



Self-Adaptive and Self-Organised Planning and Decision-Making for Multi-Robot Systems

vorgelegt von
M.Sc. Informatik
Christopher-Eyk Hrabia
ORCID: 0000-0002-5220-1627

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Oliver Brock

Gutachter: Prof. Dr. Dr. h. c. Sahin Albayrak (TU Berlin)

Gutachter: Prof. Dr.-Ing. Uwe Meinberg (BTU Cottbus-Senftenberg)

Gutachter: Prof. Dr. Mihhail Matskin (KTH Royal Institute of Technology)

Tag der wissenschaftlichen Aussprache: 14.11.2019

Berlin 2019

Abstract

Robots are leaving the friendly, well-structured world of automation and are facing the challenges of a dynamic world. The uncertain conditions in the dynamic world call for a high degree of robustness and adaptivity for individual robots as well as interactions between multiple robots and other system entities. The uncertainty makes it difficult for designers and engineers to anticipate all conditions, interactions, and side effects a system will have to deal with while the system is specified and developed. Furthermore, implicit and explicit coordination is required to perform a joint goal with multiple entities in a multi-robot system. Enabling scalability for multi-robot applications can be especially supported by means of implicit and decentralised coordination approaches. Nevertheless, robots that adapt to the dynamic environment and coordinate themselves still have to pursue their given tasks or goals.

This thesis researches how multi-purpose, mobile, multi-robot systems can be enhanced to operate more adaptively in dynamic environments. This is done by analysing and exploring the combination of so far separated research directions of goal-driven decision-making and planning as well as self-adaptation and self-organisation.

The presented hybrid decision-making and planning framework is integrated into the popular robotic middleware Robot Operating System (ROS). The solution combines symbolic planning with reactive behaviour networks, automated task delegation, reinforcement learning, and pattern-based selection of suitable self-organisation mechanisms. On that account, it brings together the advantages of bottom-up and top-down oriented architectures for task-level control of multi-robot systems. The developed framework enables a coherent and integrated design and implementation of decision-making and planning as well as coordination application logic within one software ecosystem that features a common domain model and a modular architecture. This results in a simplification of the development of adaptive multi-purpose multi-robot systems by avoiding system discontinuities and enabling a holistic view on the actual implementation.

The presented approach has been successfully evaluated in various research projects and international competitions in the field of robotics and multi-agent systems.

Zusammenfassung

Roboter verlassen ihre etablierte, strukturierte Welt der Automatisierung und stellen sich den Herausforderungen der dynamischen Umwelt. Die zum Teil unbekanntenen Bedingungen in der dynamischen Umwelt erfordern ein hohes Maß an Robustheit und Adaptivität für den einzelnen Roboter, als auch für die Interaktion mehrerer Roboter untereinander. Diese Bedingungen machen es für Entwickler schwierig alle Zustände, Interaktionen und Seiteneffekte für ein System im voraus zu spezifizieren. Zusätzlich muss für eine gemeinsame Erfüllung von Zielen durch mehrere Roboter eine explizite oder implizite Koordination erfolgen. Hier kann vor allem eine implizite und dezentrale Koordination eine gute Skalierbarkeit unterstützen. Trotz eines Fokusses auf Adaptivität, müssen die Robotersysteme aber zugleich auch gegebene Ziele und Aufgaben erfüllen.

Diese Dissertation erforscht wie multifunktionale, mobile Multi-Roboter-Systeme verbessert werden können, um adaptiver und robuster in dynamischen Umgebungen zu operieren. Dazu wird speziell eine Kombination aus den bisher unabhängig betrachteten Forschungsrichtungen der zielgerichteten Planung und Entscheidungsfindung, als auch der Selbst-Adaptation und Selbst-Organisation untersucht.

Das vorgestellte hybride Entscheidungsfindungs- und Planungsframework ist dabei integriert in die weitverbreitete Robotik-Middleware Robot Operating System (ROS). Speziell kombiniert die realisierte Lösung symbolische Planung mit reaktiven Verhaltensnetzwerken, automatischer Aufgabenverteilung, verstärkendes Lernen und musterbasierte Auswahl von Selbstorganisationsmechanismen. Auf dieser Grundlage vereint das System die Vorteile von Bottom-Up- und Top-Down-Architekturen für die Steuerung von Multi-Roboter-Systemen auf Aufgabenebene. Weiterhin ermöglicht die modulare Architektur und das übergreifende Domänenmodell innerhalb eines Softwareökosystems ein einheitliches und integriertes Design der Planungs-, Entscheidungsfindungs-, und Koordinationslogik. Das resultiert in einer Vereinfachung der Entwicklung von adaptiven und multifunktionalen Multi-Roboter-Systemen durch die Vermeidung von Systembrüchen in einem holistischen Ansatz.

Das vorgestellte System wurde erfolgreich in verschiedenen Forschungsprojekten und internationalen Wettkämpfen aus dem Bereich Multi-Agenten- und Multi-Roboter-Systeme evaluiert.

Acknowledgments

First of all, I wish to acknowledge my supervisor Prof. Dr. Dr. h. c. Sahin Albayrak for providing the opportunity, the environment, and the infrastructure that enabled me to pursue my research at DAI-Labor, Technische Universität Berlin. Furthermore, I would also like to express my very great appreciation to Prof. Dr.-Ing. Uwe Meinberg and Prof. Dr. Mihhail Matskin for taking over the role as external reviewers.

In particular, I am especially thankful for all enthusiastic colleagues that collaborated with me in various research activities, provided feedback, and shared their knowledge with me during our common time in the working groups RAS (Robotics and Autonomous Systems) and ACT (Agent Core Technologies), namely Nils Masuch, Michael Burkhardt, Orhan-Can Görür, Dr. rer. nat. Yuan Xu, Martin Berger, Christian Rackow, Tuguldur Erdene-Ochir, and Dr.-Ing. Jan Keiser.

Some colleagues from RAS and ACT deserve special acknowledgement. For this reason, I would like to offer my special thanks to the head of my group Dr.-Ing. Axel Heßler, who challenged me with uncomfortable questions but always supported me in all my research attempts, ideas, and own ways of thinking. Furthermore, I am particularly grateful for the assistance of Prof. Dr.-Ing. Johannes Fähndrich for his altruistic support, guidance, and constructive feedback at the very final stage of my dissertation. Likewise, the advice of Dr.-Ing. Marco Lützenberger put me on the right track at the beginning of my dissertation process. Furthermore, special thanks to my colleague Dr.-Ing. Tobias Küster, who was the most committed reviewer of all my research writing over the years. Moreover, my grateful thanks to my colleague Denis Pozo for sharing and discussing all our fears and doubts about pursuing a Ph.D.

