could be introduced in which entering traffic stays to the right and exiting traffic stays to the left of the room or traffic moving in the same direction could be allowed to follow each other if allocations are within a certain time span.

Investigating how traffic rules and such local multi-robot path planners improve performance in terms of throughput or occupancy, for example, is a challenging task. Nevertheless, PROSA offers the building blocks to implement such concepts in a robust and scalable way.

5.6.3 Safety supervision

In every robot application, safety should be an important factor. Typically industrial robots have watchdog or supervisor components, which are able to intervene whenever incorrect values are sent to one of motors.

In the current implementation of the HMES, these safety checks and supervisor components are assumed to be part of the machine or robot software to which operation requests are sent. More concretely, if the coordination software would send a velocity request to one of the motors of the robot, this request is not checked against the boundaries of safe operation. Rather, the HMES assumes the robot control software takes these boundaries into account.

A possible solution is to define hardware and application specific boundary checks and, for example, access control in the dispatching layer of the HMES. Here the translation of scheduled operations on resource holons to sending actual commands to the robot hardware is made. To be able to take more

complex situations into account, for instance, if executing two operations on two resources separately is fine but executing those operations simultaneously causes a problem, the dispatching layer has to be redesigned. For example, in such a situation, resource models are required in the dispatching layer to identify contradicting or counterproductive operations.

5.6.4 Execution speed

Originally, the HMES was developed for coordinating manufacturing system and this coordination is performed at a high level of abstraction. In other words, execution times, both in real and virtual time, range for a few minutes to hours and even days.

Therefore, the implementation was not optimised with respect to execution speed nor does it provide hard real-time solutions. A hard real-time application guarantees that its execution is finished before a predefined deadline.

The current implementation is, thus, not suited for robot applications in which the coordination has to meet hard deadlines. However, the holonic concepts introduced by PROSA and the short-term forecasts provided by D-MAS are also valuable in hard real-time robot applications.

A possible solution in this specific range of application would be to implement the holonic concepts in lightweight components, in a dedicated and optimised way for the specific application in order to meet the deadlines.

5.7 Conclusion

Summary:

This chapter provided, through a set of different simulation experiments, a thorough analysis of the current PROSA and D-MAS implementation, the HMES.

Firstly, tunable HMES parameters, such as exploring rate, refresh rate and hop limit, were discussed and their influence on the system performance was explained.

Secondly, a complexity analysis showed the relation between these parameters and the overall system complexity. This analysis showed the order holon contributes more to the overall complexity than the resource and product holon and one of the determining factors, in worst case, is the ant hop limit.

Thirdly, scalability experiments demonstrated how the HMES copes with a larger set of navigation requests, on the one hand, and a larger environment, on the other hand.

Fourthly, robustness against the unexpected was illustrated by blocking one of the corridors in the environment.

Finally, the current implementation limitations were highlighted and possible solutions were provided.

Chapter 6

Multi-Robot Coordination in Simulation

> *Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.*

Brian Wilson Kernighan(⋆1942–)

The experiments and results on multi-robot navigation have been published in:

J. Philips et al. [2012]. "Scalable and Robust Coordination of Multiple Mobile Robots using PROSA and delegate MAS". In: *Proceedings of the International Symposium on Robotics (ISR)*

6.1 Introduction

In this chapter, results of multi-robot coordination experiments performed in a simulated environment will be presented.

In the experiments, multiple robots navigate individually in an environment, as presented in section 1.4. A comparison is made between an approach without resource allocation and coordination and an approach using PROSA's explicit

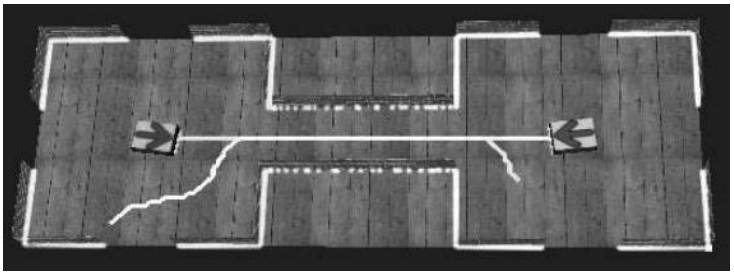


Figure 6.1: Example scenario which can lead to livelocks. Robots are represented by rectangular shapes and their desired paths by a solid line.

resource allocation combined with its delegate Multi-Agent Systems (D-MAS) for short-term forecasts. Benchmarks are proposed to evaluate the performance of both approaches.

These experiments, although performed in simulation, clearly show the need for coordination in a dynamic environment with multiple robots independently moving around. Without coordination, livelocks might occur. Also, the results show that the PROSA and D-MAS approach outperform the reactive approach with respect to the benchmarks defined for these experiments.

6.2 Problem statement

In a dynamic environment in which multiple robots and humans are present and execute, independently from each other, a task, interference between these actors might results in degraded overall task execution or completion.

6.2.1 Livelock and deadlock

In Figure 6.1 only one corridor connects the rooms and it is too narrow for two robots to cross each other. Obviously, when both robots enter the corridor they will block each other at some time resulting in a livelock.

Similar to a livelock, a **deadlock** is caused when two or more entities wait for each other and do not make progression in their task.

To avoid these situations, coordination is required. The experimental results will show that explicitly allocating these narrow corridors and using the D-MAS short-term forecasts, avoids these livelocks and deadlocks. Only when the

number of robots is too high for the environment to cope with, livelocks or deadlocks can occur. In that case, the reality, represented by the resource holons, is also unable to provide a solution.

This discussion highlights one of the major differences between a classical multi-robot coordination approach and the PROSA and D-MAS solution. Unlike classical, robocentric, approaches, PROSA always tries to reflect the reality the best it can. Only in those cases where this underlying reality has no solution or causes livelock situations, the coordination system reflecting that reality will have the same problem. The best solution is then to indicate there is no solution for the requested navigation tasks.

6.2.2 Suboptimal path execution

Another problem reactive approaches suffer from is suboptimal path execution. Due to their reactive behaviour, robots replan their path frequently. This can lead to suboptimal paths and unnecessary travel.

Figure 6.2 shows an environment in which multiple paths to a target location are possible. Each room is connected to two narrow corridors and, depending on the target, going through one corridor is more efficient than going through the other. This scenario can also cause livelocks, provided the number of robots is high. A more common problem is the ability to execute an optimal path from one room to another. If two robots enter the same corridor and thus are not aware of each other's intention, they will most likely replan their route through the other corridors to still reach their target.

This unnecessary travel, as results will show, can also be minimised through explicit resource allocation and short-term forecasts. Indeed, if robots are aware of which resources, i.e. which rooms and corridors, are still available in the near future, they are able to plan their route more accurately and, more importantly, also fix this route by allocating the relevant resources.

6.2.3 Starvation

The problem of starvation is common in priority based multi-robot path planning approaches, in which each robot plans its path individually, starting with the robot with the highest priority. Lower-priority robots take all paths planned by higher-priority robots into account.

If a higher-priority robot keeps blocking a lower priority robot from reaching its goal, starvation occurs. In Figure 6.3, such a starvation situation is depicted.

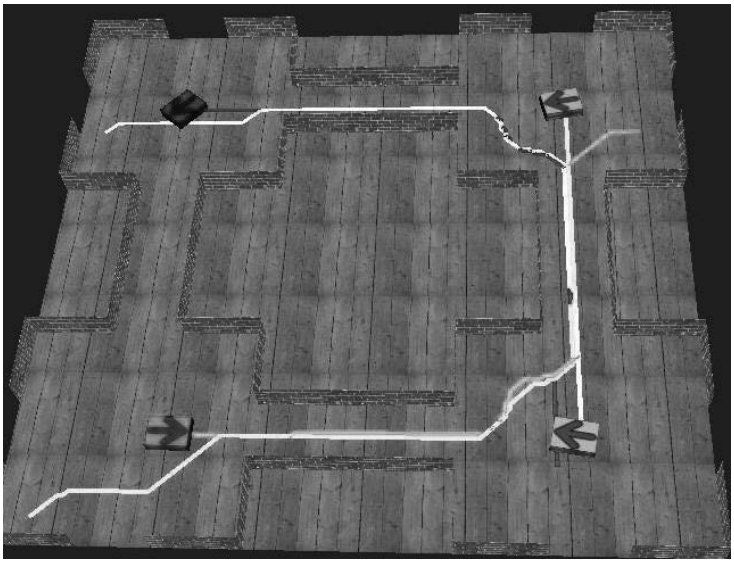


Figure 6.2: Example scenario which can lead to suboptimal path execution. Robots are represented by rectangular shapes and their desired paths by a solid line.

Robots $R1$ and $R2$ have a higher priority than robot $R3$ and each time they block $R3$'s goal. Depending on which targets robot $R1$ and $R2$ receive, robot $R3$ will perpetually be unable to execute its task. A solution to this starvation is to swap priorities from time to time, allowing robot $R3$ to plan and execute its path before $R1$ and $R2$ execute theirs.

6.3 Experimental setup

6.3.1 Environment and scenarios

The experiments were performed in MLRSimulator, a 3D simulated environment developed at PMA/KU Leuven. This simulator provides software-in-the-loop simulation, i.e. control algorithms developed for the mobile robots can be tested in simulation before testing them on the actual hardware. In appendix A, this simulator is discussed in more detail.

The environment consists of a set of rooms connected by narrow corridors, wide enough for only one robot. Robots are represented by a rectangular shape and

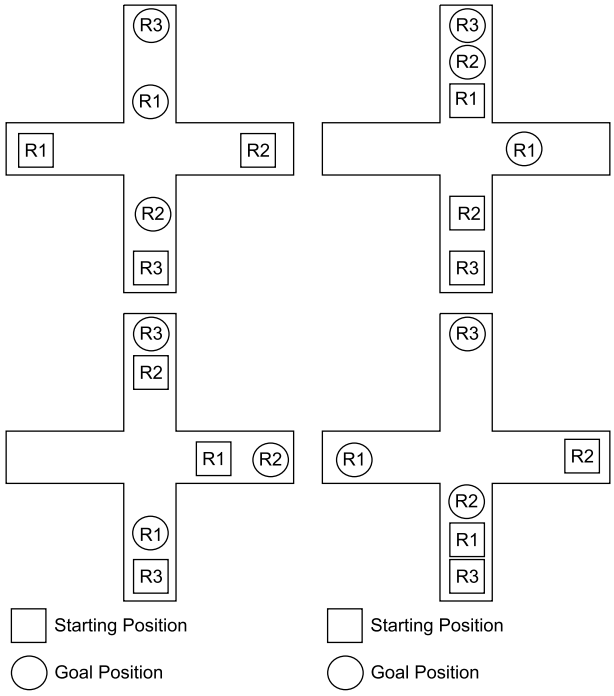


Figure 6.3: Example scenario which leads to starvation in some priority based multi-robot path planning techniques. Robot $R1$ and $R2$ have a higher priority than robot $R3$ and each time block $R3$'s goal. Depending on which targets robot $R1$ and $R2$ receive, robot $R3$ will perpetually be unable to execute its task. A solution to this starvation is to swap priorities from time to time, allow robot $R3$ to plan and execute its path before $R1$ and $R2$ execute theirs.

have a 360 degrees laser scanner with a one degree resolution. Each robot has the same path planner and path execution algorithm on board. When a robot receives a target it plans and starts executing a path to the target. At each time step, the path planner recalculates the path, thus avoiding dynamic obstacles appearing in the environment.

Two scenarios were tested, corresponding to the 2 rooms by 1 room environment shown in Figure 6.1 and the 2 rooms by 2 rooms environment shown in Figure 6.2. For both scenarios 10 runs without and 10 runs with coordination were executed and data such as robot positions, navigation commands and elapsed time were logged for analysis.

During these runs at random times navigation orders were sent to the robots.

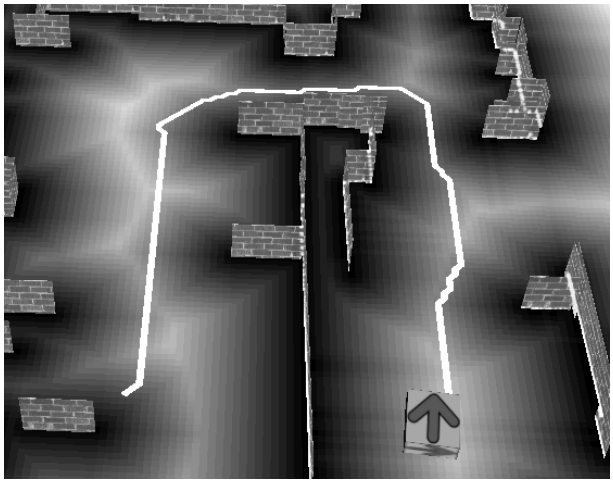


Figure 6.4: Path planner combining a breadth first search algorithm with a cost function based on path length and obstacle clearance. The trajectory is constructed from goal to start position by repeatedly updating the costs in the grid cells representing the environment. In this figure, costs are indicated by a greyscale value. White being the lowest and black being the highest cost. After the lowest-cost path is found a list of way points is calculated. These way points are chosen in such a way that in-between way points no obstacles reside. The thick white line indicates the constructed trajectory the robot will execute.

The same random times used in the experiments without coordination were used in the experiments with coordination to ensure a proper comparison could be made. In order to interpret the data gathered during the experiments, a set of benchmarks was defined. Section 6.6 discusses the chosen benchmarks for these experiments.

6.3.2 Path Planner

The path planner used in these experiments combines a breadth first search algorithm with a cost function based on path length and obstacle clearance. Figure 6.4 illustrates the search algorithm. Grid cell costs are indicated by a greyscale value. White being the lowest and black being the highest cost. The thick white line indicates the constructed trajectory the robot will execute.

After the lowest-cost path is found a list of way points is calculated. These way points are chosen in such a way that in between way points no obstacles

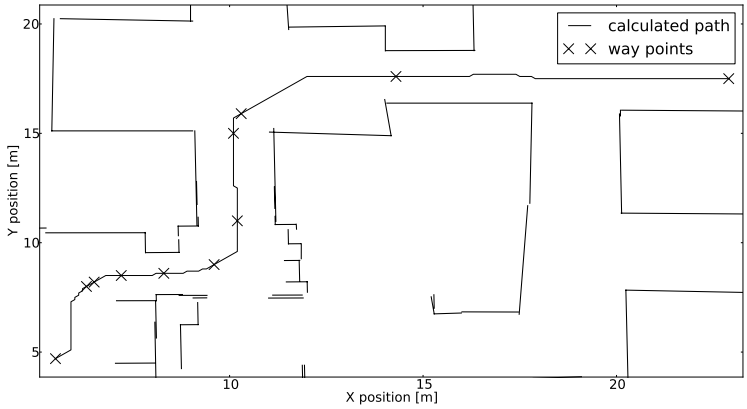


Figure 6.5: The calculated path is divided into way points, indicated by crosses. These way points are chosen in such a way that in between way points no obstacles reside and a minimum clearance from obstacles is maintained.

reside and a minimum clearance from obstacles is maintained. In Figure 6.5, an example trajectory and its way points is depicted.

6.4 Reactive approach

The PROSA & D-MAS coordination is tested against a purely reactive approach in which each robot navigates from its start to its goal position without taking the intentions of other robots into account. All simulated robots are identical with respect to physical dimensions, kinematics and dynamics. Also, each robot employs its own, but identical, path planner to calculate the trajectory and trajectory controller to execute this trajectory.

Each time step, the path planner recalculates the robot's trajectory, based on its current sensor measurement, estimated position and environment map. This newly calculated path is then used by the trajectory controller to determine the robot's next translational and rotational velocity (v and ω).

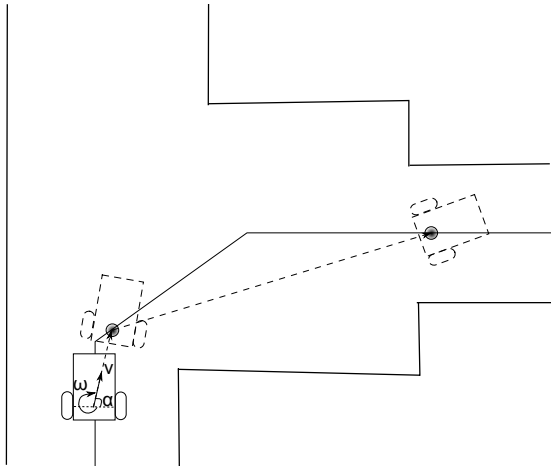


Figure 6.6: The trajectory controller in the reactive approach determines the angle (α) between the robot's current position and the first way point (circles) in the path, calculated every time step by the path planner. The robot rotates in place (ω) until aligned with the way point and moves forward (v) once aligned.

6.4.1 Trajectory controller

In these experiments, (v, ω) tuples were calculated by determining the angle (α) between the robot's current position and the first way point in the path. The robot rotates in place (at a constant, predefined rotational velocity ω) until aligned with the way point and, once aligned, moves forward (at a constant, predefined translational velocity v).

Since each time step the robot's trajectory is recalculated, continuously moving towards this first way point is sufficient to reach the goal position and avoid obstacles during navigation. In Figure 6.6 this *turn & move* behaviour is depicted.

6.5 PROSA & D-MAS approach

6.5.1 Trajectory controller

In the PROSA approach the same path planner is used to calculate the trajectories and a straightforward trajectory controller was used. Similarly

to the reactive approach, the angle between the robot's current position and the next way point is calculated and used to determine the in-place rotation. As long as the robot is aligned toward the next way point it moves forward.

The path planner also recalculates the path at each time step. The difference with the reactive approach is the time at which the navigation request is actually executed. The coordination by the HMES ensures the robot only navigates through a narrow corridor if it is allocated.

Communication between the HMES and the simulated robots is done by the communication interface discussed in 4.3.2 and appendix C.

6.5.2 Resource holons

For each room and narrow corridor in the environment a resource holon has been defined. The room resource holons have a high order capacity, while the corridor resource holons have an order capacity fixed at one. This way only one order at a time may reside on a corridor resource holon, reflecting the policy that only one robot may enter a corridor at a time.

To avoid blockades at doorways between rooms and corridors, transit zones were defined to which robots would first move before entering a corridor. The influence of these transit zones on navigation performance is discussed in the results section.

In this experiment it is sufficient to model robots moving through the environment as orders moving from one resource to the other. Doing so, no distinction is made between the navigation request of the user and the allocation of the robot best fitted to execute this request. The goal of this experiment is to highlight the need for coordination in the actual navigation and therefore abstraction was made of searching a well-suited robot.

In Figure 6.7 the multi-model of a narrow corridor resource holon is depicted. After initialisation, the resource waits for operations to execute. Two types of operations are supported, prepare operations and navigate operations. Prepare operations are used to indicate the resource has to get prepared before the operation requested by the order holon can be executed. Examples are pre-heating an oven or setting up a device. In these experiments no prepare operations were used.

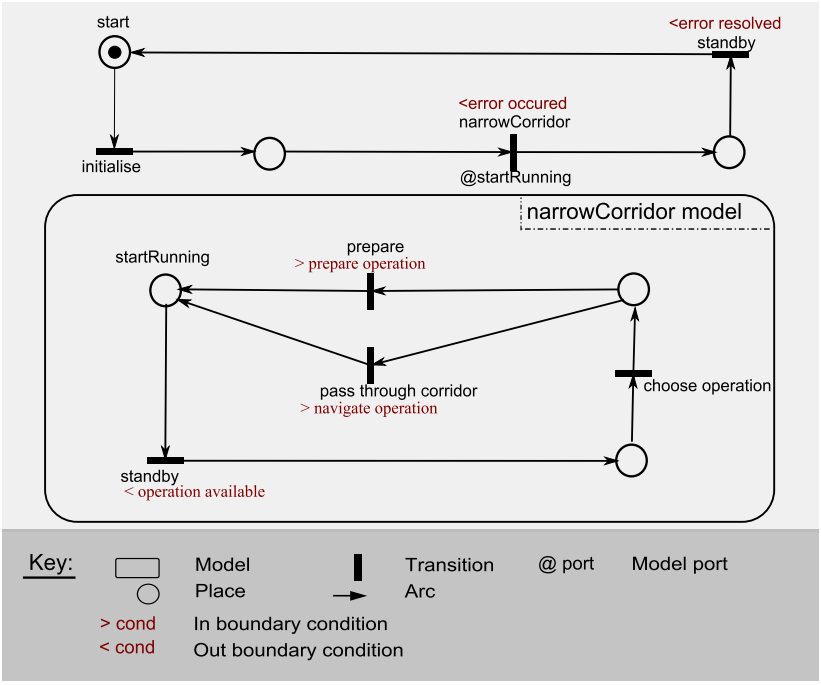


Figure 6.7: Multi-model of narrow corridor resource holon. After initialisation, the resource waits for operations to execute. Two type of operations are supported, prepare operations and navigate operations. Prepare operations are used to indicate the resource has to get prepared before the operation requested by the order holon can be executed. In these experiments no prepare operations were used.

6.5.3 Product holons

In this experiment only one product holon is created. This holon contains the process plan to navigate from one room to another. A sequence of operations was used as process plan, of which the first operation has to be executed on the resource where the robot starts and the last operation on the resource where the the robot ends. Between those two operations any number of other operations may be scheduled on any resource.

The delegate MAS of the order holon explores the different possibilities and eventually this order holon chooses a solution based on a set of criteria, such as shortest travel time and shortest path.

6.5.4 Order holons

The navigation requests, containing a navigation task of a given robot from one room to another at a particular time, are represented by order holons. Both runs without and runs with coordination used the same set of randomly chosen navigation requests.

For each navigation request, exploring ants move through the resource holons and request a virtual execution of the next operation, listed in the process plan. The ants choose their next resource holon at each junction in the resource graph randomly and try to find a solution matching the process plan. If a solution is found, the ant returns this solution to the order holon, which then chooses the best one, based on predefined selection criteria.

In this experiment, not only the time required to execute the process plan but also the number of *hops*, i.e. operations on different resources, was minimised.

If a solution is chosen, the intention ants make the relevant reservations on the resources, which initiates the actual navigation operation of the corresponding robot in simulation.

6.6 Benchmarks

The benchmarks conducted in these experiments are not intended to be exhaustive. The message here is that for each benchmark, chosen by the designer of the experiment, the PROSA approach clarifies which set of holons require adaptation to optimise that benchmark. For instance, in these navigation experiments resource holons representing narrow corridors in the environment are adapted to allow only one robot at a time.

6.6.1 Excessive distance

Since the experiments involve multiple robots navigating from one point to another, a straightforward benchmark is the travelled distance of a robot and compare this travelled distance with the optimal distance, i.e. the originally calculated route between its start position and its target. The benchmark thus measures, in meters, the extra distance a robot had to travel to reach its goal.

The longer this distance, the worse its path execution was. Both an absolute value in meters and a relative value in percentages is provided.

6.6.2 Elapsed time

Another measure that was chosen is the elapsed time navigating in the environment. This time represents the total time of the navigation task, including waiting time in the coordinated approach due to allocations, which are not granted yet. The shorter the time interval, the more efficient the robot has reached its target.

6.6.3 Hausdorff distance

Besides the above two benchmarks, a more advanced method to compare the executed path to the original path was used, namely the Haussdorf distance. This distance calculates the maximum deviation between two paths with respect to the provided metric. In these experiments two metrics were calculated. A Euclidean metric, which returns the shortest path from one trajectory to the other and a metric that measures rigid body motion, based on [Martinez et al. 1995], which also takes rotations into account. More detailed information on how these metrics are defined can be found in [Hüntemann 2011]. Intuitively, they can be interpreted as follows. A high Hausdorff distance using the Euclidean metric indicates that the robot's executed trajectory differs highly from the original path. Including the rigid body motion metric, a higher distance also indicates the robot made more turns than necessary to complete its path, which suggests it oftenly changed its path.

6.7 Results

6.7.1 Without coordination

Figure 6.8 and Figure 6.9 show two common problems of the robocentric approach, as were introduced in section 6.2. The former shows a suboptimal robot trajectory. At the beginning of the run the corridor to the left was blocked by another robot. This was detected by this robot's laser scanner. Since no information about how long it will remain blocked is provided, the robot plans a path around the obstacle through the rooms below. However, after some time, the robot which was blocking the corridor enters the room, freeing the corridor. This triggers the path planner of the other robot to change its route and calculate its path through the right corridor. All travelled distance to the south could have been avoided by waiting until the corridor was unblocked.

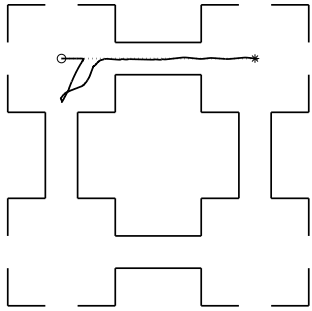


Figure 6.8: Executed trajectory of a robot without coordination. The lack of information and intention awareness causes the robot to move down first and then heading back to take the corridor to the right. The original path is indicated by a dashed line, the executed path by a solid line.

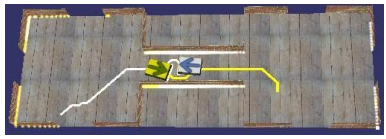


Figure 6.9: Both robots enter the same corridor and fail to reach their target. This results in a livelock, i.e. continuously blocking each other in the corridor without finding a solution. Robots are represented by rectangular shapes and their desired paths by a solid line.

Figure 6.9 shows a livelock situation which can occur if two robots enter the same corridor and no alternative route is possible due to a restricted environment. Again this problem could have been solved by ordering one of the two robots to wait until the other robot has exited the corridor. The more robots are navigating in an environment, the higher the chances for such a livelock are.

The results of the experiments without coordination are shown in Table 6.1.

In the environment with only two rooms and one corridor robots reached in 3 out 10 runs their target and in all three cases this was due to the fact that the time interval between their navigation order was high enough. This allowed the first robot to already exit the corridor before the second robot reached the entrance. In all the other cases the robots both entered the corridor causing a livelock. An example of a failed and successful run are shown in Figure 6.10. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted of robot 1 and 2 of a failed ((a)-(b)) and a successful ((c)-(d)) run. The deviations in the successful run are caused by one robot

Environment	2 by 1	2 by 2
Nr of Robots [#]	2	4
Nr of Runs [#]	10	10
Target Reached [%]	30%	100%
Absolute Excessive Distance [m]	1.060	1.142
Relative Excessive Distance [%]	10.40%	8.86%
Elapsed Time [s]	58.43	80.34
Hausdorff (Euclidean) [m]	3.813	1.795
Hausdorff (Rigid Body Motion) [m]	4.107	2.823

Table 6.1: Benchmarks of runs without coordination.

avoiding the other.

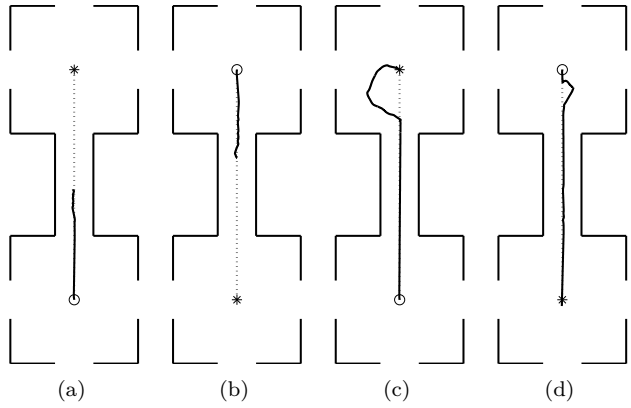


Figure 6.10: Simulation runs of 2 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted of robot 1 and 2 of a failed ((a)-(b)) and a successful ((c)-(d)) run. The deviations in the successful run are caused by one robot avoiding the other.

The elapsed time and excessive distance benchmarks shown in Table 6.1 only take successful runs into account, i.e. if the robots reached their target. If the target was not reached these benchmarks have little meaning, since elapsed time in a livelock would be infinite and excessive distance negative. Nevertheless, the Hausdorff distance can be interpreted, for runs not reaching the target, as a measure for the distance between the robot's end position and its target. Only taking the successful runs into account results in Hausdorff distances of 0.86 and 1.74 for respectively the Euclidean and Martinez-Duffy metric. These distances are, evidently, lower compared to the unsuccessful runs but still higher than

the runs with coordination. Results with coordination are shown in Table 6.2.

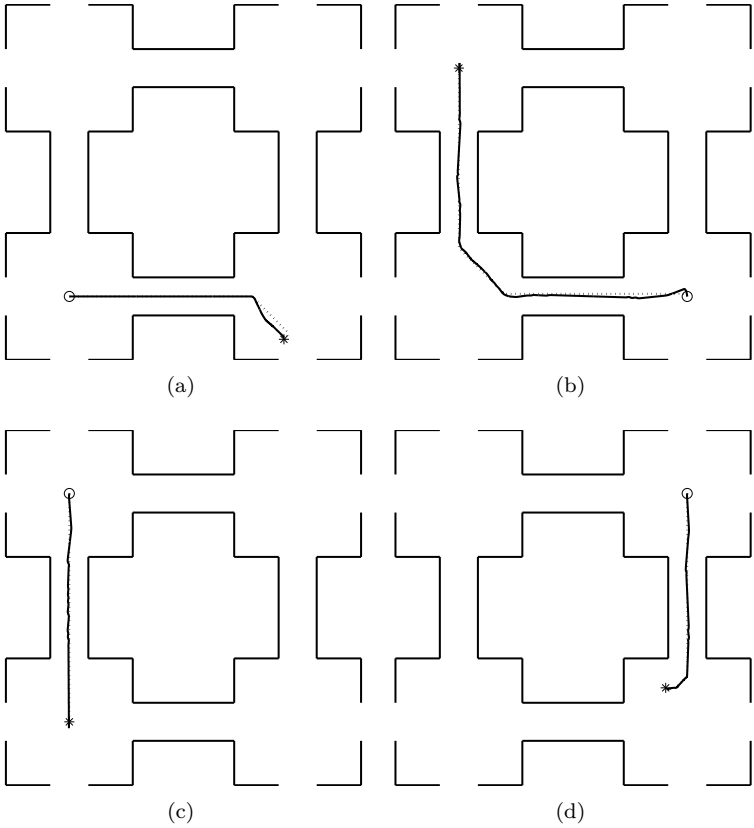


Figure 6.11: Simulation run of 4 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. In this run all robots successfully reach their target without much excessive travelled distance.

The experiments in the small environment with only two rooms and one corridor clearly show the need for coordination at bottlenecks, in this case a narrow corridor used by more than one robot in the environment. How this coordination is done, depends on the implementation of the corridor resource. In these experiments the resource scheduler only allowed one order at a time, thus allowing only one robot in the corridor at once. This scheduler could be replaced by a local multi-robot path planner allowing, for instance, that robots going in the same direction can enter the corridor together. Since this implementation is

separated from the other agents, such a change has little effect on the remainder of the system.

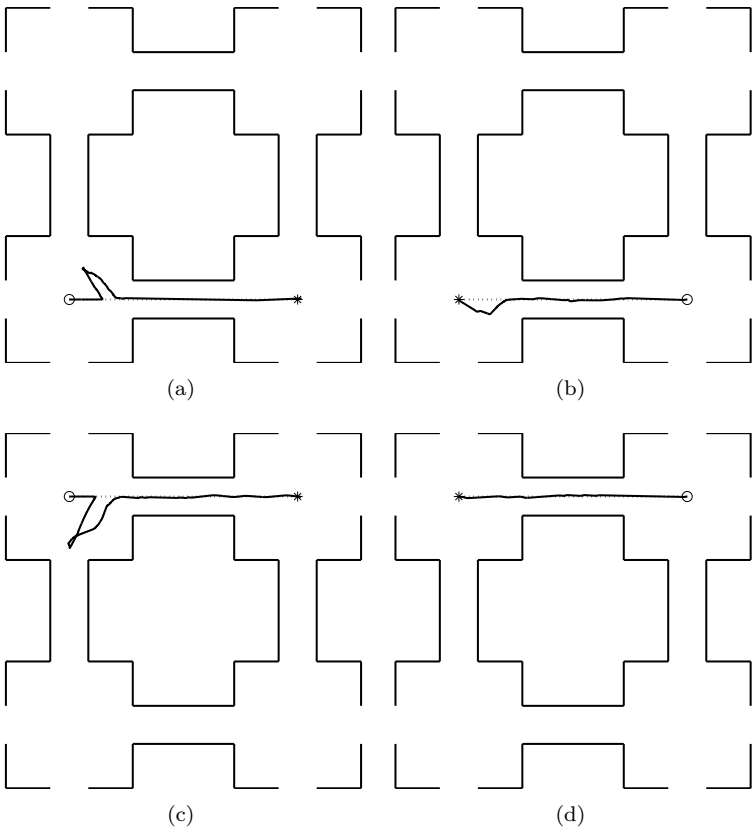


Figure 6.12: Simulation run of 4 robots without coordination. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. In this run robot 1 ((a)) and robot 3 ((c)) both drastically change their trajectory resulting in excessive travelled distance

In the larger environment with four rooms, four corridors and four robots, all targets were reached since at any time if one robot blocks in a specific corridor, a solution can be found using the other corridor. However, this can lead to unwanted behaviour, in which a robot first moves to one corridor, detects an obstacle and then moves to the other corridor or a previously detected obstacle disappears and the newly found shortest path makes the robot change its direction again. In Figure 6.11, a successful run of 4 robots without coordination

is shown, while in Figure 6.12 this unwanted behaviour is depicted.

In the run shown in Figure 6.12, robot 1 ((a)) and robot 3 ((c)) both drastically change their trajectory resulting in excessive travelled distance. Again, planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. This is caused by robot 2 and 4 entering the lower and upper narrow corridor respectively, blocking the passage for robot 1 and 3, which replan their route through the other corridor. Once robot 2 and 4 exit the narrow corridor, the shortest path is available again and robot 1 and 3 change their trajectory again.

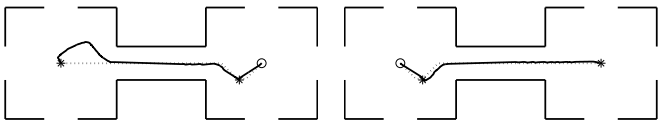
Although in this larger environment the Hausdorff distances are smaller than those in the 2 by 1 environment, compared to only the successful runs of the smaller environment the difference is significant. In most runs one or more robots were *backtracking* and altering their path due to interference from others inside a corridor. This led to high Hausdorff distances.

6.7.2 With coordination

In the experiments with coordination through PROSA and D-MAS, robots are aware of each other's intentions and only enter a corridor after the scheduler of that particular corridor resource has given them a valid slot. This way livelock or backtracking is avoided. Whenever a robot is not able to enter a corridor yet, it waits in a transit zone in front of the corridor's entrance. These transit zones are a chosen in such a way that robots already in the corridors are not blocked by waiting robots during their exit from the corridor. In Figure 6.13 such a trajectory is depicted.

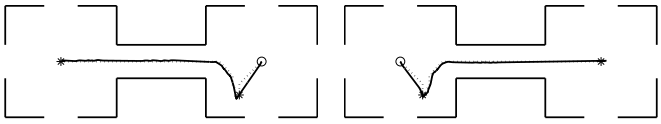
After waiting in the transit zone, the robot has allocated the corridor resource and can move through it. Notice that the original path and executed path, indicated by respectively a dashed and a solid line, almost completely overlap inside the corridor. Since the robot has allocated this corridor resource before starting to navigate through it, it is certain its original path can be executed. In the room resource the robot deviates from its original path to avoid a collision with the other robot waiting in the transit zone. This deviation can be reduced by making this transit zone larger, as is shown in Figure 6.14. The second robot moving through the corridor, depicted in Figure 6.13(b) has no obstacles on its path and thus its executed path and original path completely overlap.

In Figure 6.15 a simulation run of 4 robots with coordination in the 2 by 2 environment is shown. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. The use of transit zones is highlighted by the intermediate targets. The robot navigates from



(a) Robot 1 (b) Robot 2

Figure 6.13: Executed trajectories of two robots with coordination of a shared corridor. The robots first move to a transit zone at the side of the corridor's entrance and wait for an available slot on the corridor resource. Whenever a robot has successfully allocated the corridor, it can move through it. The original path, indicated by a dashed line, and the executed path, indicated by a solid line, almost completely overlap inside the corridor.



(a) Robot 1 (b) Robot 2

Figure 6.14: Executed trajectories of two robots with coordination of a shared corridor, using a larger transit zone. This way the deviation due to collision avoidance is almost completely reduced to zero.

transit zone to transit zone until it reaches its target. Although its path is suboptimal in distance, the execution is guaranteed to be smooth since corridors are allocated beforehand.

The results of these experiments with coordination are shown in Table 6.2. Overall, with properly chosen transit zones, a significant decrease of excessive distance is achieved, both in absolute and relative values. The higher elapsed time compared to the reactive approach is largely due to the slightly longer trajectories and the intermediate stops at the transit zones causing lower travel speed. Also Hausdorff distances are significantly lower than the approach without coordination. Performing a Wilcoxon rank-sum test on the excessive distance data results in a z-score of -4.205035 and p-value of 0.00001. This z-score is lower than the z-score of the one-tailed significance level of 5%, -1.64485, thus the null hypothesis $\mu_{coordinated} > \mu_{reactive}$ can be rejected. Analogously, hypothesis tests on the Hausdorff distances with Euclidean metric and rigid body motion metric results in the rejection of the null hypotheses, with respectively z-scores of -3.06958 and -2.75204 and p-values of 0.00107 and 0.00296, at the same significance level of 5%. It can also be shown that although the mean execution time is higher in the coordinated approach, the

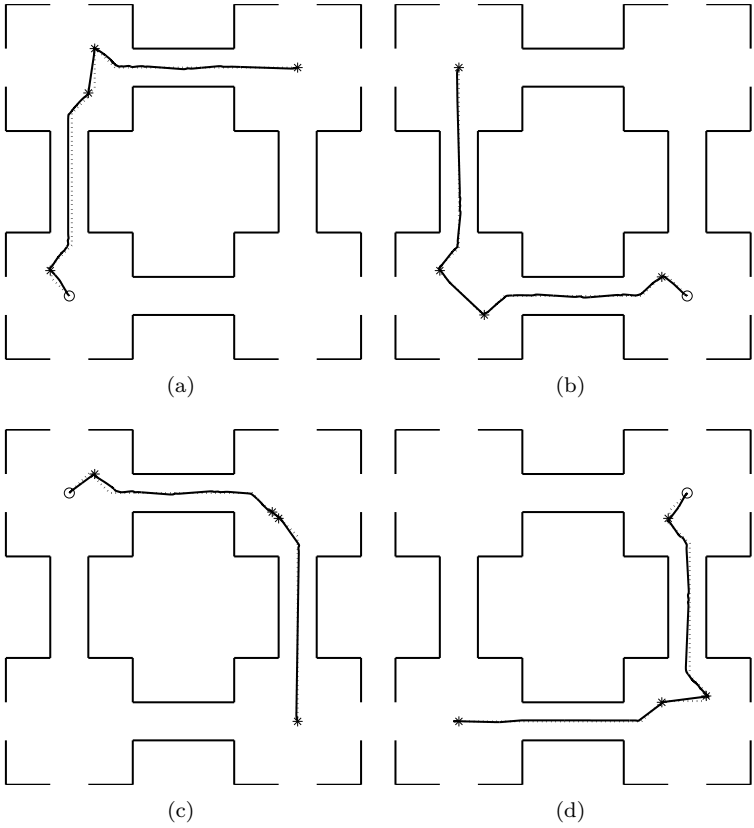


Figure 6.15: Simulation run of 4 robots with coordination through PROSA and D-MAS. Planned (dotted line) and executed (solid line) trajectories from start (circle) to goal (star) are depicted. The use of transit zones is highlighted by the intermediate targets. The robot navigates from transit zone to transit zone until it reaches its target. Although its path is suboptimal in distance, the execution is guaranteed to be smooth since corridors are allocated on beforehand.

difference is not significant, with z-score 0.19245 and p-value 0.42370.

Environment	2 by 1	2 by 2
Nr of Robots [#]	2	4
Nr of Runs [#]	10	10
Target Reached [%]	100%	100%
Absolute Excessive Distance [m]	0.098	0.315
Relative Excessive Distance [%]	0.91%	2.63%
Elapsed Time [s]	66.03	83.42
Hausdorff (Euclidean) [m]	0.239	0.383
Hausdorff (Rigid Body Motion) [m]	1.340	1.514

Table 6.2: Benchmarks of runs with coordination.