Not to forget, I am particularly grateful for all the insights and ideas I could develop through the discussions and provided feedback during the supervision of many Bachelor's and Master's theses. It was a pleasure to mentor Stephan Wypler, Phillip Erik Rieger, Tanja Katharina Kaiser, Alexander Wagner, Patrick Marvin Lehmann, Vito Felix Mengers, and Michael Franz Ettliger.

Most importantly, very special thanks to my wife Anne, for her love, overall support in my life, and acceptance of all the necessary extra hours that allowed me to pursue my dissertation.

Table of Contents

Table of Contents	ix
List of Figures	xiii
List of Tables	xv
List of Listings	xvii
Acronyms	xviii
Publications	xxi
I. Introduction	1
1. Motivation	2
2. Problem Statement	3
3. Research Questions	6
4. Structure of the Document	8
II. Analysis	13
5. Foundational Concepts	15
5.1. Autonomous Agents and Multi-Agent Systems	16
5.2. Emergence and Swarm Intelligence	18
5.3. Swarm Robotics	20
5.4. Self-Organisation and Self-Adaptation	22
5.5. Common Characteristics of Emergence, Swarm Intelligence, Self-Adaptation, and Self-Organisation	24
6. Self-Organisation Mechanisms and their Application	25
7. Related Surveys about Emergence, Self-Organisation and Self-Adaptation	28
8. Self-Adaptation and Self-Organisation Software Development Methodologies	31
9. Middleware Solutions and Frameworks	35
10. Mechanism Design and Implementation	41
10.1. Evolution, Learning and Simulation	42
10.2. Functional Languages	43
10.3. Application-specific Approaches	44
10.4. Declarative Approaches	45

Table of Contents

10.5. Analysis, Verification and Validation	48
11. Planning and Decision-Making in Robotics	49
12. Task Allocation, Decomposition, and Delegation	55
12.1. Decomposition	56
12.2. Task Allocation and Delegation	57
13. Reinforcement Learning	58
13.1. Problem Classification	59
13.2. Reinforcement Learning Methods	60
13.3. Comparing Value Function and Policy Gradient Methods	61
13.4. Deep Q-Network	62
13.5. Exploration vs. Exploitation	64
14. Analysis Summary: Opportunities and Challenges	64
III. Concept	71
15. Objectives	72
16. Methodology	75
17. Architecture	84
IV. Detailed Concepts and Implementation	93
18. Hybrid Behaviour Planner Core	95
18.1. Behaviour-Network Base	95
18.1.1. Model	96
18.1.2. Decision-Making	99
18.2. Symbolic Planner Extension	105
18.3. ROS-Integration	106
18.4. Knowledge Base: Sharing information in a hybrid behaviour network . .	110
18.5. Cores of Cores: Hybrid Behaviour Networks in a Hierarchy	116
18.6. Summary Decision-Making and Planning Core	119
19. Combining Self-Organisation and Hybrid Behaviour Planning	119
19.1. Self-Organisation Framework	119
19.2. Self-Organisation Pattern Implementations	124
19.2.1. Movement Patterns	125
19.2.2. Decision Patterns	127
19.3. Combining Decision-Making, Planning and Self-Organisation	128
19.4. Selecting Self-Organisation Mechanisms	132

19.5. Summary Combining Decision-Making and Planning with Self-Organisation	135
20. Increasing Adaptability with Reinforcement Learning	136
20.1. Integration Concept	136
20.2. Algorithmic Integration	139
20.2.1. Activation Algorithm	139
20.2.2. Exploration Strategy	141
20.2.3. Reward	142
20.3. Reinforcement Learning Extension Architecture	143
20.4. Reinforcement Learning Algorithm Implementation	147
20.4.1. Adaptation of the DDQN approach	148
20.4.2. Parametrisation	151
20.5. Reinforcement Learning Integration Summary	154
21. Task Decomposition, Allocation and Delegation in a Multi-Robot System	154
21.1. Delegation Component Concept	156
21.2. Decomposition	158
21.3. Allocation	161
21.3.1. Selecting an Agent	161
21.3.2. Delegation	166
21.3.3. Communication	167
21.4. Task Delegation Component Architecture	168
21.4.1. Delegation Module	168
21.4.2. RHBP Decomposition	170
21.5. Summary Decomposition, Allocation, and Delegation	172
V. Application and Evaluation	173
22. Experiments	174
22.1. Maes Automation Scenario	175
22.2. Planning and Decision-Making in a Multi-Robot Simulation	179
22.3. Multi-Robot Experiment Self-Organisations with Decision-Making and Planning	184
22.4. Learning and Self-Adaptation in the Maes Automation Scenario	189
23. Projects	192
23.1. SpaceBot Cup 2015 Project	193

Table of Contents

23.2. Multi-Agent Programming Contest	197
23.2.1. Multi-Agent Programming Contest 2017	198
23.2.2. Multi-Agent Programming Contest 2018	210
23.2.3. Multi-Agent Programming Contest Conclusion	222
23.3. EffFeu Project: Efficient Operation of Unmanned Aerial Vehicles for Industrial Firefighters	223
23.3.1. EffFeu System Architecture	225
23.3.2. Mission-guided Control with RHBP	227
23.3.3. EffFeu Project Summary	240
24. Comparison with other Frameworks	241
24.1. Comparison against symbolic planning frameworks, behaviour networks and hybrid planning approaches	242
24.2. Comparison against other decomposition and delegation approaches . . .	243
VI. Summary and Discussion	249
25. Conclusion	250
26. Revisiting Research Questions, Objectives, and Requirements	252
27. Discussion	256
28. Future Work	259
Bibliography	261