6.8 Conclusion

Summary:

The simulation experiments discussed in this chapter illustrated the benefits of resource awareness and making each shared resource in the environment a first-class citizen in the software design. Decoupling this knowledge and placing it at the right software components instead of the robot control system enhances system flexibility and scalability. The planning in a multi-robot application is distributed to these resources responsible for the planning, i.e. allocation, in the physical entity they represent. Also, restrictions on the allocation of particular resources, such as narrow corridors, is applied on these specific components without the need to change other components.
Experimental data in simulation showed a coordinated approach with explicit resource allocation achieves better performance in path execution, measures by the defined benchmarks.

Chapter 7

Coordination of Multiple Robotic Wheelchairs

> *No matter how slick the demo is in rehearsal, when you do it in front of a live audience, the probability of a flawless presentation is inversely proportional to the number of people watching, raised to the power of the amount of money involved.*
>
> Mark Gibbs

7.1 Introduction

In this chapter two robotic wheelchairs at PMA/KU Leuven, used in navigation experiments, are introduced. Also, the autonomous navigation software developed for this multi-robot coordination experiment is explained.

The coordination of HMES and its integration with robot software developed at PMA/KU Leuven is also discussed and an experiment with two robotic wheelchairs, coordinated by HMES and moving between rooms illustrates the approach in a real environment with robot hardware.



Figure 7.1: The robotic wheelchair SARA has mid wheel drive and is equipped with two Hokuyo URG-04LX laser scanners and has Baumer magnetic encoders on both wheels. The joystick module and encoders are read by a National Instruments NI PCI-6225 data acquisition (DAQ) card. The wheelchair control unit, running Ubuntu 10.04, is a mini-ITX board with an Intel Atom 1.6GHz CPU, 1GB memory, on board Intel GPU and a 160GB hard drive. Interaction with the platform goes through the 100Mbps Ethernet card or the USB WiFI receiver.

7.2 Robotic wheelchairs

Both wheelchair control units run the same robot software, developed in C++. This software separates platform independent components, such as algorithms and generic sensor and actuator classes, from platform specific components, such as device drivers. The operating system Ubuntu 10.04 is configured on both systems.

7.2.1 SARA

Figure 7.1 shows robotic wheelchair SARA, Semi-Autonomous Robot Assistant.

This power wheelchair has mid wheel drive and is equipped with two Hokuyo URG-04LX laser scanners and has Baumer magnetic encoders on both wheels. The joystick module and encoders are read by a National Instruments NI PCI-6225 data acquisition (DAQ) card. This card also allows to set joystick values from within the software, e.g. to correct the user's input. Additional analogue channels can be used to read extra sensors such as infra red or ultra sound.

The wheelchair control unit is a mini-ITX board with an Intel Atom 1.6GHz CPU, 1GB memory, on board Intel GPU and a 160GB hard drive. Interaction with the platform goes through the 100Mbps Ethernet card or the USB WiFI card.

7.2.2 INGA

Figure 7.2 shows robotic wheelchair INGA, Intelligent Navigation Assistant. This power wheelchair has front wheel drive and is also equipped with two Hokuyo URG-04LX laser scanners and Baumer magnetic encoders on both wheels. In addition to its standard joystick connected by serial port, it has a haptic joystick. A National Instruments NI PCIe-6251 is used to read in the encoders.

The wheelchair control unit is a ZOTAC Z68-ITX WiFi mini-ITX board with an Intel i5-2405S quad core CPU, 4GB memory, a NVIDIA GeForce GT 430 GPU with 1GB memory and a 160GB OCZ Vertex3 solid state drive. Interaction with the platform goes through the 1Gbps Ethernet card or the on board WiFI card.

The specifications of this robotic wheelchair are much higher, since it is also used in a project which involves online video processing.

7.3 Autonomous navigation

In contrast to previous research on robotic wheelchairs at PMA/KU Leuven, which focused on sharing control between machine and human, autonomous navigation is required for the multi-robot coordination scenario envisaged in this thesis.

Figure 7.3 depicts the interaction of the different software components running on the robotic wheelchairs. Communication between components is message-based using a publish-subscribe methodology. This way the components are not



Figure 7.2: The robotic wheelchair INGA has front wheel drive and is also equipped with two Hokuyo URG-04LX laser scanners and Baumer magnetic encoders on both wheels. In addition to its standard joystick connected by serial port, it has a haptic joystick. A National Instruments NI PCIe-6351 is used to read in the encoders. The wheelchair control unit is a ZOTAC Z68-ITX WiFi mini-ITX board with an Intel i5 CPU, 4GB memory, a NVIDIA GeForce GT 430 GPU with 1GB memory and a 160GB OCZ Vertex3 solid state drive. Interaction with the platform goes through the 1Gbps Ethernet card or the on board WiFI card.

dependent on a specific component but only on particular input data, which might originate from any component.

7.3.1 State estimator

The state estimator collects sensor data from laser scanners and wheel encoders to estimate current robot pose and velocities. Pose estimation in world coordinates is performed by matching laser scan data onto an a priori map of line segments and an initial pose guess. Every time new sensor data arrive, the state estimator updates the robot state.

A Kalman filter is used to keep track of the estimated robot state and its uncertainty. In this case, *state* corresponds to the robot's pose (x_e, y_e, θ_e) and velocity (v_e, ω_e), where (x_e, y_e) corresponds to the global position in a learned map.

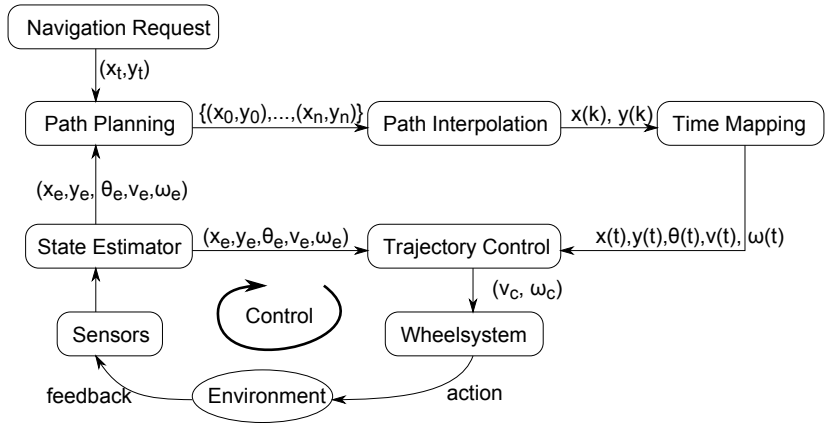


Figure 7.3: Control scheme for autonomous navigation of the robotic wheelchairs.

More information on the state estimator and its implementation can be found in [Demeester 2007].

The estimated state $(x_e, y_e, \theta_e, v_e, \omega_e)$ is then used by the path planner to calculate a path from the robot's current position, and by the trajectory controller to calculate the control errors on position, orientation and velocity.

7.3.2 Path planning

The same path planner is used as in the simulation experiments discussed in chapter 6.

A navigation request is issued either by a user through the robot graphical user interface, cf. appendix B, or the HMES. The request consists of a target position (x_t, y_t) and is sent to the path planner to calculate a path from the robot's current position to the supplied target position.

The path planner uses an a priori map of the environment and optionally laser scan data to plan a path around obstacles in the environment that are not in the map. First, it divides the environment into grid cells with a configurable resolution and calculates an occupancy grid. Then it uses a breadth first search algorithm that finds a path which makes a compromise between path length and other costs such as distance to obstacles.

The result is a set of grid cells (i, j), which is converted into a set of coordinates (x, y). From this set of positions a subset of way points, $(x_0, y_0), \ldots, (x_n, y_n)$ is

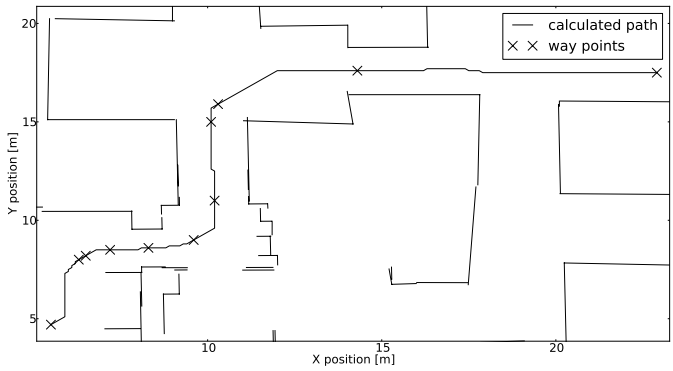


Figure 7.4: The set of 262 positions in the calculated path is narrowed down by selecting 11 way points in this example.

calculated. First the start position of the path is added to this set, then the path is traversed as long as a straight line can be drawn between the current position and the previous way point without colliding with an obstacle. Once a collision is reached, the previous position is added as a new way point.

This results in small a set of positions onto which smooth path interpolation can be applied. More details on the path planner can also be found in [Demeester 2007].

Figure 7.4 shows how a calculated path of 262 positions is narrowed down to 11 way points. Computation time required by the path planner is around 1 to 5ms, depending on the path length and number of grid cells.

7.3.3 Path interpolation

In this step, a set of natural cubic splines is calculated. Each spline goes through two subsequent way points of the calculated path. Piecewise cubic spline interpolation is known for its smooth and stable characteristics [McKinley et al. 1998; Kruger 2002]. Unlike global interpolation, in which one high order polynomial would be fitted through all way points, cubic splines suffer much less from oscillations and overshoot at the way points.

A natural cubic spline is a third degree polynomial continuous up to the second order derivative. For $n + 1$ way points $(x_0, y_0), \ldots, (x_n, y_n)$, the piecewise function $S(x)$ consists of n splines s_i, each taking care of a $x_i < x < x_{i+1}$

segment. Thus, the spline left of way point (x_i, y_i) is denoted as s_i and right of this way point as s_{i+1}.

Each spline s_i can be represented parametrically as:

$$s_i(k) = (X_i(k), Y_i(k))$$

with $i = 1, \ldots, n$ and $k \in [0, 1]$ non-uniform defined along the spline segment [van den Brink 2005]. $X_i(k)$ is chosen in such a way it is only defined by the way point x coordinates and $Y_i(k)$ by the way point y coordinates.

Separating x and y coordinates allows the representation of trajectories crossing the same x coordinate more than once, such as vertical lines and circular paths. The derivation of $X_i(k)$ and $Y_i(k)$ is identical, so only $X_i(k)$ will be discussed here. Thus, the position in $x(k)$ is defined as

$$X_i(k) = a_i^x + b_i^x x + c_i^x x^2 + d_i^x x^3$$

Each $X_i(k)$ has to meet the interpolation conditions:

$$\begin{cases} X_i(1) = x_i \\ X_{i+1}(0) = x_i \end{cases}$$

And continuity conditions:

$$\begin{cases} X_i'(1) = X_{i+1}'(0) \\ X_i''(1) = X_{i+1}''(0) \end{cases}$$

Finally, *natural* cubic splines have the following border conditions:

$$\begin{cases} X_1(0) = x_0 \\ X_n(1) = x_n \\ X_1''(0) = 0 \\ X_n''(1) = 0 \end{cases}$$

This results in 4 equations and 4 variables for each $X_i(k)$ thus its coefficients a_i^x, b_i^x, c_i^x and d_i^x are uniquely determined. For a complete derivation of this calculation, see [van den Brink 2005] or e.g. [McKinley et al. 1998].

In Figure 7.5 an example of the spline interpolation is depicted. Computation time depends on the number of way points but is very fast, since it only requires to solve a matrix of $4n$ equations and $4n$ variables.

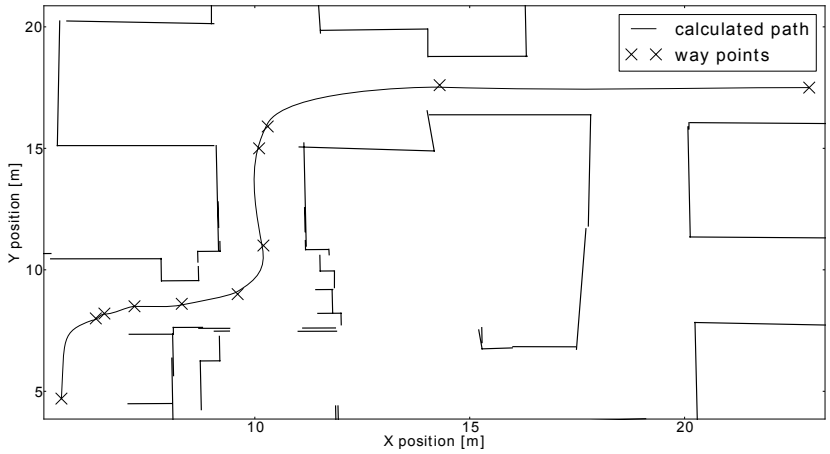


Figure 7.5: Splines connecting each way point to create a smooth interpolation.

Although this approach results in a smooth path, no collision checking is performed along the path and dynamic constraints are not taken into account. The latter will be addressed in the time mapping step, explained below.

Moreover, orientation along the path is not integrated either. To partially compensate for this, the robot will first turn on the spot, if its current orientation is not aligned in the direction of the first spline.

7.3.4 Time mapping

After having calculated the interpolating splines, the obtained path $S(x)$ has to be expressed in function of time to determine which position, orientation and velocity the robot has to reach to at a given time.

The time mapping also applies constraints on velocity and acceleration to meet the robotic wheelchair's physical limitations.

The resulting $x(t), y(t), \theta(t), v(t)$ and $\omega(t)$ are sent to the trajectory controller as desired input.

In the current implementation, an iterative algorithm is used to determine suitable input values, which is not very efficient and can take up to 50ms depending on the path length on the control unit of SARA and 10ms on the control unit of INGA.

7.3.5 Trajectory control

Past research in trajectory control on a different mobile robot at PMA/KU Leuven [Van Dorpe 1997] resulted in a state space controller which is used here to execute the trajectory. As input, this controller receives the desired position, orientation and velocity in function of time from the path interpolation component on the one hand and the robot's current position, orientation and velocity from the state estimator on the other hand. Every control cycle it then compares the desired position (x_d, y_d), orientation θ_d, and velocity (v_d, ω_d) at that time with the current estimates $(x_e, y_e, \theta_e, v_e, \omega_e)$.

Based on these inputs, the controller calculates the error on the robot's position both tangent (e_t) and lateral (e_p) to the given trajectory and the error on the orientation (e_a):

$$\begin{cases} e_t = (y_d - y_e)\sin(\theta_d) + (x_d - x_e)\cos(\theta_d) \\ e_p = (y_d - y_e)\cos(\theta_d) - (x_d - x_e)\sin(\theta_d) \\ e_a = \theta_d - \theta_e \end{cases}$$

Using these errors the required adjusted values (v_{adj}, ω_{adj}) are calculated as follows:

$$\begin{pmatrix} v_{adj} \\ \omega_{adj} \end{pmatrix} = \begin{pmatrix} 0 & 0 & -p_0 \\ -p_2 - p_1 & \frac{p_1 p_2}{v_e} & 0 \end{pmatrix} \cdot \begin{pmatrix} e_a \\ e_p \\ e_t \end{pmatrix}$$

This implies that the correction of e_t is performed by only on v_c and the correction of e_a and e_p by ω_c. Furthermore, the pole p_0 only influences v_c, while poles p_1 and p_2 only influence ω_c.

The control values (v_c, ω_c) then defined as the difference between desired and adjusted values:

$$\begin{cases} v_c = v_d - v_{adj} \\ \omega_c = \omega_d - \omega_{adj} \end{cases}$$

A complete derivation of this controller can be found in [Van Dorpe 1997].

The resulting control values are sent to a lower level PI-controller running at 40Hz. This PI-controller receives feedback from the magnetic encoders, which are read at 200Hz, and controls individual wheel speeds ω_l and ω_r, for respectively left and right wheel:

$$\begin{cases} \omega_l = \frac{2v_c - w_c \ell}{d_l} \\ \omega_r = \frac{2v_c + w_c \ell}{d_r} \end{cases}$$

with ℓ the distance between left and right wheel and d_l and d_r the diameter of respectively left and right wheel.

7.4 Coordination

The HMES coordination is exactly the same as for the simulation experiments. Rooms and corridors are represented by resource holons and robots moving through these resources are modelled as order holons. The product holon contains a simple process plan: a sequence of operations of which the first operation has to be executed on the resource where the robot starts and the last operation on the resource where the robot ends. Between those two operations any number of other operations may be scheduled on any resource.

The narrow corridors or hallways in the environment where the robots cannot easily pass require allocation and allow only one order at a time. Larger rooms or wider corridors also require allocation but have larger capacity, i.e. order holons are allowed to simultaneously allocate these resource holons.

7.5 Integration

Since the planning and control software running on the robot and the coordination software, namely the HMES, are completely separated, the communication interface, discussed in appendix C, is used to integrate both applications.

Keeping these two applications separated has many advantages. It allows independent development on both the robot platforms and in the HMES without interference. Existing software can be reused by connecting the communication interface at the robot side to the events and messages that, for instance, the path planner and trajectory controller send. Moreover, real robots and virtual, simulated robots as well as real environment and simulated rooms and corridors can be combined without having to change anything at the HMES side. All communication between the HMES and the robotic wheelchairs is exactly the same as that sent between the HMES and the multi-robot simulator.

Figure 7.6 shows such an augmented environment in which the existing first floor robot lab and mechatronics lab are combined with virtual rooms and corridors. The HMES can use this map for hybrid executions, since it sends identical navigation commands to real and virtual robots.

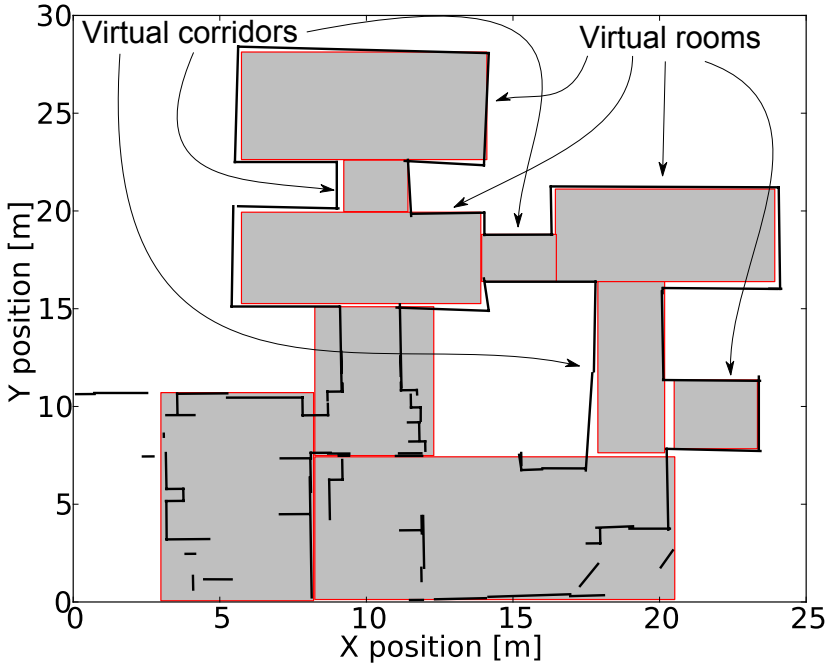


Figure 7.6: Augmented environment, in which the existing first floor robot lab and mechatronics lab are combined with virtual rooms and corridors. The HMES can use this map for hybrid executions, since it sends identical navigation commands to real and virtual robots.

7.6 Experimental setup

Figure 7.7 shows the experimental setup. Two robotic wheelchairs will be coordinated to navigate between robotics and mechatronics lab on the first floor at PMA/KU Leuven. Start positions are indicated by $S1$ and $S2$, goal positions by $G1$ and $G2$. Example trajectories are drawn between start and goal positions and resource holons are indicated by the light grey rectangles. The other black lines represent the 2D lines map used by the path planner.

The holons defined in this experiment are the same as mentioned in section 7.4 and, thus, identical to those used in the simulation experiments.

The HMES parameters were configured as follows: exploring ants are sent out every second, refreshing is done once every five seconds and solutions evaporate after 15 seconds. The hop limit is lowered to 10 since the graph diameter is only 3 in this experiment.