List of Figures

4.1.	Document at a glance: Argumentation structure and section relationships.	10
17.1.	Combined three to four tiered architecture for realising robotic systems.	85
17.2.	System architecture and architecture layers of a single agent.	86
17.3.	System architecture with multiple agents on different hierarchy/abstraction levels.	87
18.1.	Behaviour network components and their relationship.	97
18.2.	RHBP core ROS node and communication architecture.	107
18.3.	Graphical RHBP Monitoring and control in ROS.	109
18.4.	Graphical knowledge base monitoring, inspection and control in ROS. .	115
18.5.	UML Class diagram illustrating the architectural integration of NetworkBehaviours in the RHBP framework.	117
19.1.	Design patterns for self-organisation and their relationships	120
19.2.	Framework of bio-inspired self-organisation design pattern dependencies.	121
19.3.	Visualised gradient with attractive/repulsive center, goal radius, diffusion radius, and orientation.	122
19.4.	Architecture for the integration of self-organisation into the RHBP. . . .	129
19.5.	Integration of the Coordination Mechanism Selector in the structure of the RHBP.	132
19.6.	Architecture of the Coordination Mechanism Selector.	133
20.1.	Details about the Reinforcement Learner component within a simplified system architecture for a single agent.	144
20.2.	DDQN learning and prediction architecture and process.	150
21.1.	The process of the auctioneer.	162
21.2.	The process of the auction participant.	164
21.3.	Object-oriented delegation module and RHBP integration architecture. .	169

List of Figures

22.1.	RHBP behaviour model of the Maes' scenario.	177
22.2.	RHBP activation plot of the hybridly planned scenario from Maes.	178
22.3.	RHBP activation plot of the scenario from Maes with behaviour network only.	179
22.4.	RHBP behaviour model of the multi-robot turtle experiment.	181
22.5.	Turtlesim multi-robot example scenario.	183
22.6.	Visualised self-organisation simulation scenario.	186
22.7.	Model of the RHBP solution for an individual robot of the experiment illustrating the relationships with preconditions, behaviours and effects.	187
22.8.	Required decision steps until mission success or failure over learning episodes.	191
23.1.	SpaceBot Cup behaviour model.	195
23.2.	Agent architecture of a single agent in MAPC 2017.	201
23.3.	RHBP Behaviour model for agent task execution of TUBDAI 2017.	204
23.4.	System load during experimental match with 3 simulations á 1000 steps.	207
23.5.	Behaviour model for agent task execution of Dumping to gather.	215
23.6.	High-level decision-making behaviour model for agent task execution of TUBDAI 2018.	220
23.7.	EffFeu system architecture. Arrows indicate the direction of the data flow.	225
23.8.	RHBP component relationships and external user goal API.	229
23.9.	RHBP component relationships and external user goal API in a multi-robot setup.	230
23.10.	EffFeu project mid-project milestone scenario behaviour network model.	231
23.11.	More complex behaviour network model including more and alternative behaviours for similar effects and additional goals.	233
23.12.	Comparison between different characteristic symbolic planning influences specified by weights.	236
23.13.	Multi-robot scenario applying task decomposition, allocation, and delegation.	239

List of Tables

6.1. Classification of self-organisation mechanisms to patterns and applications in multi-robot systems.	27
14.1. Comparison of the properties of reviewed approaches from self-adaptation, self-organisation and swarm robotics.	66
14.2. Comparison of robot planning and decision-making approaches.	68
16.1. Comparison of symbolic planning frameworks, behaviour networks and hybrid planning approaches.	81
21.1. Message types with descriptions, most relevant content parameters, and ROS realisation.	168
22.1. Comparison of a RHBP behaviour network layer only configuration against the hybrid-planning configuration in a multi-robot path finding scenario.	184
22.2. Experiment results of a comparison between different available self-organisation patterns for collision avoidance.	188
22.3. Mission steps and success differentiated by exploration.	192
23.1. MAPC 2017 final ranking.	208
23.2. MAPC 2017 simulation match results.	209
23.3. MAPC 2018 final ranking.	211
23.4. MAPC 2018 simulation match results.	212
24.1. Comparison of RHBP against existing symbolic planning frameworks, behaviour networks and hybrid planning approaches.	244
24.2. Comparing RHBP decomposition features against existing solutions.	246
24.3. Comparison of the RHBP delegation features against existing solutions.	247

List of Listings

18.1. Example of a RHBP model definition in Python.	111
19.1. The gradient data message structure in pseudocode.	123
19.2. Python example of the RHBP self-organisation extension applied in a simplified behaviour model definition.	131
19.3. Python example of the application of the SO Coordinator within a RHBP behaviour model definition.	135
23.1. Example PDDL domain generated by RHBP for one agent.	205
23.2. Example PDDL problem generated by RHBP for one agent.	206

Acronyms

ADL Action Description Language.

AI Artificial Intelligence.

ANN Artificial Neural Network.

AOSE Agent-oriented Software Engineering.

API Application Programming Interface.

AUML Agent Unified Modelling Language.

BDI Belief, Desire, and Intention.

CFP Call for Proposal.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DDQN Double DQN.

DMSL Decision Making Specification Language.

DQN Deep Q-Network.

eBT extended Behaviour Tree.

FACPL Formal Access Control Policy Language.

FESAS Framework for Engineering Self-Adaptive Systems.

FIFO First In - First Out.

GNSS Global Navigation Satellite System.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

HSM Hierarchical State Machine.

HTN Hierarchical Task Network.

ID Identification.

IDE Integrated Development Environment.

JIAC Java-based Intelligent Agent Componentware.

LDP Locally Distributed Predicate.

LSA Live Semantic Annotation.

MADP Multi-Agent Decision Process.

MAPC Multi-Agent Programming Contest.

MAPE Monitoring, Analysis, Planning and Execution.

MAPE-K Monitoring, Analysis, Planning, Execution and Knowledge.

MARL Multi-Agent Reinforcement Learning.

MASSim Multi-Agent Systems Simulation Platform.

MDP Markov Decision Process.

OpenUP Open Unified Process.

OSL Origami Shape Language.

OWL Web Ontology Language.

OWL-DL Web Ontology Language Description Logics.

PDDL Planning Domain Definition Language.

POMDP Partially-Observable Markov Decision Process.

PPDDL Probabilistic Planning Domain Definition Language.

PSCEL Policed SCEL.

QoS Quality of Service.

RDDL Relational Dynamic Influence Diagram Language.

RDF Resource Description Framework.

RGBD Red Green Blue Depth.

RHBP ROS Hybrid Behaviour Planner.

RL Reinforcement Learning.

ROS Robot Operating System.

ROSCo ROS Commander.

RTOS Real Time Operating System.

SCEL Service Component Ensemble Language.

SLAM Simultaneous Localization and Mapping.

SPARQL SPARQL Protocol and RDF Query Language.

SPARUL SPARQL/Update.

StocS Stochastic Extension of SCEL.

TCP Transmission Control Protocol.

TDL Task Description Language.

TOTA Tuples on the Air.

TUB Technische Universität Berlin.

TuCSoN Tuple Centres Spread over the Network.

UAS Unmanned Aerial System.

UAV Unmanned Aerial Vehicle.

UML Unified Modelling Language.

Publications

The major contributions of this thesis are based on peer-reviewed conference papers, journal papers and book chapters. Nevertheless, in-depth details about the approach, implementation, and evaluation are made available for the first time within this document.

Journal Papers and Peer-reviewed Book Chapters

- Christopher-Eyk Hrabia, Michael Franz Ettliger, and Axel Hessler. ROS Hybrid Behaviour Planner: Behaviour Hierarchies and Self-Organisation in the Multi-Agent Programming Contest. *The Multi-Agent Programming Contest (MAPC 2018) (Lecture Notes in Computer Science)*, Springer International Publishing, to appear.
- Christopher-Eyk Hrabia, Marc Schmidt, Andrea Marie Weintraud, and Axel Hessler. Distributed Decision-Making based on Shared Knowledge in the Multi-Agent Programming Contest. *The Multi-Agent Programming Contest (MAPC 2018) (Lecture Notes in Computer Science)*, Springer International Publishing, to appear.
- Christopher-Eyk Hrabia, Axel Hessler, Yuan Xu, Jacob Seibert, Jan Brehmer, and Sahin Albayrak. EffFeu Project: Towards Mission-Guided Application of Drones in Safety and Security Environments. *Sensors (Volume 19) - Special Issue Unmanned Aerial Vehicle Networks, Systems and Applications*, MDPI Basel, 2019.
- Christopher-Eyk Hrabia, Marco Lützenberger, and Sahin Albayrak. Towards Adaptive Multi-Robot Systems: Self-Organization and Self-Adaptation. *The Knowledge Engineering Review*, e16, Cambridge University Press, 2018.
- Christopher-Eyk Hrabia, Patrick Marvin Lehmann, Nabil Battjbuier, Axel Hessler, and Sahin Albayrak. Applying Robotic Frameworks in a Simulated Multi-Agent Contest. *Annals of Mathematics and Artificial Intelligence*, 1–22, Springer International Publishing, 2018.
- Christopher-Eyk Hrabia, Martin Berger, Axel Hessler, Stephan Wypler, Jan Brehmer, Simon Matern, and Sahin Albayrak. An autonomous companion UAV for the

SpaceBot Cup competition 2015. *Robot Operating System (ROS) - The Complete Reference (Volume 2)*, 345–385, Springer International Publishing, 2017.

Conference Papers

- Christopher-Eyk Hrabia, Patrick Marvin Lehmann, and Sahin Albayrak. Increasing Self-Adaptation in a Hybrid Decision-Making and Planning System with Reinforcement Learning. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, USA, July 2019*, 469–478, IEEE, 2019.
- Christopher-Eyk Hrabia, Axel Hessler, Yuan Xu, Jan Brehmer, and Sahin Albayrak. EffFeu project: Efficient Operation of Unmanned Aerial Vehicles for Industrial Fire Fighters. In *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, DroNet 2018, Munich, Germany, June 15, 2018*, 33–38, ACM, 2018.
- Christopher-Eyk Hrabia, Tanja Katharina Kaiser, and Sahin Albayrak. Combining Self-Organisation with Decision-Making and Planning. In *Multi-Agent Systems and Agreement Technologies: 15th European Conference, EUMAS 2017, and 5th International Conference, AT 2017, Evry, France, December 14-15, 2017*, 385–399, Springer International Publishing, 2017.
- Christopher-Eyk Hrabia, Stephan Wypler, and Sahin Albayrak. Towards Goal-driven Behaviour Control of Multi-Robot Systems. In *2017 IEEE 3rd International Conference on Control, Automation and Robotics (ICCAR), Nagoya, Japan, April 2017*, 166–173, IEEE, 2017.
- Christopher-Eyk Hrabia. A Framework for Adaptive and Goal-Driven Behaviour Control of Multi-Robot Systems. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), Augsburg, Germany, September 2016*, 166–173, IEEE, 2016. (*Won Best Doctoral Symposium Paper Award*)
- Christopher-Eyk Hrabia, Nils Masuch, and Sahin Albayrak. A Metrics Framework for Quantifying Autonomy in Complex Systems. In *Multiagent System Technologies: 13th German Conference, MATES 2015 Revised Selected Papers (Lecture Notes in Computer Science), Cottbus, Germany, September 2015*, 22–41, Springer International Publishing, 2015.

- Christopher-Eyk Hrabia. Simplified planning and control with biologically inspired mechanisms for multi-robot systems. In *MATES Doctoral Consortium 2014, Clausthal, Germany, September 2014*, 5–10, Technische Universität Clausthal, 2014.

Supervised Bachelor's and Master's Theses

- Michael Franz Ettliger. Self-Organised Coordination and Hybrid Behaviour Planning in the Multi-Agent Programming Contest 2018.
Master's Thesis, Technische Universität Berlin, December 2018.
- Vito Felix Mengers. Automated task decomposition within a hybrid behaviour network architecture of multi-robot systems.
Bachelor's Thesis, Technische Universität Berlin, October 2018.
- Patrick Marvin Lehmann. Integration von Reinforcement Learning in ein hybrides Multi-Agenten Planungssystem.
Master's Thesis, Technische Universität Berlin, August 2018.
- Alexander Wagner. Vergleichende Analyse von ROS-integrierten Taskplanungsansätzen in Multi-Roboter-Szenarien.
Bachelor's Thesis, Technische Universität Berlin, November 2017.
- Tanja Katharina Kaiser. Combining Self-Organization with Hybrid Behaviour Networks.
Master's Thesis, Technische Universität Berlin, May 2017.
- Phillip Erik Rieger. Improving Modelling of a Hybrid Planning System.
Bachelor's Thesis, Technische Universität Berlin, May 2017.
- Leonard Slonina. Incorporating Learning into Hybrid Behaviour Planning.
Bachelor's Thesis, Technische Universität Berlin, November 2016.
- Stephan Wypler. Development of a Hybrid Planner for ROS.
Master's Thesis, Technische Universität Berlin, January 2016.

Part I.

Introduction

1. Motivation

Robotic systems are able to support or replace humans in a wide variety of tasks and are capable of operating in various situations and environmental conditions. Deployed robot systems in the industry, like in manufacturing or recycling, enable operation in environments that are dangerous for humans. Furthermore, some robots allow processing higher workloads or achieve higher precision than their human counterparts. This successful application is possible due to static and predictable environments, where engineers have considered and tested all possible world and system states in advance. The required complete considerations and tests generate great effort during the design and development of robots.