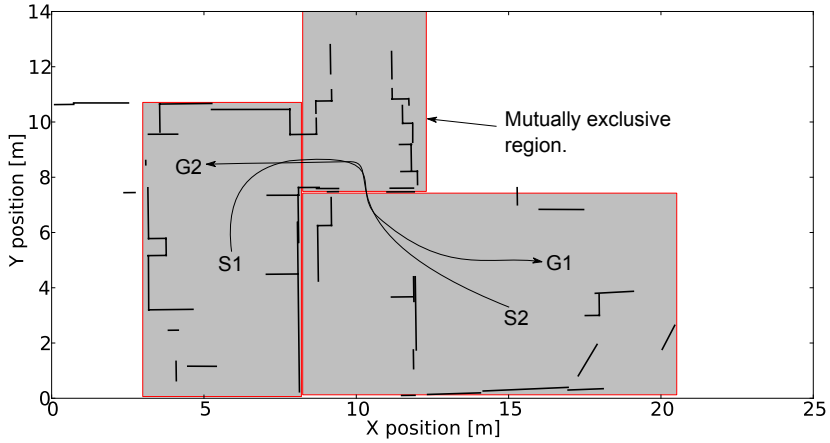


Figure 7.7: Experimental setup in which two robotic wheelchairs are coordinated to navigate between robotics and mechatronics lab on the first floor at PMA/KU Leuven.

Motivation for the low exploring rate is the low average speed of the robotic wheelchairs. Detecting disturbances, that might occur in the environment and affect the resource allocation, is still done fast enough for the robotic wheelchair to react and pursue an alternative solution.

7.7 Results

7.7.1 Trajectory Control

Figure 7.8 depicts a trajectory followed by robot Inga during the tuning process of the state-space controller. Pole values p_0, p_1 and p_2 in this trial were respectively set to -0.8, -0.5 and -0.8. The desired trajectory, generated using the spline interpolation, is depicted by the dashed line, while the actual trajectory followed by the robot, estimated by the state estimator, is depicted by the solid line.

This shows the executed trajectory accurately follows the desired trajectory. To quantify this accuracy, Figure 7.9 plots the corresponding angular, parallel and tangent errors in absolute values.

These errors show a maximum angular error of ± 0.2 radians or ± 11.5 degrees, and maximum parallel and tangent errors of ± 6 cm. The dashed line indicates

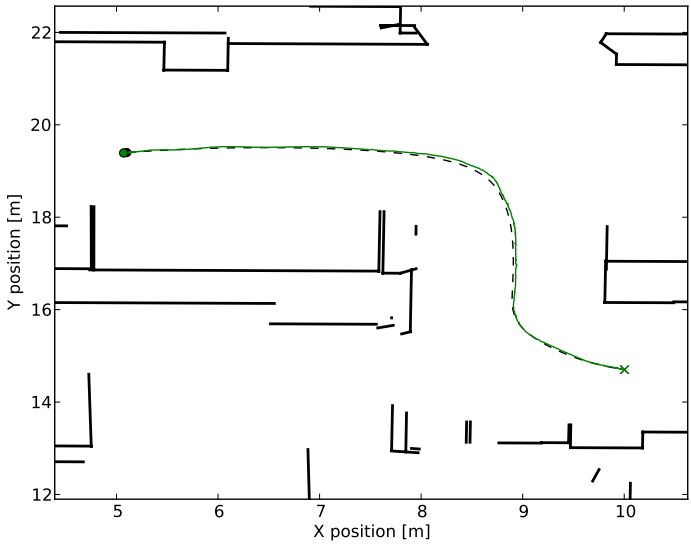


Figure 7.8: Trajectory followed by robot Inga during tuning process of state-space controller. The desired trajectory, generated using the spline interpolation is depicted by the dashed line, while the actual trajectory followed by the robot, estimated by the state estimator, is depicted by the solid line.

the mean of each error and its value is also depicted. On average, the robot remains within 1.6 cm of the desired trajectory and its orientation differs only 3.4 degrees.

The increase of the parallel error after ±30 seconds corresponds to the slightly larger offset the robot had while turning into the doorway, visible in Figure 7.8. However, the state space controller is able to recover from this offset resulting in the much lower parallel error further down the trajectory.

7.7.2 Coordination

During the experimentation period, the test environment often, although only slightly, changed. Since the path planner requires an a priori map of the environment, each time the environment changes, a new map has to be constructed. Data discussed in this section relate to the environment shown in Figure 7.10.

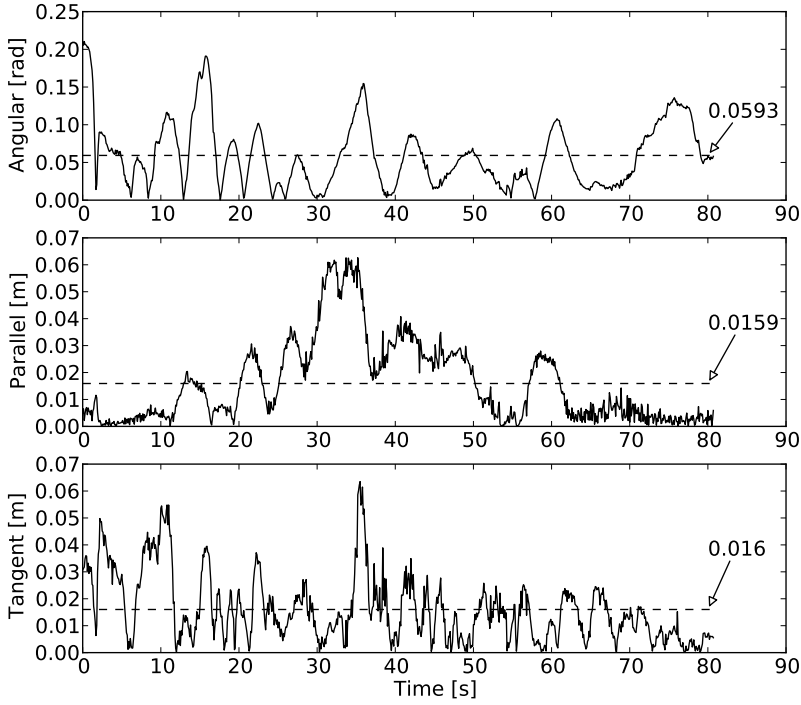


Figure 7.9: Evolution of angular, parallel en tangent control errors in absolute values during the execution of the trajectory depicted in Figure 7.8.

Figure 7.11 shows resource allocations during the coordination experiment. Robot $R1$ represents robotic wheelchair Inga going from $Room0$ to $Room1$, while robot $R2$ represents robotic wheelchair Sara, going from $Room1$ to $Room0$. Both robots first move to their respective transit zones in their starting rooms. In this experiment, $R2$ is granted access to the mutually exclusive resource $NarrowCorridor0$ first. Immediately after $R2$ exits this resource, $R1$ is able to enter. The trajectories are depicted in Figures 7.12, 7.13, 7.14, 7.15 and 7.16.

Figure 7.12 shows the $3D$ plot of the trajectories. The solid lines are the executed trajectories with respect to the time and the dashed lines indicate the corresponding trajectory projected onto the environment. The time dimension shows that the robot starting at the left (Inga) waits until the robot at the right (Sara) passes through the area where their trajectories overlap. The fact that

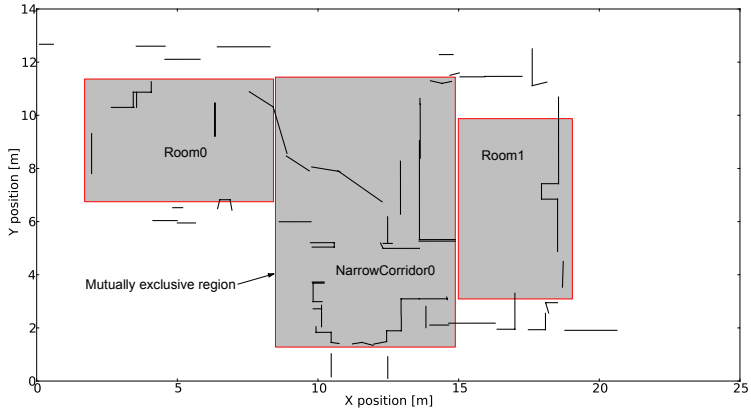


Figure 7.10: The environment used during the coordination experiment in which two robotic wheelchairs changed rooms. Access to the mutually exclusive region *NarrowCorridor0* is coordinated by PROSA and D-MAS.

the solid trajectories do not intersect indicates no collisions occurred.

Finally, Figures 7.13, 7.14, 7.15 and 7.16 show different steps in the coordinated execution. In the first figure, robot $R2$ has reached its transit zone, while $R1$ is still moving toward its transit zone, in the second figure, robot $R1$ has reached its transit zone, while $R2$ is already toward the narrow corridor. In the third figure, robot $R2$ has reached its goal position, while $R1$ is still waiting in the transit zone. Finally, in the fourth figure, both robots have reached their goal position.

7.7.3 Discussion

The experiment demonstrated, on the one hand, the autonomous navigation using the grid-based $2D$ planner, spline interpolation and state-space trajectory controller. On the other hand, the coordination of the two robotics wheelchair by PROSA and D-MAS was demonstrated.

However, it was assumed that within the room resources ($Room0$ and $Room1$) the robots were not able to obstruct each other. In other words, although resource allocation was required, these resources allow multiple allocations at the same time. Thus, no coordination within the same room was applied and both robots could, for instance, move to the same transit zone or area in the

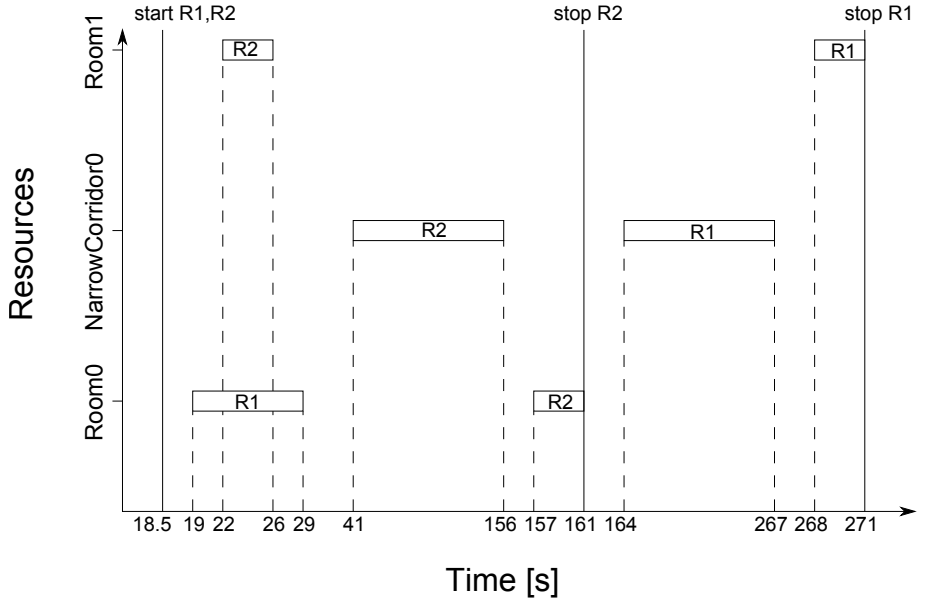


Figure 7.11: Allocations on resources during the coordination experiment. Robot $R1$ represents robotic wheelchair Inga going from $Room0$ to $Room1$, while robot $R2$ represents robotic wheelchair Sara, going from $Room1$ to $Room0$. Both robots first move to their respective transit zones in their starting rooms. In this experiment, $R2$ is granted access to the mutually exclusive resource $NarrowCorridor0$ first. Immediately after $R2$ exits this resource, $R1$ is able to enter.

room, resulting in unsafe situations. Future work should investigate how transit zones or room resources could be better managed. For example, the resource holon could manage which robot is positioned where.

Moreover, during the trajectory execution in this experiment, no obstacle avoidance was applied. Whenever a navigation request was received, a path was calculated based on the a priori map and the current sensor data. In future experiments, to increase safety, obstacle avoidance should be added online. At the robotics division of PMA/KU Leuven such algorithms already exist and have been tested on the robotic wheelchairs during different projects. Therefore, integration of one of those algorithms is a necessary next step.

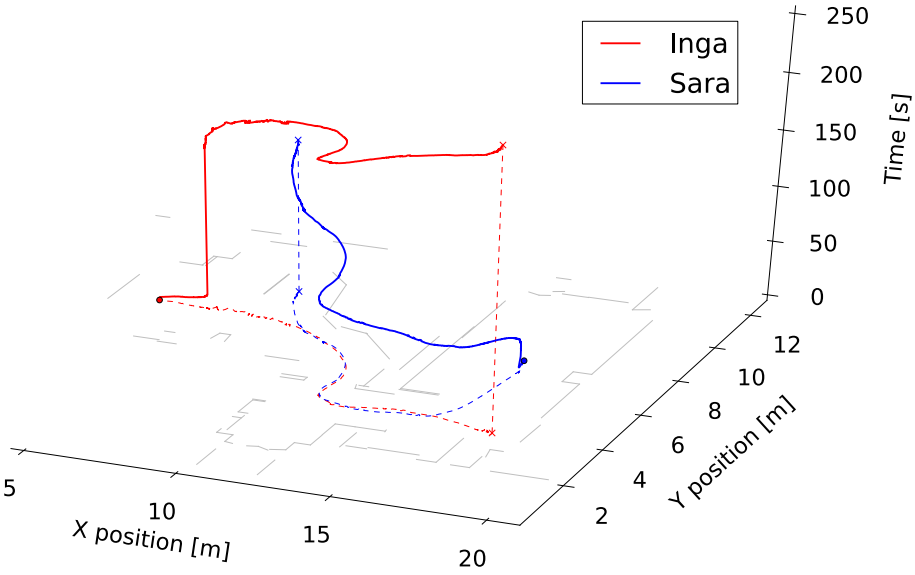


Figure 7.12: 3D plot of the trajectories followed by both robotic wheelchairs. The solid lines are the executed trajectories with respect to the time and the dashed lines indicate the corresponding trajectory projected onto the environment. The time dimension shows the robot starting at the left (Inga) waits until the robot at the right (Sara) passes through the area where their trajectories overlap. The fact that the solid trajectories do not intersect indicates no collisions occurred.

7.8 Conclusion

Summary:

This chapter focussed on the integration of the HMES coordination software and autonomous navigation software running on robotic wheelchairs.

Firstly, the hardware platforms were described and the different software components running on the robots to enable autonomous navigation were discussed.

Secondly, this chapter discussed how the same coordination configuration was used as in the simulation experiments and integration between the HMES and

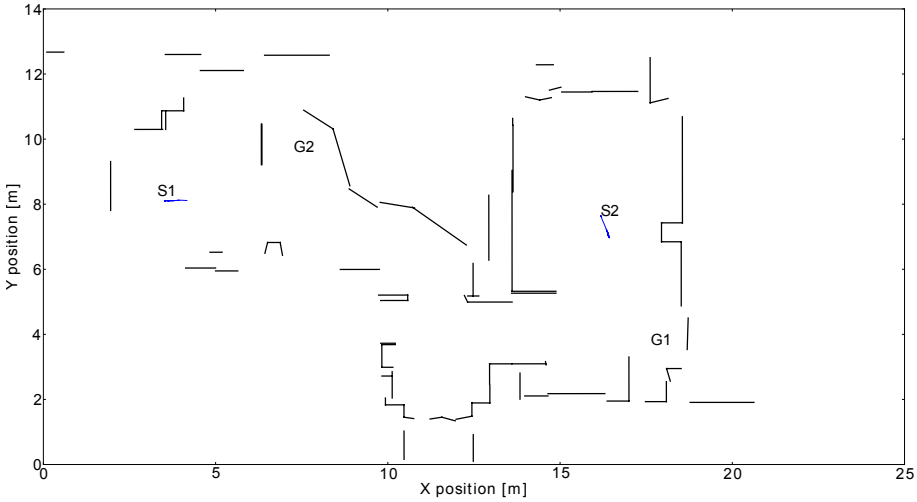


Figure 7.13: Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R2$ has reached its transit zone, while $R1$ is still moving toward its transit zone.

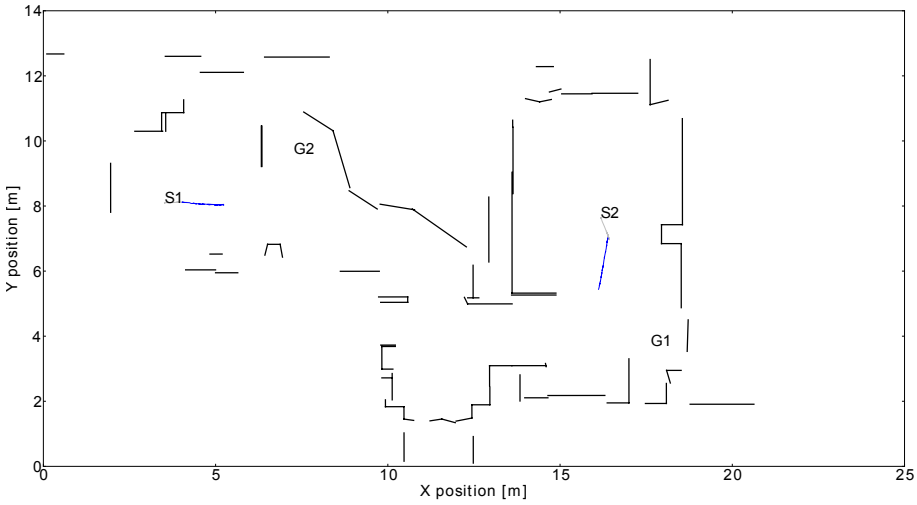


Figure 7.14: Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R1$ has reached its transit zone, while $R2$ is already toward the narrow corridor.

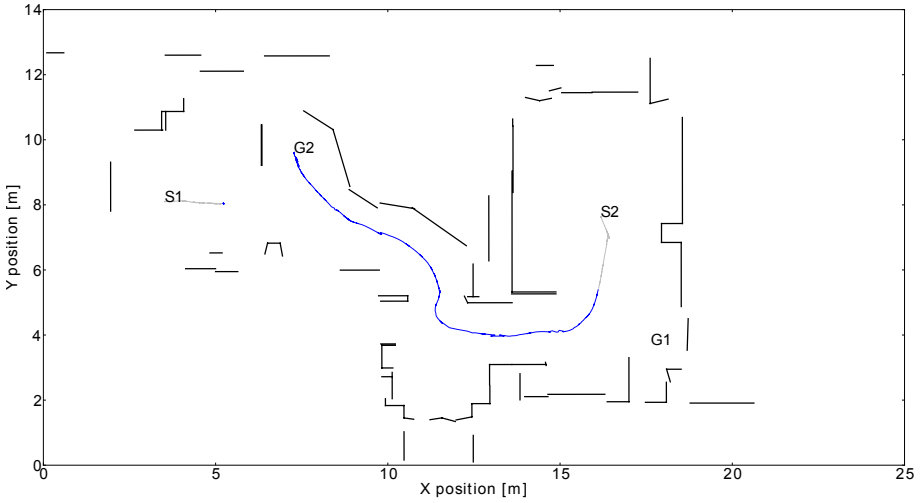


Figure 7.15: Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here robot $R2$ has reached its goal position, while $R1$ is still waiting in the transit zone.

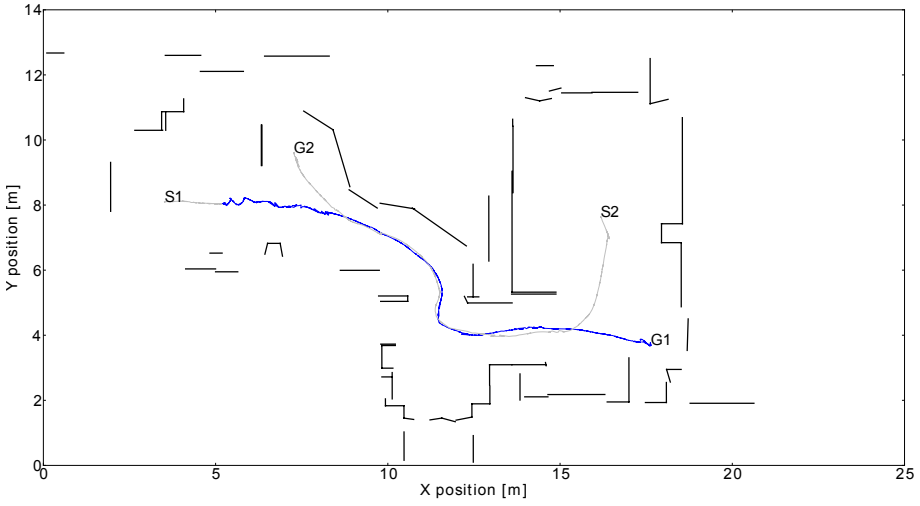


Figure 7.16: Trajectories of both robotic wheelchairs are represented by the solid lines. Start and goal positions of robot $R1$ and $R2$ are indicated by $S1$, $S2$ and $G1$, $G2$. Here both robots have reached their goal position.

robot software was illustrated.

Thirdly, experimental results of two robotic wheelchairs coordinated by the HMES were provided. Data analysis of the trajectory controller showed the positional and angular errors are well within application boundaries on the one hand and allocation of a two rooms and a narrow corridor between the robot wheelchairs was successfully shown.

Chapter 8

Conclusion and Future Work

> “ *Μηδ’ ὕπνον μαλακοῖσιν ἐπ’ ὄμμασι προσδέξασθαι,*
> *πρὶν τῶν ἡμερινῶν ἔργων τρὶς ἕκαστον ἐπελθεῖν·*
> *πῆ παρέβην; τί δ’ ἔρεξα; τί μοι δέον οὐκ ἐτελέσθη;*
> *ἀρξάμενος δ’ ἀπὸ πρώτου ἐπέξιθι· καὶ μετέπειτα*
> *δειλὰ μὲν ἐκπρήξας, ἐπιπλήσσεο· χρηστὰ δὲ, τέρπου.*
>
> *Let not sleep fall upon thy eyes till thou hast thrice reviewed the transactions of the past day. Where have I turned aside from rectitude? What have I been doing? What have I left undone, which I ought to have done? Begin thus from the first act, and proceed; and in conclusion, at the ill which thou hast done be troubled, and rejoice for the good.* ”
>
> Πυθαγόρας Σαμιος (Pythagoras of Samos) (⋆ c.570 BC – †c.495 BC)
> Χρύσεα Ἔπη (The Golden Verses),
> translation by Samuel Johnson in *The Rambler*, section 50 (1750)

Of the three questions Pythagoras states above, this chapter tries to provide answers to the last two, which are most relevant here. Firstly, to summarise *what has been done*, it evaluates the contributions made in this dissertation and reviews to which extent the proposed research objectives are accomplished. Secondly, for each of those contributions, future research, i.e. work *left undone*, is discussed.

8.1 Contributions and objectives revisited

The objectives in this dissertation were as follows:

Applicability: investigate how well the holonic concepts fit in the field of robotics. Provide an evaluation by implementing an application scenario.

Robustness: provide guidelines and considerations in the development of robust robot software which is able to cope with external disturbances and changes in the environment.

Integration: decompose this software in such a way that integration with other existing or new software is feasible and straightforward.

Scalability: provide a solution which scales well with respect to an increase in either the number of robots or either the size of the environment in which these robots operate. This increase should result in a linear or polynomial increase of the system's overall computation time or complexity.

In the following paragraphs each objective and relevant contributions will be evaluated.

8.1.1 Applicability (cf. chapters 2, 4, 5, 6 and 7)

During this dissertation a multi-robot navigation scenario was defined in which a small set of autonomous mobile robots, e.g. robotic wheelchairs, provide navigation services for a group of users, such as patients in a hospital or inhabitants of a retirement home or school for the physically impaired.

One of the benefits of such an approach is that wheelchair user regain some independence and caretakers are only required to help the user mount or dismount the wheelchair. While navigating, the robot autonomously finds it way and is able to avoid obstacles using its range sensors. If desired, the user is also able to take control of the wheelchair and drive it manually or in a *shared control* manner.

One of the key criteria for a successful application is to minimise user discomfort. Not only should the autonomous navigation be smooth, but also interference between two or more robotic wheelchairs should be avoided. For instance, if two or more robotic wheelchairs enter a narrow corridor from either side, at least one robot will be forced to retreat.

The investigation on how well the holonic concepts, in general, and PROSA and D-MAS, in specific, fit in the field of robotics focussed around this multi-robot navigation scenario.

One of the challenges in the mapping or *translation* to robotics was the introduction of a continuous environment in which the robots operate. Defining obstacle-free paths for multiple robots in a dynamic environment is a difficult task. The choice was made to let PROSA and D-MAS handle global coarse-grained path planning and let the robots use their on board obstacle avoidance algorithms to allow local deviations from a proposed path.

This also showed how PROSA can be integrated with existing robot software and the multi-robot path planning problem was even further reduced by distributing the path planning among the rooms and corridors in the environment, which were represented in PROSA as resource holons and required explicit allocation.

Although this structural decomposition simplified the planning problem, another challenge was how to make a transition from one room to the other without blocking the doorway for other robots. Transition zones were defined near the doorways where robots could temporarily reside. Whenever a robot, previously occupying a narrow corridor, left, the robot which was stationary in the transition zone could continue its path.

Multi-robot navigation experiments have been conducted to illustrate the applicability of holonics in robotics. A first experiment in simulation showed how bottlenecks and unnecessary travel can be avoided by employing PROSA and its structural decomposition and D-MAS to provide short-term forecasts. The experiment also illustrated that an increase of the number of robots and rooms has no effect on the overall performance. A second experiment on the robot hardware platforms at PMA/KU Leuven shows how D-MAS allows two robotic wheelchairs to propagate their intention to each other and how explicit resource allocation combined with this intention awareness improves the overall multi-robot navigation task.

Besides the elaboration of these multi-robot navigation experiments in chapters 6 and 7, other robot applications adopting the holonic concepts were also discussed in chapter 4, and guidelines in using PROSA and D-MAS for the development of robot applications were provided 2. Moreover, in chapter 5 on the HMES analysis, the limitations of the current implementation were highlighted and possible extensions were provided.

8.1.2 Robustness (cf. chapters 5 and 6)

By jointly using D-MAS and PROSA, the coordination mechanism in the robot system evolves from reaction to prediction of disturbances in the environment. Indeed, coupling resource allocation with D-MAS's short-term forecasting power, allows the system to anticipate and avoid bottlenecks or other navigational issues rather than have to deal with them.

This type of control is often referred to as **preview control** which, in contrast to feed forward or feed back control, deals with near future events which are estimated or forecast. In previous research preview control has mostly been applied in immediate robot control or shared control with a user. With PROSA and D-MAS this technique is also adopted to increase the robustness in global planning.

More concretely, if in the navigation application a particular room or corridor becomes unavailable, due to some unexpected problem, the exploring ants detect this problem. The order holon sending out these exploring ants is notified and adapts its solution. Thus, the forecasting or preview control is not only used to find a initial path to follow. These ants are continuously sent out, also during execution, to keep up with reality and react to disturbances.

The robustness of the current PROSA and D-MAS implementation, the Holonic Manufacturing Execution System (HMES), was tested during simulation experiments, discussed in chapters 5 and 6. In the first experiment one of the rooms of a chosen path was deliberately blocked during execution. Results showed the exploring ants had already detected the problem before the robot reached the problem area and an alternative route was chosen. The second experiment showed how this technique results in better performance with respect to travelled distance and elapsed time, compared to a reactive approach.

8.1.3 Integration (cf. chapters 2, 4 and C)

By using PROSA in robotics at a higher coordination level, both legacy hardware and existing state of the art can be integrated with newer hardware and software to the degree they are compliant. Three groups can be identified: not compliant, to some extent compliant and fully compliant.

The robots which are not compliant do not follow any resource allocation rules and, thus, will be only treated as disturbances in the system. The detection of these robots is similar to any other above-described disturbance. For instance, sensors in the environment or on a compliant robot might be able to detect

them as unexpected obstacles. Nevertheless, the system is able to handle these disturbances and thus non compliant entities are allowed in the environment.

Compliant robots either provide full support the PROSA and D-MAS system or only partially support it. For instance, a robot might support resource allocation without providing own sensor data to update resource holons with occupancy information.

In the experiments performed in this thesis only robots with the same level of compliance were used, either simulated or actual hardware platforms.

In order to communicate between all the different actors in the multi-robot coordination application, a common communication protocol was designed. The protocol makes use of *protocol buffers* to optimise the encoding of structured data sent *over the wire*.

Both the communication between the robots and PROSA but also the data sent from an to the graphical user interfaces and multi-robot simulator follow this protocol. This way there is no distinction between data coming from a real robot or a virtual robot in the simulator and both can be used in the multi-robot application, even simultaneously.

In chapter 2, integration in general was discussed, while in chapter 4 concrete information with respect to integration of the HMES implementation and the existing robot software was provided. Finally, in appendix C the communication protocol facilitating this integration was discussed.

8.1.4 Scalability (cf. chapter 5)

The most apparent influence or contribution is the paradigm shift from functional to structural decomposition of robot software. Introducing **holons** in robotics is, to our knowledge, a novel approach and adopting single source of truth has provided a **scalable** solution to multi-robot navigation.

Both scalability with respect to the number of coordinated robots and the size of the environment, i.e. the rooms and corridors these robots navigate through, has been investigated and in both cases holonic systems provide a more scalable solution than most common approaches in robotics. These results are presented in chapter 5.

Simulation experiments were conducted with 2 to 32 robots and environments as small as two rooms and one corridor to as large as 32 rooms and 52 corridors.

Total execution time and throughput was analysed and a correlation between number of conflicts robots have in the environment and total execution time was

clearly present. The lower the number of conflicts, the more robots executed their navigation task in parallel without interfering each other and total execution time remained constant. For each added conflict a small penalty in throughput was identified. Nevertheless, due to PROSA's structural decomposition and the distribution of the multi-robot path planning, parallel execution is high and resulted in a highly scalable system.

Next to the scalability and robustness experiments, the HMES performance has been analysed on various other aspects. The influence of configurable parameters on system performance was investigated and a complexity analysis with respect to computation time has been conducted.

The computational complexity of the HMES does not depend on a single input but a large set of parameters, such as the number of orders, the number resources, the complexity of the process plans, and some tunable parameters. Therefore, a practical approach was chosen to determine its computational complexity. The resulting analysis indicated that the execution cycles in the order holon and the resource scheduler largely determine the complexity. For instance, optimising the resource scheduler could already greatly improve the overall performance.

However, there is always a trade off to be made between optimisations specific to an applications and the general applicability of the system. Also, it is not guaranteed an optimised scheduler with respect to computational complexity will deliver similar results with respect to the application.

These performance tests also provided guidelines on how to configure the HMES for specific applications.

8.1.5 Experimentation (cf. chapters 6 and 7)

Next to the simulation experiments illustrating the scalability and robustness of the approach, two multi-robot navigation experiments have been conducted, both in simulation and using robotic wheelchairs. These experiments highlighted the applicability and integration of PROSA & D-MAS with existing robot software.

A first experiment showed in simulation how bottlenecks and unnecessary travel can be avoided by employing PROSA and its structural decomposition and D-MAS to provide short-term forecasts. The experiment also illustrated that an increase of the number of robots and rooms had no effect on the overall performance. The approach was compared to a reactive approach in which all robots continuously replan to determine their next navigation command.

The second experiment showed how D-MAS allows two robotic wheelchairs to propagate their intention to each other and how explicit resource allocation combined with this intention awareness improves the overall multi-robot navigation task.

The results of these experiments are discussed in chapters 6 and 7.

8.1.6 Graphical tools (cf. chapters A and B)

During this thesis, several graphical user interfaces and tools were developed to enhance feedback of system data to a user or debugger and to speed up experiment configuration. Besides these user interfaces, the existing MLRSimulator [Vanacker 2008] has been redesigned to allow the simulation of multiple robots and increase the feedback of information.

The most elaborated user interface, named Robot GUI, allows remote control of robotic wheelchairs, visualises sensor and algorithm data, and contains a map editor.

Each of these user interfaces has been completely separated from the system code and communicates over network sockets. By doing so, the Robot GUI can run on any system as long it is connected to the same network as the robot.

8.2 Future work

Although the application of holonics in the multi-robot navigation scenario was thoroughly investigated, there is plenty of room for future work in both the navigation topic as well as other robotics domains, which were only touched briefly.

8.2.1 Application specific holons

The multi-models, i.e. multi-level Petri nets and Java models, developed in this thesis, remain fairly basic. For instance, the robots navigating through the environment were represented as order holons. In future research application specific resource holons, e.g. representing a robot wheelchair, with its characteristics, could be created. An order holon would then represent the actual navigation request and execution would require both a robot resource and a set of room and corridor resources.

In chapter 5, the limitation of the current HMES implementation was discussed. Also this problem of multi-resource allocation has been briefly addressed.

The challenge here is that not only time, i.e. during which time intervals do which resources need to be allocated, but also space, i.e. where do these (robot) resources have to be, becomes a factor in the exploration of solutions. Indeed, the allocation of a robot resource for a particular time interval is not sufficient. This robot also has to follow a particular path in the environment.

In order to solve this problem, the exploring D-MAS need to support multi-resource allocation and the resource scheduler of *moving* resources could be adapted to incorporate location information.

A similar challenge can be identified in sharing a sensor between two robots. Allocation of this sensor is not necessarily the problem, but finding a sensor at the right time and pointing in the right direction is more difficult. Again, the exploring D-MAS have to be able to query not only when this sensor would be available but also where it will be at that time.

Another possible solution to these challenges is the introduction of application specific staff holons. These holons could maintain a list of which robot will be where at which time and provide the exploring ants with this information.

8.2.2 World model extensions

The transit zones defined at the connection points between room or corridor resources already accommodated for a deadlock problem which can occur if one robot is waiting to enter a room and another is waiting to exit that room. However, the resource holons could be largely improved by integrating a local multi-robot path planner. Since each room would have its own planner, the computational complexity would remain low and this way the resource holons could specify a free spot to any robot requesting to pass through. The local multi-robot path planning would ensure the provided positions are free at the time to robot would reside there. Also, the resource holon could manage how many robots are still allowed in its room or corridor by taking the size of the robots into account.

Moreover, rules could be introduced in which entering traffic stays to the right and exiting traffic stays to the left of the room or traffic moving in the same direction could be allowed to follow each other if allocations are within a certain time span.

Investigating how traffic rules and such local multi-robot path planners improve performance in terms of throughput or occupancy, for example, is a challenging

task. Nevertheless, PROSA offers the building blocks to implement such concepts in a robust and scalable way.

8.2.3 Case study

A possible first step in the continuation of this research could be the evaluation of the current multi-robot navigation application in a case study at, for example, a hospital or school for physically impaired children. Although many experiments have been conducted to thoroughly test the implementation, most of them were in simulation and practical issues which will arise at a test site will provide insights in how to improve particular components in the application.

Also user evaluations provide an added value and can indicate weak points. Possible collaborations in Leuven are Windekind, a school for physically or mentally impaired children and University Hospital Gasthuisberg. Windekind has already collaborated with PMA/KU Leuven in European projects and has showed interest in this application scenario. A collaboration with Gasthuisberg could also involve the scheduling of linen or bed transport throughout the hospital.

8.2.4 Explore robotics domains

This thesis focussed on the problem of multi-robot navigation and provided a solution using the holonic architecture PROSA and short-term forecasting D-MAS.

In the chapter on holonic robot applications other applications have been briefly discussed. Introducing holonics in any of those subdomains of robotics is an obvious extension of this research. The guidelines provided in this thesis will help adopt PROSA and D-MAS in other domains faster. Also the computational complexity and scalability analyses of the current implementation help decide whether the implementation of application-specific holons or for instance a different scheduler would improve the performance of the HMES for a particular application.

Appendix A

Multi-Robot Simulator

A.1 Background

The development of a 3D robotic wheelchair simulator started in 2005 with the European FP6 project MAIA. The goal of this project was to create a brain actuated robotic wheelchair and to test the discrete movement command sent by the brain computer interface a simulator was required.

The following years this MLRSimulator was used in several mobile robotics projects at PMA/KU Leuven and development continued.

In the context of this thesis, the simulator required a refactoring phase to accommodate multiple robotic wheelchairs. In the past, the focus was always on one robot.

The resulting 3D simulator could then be used to run multi-robot simulation experiments, since its communication protocol is compatible with the one used by the HMES interface to the robots.

A.2 Overview

The 3D simulator uses the `OpenSceneGraph` library, which is a wrapper around Open GL. It is written in standard C++ and is platform independent. This library defines a graph of which each node represents part of the simulation. For instance, the wall and floor nodes define the environment, the wheelchair



Figure A.1: MLRSimulator showing an environment with multiple robots. Each robot has its own laser scanner for collision detection.

nodes represent the robotic wheelchairs and each wheelchair node contains a laser scanner node, representing their sensor.

This simulator provides software-in-the-loop simulation, i.e. control algorithms developed for the mobile robots can be tested in simulation, as is, before testing them on the actual hardware.

Figure A.1 shows a simulated environment with multiple robots. Each robot has its own laser scanner for collision detection.

A.3 Collision detection

The simulated laser scanner scans the environment by sending out rays in its view plane. It then calculates collisions with the environment or other wheelchairs.



Figure A.2: The intersection visitor calculates collisions between laser scanner rays and the environment. The white spheres indicate detected collisions.

These collisions are detected using OpenSceneGraph's intersection visitor. This component traverses the scene graph and checks for each ray if it collides with one of the environment nodes.

In Figure A.2, a resulting set of scan points is displayed.

A.4 3D Models

OpenSceneGraph also provides a set of plugins to load 3D models from file. In more recent experiments with simulated robotic wheelchairs a 3D model of a wheelchair is used to represent the wheelchairs and some furniture models are added to reflect a house environment.

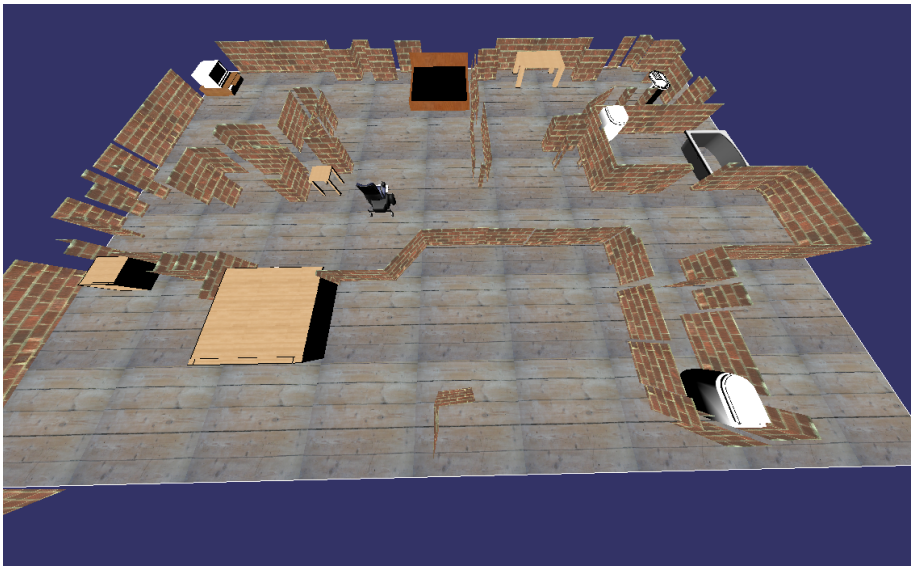


Figure A.3: Top view of MLRSimulator.

Figure A.3, A.4 and A.5 show the use of these models in the three available viewpoints in the simulator: top view, third person view and first person view.

A.5 Conclusion

Summary:

This appendix provided some background information about the 3D simulator, which was developed at PMA/KU Leuven. In the context of this thesis, changes to this simulator were made to allow multiple robots which can be coordinated through the HMES. Coordinated multi-robot navigation experiments, conducted using this simulator, are discussed in chapter 6.