Nowadays, robotic systems are about to replace and support humans in more dynamic environments. They are mobile, execute different context-dependent actions, and get more and more feature-rich. Robots are leaving the friendly, well-structured world of automation and are facing the challenges of a dynamic world. Examples for upcoming applications are robot-human co-workers in the industry, service-robots in households, restaurants, offices, or tourist guides in public space. At the moment the application of robots beyond automation is mostly limited to single robots like service robots for simple domestic applications (cleaning) (Kleiner et al., 2017), social entertainment for children (Eguchi and Okada, 2018) or flying robots (drones) for surveillance (Li et al., 2018), as well as inspection tasks (Yang et al., 2015b).

Furthermore, many applications would also benefit from the operation of more than one robot, a comprehensive list of potential advantages and disadvantages of multi-robot systems is collected by Arkin, 1998 and Yogeswaran and Ponnambalam, 2010. On the one hand, the following advantages are mentioned. *Parallelism* allows to perform divisible tasks in parallel to increase efficiency. Likewise, certain tasks might be only feasible for a group of robots due to different capabilities or the required *combination of forces*. Moreover, distributed robots are automatically increasing the spatial coverage for *distributed detection*, which also results in a wider detection range in comparison to a

single robot. Similarly, applying *distributed actions* with multiple robots enables to act at different places at the same time. Using multiple robots can increase the *robustness* of the system, thanks to *redundancy* because the failure of a single robot does not imply a failed mission. Further, a distributed system with multiple robots can easily be *scaled-up* by adding additional entities. Correspondingly, the *heterogeneity* of different specialist robots allows to increase the efficiency of the system by utilising their special properties. Additionally, using multiple robots allows to select the right amount and best matching roles dynamically depending on the problem to solve, which increases the *flexibility*. Using multiple simple, potentially specialist, robots can be *cheaper* than having a single powerful robot for each separate task.

On the other hand, only a few disadvantages are listed. A group of robots can suffer from *interferences* in terms of perception, motion, communication, and operation. If a system consists of multiple robots, not all information can be omnipresent in the system, which leads to *uncertainty* for the individuals. A multi-robot system is not necessarily cheaper than a single more powerful robot, thus *total system cost* can eventually be higher.

Various possible application domains for multi-robot systems are conceivable and are already explored, for example, space exploration (Yliniemi, Agogino, and Tumer, 2014) as well as disaster rescue operations with aerial, and ground robots (Sheh, Schwertfeger, and Visser, 2016). Applications that are more industrial are warehouse logistics (Digani et al., 2015) or service robots operating as a team in future smart-buildings environments, such as hospitals (Mettler, Sprenger, and Winter, 2017). In consequence, an increasing application of mobile multi-robot systems in various scenarios is expected in the near future. The upcoming and envisioned applications of multi-robot systems in dynamic environments are difficult to realise with established approaches and have to encounter mentioned disadvantages as well as other induced challenges that are discussed in the following.

2. Problem Statement

The aforementioned application domains introduce dynamic environments and require complex interactions between multiple robots and other system entities, like humans or software components. The dynamics and complex interaction induce uncertainty. ‘Uncertainty is a system state of incomplete or inconsistent knowledge [...]’ (Ramirez, Jensen, and Cheng, 2012, p.101). Uncertainty makes it difficult for designers and engineers to anticipate all conditions, interactions, and side effects a system will have to deal with, while the system is specified and developed. If it is not impossible to cover all those conditions, it will increase the effort exponentially. In consequence, robots cannot simply apply fixed sequences of actions, control and feedback loops, or simple state-machines in such environments anymore. Instead, robots have to select dynamically the most appropriate action that addresses the current context as well as the targeted system goal in an adaptive manner. For this reason, robots have to exhibit more adjustable autonomy. Adjustable autonomy refers to the ability to make decisions without external constraints in a specified scope (Sierhuis et al., 2003). Such adjustable autonomy allows for an adaptive selection of suitable actions and increases the ability to handle unexpected situations in a more appropriate manner. The need to develop multi-robot systems that operate in uncertain environments generates demand for systems that put emphasis on robustness and adaptivity rather than on behaving in an optimal fashion, as for instance required in static automation tasks, like car manufacturing.

Addressing more complex problems or tasks with multiple robots allows exploiting various advantages that have been already mentioned. For instance, using multiple robots can improve the reliability and efficiency of a system by making use of parallel operation as well as on-demand takeovers between the entities. Other problems are not feasible for single robots, e.g. in construction, because of missing capabilities. In this case, other team members might provide the missing capability. Alternatively, a task is executed in a collective fashion, combining the strength of several entities. Hence, especially the distribution of a task amongst several robots can be necessary or beneficial to achieve a system goal. However, coordination is required to perform a joint goal with a multi-robot system. Coordination allows for the consistence of the activities within a group of robots (Gancet and Lacroix, 2007) and it is enabled through mechanisms that man-

age resource conflicts. Resource conflicts might occur, e.g. for time, space, energy, and specific items (Gancet and Lacroix, 2007). Coordination can be accomplished explicitly or implicitly (Parker, 1998). The explicit assignment of tasks to robots or teams of robots is described as multi-robot task allocation. Alternatively, in implicit or emergent approaches (Gerkey and Mataric, 2002) the coordination arises from local interactions between members of the robot team and the environment. Explicit coordination is more concerned about optimality while implicit coordination fosters adaptability and robustness (Prehofer and Bettstetter, 2005).

Independent of the required adaptation to the dynamic environment, robots still have to pursue their given individual tasks and goals. General-purpose robots (Simmons, 1994), like service robots or advanced autonomous drones, are more often using decision-making algorithms or even task-level planning, also called Artificial Intelligence (AI) planning, for a goal-oriented mission control (Keller, Eyerich, and Nebel, 2012; Tomic et al., 2012). In this sense, general-purpose robots describe robots that execute their mission autonomously in dynamic environments and which are able to perform several different tasks. This is in contrast to robots that just have a very specific application, like manufacturing robots or simple drones that are only processing preprogrammed trajectories.

In general, adaptivity and robustness for individual systems as well as for the implicit coordination of multiple systems correspond to the concepts and ideas of the research fields of emergence (De Wolf and Holvoet, 2005), self-adaptation (Brun et al., 2009), self-organisation (Wolf and Holvoet, 2004), and swarm robotics (Şahin, 2005). While the particular characteristics of emergence, swarms, self-adaptation, and self-organisation may differ, the general objective of all of these concepts is exceptionally coherent, that is: Having ensembles of robust systems that maintain their structure, in the sense of organisation, and feature a high level of adaptation. The terms emergence, swarm robotics, self-adaptation, and self-organisation will be discussed in more detail later in this document, see Chapter 5.

Implicit coordination of mobile multi-robot systems in a self-organised manner is not only beneficial for simple, insect-inspired robots as in common swarm robotics scenarios like shown by Şahin, 2005 and Dantu et al., 2011. Likewise, more complex general-purpose robot systems, for instance in disaster rescue teams, can benefit as well from

implicit decentralised coordination while operating complex individual tasks. Here, the term complex refers to tasks that require robots with various capabilities, like providing a rich set of alternative actions, multiple goals, dependent task steps, and compound tasks consisting of several sub-operations.

The following illustrative example illustrates the idea. Rescue robots could use a self-organised exploration or search strategy while applying a certain rescue procedure to help individual victims once they are discovered. The rescue procedure consists of several dependent tasks like uncovering the human from snow or blowing out a fire before the human can be examined and transported. In this scenario, the self-organised exploration or search operation allows to increase the robustness of the system to malfunctioning individuals and improve the scalability by enabling a dynamic adjustment of the robot team size without requiring explicit coordination (Masar, 2013). Here, it would also be imaginable that transportation is achieved in a self-organised way again by a group of robots working together. A self-organised transport would allow for adaptation to the size and weight of the load as well as the environment structure (Campo et al., 2006). Alternatively, the transport could be realised with an explicitly coordinated operation. At the same time, the dependent tasks would require planning to combine the necessary primitive tasks in the right order to achieve the rescue procedure goal.

Nevertheless, there is a research gap in between these worlds of individual decision-making and planning for single systems, centralised explicit coordination, decision-making and planning of general-purpose multi-robot systems and self-organised coordination of many simple robots. Current research is either focusing on one or the other direction. In order to address the requirements of creating adaptive and goal-driven multi-robot systems, it is desirable to combine and integrate goal-directed planning, coordination and decision-making approaches with self-adaptation and self-organisation mechanisms.

3. Research Questions

Bringing it all together, there is a demand for adaptive multi-robot systems that operate in uncertain environments. Moreover, current multi-robot systems are either rather simple if they apply implicit coordination algorithms, respectively self-organisation, or the coordination is strictly separated from more advanced decision-making and planning approaches. This thesis addresses the problem of how to develop general-purpose, mobile, multi-robot systems that operate in a dynamic environment in an adaptive and robustly coordinated fashion. This is done by analysing and exploring the combination of so far separated research directions of goal-driven decision-making and planning, multi-robot coordination as well as self-adaptation and self-organisation. In particular, the elaboration is first considering if the integration of self-adaptation capabilities into common decision-making and planning approaches improves the operation of single or multi-robot systems in dynamic environments. In order to address this, it is necessary to establish a decision-making and planning approach that allows the integration of self-adaptation. Secondly, it is researched how a direct combination with self-organisation features can enable the development of more flexible and robust coordination of general-purpose multi-robot systems. To achieve this, it is required to develop an approach that allows to generalise existing application-specific self-organisation algorithms. Likewise, a possible combination of implicit coordination through self-organisation and other decentralised coordination approaches requires further considerations. Moreover, it is the goal to enable an evaluation of the taken approach in practice. For this reason, the focus of this work is on applicable software frameworks, libraries, respectively Application Programming Interfaces.

In detail, this thesis is targeting the following research questions.

- Q1** How does the integration of self-adaptive decision-making and task-level planning allow robotic systems to cope with dynamic environments?

- Q2** In what way can the combination of task-level decision-making and planning with self-organisation improve the robustness of coordination for multi-robot systems?

In order to answer the formulated research questions, it is the goal to develop a software framework that allows to design and implement multi-robot systems, which exhibit self-adaptive task-level decision-making and planning as well as self-organised coordination amongst each other. Particularly, it is intended to enable a coherent and integrated design and implementation of the decision-making and planning as well as coordination application logic within one software ecosystem that features a common domain model and a modular architecture. Such an approach simplifies the development by avoiding system discontinuities and enables a holistic view on the actual implementation. Furthermore, the planned evaluation of the software framework in different application scenarios guarantees the transferability and applicability in practice to enable comparison and exchange with other researchers.

The following chapter describes the structure of the entire document and provides hints on how it can be read depending on the particular interests of the reader.

4. Structure of the Document

This chapter gives some background about the structure of the document at hand, some recommendation about how it can be read, and used conventions.

This thesis comprises six major parts following a typical structure of scientific documents with introduction, analysis, concept, detailed concept and implementation, evaluation, and conclusion. The structure of argumentation and the relationship of individual sections of the document are comprehensively visualised in Figure 4.1. The diagram tries to combine an overview of the document structure, covering the most relevant chapters and sections, with major statements directly extracted from the chapter content. Nevertheless, the statements are sometimes shortened and taken out of context, hence it is recommended to read the entire sections to follow the argumentation. Relationships between certain arguments and sections of the document are colour encoded. The six major parts of the document are depicted in Orange, whereas important chapters are shown in Light Blue. If an argument or section is related to multiple others, a colour gradient is used. In detail, general multi-robot considerations, including explicit coordi-

nation, are coloured Green. Everything related to general-purpose decision-making and planning is depicted in Yellow. Blue-coloured are all items related to self-organisation, while violet colour relates to self-adaptation. These multiple dimensions allow the reader to follow the line of argumentation that is most interesting for him or to look up certain aspects later during chronological reading. Furthermore, the used Identifications (IDs) for certain statements such as objectives or requirements are used throughout the text to help the reader to follow the argumentation and simplify the connection across various chapters. Especially in this aspect, the Figure 4.1 can also be used as a reference to find the chapter in which a statement has been introduced and argued.

The following paragraphs provide an outline of the different parts and highlight essential chapters.

The first *Part I Introduction* motivates the work, describes the problem, and formulates the research questions. If the reader has read chronologically up to this point, this part is already completed.

In order to have a solid foundation for the remaining thesis, the second *Part II Analysis* starts with a differentiation and description of some foundational concepts that are important to understand this thesis before an in-depth analysis of all related research fields is presented. A summary of the deduced insights is given in Chapter 14. This summary is also reflected in Figure 4.1 instead of classifying the individual chapters of that part. If particular details of related work are of interest to the reader, it is recommended to select the relevant chapter/section according to the title from the table of contents.