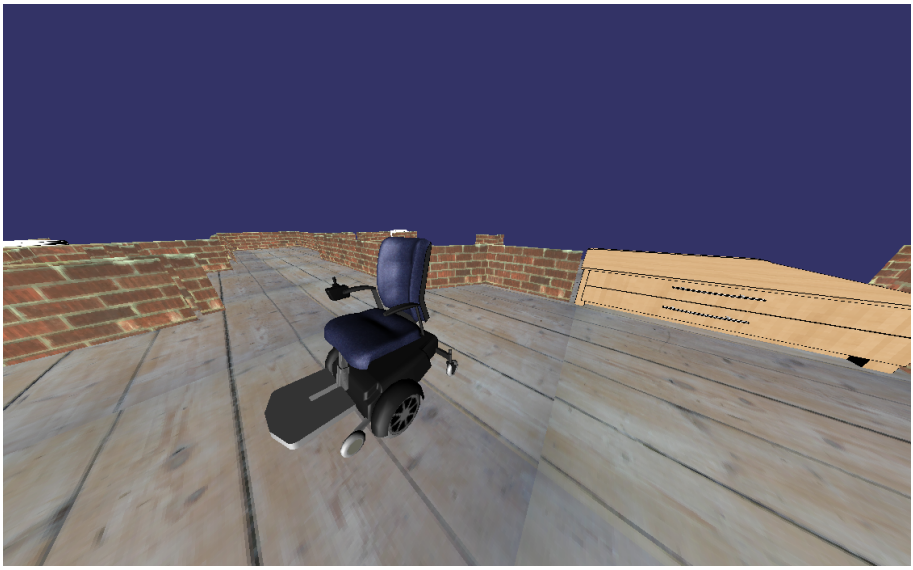


Figure A.4: Third person view of MLRSimulator.

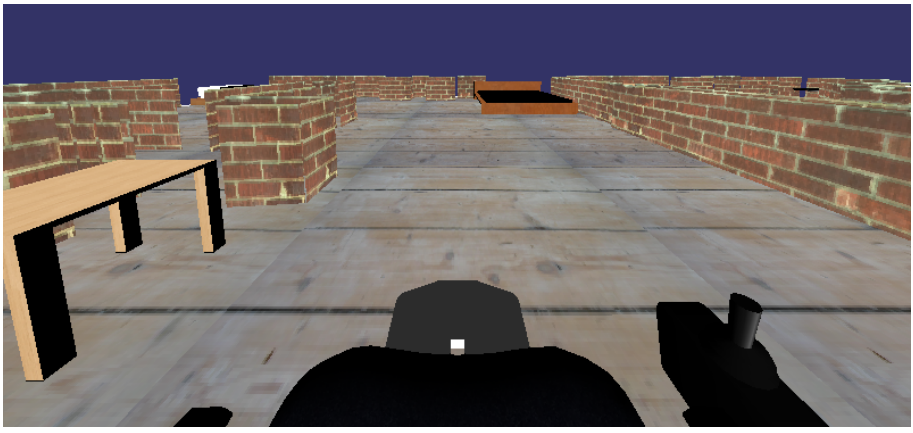


Figure A.5: First person view of MLRSimulator.

Appendix B

Robot Graphical User Interface

B.1 Motivation

This appendix discusses the robot graphical user interface (Robot GUI), which was created during this research. The motivation to develop this interface was twofold.

Firstly, instead of performing tasks on the robots and analysing the log file afterwards, there was need for online evaluation of the robot's algorithms, its sensors and actuators.

Secondly, a interface to control which algorithm was running on the robotic wheelchairs was required and functionality to design environments in which, real of simulated, robots could navigate, was desired.

Moreover, the graphical interface should be usable on all robotic wheelchairs through a common interface and independent from the existing code.

B.2 Overview

In Figure B.1 the main window of the Robot GUI is depicted. The different interaction sections are marked from ① to ⑨ and are explained in the following paragraphs.

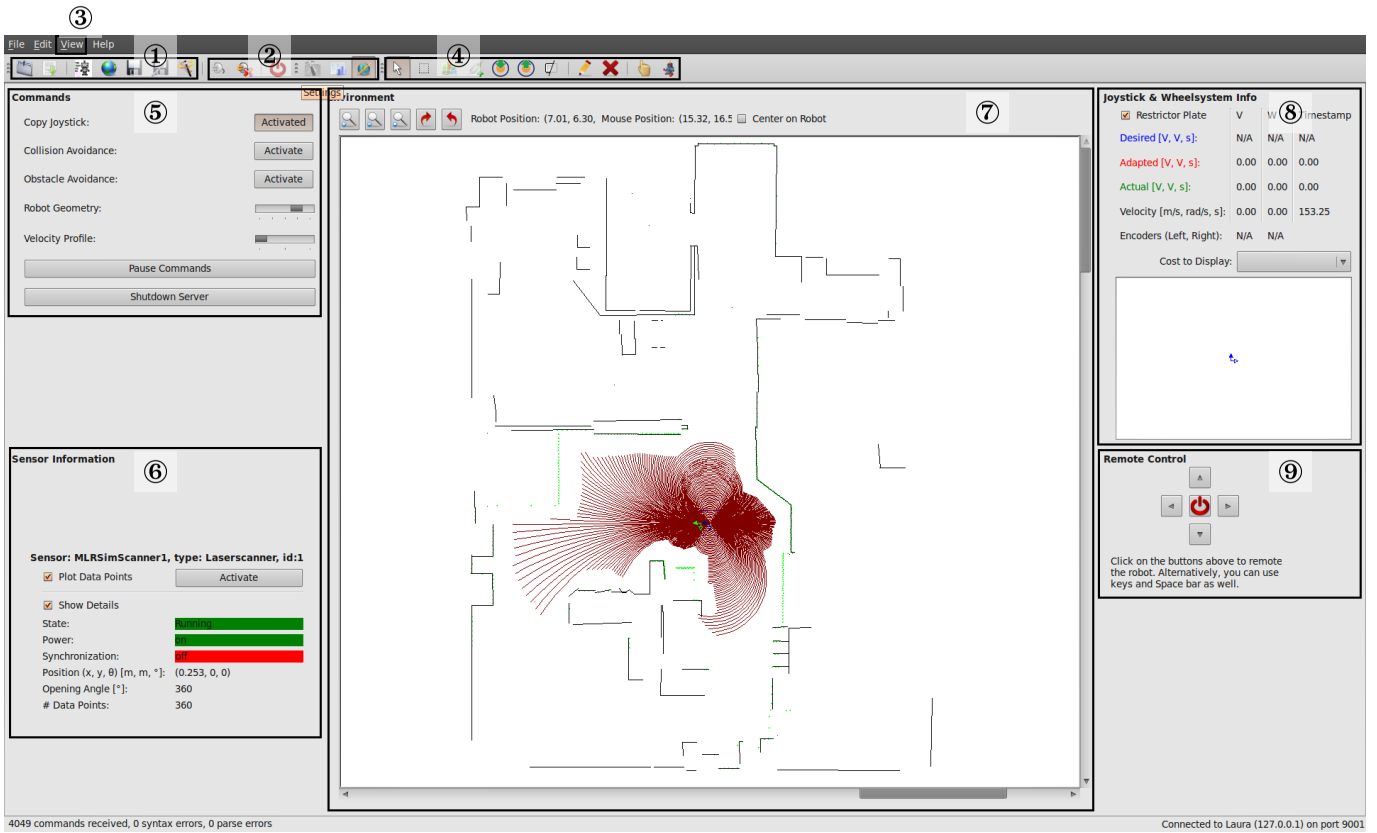


Figure B.1: The main window of the Robot GUI. Different sections are marked from ① to ⑨.

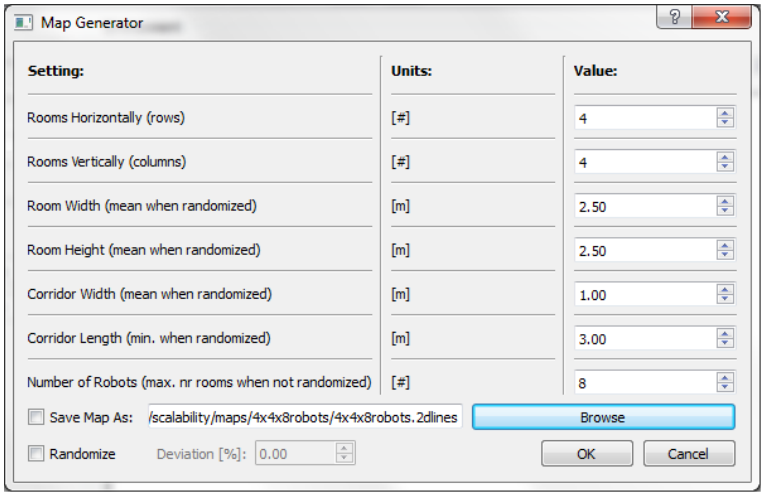


Figure B.2: The `Map Generator` window allows the user to specify settings to generate a custom map.

① Loading & saving

Data sent to and received from the robot appear in the connection log. This log can be loaded ( ) and saved ( ) through these buttons or via the `File` menu.

Also a map of the environment can be loaded ( ) and saved (,) to file. The map file can be saved to formats readable by both the 3D simulator (`.3d`) and algorithms running on the robot (`.2dlines`). Together with the map all robot positions will be saved as well, and in a format compatible with the algorithms (`.pose`).

The Robot GUI can also generate environments using a set of criteria given by the user. The map generator button ( ) opens a separate window in which these criteria can be entered. Figure B.2 shows this window. It allows the user to specify the number of rooms, their width an height, how many robots to generate and whether to randomise robot positions and room sizes.

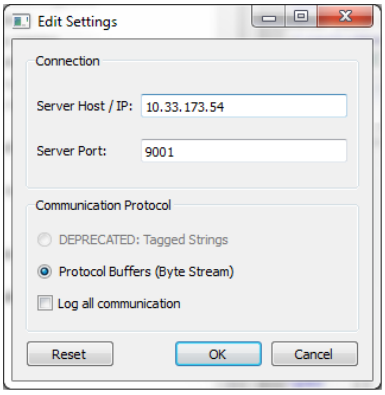


Figure B.3: The `Settings` window allows the user to specify connection settings such as the robot's IP address and port number.

② Editing & viewing settings

Settings, such as window position and size, connection settings and log or map file locations are automatically saved and made persistent when the user closes the Robot GUI. This way, the user is not obliged to reenter all settings every time. In Windows these settings are saved to the registry while in Linux a small settings file is kept in the user's home directory.

The user can enter connection settings through the `Settings` window (), which is also accessible via the `Edit` menu. This window is depicted in Figure B.3. All communication between the Robot GUI and the robotic wheelchairs is done using the same communication protocol used between the HMES and the robots or the simulator. Messages are sent over TCP/IP and formatted using Google Protocol Buffers. In appendix C, this protocol is discussed in more detail.

After a connection with the robot is established through the `Connect` button (), the robot's current set of parameters can be consulted through the `Parameters` window, shown in Figure B.4. This window is accessible through the `Edit` menu or its button (). Disconnecting from the robot is done using the `Disconnect` button (().

Finally, the `Map Editor` () can also be activated through the `Edit` menu. Its functionality will be discussed separately.

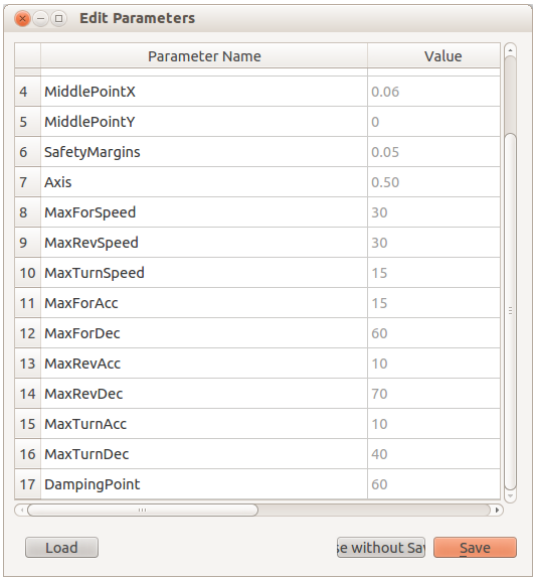


Figure B.4: The `Parameters` window shows the parameters the robot is currently using.

③ Customising views and log

Through the `View` menu the connection log can be accessed and the user can customise which windows and toolbars are shown in the main window. The connection log gathers all data sent to and received from the robot and labels data to make a distinction between `Info`, `Error`, `Send` and `Receive` messages. Optionally, the log can be filtered by unchecking the `Log all communication` checkbox in the `Settings` window. This way only critical messages will be added to the log file.

Figure B.5 shows the `Connection Log` window and an example of messages appearing in this log.

④ Editing the map

The map editor provides a set of buttons to create and edit custom maps within the Robot GUI. These maps can then be saved and used on the robot or in simulation.

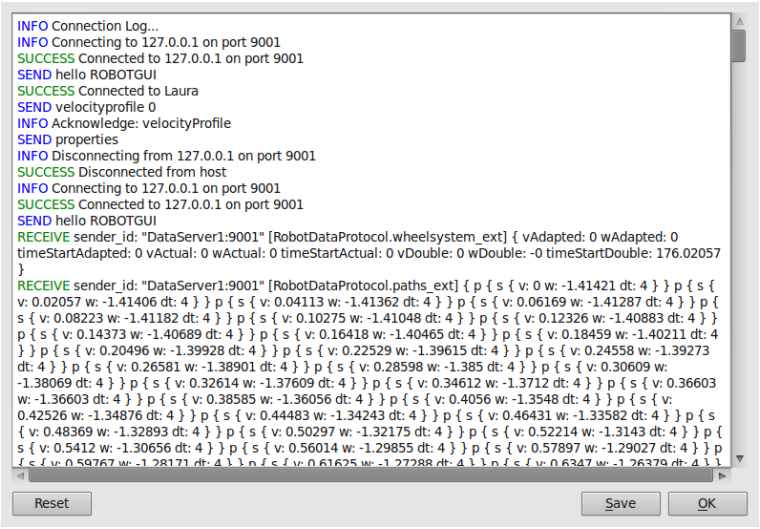


Figure B.5: The `Connection Log` window shows messages sent to and received from the robot.

Select region

This button is used to select a region in the environment. The selection can then be deleted or saved as a named region. If deleted, also robots within the selected region will be deleted.

Add region

This button saves the current selection as a named region. These regions are used to identify parts of the environment and define resource holons for the HMES.

Add link

Named region can be linked to each other using this button. The current selection will be used as link area. These areas are using to define the transit zones in resource holons used by the HMES. Links between regions are shown in their tooltip when hovering over them.

Save regions

This button opens a `Save as` dialogue to save the named regions to a `.region` file. The format is compatible with the main program running the HMES. This program can then read in the regions and their specifications (Room or corridor, size, maximum capacity, transit zones).

Load regions

Region files saved earlier can be loaded into the Robot GUI. Although this loads all regions and their specifications, it does not check if the regions match the current map of the environment.

Show or hide regions

During execution, sensor data and the robot's position will be drawn in the environment. To keep this information readable, this button toggles whether or not to show the named regions and their link areas.

Add line

Lines can be added through this button. A grid will appear in the environment to facilitate drawing. However, snap to grid is not implemented in this graphical interface.

Delete selection

Selected lines, regions or robots can be deleted using this button.

Move robot

A selected robot can be moved using this button. First the user moves the mouse pointer to a desired target position, then by clicking and holding the left mouse button, moving the mouse pointer will rotate the robot.

Add robot

Adding a robot is similar to moving a selected robot. First the user moves the mouse pointer to a desired target position, then by clicking and holding the left mouse button, moving the mouse pointer will rotate the robot. Keep in mind this only add robots to the pose file. Only one robot can be active, i.e. connected to, at the same time. This functionality was mainly added to created multi-robot maps for simulation.

⑤ Switching between algorithms

The `Commands` window provides sliders and buttons to activate particular algorithms on the robot. For example, the user can choose between user control (copy joystick), collision avoidance and obstacle avoidance.

The sliders allow the user to change the velocity profile and robot geometry used by the algorithms. The `Pause Commands` button ignores incoming robot messages, which can be helpful to identify a problem with current data. The `Shutdown Server` button sends a shutdown request message to the robot.

⑥ Receiving sensor info

The `Sensor Information` window shows status information and specifications of the sensors connected to the robot. This helps to identify sensor malfunctioning and to see if sensor data is interpreted correctly. Also, each sensor has a checkbox whether or not its data points should be plotted in the environment.

⑦ Viewing the environment as perceived by the robot

In the `Environment` window all sensor data coming from the robot, the map loaded from file or created using the map editor and additional information coming from algorithms running on the robot are drawn. Also the robot geometry, its frame and all sensor frames are drawn to facilitate data interpretation.

The buttons on top allow to rotate (,) and zoom (, ,) the entire environment.

⑧ Receiving actuator info

The `Joystick & Wheelsystem Info` window shows joystick and wheelsystem values during execution and also plots the restrictor plate. If the dynamic window algorithm is used different costs will be displayed on top if this restrictor plate as well.

⑨ Sending motion commands

The `Remote Control` window provides buttons to move the robot forward, backward, left or right or stop its movement. Although this was added for debugging purposes only, it can be used as a simple joystick interface to steer the robot.

B.3 Conclusion

Summary:

This appendix provided an overview of the robot graphical user interface, which was developed to facilitate data analysis and debugging. It has the same communication interface to the robotic wheelchairs as the 3D simulator and the HMES and provides basic functionality to send commands to the robot, create maps and regions and save or load data from file.

Appendix C

Communication Protocol

C.1 Motivation

In the past, communication between the robot software and a graphical interface was done by sending plain text strings over TCP/IP. Although this works fairly well it has some limitations with respect to formatting and is far from efficient with respect to memory and time.

Since in the context of this thesis multiple robots would be coordinated and both code written in C++ and JAVA had been developed, a library was required supporting both programming languages and providing at least basic optimisation with respect to memory and time.

Benchmarks found at [*Benchmarks for various JAVA serializers*] showed Google Protocol Buffers scored well on all tests and also supports both JAVA and C++.

C.2 Google protocol buffers

From its documentation:

> “Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data, think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams

> and using a variety of languages. You can even update your data structure without breaking deployed programs that are compiled against the *old* format. ”

In other words, the message format for each type of message between the robot software, its graphical user interface, the 3D simulator or the HMES is defined once. Out of this *protocol* C++ and JAVA code is generated and this code is used to create and populate messages in the different software modules.

The protocol is saved in a `.proto` file, which can be compiled by Google's protocol compiler resulting in the generated language specific data access classes.

Each message defined in this file contains a set of numbered fields specifying a name and type, if its a required field or optional and, optionally, its default value. Listing C.1 shows the complete protocol used in the robot applications, discussed in this text.

A wrapper message type was defined to be able to send messages which can be of different types. Since the protocol buffer parser cannot determine the message type on the content only, this technique still allows to send different kind of messages and the recipient application will be able to decode them correctly.

Each optional field in the wrapper message defines a possible message type and in practice only one of those fields will be filled in.

Another advantage of optional fields is the ability to provide a default value. This means if that particular field of the message is not filled in by the source application, no bytes are used to serialise that field, but still during message decoding at the target application, the default value can be filled in. Moreover, if some software module does not provide this field yet, the message can still be sent and parsed correctly.

C.3 Robot data protocol

In this section the different messages in the robot data protocol will be briefly explained. Keep in mind this protocol was used for internal use only and is definitely not self-contained.

Also, it lacks the enforcement of correct units for each value and coordinates are not specified with respect to a coordinate frame.