Part III constructs the high-level concept of the approach developed in this thesis on the foundation of the analysis of the previous Part II and the proposed research questions. This is achieved by deducing the objectives first in Chapter 15 before the methodological approach is outlined in Chapter 16, and a high-level architecture of the approach is presented in Chapter 17. In the next Part IV, we combine the detailed concepts of the particular components with insights about the implementation. This is done due to certain aspects of the concept depend on implementation details of other components. The part is divided into four major chapters. The first Chapter 18 introduces the core components of the approach, while the other three Chapters are constructed on top of it. Here, each of the other chapters is focusing on a different aspect. The Chapter 19 is

4. Structure of the Document

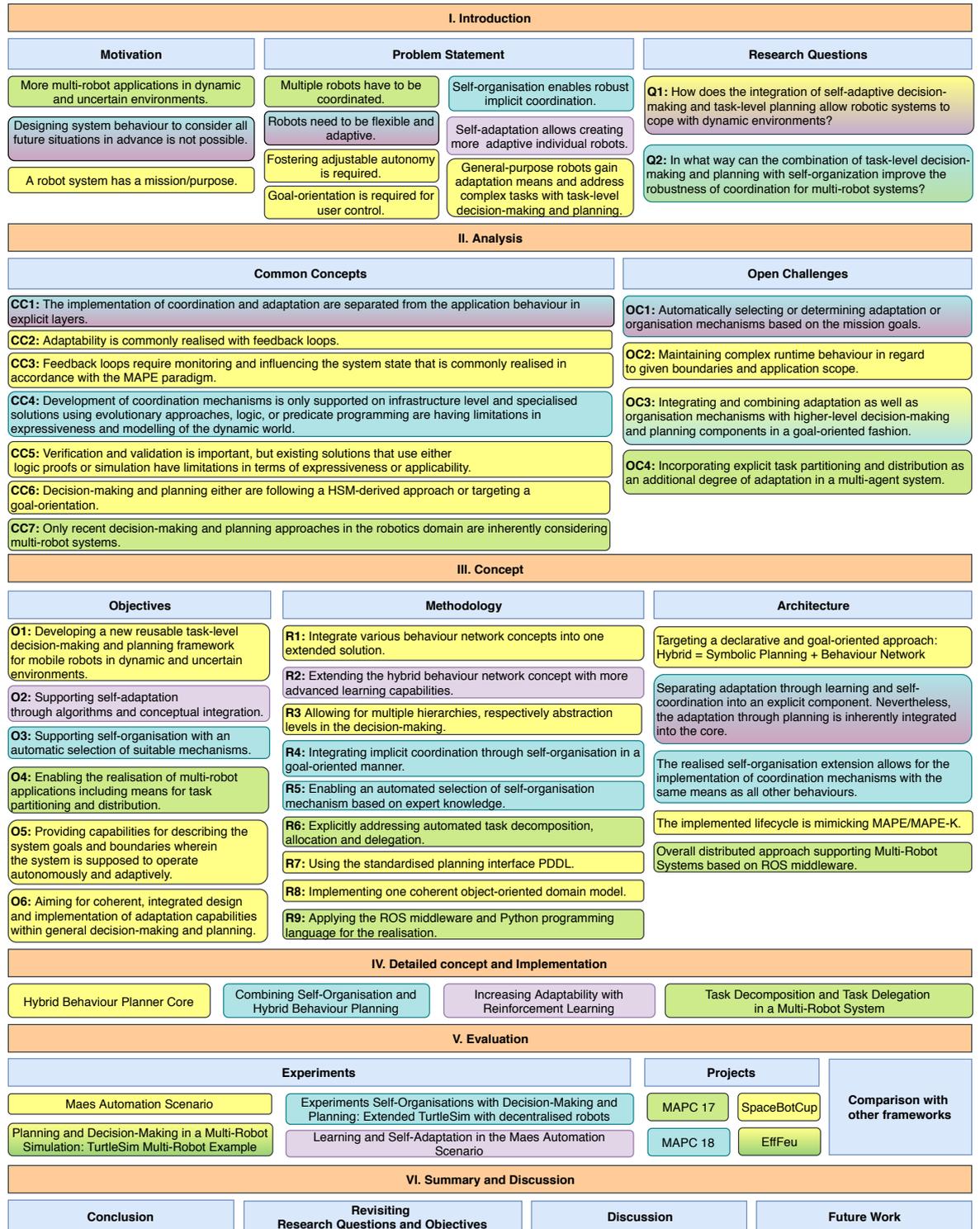


Figure 4.1.: Document at a glance: Argumentation structure and section relationships. Colours and gradients between colours encode relationships.

concerned about the combination of self-organisation, decision-making and planning for implicit coordination; Chapter 20 focuses on the integration of learning capabilities to foster self-adaptation, and Chapter 21 is covering the details of an explicit coordination approach with task decomposition. Subsequently, Part V covers the evaluation of the approach. It is partitioned in conducted experiments in Chapter 22, insights from the application in research projects in Chapter 23, and an overall qualitative comparison of the capabilities against the related work in Chapter 24. Finally, the last Part VI contains a summary of the achievements presented in this dissertation. This part is subdivided into an overall conclusion in Chapter 25, a reconsideration of the initially formulated research questions in Chapter 26, a discussion of limitations in Chapter 27, and an outlook to future research possibilities in Chapter 28.

In order to gain a general understanding of this thesis without going into depth, it is recommended to read at least Part I, the summary of Part II in Chapter 14, the entire Part III, and the final conclusion provided in Part VI. This minimal reading will provide an impression of the developed approach, without details about the related work, the concept, and evaluation. On this foundation, it is still possible to extend the reading depending on the particular interest in one or several of the only touched parts.

Part II.

Analysis

This part aims to provide a foundation for the dissertation goal of developing a concept and implementation for a software framework that combines self-adaptation and self-organisation with explicit task coordination, decision-making and planning in the domain of multi-robot systems. This is done by reviewing related work from various relevant fields. In particular, it is the aim to identify ways, mechanisms, and approaches that can help to gear emergent behaviour of mobile multi-robot systems towards their designated purpose and thus to profit from a goal-oriented autonomous system that features a high level of robustness and adaptivity.

To approach this aim, we survey existing work and compare approaches from fields that are dealing with the questions targeted in this dissertation. Notably, we analyse concepts and approaches from the areas of self-adaptation, self-organisation, emergent systems, and robotics. In doing so, we aim to identify similarities, differences, missing links and open gaps. We put particular emphasis on identifying ways to link emergent low-level behaviour with the global system's perspective and have a closer look at how bottom-up and top-down approaches are related to each other. Finally, we put everything into the relation of multi-robot system applications and the integration with existing task-level decision-making and planning.