Listing C.1: Robot Data Protocol

```
1  package RobotDataProtocol;
2
3  option optimize_for = SPEED;
4  option java_package = "mrc.com.protocol";
5
6  message WrapperMessage {
7      extensions 100 to max; // allow users of this proto file to
           define their own message types.
8      optional string sender_id = 1 [default = "localhost:9001"];
9  }
10 extend WrapperMessage {
11     optional Hello hello_ext = 100;
12     optional RobotGeometry robotGeometry_ext = 101;
13     optional Sensor sensor_ext = 102;
14     optional Paths paths_ext = 103;
15     optional Wheelsystem wheelsystem_ext = 104;
16     optional Joystick joystick_ext = 105;
17     optional DynamicWindow dynamicWindow_ext = 106;
18     optional RestrictorPlate restrictorPlate_ext = 107;
19     optional Assistance assistance_ext = 108;
20     optional VelocityProfile velocityProfile_ext = 109;
21     optional Shutdown shutdown_ext = 110;
22     optional Acknowledge acknowledge_ext = 111;
23     optional State state_ext = 112;
24     optional Navigate navigate_ext = 113;
25     optional Map map_ext = 114;
26     optional Properties properties_ext = 115;
27     optional Trajectory trajectory_ext = 116;
28     optional TrajectoryController trajectoryController_ext =
           117;
29     optional Encoders encoders_ext = 118;
30     optional TrajectoryGenerator trajectoryGenerator_ext = 119;
31 }
32 message Encoders {
33   required float left = 1; // [rad]
34   required float right = 2; // [rad]
35 }
36 message Trajectory {
37   repeated Point waypoints = 1;
38 }
39 message TrajectoryController {
40   repeated TargetVelocity targetVelocities = 1;
41 }
42 message TrajectoryGenerator {
43   repeated TrajectoryState states = 1;
44 }
45 message TrajectoryState {
46   required float x = 1; // [m]
47   required float y = 2; // [m]
48   required float th = 3; // [rad]
49   required float v_x = 4; // [m/s]
50   optional float v_y = 5 [default = 0]; // [m/s]
```

```
51   required float w = 6; // [rad/s]
52   required float time = 7; // [s]
53 }
54 message TargetVelocity {
55   required float v = 1; // [m/s]
56   required float w = 2; // [rad/s]
57   required float dt = 3; // [s]
58   required float timestart = 4; // [s]
59 }
60 message Properties {
61   repeated PropertyValue propertyValue = 1;
62   optional string text = 2;
63 }
64 message PropertyValue {
65   required string name = 1;
66   required string value = 2;
67   optional bool readonly = 3;
68 }
69 message Map {
70     repeated Line lines = 1;
71 }
72 message Line {
73     required Point start = 1;
74     required Point end = 2;
75 }
76 message Navigate {
77     required Point target = 1;
78     optional float orientation = 2; // [rad]
79     optional string targetString = 3;
80     optional bool execute = 4 [default = false];
81     optional Point source = 5;
82     optional string sourceString = 6;
83     optional int32 algorithm = 7 [default = 1];
84 }
85 message State {
86     required Point pos = 1;
87     optional float orientation = 2; // [rad]
88     optional int32 state = 3 [default = 1];
89 }
90 message Hello {
91   optional string text = 1 [default = "hello"];
92 }
93 message Acknowledge {
94   optional string text = 1 [default = "ack"];
95 }
96 message RobotGeometry {
97     optional uint32 profile = 1;
98   message Polygon {
99     required float zMin = 1; // [m]
100    required float zMax = 2; // [m]
101    repeated Point points = 3;
102  }
103  repeated Polygon polygons = 2;
```

```
104 }
105 message Point {
106   optional float x = 1 [default = 0]; // [m]
107   optional float y = 2 [default = 0]; // [m]
108   optional int32 valid = 3 [default = 1];
109 }
110 message Sensor {
111   optional uint32 id = 1;
112   optional string type = 2;
113   optional string name = 3;
114   optional bool power = 4;
115   optional bool sync = 5;
116   optional string state = 6;
117   required float x = 7; // [m]
118   required float y = 8; // [m]
119   optional float z = 9; // [m]
120   required float theta = 10; [rad]
121   optional float openingAngle = 11; // [rad]
122   repeated Point points = 12;
123   repeated float ranges = 13;
124 }
125 message Paths {
126   message Segment {
127     required float v = 1; // [m/s]
128     required float w = 2; // [rad/s]
129     required float dt = 3; // [s]
130   }
131   message Path {
132     optional bool marked = 1 [default = false];
133     repeated Segment s = 2;
134   }
135   repeated Path p = 1;
136 }
137 message Wheelsystem {
138   required float vAdapted = 1; // [m/s]
139   required float wAdapted = 2; // [rad/s]
140   required float timeStartAdapted = 3; // [s]
141   required float vActual = 4; // [m/s]
142   required float wActual = 5; // [rad/s]
143   required float timeStartActual = 6; // [s]
144   required float vDouble = 7; // [m/s]
145   required float wDouble = 8; // [rad/s]
146   required float timeStartDouble = 9; // [s]
147 }
148 message Joystick {
149   required float v = 1; // [m/s]
150   required float w = 2; // [rad/s]
151   required float timeStart = 3; // [s]
152 }
153 message DynamicWindow {
154   required uint32 gridSize = 1;
155   required int32 w = 2;
156   required int32 v = 3;
```

```
157   required int32 wMin = 4;
158   required int32 wMax = 5;
159   required int32 vMin = 6;
160   required int32 vMax = 7;
161   required uint32 iAcc = 8;
162   required uint32 jAcc = 9;
163   repeated float totalCost = 10 [packed = true];
164   repeated float costGradient = 11 [packed = true];
165   repeated float costUserDifferentV = 12 [packed = true];
166   repeated float costUserDifferentRho = 13 [packed = true];
167   repeated float costClosenessToObst = 14 [packed = true];
168   repeated float costCollisionSpeed = 15 [packed = true];
169   repeated float costV = 16 [packed = true];
170 }
171 message RestrictorPlate {
172   optional float centerX = 1;
173   optional float centerY = 2;
174   repeated Point points = 3;
175 }
176 message Assistance {
177   optional uint32 level = 1 [default = 1];
178 }
179 message VelocityProfile {
180     optional uint32 profile = 1 [default = 1];
181 }
182 message Shutdown {
183   optional string text = 1;
184 }
```

Acknowledge

This message is used to send an acknowledgement to a sending party. An optional string message can be accompanied.

Assistance

If a user switches the assistance level in the graphical user interface this message is sent. Currently supported levels are user control (1), collision avoidance (2) and obstacle avoidance (3).

DynamicWindow

If collision or obstacle avoidance is active and uses the dynamic window implementation developed at PMA/KU Leuven, this message can be sent to the graphical user interface to display the several costs.

Encoders

This message is sent by robotic wheelchairs with encoders on both their left and right wheel.

Hello

The `Hello` message is sent whenever a connection between two applications is established. Typically, the sender application provides its name in this message.

Joystick

This message contains the joystick values converted to translational and rotational velocities and a timestamp to identify when this measurement took place.

Map

If a map needs to be shared among two applications, this message can be sent. It contains a repeated, i.e. 1 or more, set of lines, which are defined by their start and end coordinates with respect to the world frame.

Navigate

This message is sent to issue a navigation task to a robot or a simulated robot. It contains the source and target, both in coordinates and their region name. It also contains a boolean to indicate if the task should be executed or only the path should be calculated and a field to specify which algorithm to use.

Paths

This message contains the set of feasible circle segments a robotic wheelchair is able to execute at a particular time. It is sent to the graphical user interface to display where the robot can move.

Properties

If required, the robot application is able to send its current set of parameters using this message. Sending an empty `Properties` message is interpreted as a request to send the parameters.

RestrictorPlate

This message contains a set of points representing the joystick's restrictor plate. It is optionally sent to the graphical user interface to display the restrictor plate in the joystick info window.

RobotGeometry

The robot geometry can be displayed in the graphical user interface by decoding this message. Each robot geometry is defined by a set of polygons at particular heights.

Sensor

This message is used to send sensor data and status information to the graphical user interface.

Shutdown

If a user issues a shutdown command in the graphical user interface, this message is sent to the robot.

State

A `State` message contains the robot's position, its orientation and its state. The state can either be running, idle or error.

Trajectory

A calculated trajectory can be sent using this message. It is defined as a set of waypoints.

TrajectoryController

This message is used to send a set of velocities to the robot which need to be executed at particular time intervals.

TrajectoryGenerator

This message contains a trajectory specified not only by x and y coordinates but also orientations and velocities at particular times.

VelocityProfile

This message is sent if the user changes the velocity profile in the graphical user interface and whenever a connection is established with a robot to inform the user which profile is currently used.

Wheelsystem

This message is sent to the graphical user interface to indicate the current velocities of the robot.

C.4 Conclusion

This appendix discussed the use of Google protocol buffers in the robotic applications developed at PMA/KU Leuven and briefly indicated how each message type, defined in the robot data protocol, is used in practice.

Bibliography

AIST (2012). *RT-Middleware Overview*. URL: `http://openrtm.org/openrtm/en/content/rt-middleware-overview` (cit. on p. 57).

Alami, R., R. Chatila, R. Fleury, M. Ghallab, and F. Ingrand (1998). "An Architecture for Autonomy". In: *The International Journal of Robotics Research* 17.4, pp. 315–337 (cit. on pp. 2, 62, 63).

Ali, O. (Sept. 2010). "Operational Planning for Outdoor Engineering Processes". PhD thesis. KU Leuven (cit. on pp. 48, 74).

Ando, N., T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon (Aug. 2005). "RT-middleware: distributed component middleware for RT (robot technology)". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3933–3938 (cit. on pp. 2, 56).

Arbib, M. A. (1969). *Theories of abstract automata (Prentice-Hall series in automatic computation)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (cit. on p. 36).

Arkin, R. C. and T. Balch (1997). "AuRA: Principles and Practice in Review". In: *Journal of Experimental and Theoretical Artificial Intelligence* 9, pp. 175–189 (cit. on p. 61).

Babiceanu, R. F. and F. F. Chen (Feb. 2006). "Development and applications of holonic manufacturing systems: a survey". In: *Journal of Intelligent Manufacturing* 17.1, pp. 111–131 (cit. on pp. 16, 29).

Bartholdi III, J. J. and K. R. Gue (Nov. 2000). "Reducing Labor Costs in an LTL Crossdocking Terminal". In: *Operations Research* 48.6, pp. 823–832 (cit. on p. 49).

Bass, L., P. Clements, and R. Kazman (2003). *Software Architecture in Practice*. Addison-Wesley Professional (cit. on p. 29).

Benchmarks for various JAVA serializers. URL: `http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking` (cit. on p. 201).

Bennewitz, M., W. Burgard, and S. Thrun (2002). "Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots". In: *Robotics and Autonomous Systems* 41.2, pp. 89–99 (cit. on pp. 18, 69).

Bensalem, S., M. Gallien, F. Ingrand, I. Kahloul, and T.-H. Nguyen (2009). "Toward a More Dependable Software Architecture for Autonomous Robots". In: *Special issue on Software Engineering for Robotics of the IEEE Robotics and Automation Magazine* 16, pp. 67–77 (cit. on p. 62).

Bondi, A. B. (2000). "Characteristics of scalability and their impact on performance". In: *Proceedings of the 2nd international workshop on Software and performance*. WOSP '00. ACM, pp. 195–203 (cit. on p. 26).

Bongaerts, L., L. Monostori, D. McFarlane, and B. Kádár (2000). "Hierarchy in distributed shop floor control". In: *Computers in Industry* 43.2, pp. 123–137 (cit. on p. 66).

Boysen, N. (Jan. 2010). "Truck scheduling at zero-inventory cross docking terminals". In: *Computers & Operations Research* 37.1, pp. 32–41 (cit. on p. 49).

Brooks, A., T. Kaupp, A. Makarenko, W. Stefan, and A. Orebäck (2005). "Towards Component-Based Robotics". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)* (cit. on pp. 2, 55).

Brummit, B. and A. Stentz (May 1998). "GRAMMPS: A Generalized Mission Planner for Multiple Mobile Robots". In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1564–1571 (cit. on p. 70).

Bruyninckx, H. (2010). *Robotics software framework harmonization by means of component composability benchmarks. "The manifolds of four"*. BRICS Deliverable D8.1. URL: `http://www.best-of-robotics.org/pages/publications/BRICS_Deliverable_D8.1.pdf` (cit. on pp. 4, 11, 17, 20–23, 50, 56, 74).

Bruyninckx, H., P. Soetens, and B. Koninckx (2003). "The Real-Time Motion Control Core of the Orocos Project". In: *IEEE International Conference on Robotics and Automation*, pp. 2766–2771 (cit. on pp. 2, 58).

Collett, T. H. J., B. MacDonald, and B. Gerkey (2005). "Player 2.0: Toward a Practical Robot Programming Framework". In: *Proceedings of Australasian Conference on Robotics and Automation (ACRA'05)* (cit. on p. 61).

Demeester, E. (May 2007). "User-Adapted Plan Recognition and Shared Control for Wheelchair Driver Assistance under Uncertainty". PhD thesis. K.U.Leuven (cit. on pp. 40, 159, 160).

De Schutter, J., T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx (2007). "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty". In: *The International Journal of Robotics Research* 26.5, pp. 433–455 (cit. on p. 59).

Dias, M., R. Zlot, N. Kalra, and A. Stentz (July 2006). "Market-Based Multirobot Coordination: A Survey and Analysis". In: *Proceedings of the IEEE* 94.7, pp. 1257–1270 (cit. on p. 72).

Dias, M. and A. Stentz (2000). "A free market architecture for distributed control of a multirobot system". In: *In 6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pp. 115–122 (cit. on p. 71).

Dorigo, M. (2001). *Swarm-bots (IST-2000-31010)*. URL: `http://www.swarm-bots.org` (cit. on p. 72).

— (2006). *Swarmanoid (IST-022888)*. URL: `http://www.swarmanoid.org` (cit. on p. 72).

Dorigo, M. and T. Stützle (2004). *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company (cit. on p. 72).

Elkady, A. and T. M. Sobh (2012). "Robotics Middleware: A Comprehensive Literatur Survey and Attribute-Based Bibliography". In: *Hindawi Journal of Robotics*. Ed. by Y. Li (cit. on p. 55).

Fishwick, P. A. and B. P. Zeigler (Jan. 1992). "A Multimodel Methodology for Qualitative Model Engineering". In: *ACM Transactions on Modeling and Computer Simulation* 2.1, pp. 52–81 (cit. on p. 43).

Gat, E., R. P. Bonnasso, R. Murphy, and A. Press (1997). "On Three-Layer Architectures". In: *Artificial Intelligence and Mobile Robots*. AAAI Press, pp. 195–210 (cit. on p. 61).

Gerkey, B. P. and M. J. Mataric (2002). "Sold!: auction methods for multirobot coordination". In: *Robotics and Automation, IEEE Transactions on* 18.5, pp. 758–768 (cit. on pp. 71, 72).

Gerkey, B., R. T. Vaughan, and A. Howard (2003). "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". In: *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)* (cit. on pp. 2, 61).

Hadeli, P. Valckenaers, P. Verstraete, B. Germain, and H. Van Brussel (2006). "A Study of System Nervousness in Multi-agent Manufacturing Control System". In: *Engineering Self-Organising Systems*. Ed. by S. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli. Vol. 3910. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 232–243 (cit. on p. 110).

Harary, F. (1994). *Graph Theory*. Addison-Wesley (cit. on p. 109).

Hausknecht, M., T.-C. Au, and P. Stone (Sept. 2011). "Autonomous Intersection Management: Multi-Intersection Optimization". In: *Proceedings of IROS 2011-IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*. San Francisco, USA (cit. on p. 72).

Henning, M. (Jan. 2004). "A New Approach to Object-Oriented Middleware". In: *IEEE Internet Computing* 8.1, pp. 66–75 (cit. on p. 55).

Hexmoor, H., J. Lammens, and S. Shapiro (1993). "Embodiment in GLAIR: a grounded layered architecture with integrated reasoning for autonomous agents". In: *Proceedings of the Sixth Florida AI Research Symposium (FLAIRS'93)*, pp. 325–329 (cit. on p. 61).

Hill, M. D. (Dec. 1990). "What is scalability?" In: *SIGARCH Comput. Archit. News* 18 (4), pp. 18–21 (cit. on p. 26).

Holvoet, T. and P. Valckenaers (May 2006). "Beliefs, Desires and Intentions Through the Environment". In: *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*. Ed. by P. Stone and G. Weiss. Hakodate, Japan, pp. 1052–1054 (cit. on p. 42).

Holvoet, T., D. Weyns, and P. Valckenaers (Sept. 2009). "Patterns of Delegate MAS". In: *Proceedings of the 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*. San Francisco, California, USA, pp. 1–9 (cit. on pp. 5, 28, 42).

Huang, M. (2011). "Task Learning and Execution for Behavior-Based Mobile Manipulation". PhD thesis. KU Leuven (cit. on p. 88).

Hüntemann, A. (Mar. 2011). "Probabilistic Human-Robot Navigation - Plan Recognition, User Modelling and Shared Control for Robotic Wheelchairs". PhD thesis. Celestijnenlaan 300B: KU Leuven (cit. on pp. 40, 146).

Intermodalics (2011). *An introduction to the Orocos Toolchain v2*. URL: `http://people.mech.kuleuven.be/~orocos/euRobotics2011/RTT-Overview-Short.pdf` (cit. on p. 59).

JAUS (July 2007). *Reference Architecture Specification. Version 3.3, vol. II, part 1, The Joint Architecture for Unmanned Systems*. URL: `http://www.jointrobotics.com/documents/jaus/FinalRAV3-3Part1.doc` (cit. on p. 64).

Koestler, A. (1968). *The Ghost in the Machine*. The Macmillan Company (cit. on pp. 5, 25).

Kollingbaum, M., T. Heikkila, P. Peeters, J. Matson, P. Valckenaers, D. Mcfarlane, and G. Bluemink (2000). "Emergent Flow Shop Control Based On Mascada Agents". In: *Proceedings of MIM 2000* (cit. on p. 5).

Kruchten, P. (2000). *The Rational Unified Process: An Introduction*. Addison-Wesley (cit. on p. 29).

Kruger, C. (Aug. 2002). *Constrained Cubic Spline Interpolation for Chemical Engineering Applications*. URL: http://www.korf.co.uk/spline.pdf (cit. on p. 160).

Leitão, P. and F. Restivo (2006). "ADACOR: A holonic architecture for agile and adaptive manufacturing control". In: *Computers in Industry* 57.2, pp. 121–130 (cit. on p. 67).

Leitão, P., A. Colombo, and F. Restivo (Feb. 2005). "ADACOR: a collaborative production automation and control architecture". In: *Intelligent Systems, IEEE* 20.1, pp. 58–66 (cit. on pp. 5, 67).

Makarenko, A., A. Brooks, and T. Kaupp (2006). "Orca: Components for robotics". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*, pp. 163–168 (cit. on pp. 55, 61).

— (Nov. 2007). "On the Benefits of Making Robotic Software Frameworks Thin". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07) Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*. Ed. by E. Prassler, K. Nilsson, and A. Shakhimardanov. San Diego, CA (cit. on p. 55).

Martinez, J. M. R. and J. Duffy (1995). "On the Metrics of Rigid Body Displacements for Infinite and Finite Bodies". In: *Journal of Mechanical Design* 117.1, pp. 41–47 (cit. on p. 146).

McCulloch, W. (1945). "A heterarchy of values determined by the topology of nervous nets". In: *Bulletin of Mathematical Biology* 7 (2), pp. 89–93 (cit. on p. 25).

McCulloch, W. S. (1988). *Embodiments of Mind*. MIT Press (cit. on p. 25).

McFarlane, D. C. and S. Bussmann (Sept. 2000). "Developments in Holonic Production Planning and Control". In: *International Journal of Production Planning and Control* 11.6, pp. 522–536 (cit. on pp. 16, 29).

McKinley, S. and M. Levine (1998). *Cubic Spline Interpolation*. URL: http://online.redwoods.cc.ca.us/instruct/darnold/laproj/fall98/skymeg/proj.pdf (cit. on pp. 160, 161).

McWilliams, D. L. (May 2009). "Genetic-based scheduling to solve the parcel hub scheduling problem". In: *Computers & Industrial Engineering* 56.4, pp. 1607–1616 (cit. on p. 49).

Michal, D. S. and L. Etzkorn (2011). "A comparison of Player/Stage/Gazebo and Microsoft Robotics Developer Studio". In: *Proceedings of the 49th Annual Southeast Regional Conference*. ACM-SE '11. Kennesaw, Georgia: ACM, pp. 60–66 (cit. on p. 55).

Microsoft (2008). *Microsoft Robotics Developer Studio* (cit. on p. 59).

— (2012). *VPL Introduction*. URL: http://msdn.microsoft.com/en-us/library/bb483088.aspx (cit. on p. 60).

Mondada, F., G. C. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo (2002). "SWARM-BOT: A Swarm of Autonomous Mobile Robots with Self-Assembling Capabilities". In: *Proceedings of the International Workshop on Selforganisation and Evolution of Social Behaviour*. University of Zurich, pp. 11–22 (cit. on p. 72).

Monostori, L. and B. Kádár (June 1998). "Agent Based Architectures for Mastering Changes and Disturbances in Manufacturing". In: *Tasks and Methods in Applied Artificial Intelligence, 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-98-AIE), Proceedings*. Ed. by A. Pasqual del Pobil, J. Mira, and M. Ali. Benicàssim, Castellón, Spain, pp. 755–764 (cit. on pp. 16, 29).

Monostori, L., J. Váncza, and S. Kumara (2006). "Agent-Based Systems for Manufacturing". In: *CIRP Annals - Manufacturing Technology* 55.2, pp. 697–720 (cit. on pp. 5, 28, 66).

Nesnas, I. A. D., R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu, and D. Apfelbaum (2006). "CLARAty: Challenges and Steps Toward Reusable Robotic Software". In: *International Journal of Advanced Robotic Systems* 3.1, pp. 23–30 (cit. on pp. 2, 61, 62).

Nestinger, S. (2012). *Mobile-R Robot Control System Framework*. URL: `http://mobile-r.org/rcs.php` (cit. on p. 66).

Nestinger, S. and H. Cheng (May 2011). "Mobile-R: A reconfigurable cooperative control platform for rapid deployment of multi-robot systems". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 52–57 (cit. on pp. 64, 65).

Olmi, R., C. Secchi, and C. Fantuzzi (Sept. 2011). "An efficient control strategy for the traffic coordination of AGVs". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 4615–4620 (cit. on p. 72).

Parunak, H. V. D. (1997). "'Go to the Ant': Engineering Principles from Natural Multi-Agent Systems". In: *Annals of Operations Research, special issue on Artificial Intelligence and Management Science* 75, pp. 69–101 (cit. on pp. xxv, 24).

Peasgood, M., C. Clark, and J. McPhee (Apr. 2008). "A Complete and Scalable Strategy for Coordinating Multiple Robots Within Roadmaps". In: *Robotics, IEEE Transactions on* 24.2, pp. 283–292 (cit. on p. 69).

Peeters, P., H. Van Brussel, P. Valckenaers, J. Wyns, L. Bongaerts, M. Kollingbaum, and T. Heikkilä (Oct. 2001). "Pheromone based emergent shop floor control system for flexible flow shops". In: *Artificial Intelligence in Engineering* 15.4, pp. 343–352 (cit. on p. 46).

Petri, C. A. (1966). "Communication with automata". eng. PhD thesis. Universität Hamburg (cit. on pp. 36, 80).

Petri, C. A. and W. Reisig (2008). "Petri net". In: *Scholarpedia* 3.4, p. 6477 (cit. on p. 80).

Philips, J., P. Valckenaers, E. Aertbeliën, J. Van Belle, B. Germain, H. Bruyninckx, and H. Van Brussel (2011). "PROSA and Delegate MAS in Robotics". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V. Marík, P. Vrba, and P. Leitão. Vol. 6867. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 195–204 (cit. on p. 79).

Philips, J., P. Valckenaers, H. Bruyninckx, and H. Van Brussel (2012). "Scalable and Robust Coordination of Multiple Mobile Robots using PROSA and delegate MAS". In: *Proceedings of the International Symposium on Robotics (ISR)* (cit. on p. 135).

Py, F. and F. Ingrand (2004). "Real-Time Execution Control for Autonomous Systems". In: *In Proceedings of the 2nd European Congress ERTS, Embedded Real Time Software*, pp. 21–23 (cit. on p. 62).

Quigley, M., B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng (2009). "ROS: an open-source Robot Operating System". In: *Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)* (cit. on pp. 57, 58).

Reitbauer, A., A. Battino, B. Saint Germain, A. Karageorgos, N. Mehandjiev, and P. Valckenaers (2005). "The MaBE middleware". In: *Emerging Solutions for future manufacturing systems* 159, pp. 53–60 (cit. on p. 5).

Ruiz-del-Solar, J., E. Chown, and P. Plöger (2011). *RoboCup 2010: Robot Soccer World Cup XIV*. Springer (cit. on p. 19).

Saint Germain, B. (2010). "Distributed Coordination and Control for Networked Production Systems". PhD thesis. KU Leuven (cit. on pp. 80, 82).

Saint Germain, B., P. Valckenaers, H. Van Brussel, Hadeli, O. Bochmann, C. Zamfirescu, and P. Verstraete (Apr. 2003). "Multi-agent manufacturing control: an industrial case study". In: *Intelligent Manufacturing Systems 2003, A Proceedings volume from the 7th IFAC Workshop*. Ed. by L. Monostori, B. Kádár, and G. Morel. Budapest, Hungary, pp. 207–212 (cit. on pp. 5, 46).

Saint Germain, B., P. Valckenaers, H. Van Brussel, and J. Van Belle (2011). "Networked manufacturing control: An industrial case". In: *CIRP Journal of Manufacturing Science and Technology* 4.3, pp. 324–326 (cit. on p. 47).

Saint Germain, B., P. Valckenaers, P. Verstraete, Hadeli, and H. Van Brussel (Nov. 2007). "A multi-agent supply network control framework". In: *Control Engineering Practice* 15.11, pp. 1394–1402 (cit. on p. 47).

Simon, H. A. (1968). *The Sciences of the Artificial*. MIT Press (cit. on p. 25).

Skobelev, P. (2011). "Multi-Agent Systems for Real Time Resource Allocation, Scheduling, Optimization and Controlling: Industrial Applications". In: *HoloMAS*. Ed. by V. Marík, P. Vrba, and P. Leitão. Vol. 6867. Lecture Notes in Computer Science. Springer, pp. 1–14 (cit. on pp. 5, 67).

Sprenger, R. and L. Mönch (2011). "iCoMAS: An Agent-Based System for Cooperative Transportation Planning in the Food Industry". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V. Marík, P. Vrba, and P. Leitão. Vol. 6867. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 175–184 (cit. on p. 68).

Tsui, L. Y. and C.-H. Chang (Jan. 1990). "A microcomputer based decision support tool for assigning dock doors in freight yards". In: *Computers & Industrial Engineering* 19.1-4, pp. 309–312 (cit. on p. 49).

Valckenaers, P. and H. Van Brussel (Feb. 2003). "Deadlock avoidance in flexible flow shops with loops". In: *Journal of Intelligent Manufacturing* 14.1, pp. 137–144 (cit. on p. 46).

Valckenaers, P., H. Van Brussel, M. Kollingbaum, and O. Bochmann (July 2001). "Multi-Agent Coordination and Control Using Stigmergy Applied to Manufacturing Control". In: *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001, Selected Tutorial Papers*. Ed. by M. Luck, V. Mařík, O. Stepánková, and R. Trappl. Prague, Czech Republic, pp. 317–334 (cit. on p. 46).

Valckenaers, P., J. Van Belle, and O. Ali (Sept. 2011). "PROSA and Delegate MAS for Open-Air Engineering Processes". In: *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*. Toulouse, France (cit. on p. 48).

Valckenaers, P. and H. Van Brussel (2005). "Holonic Manufacturing Execution Systems". In: *CIRP Annals - Manufacturing Technology* 54.1, pp. 427–432 (cit. on pp. 42, 46).

Valckenaers, P., H. Van Brussel, L. Bongaerts, and J. Wyns (1997). "Holonic manufacturing systems". In: *Integrated computer-aided engineering* 4, pp. 191–201 (cit. on p. 5).

Vanacker, G. (July 2008). "Adaptive Steering Behaviour Modelling for Power Wheelchair Control". PhD thesis. K.U.Leuven (cit. on pp. 10, 181).

Van Belle, J., J. Philips, O. Ali, B. Saint Germain, H. Van Brussel, and P. Valckenaers (2012). "A Service-Oriented Approach for Holonic Manufacturing Control and Beyond". In: *Service Orientation in Holonic and Multi-Agent Manufacturing Control*. Ed. by T. Borangiu, A. Thomas, and D. Trentesaux. Vol. 402. Studies in Computational Intelligence. Springer Berlin / Heidelberg (cit. on p. 45).

Van Belle, J., B. Saint Germain, P. Valckenaers, H. Van Brussel, R. Bahtiar, and D. Cattrysse (Aug. 2011a). "Intelligent Products in the Supply Chain Are Merging Logistic and Manufacturing Operations". In: *Proceedings of the 18th IFAC World Congress*. Milan, Italy (cit. on p. 50).

Van Belle, J., B. Saint Germain, P. Verstraete, P. Valckenaers, O. Ali, H. Van Brussel, and D. Cattrysse (Aug. 2009). "A Holonic Chain Conveyor Control System: An Application". In: *Holonic and Multi-Agent Systems for Manufacturing, 4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2009, Proceedings*. Ed. by V. Mařík, T. Strasser, and A. Zoitl. Linz, Austria, pp. 234–243 (cit. on p. 49).

Van Belle, J., P. Valckenaers, B. Saint Germain, R. Bahtiar, and D. Cattrysse (July 2011b). "Bio-Inspired Coordination and Control in Self-Organizing Logistic Execution Systems". In: *Proceedings of the 9th IEEE International Conference on Industrial Informatics (INDIN 2011)*. Caparica, Lisbon, Portugal, pp. 713–718 (cit. on p. 49).

Van Belle, J., B. Saint Germain, R. Bahtiar, P. Valckenaers, H. Van Brussel, and D. Cattrysse (Nov. 2010). "Towards holonic control for cross-docks". In: *Bijdragen vervoerslogistieke werkdagen 2010*. Ed. by F. J. A. Witlox and S. Weijers. Antwerp, Belgium, pp. 319–332 (cit. on p. 50).

Van Brussel, H., P. Valckenaers, J. Wyns, P. Peeters, and L. Bongaerts (June 2000). "Holonic manufacturing systems: architectural and manufacturing control issues". In: *Proceedings of 2nd CIRP International Seminar on Intelligent Computation in Manufacturing Engineering (ICME 2000)*. Capri, Italy, pp. 19–29 (cit. on p. 46).

Van Brussel, H., J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters (Nov. 1998). "Reference Architecture for Holonic Manufacturing Systems: PROSA". In: *Computers in Industry* 37.3. Special issue on intelligent manufacturing systems, pp. 255–274 (cit. on pp. 5, 16, 28, 29, 40).

van den Berg, J., J. Snoeyink, M. Lin, and D. Manocha (June 2009). "Centralized path planning for multiple robots: Optimal decoupling into sequential plans". In: *Proceedings of Robotics: Science and Systems*. Seattle, USA (cit. on p. 69).

van den Brink, S. N. (2005). *Trajectory generation for a four axis robot with linear kinematics*. Tech. rep. Technische Universiteit Eindhoven (cit. on p. 161).

Van Dorpe, J. (1997). "Navigation techniques for the mobile robot LiAS". PhD thesis. KU Leuven (cit. on p. 163).

Verstraete, P. (May 2009). "Integrating existing scheduling techniques into the Holonic Manufacturing Execution System". PhD thesis. KU Leuven (cit. on pp. 43, 46, 80, 106).

Verstraete, P., B. Saint Germain, K. Hadeli, P. Valckenaers, and H. Van Brussel (Sept. 2006). "On applying the PROSA reference architecture in multi-agent manufacturing control applications". In: *Multiagent Systems and Software Architecture, Proceedings of the Special Track at Net.ObjectDays (NODe 2006)*. Ed. by D. Weyns and T. Holvoet. Erfurt, Germany, pp. 31–47 (cit. on p. 46).

Verstraete, P., B. Saint Germain, P. Valckenaers, H. Van Brussel, J. Van Belle, and Hadeli (Jan. 2008). "Engineering manufacturing control systems using PROSA and delegate MAS". In: *International Journal of Agent-Oriented Software Engineering* 2.1, pp. 62–89 (cit. on pp. 30, 40).

Vis, I. F. and K. J. Roodbergen (Apr. 2008). "Positioning of goods in a cross-docking environment". In: *Computers & Industrial Engineering* 54.3, pp. 677–689 (cit. on p. 49).

Vokřínek, J., P. Novák, and A. Komenda (2011). "Ground Tactical Mission Support by Multi-agent Control of UAV Operations". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V. Maríık, P. Vrba, and P. Leitão. Vol. 6867. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 225–234 (cit. on p. 68).

Wagner, G. and H. Choset (Sept. 2011). "M*: A complete multirobot path planning algorithm with performance bounds". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 3260–3267 (cit. on p. 69).

Wen, M., J. Larsen, J. Clausen, J.-F. Cordeau, and G. Laporte (Dec. 2009). "Vehicle routing with cross-docking". In: *Journal of the Operational Research Society* 60.12, pp. 1708–1718 (cit. on p. 49).

Willow Garage (Feb. 2012). *ROS Concepts*. URL: `http://www.ros.org/wiki/ROS/Concepts` (cit. on p. 58).

Wyns, J. (1999). "Reference Architecture for Holonic Manufacturing Systems". PhD thesis. KU Leuven (cit. on p. 29).

Yu, W. and P. J. Egbelu (Jan. 2008). "Scheduling of inbound and outbound trucks in cross docking systems with temporary storage". In: *European Journal of Operational Research* 184.1, pp. 377–396 (cit. on p. 49).

Zamfirescu, C., P. Valckenaers, Hadeli, H. Van Brussel, and B. Saint Germain (Sept. 2003). "A Case Study for Modular Plant Control". In: *Holonic and Multi-Agent Systems for Manufacturing, 1st International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2003, Proceedings*. Ed. by V. Mařík, D. McFarlane, and P. Valckenaers. Prague, Czech Republic, pp. 268–279 (cit. on pp. 5, 46).

Zhang, K., E. G. C. Jr., and A. Barbu (2010). "A novel Stochastic Clustering Auction for task allocation in multi-robot teams". In: *2010 IEEE/RSJ*

International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan. IEEE, pp. 3300–3307 (cit. on p. 72).

Curriculum Vitae

Personal data

Name:	Johan Philips
Date of birth:	14th of January, 1983
Place of birth:	Ekeren (Antwerp), Belgium
Work Address:	Robotics Division Department of Mechanical Engineering KU Leuven Celestijnenlaan 300B bus 2420 B-3001 Heverlee, Belgium
Tel:	+32 16 32 24 80
Email:	johan.philips@mech.kuleuven.be

Education

2007-2012:	PhD candidate at the Robotics Division, Department of Mechanical Engineering, KU Leuven.
2005-2006:	Master of Artificial Intelligence at the KU Leuven. Magna cum laude.
2003-2005:	Licentiaat in de Informatica at the KU Leuven. Cum laude.
2001-2003:	Kandidaat in de Informatica at the KU Leuven. Cum laude.
1995-2001:	Major in Mathematics and Greek at St. Michielscollege, Brasschaat. Cum laude.

Experience

Research on:
- Multi-robot path planning and coordination
- Holonic robotic systems
- Robot software engineering

Teaching assistant activities:
- Daily supervisor of one master thesis
- Teaching assistant for Uncertainty Reasoning in Knowledge Systems
- Teaching assistant for Sensors and Measurement Systems

Supporting activities:
- Website and databases maintenance
- Organising committee member, MAIA workshop, Leuven, 2007
- Organising committee member, Erlang workshop, INCOM, 2012

List of Publications

Publications in international peer reviewed journals

Galán, F., M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips, and J. R. Millán (July 2008). "A brain-actuated wheelchair: asynchronous and non-invasive brain-computer interfaces for continuous control of robots". In: *Clinical Neurophysiology* 119, pp. 2159–2169.

Valckenaers, P., H. Van Brussel, H. Bruyninckx, B. Saint Germain, J. Van Belle, and J. Philips (Aug. 2011). "Predicting the unexpected". In: *Comput. Ind.* 62 (6), pp. 623–637.

Vanacker, G., J. R. Millán, E. Lew, P. W. Ferrez, F. Galán, J. Philips, H. Van Brussel, and M. Nuttin (Jan. 2007). "Context-based filtering for assisted brain-actuated wheelchair driving". In: *Computational Intelligence and Neuroscience* 2007, p. 3.

Van Belle, J., J. Philips, O. Ali, B. Saint Germain, H. Van Brussel, and P. Valckenaers (2012). "A Service-Oriented Approach for Holonic Manufacturing Control and Beyond". In: *Service Orientation in Holonic and Multi-Agent Manufacturing Control*. Ed. by T. Borangiu, A. Thomas, and D. Trentesaux. Vol. 402. Studies in Computational Intelligence. Springer Berlin / Heidelberg.

Vanhooydonck, D., E. Demeester, A. Hüntemann, J. Philips, G. Vanacker, H. Van Brussel, and M. Nuttin (Aug. 2010). "Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model". In: *Robot. Auton. Syst.* 58 (8), pp. 963–977.

Papers at international conferences, published in full in proceedings

Demeester, E., J. Philips, A. Hüntemann, E. Vander Poorten, and H. Van Brussel (June 2011). "Active, lifelong sensor synchronization: A Kalman filtering approach". In: *Advanced Robotics (ICAR), 2011 15th International Conference on*, pp. 501–506.

Demeester, E., A. Hüntemann, J. Philips, and E. Vander Poorten (2012). "Probabilistic navigation assistance for robotic wheelchairs: Comparison of ML, MAP and greedy POMDP techniques". In: *Proceedings of the International Symposium on Robotics (ISR)*.

Galán, F., M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips, H. V. Brussel, and J. Millán (Nov. 2007a). "An Asynchronous and Non-Invasive Brain-Actuated Wheelchair". In: *13th International Symposium on Robotics Research (ISRR 2007)*. Hiroshima, Japan.

Galán, F., M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips, H. Van Brussel, and J. R. Millán (Nov. 2007b). "An asynchronous and non-invasive brain-actuated wheelchair". In: *Proceedings of the 13th International Symposium on Robotics Research*. Vol. 128. Hiroshima, Japan.

Galán, F., M. Nuttin, D. Vanhooydonck, E. Lew, P. W. Ferrez, J. Philips, and J. R. Millán (Sept. 2008). "Continuous brain-actuated control of an intelligent wheelchair by human eeg". In: *4th International Brain-Computer Interface Workshop & Training Course*. IDIAP-RR 08-53. Graz University of Technology, Graz, Austria.

Grave de Peralta, R., Q. Noirhomme, J. Philips, G. Vanacker, D. Vanhooydonck, M. Nuttin, and S. Gonzalez Andino (2007). "Telepresence based on remote real time control of a robot with brain waves". In: *Annual Conference Proceedings of 10th Annual International Workshop on Presence*, pp. 383–384.

Hüntemann, A., E. Demeester, G. Vanacker, D. Vanhooydonck, J. Philips, H. Van Brussel, and M. Nuttin (Nov. 2007). "Bayesian plan recognition and shared control under uncertainty: assisting wheelchair drivers by tracking fine motion paths". In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 3360–3366.

Millan, J., F. Galan, D. Vanhooydonck, E. Lew, J. Philips, and M. Nuttin (Sept. 2009). "Asynchronous non-invasive brain-actuated control of an intelligent wheelchair". In: *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pp. 3361–3364.

Millán, J., F. Galán, E. Lew, P. W. Ferrez, J. Philips, G. Vanacker, and M. Nuttin (2007). "Shared-Control of a Brain-Actuated Wheelchair in Natural Environments". In: *Swiss Society of Biomedical Engineering 2007 Meeting (SSBE 2007)*.

Philips, J., J. R. Millán, G. Vanacker, E. Lew, F. Galán, P. W. Ferrez, H. Van Brussel, and M. Nuttin (June 2007). "Adaptive shared control of a brain-actuated simulated wheelchair". In: *Proceedings of the 10th IEEE International Conference on Rehabilitation Robotics*. Noordwijk, The Netherlands, pp. 408–414.

Philips, J., P. Valckenaers, E. Aertbeliën, J. Van Belle, B. Germain, H. Bruyninckx, and H. Van Brussel (2011). "PROSA and Delegate MAS in Robotics". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V. Marík, P. Vrba, and P. Leitão. Vol. 6867. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 195–204.

Philips, J., P. Valckenaers, H. Bruyninckx, and H. Van Brussel (2012). "Scalable and Robust Coordination of Multiple Mobile Robots using PROSA and delegate MAS". In: *Proceedings of the International Symposium on Robotics (ISR)*.

Vander Poorten, E., E. Demeester, E. Reekmans, J. Philips, A. Hüntemann, and J. De Schutter (2012). "Powered Wheelchair Navigation Assistance through Kinematically Correct Environmental Haptic Feedback". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*.