The remainder of this chapter is structured as follows: In Chapter 5, we elaborate on foundational concepts to create a common understanding of the terms for the following chapters and parts. In particular, the concepts of agent and multi-agent system, emergence, self-adaptation, self-organisation, and swarm robotics are introduced amongst other related concepts. In doing so, we identify and detail on characteristics and challenges that can be found in each of the presented concepts.

Before we focus on particular approaches that address the common aspects of the presented concepts in a general reusable fashion, we first motivate this by presenting a collection and classification of various scenario-specific algorithms from the robotics domain in Chapter 10. This provides an overview of how such mechanisms can be applied in practice and for which purpose. Here, we explicitly focus on implicitly coordinated multi-robot applications to emphasise the applicability of such approaches.

In Chapter 7, we discuss existing surveys with relationship to the formerly presented concepts that were done in related or similar research fields, in order to emphasise the requirement of an additional in-depth review. In Chapter 8, we have a closer look

into general engineering processes and methodologies that foster the development of self-adaptive, self-organised, or emergent systems. The Chapter 9 presents practical realisations that were done by means of existing middleware frameworks that support the development of any kind of adaptive or self-organising system. In Chapter 11, we forge the link to our particular application domain, namely robotics. In doing so, we focus on how the identified mechanisms can be integrated into planning and decision-making concepts for multi-robot systems by analysing existing approaches.

Before we combine the former findings, Chapter 12 provides additional background and related work about the decomposition, allocation, and delegation of tasks, which can potentially give additional freedom for more adaptive coordination in multi-robot systems. Similarly, Chapter 13 contains background and related work about Reinforcement Learning (RL), which is relevant for realising self-adaptation. These both chapters will become especially important for the *Reinforcement Learner* as well as the *Task Decomposition and Delegation* extensions of the developed framework that are presented later in Chapter 20 and Chapter 21.

Finally, in Chapter 14 of this part we highlight common characteristics of the presented approaches, open challenges, and connect this with the proposed research questions of this dissertation.

This part is based on the journal paper (Hrabia, Lützenberger, and Albayrak, 2018) but the content is completely revised and extended in several directions. In particular, the foundational concepts are extended with Section 5.1 about autonomous agents and multi-agent systems. Furthermore, Chapter 6 extends the analysis part with example applications of self-organisation algorithms and a corresponding classification. Moreover, additional background and related work chapters about task allocation, decomposition, and delegation in Chapter 12 as well as about reinforcement learning in Chapter 13 are integrated into this part, too. Here, Chapter 13 has its foundation in the corresponding parts of the conference paper (Hrabia, Lehmann, and Albayrak, 2019). Additionally, Chapter 11 about planning and decision-making in robotics incorporates additional content from the conference paper (Hrabia, Wypler, and Albayrak, 2017) aside from other literature extensions. Finally, Chapter 14 is completely revised to reflect former mentioned improvements.

5. Foundational Concepts

In this chapter, frequently used terms and their mutual dependencies to and relationships with the targeted problem domain are explained to provide a common foundation for the reader of this work. First, the target domain of this dissertation, robots and multi-robot systems, is connected to the understanding of the related field of autonomous agents and multi-agent systems. Then we distinguish and clarify the often synonymously used terms of emergence, swarm intelligence, swarm robotics, and self-organisation. Furthermore, we interrelate self-organisation and self-adaptation. Finally, we highlight the characteristic properties of the mentioned approaches to underline their relevance for the state-of-the-art analysis that is done in the remainder of this part.

5.1. Autonomous Agents and Multi-Agent Systems

This section gives a clarification of the term agent and multi-agent system, which is important for this dissertation because robot and multi-robot systems are commonly considered as agents, respectively multi-agent systems. Therefore, we will also consider robots as agents throughout this thesis, after we have grounded our understanding, but keeping in mind that one robot can also consist of several software- or hardware-agents. However, the term robot and agent are used synonymously in this dissertation if not explicitly stated differently.

There exists a broad range of definitions for the term agent in the field of computer science, often also referred to more specifically as autonomous (Jennings, Sycara, and Wooldridge, 1998), rational (Wooldridge, 2000) or intelligent agent (Weiss, Braubach, and Giorgini, 1999). In (Franklin and Graesser, 1997) the authors provide a classification of various existing concepts of autonomous agents. On this foundation, Franklin and Grasser developed their own definition (Franklin and Graesser, 1997, p.25): ‘An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.’

A popular definition is presented by Weiss, Braubach, and Giorgini, 1999, p.360: ‘An intelligent agent is a self-contained software/hardware unit that can handle its tasks in

5.1. Autonomous Agents and Multi-Agent Systems

a knowledge-based, flexible, interactive, and autonomous way.’ This definition is also consistent with the understanding of agenthood from Jennings, Sycara, and Wooldridge, 1998.

Both presented definitions cover hardware and software systems and highlight the inner motivation of the agent, named tasks or agenda. The difference is that Weiss et al. do also explicitly mention knowledge as persistence of former experience of the system as well as important properties of being flexible, interactive, and autonomous. Furthermore, Weiss, Braubach, and Giorgini, 1999 state that extended notions of agents also consider situatedness/embeddedness for the close sensory and actuator coupling with the environment similar to Franklin and Grasser. Even though there is not one agreed on definition, most existing definitions are coherent in terms of capabilities that are required for a system to be considered an agent-based system. These capabilities are commonly classified in key and extended attributes.

Following Franklin and Grasser, 1997; Jennings, Sycara, and Wooldridge, 1998; Weiss, Braubach, and Giorgini, 1999 key capabilities can be summarised like below.

Flexibility: An agent handles events in the world either *reactively* in a reasonable time or applies planning and prediction *proactively* to pursue its goals.

Interactivity/communicativity: An agent is able to interact with its environment, especially with other agents with an interface that covers its internal state.

Autonomy: An agent decides on its own which action it takes in a certain situation without consulting any other system entity.

These key capabilities might be extended with the following attributes to narrow the scope of what can be seen as an agent in specific applications.

Situatedness/embeddedness: There is close coupling between the agent and the environment through sensors and actors.

Learning capability/adaptivity: An agent is able to improve itself with respect to its goals over time by learning from experience.

Mobility: An agent is able to transport itself within its environment.

Character: An agent has a ‘personality’ and emotional states.

5. Foundational Concepts

Continuity: An agent is executed continuously, and not just started for a specific operation.

In this work, we follow above understanding of Weiss, Braubach, and Giorgini, 1999, because the explicit notion of software/hardware units intuitively includes robots that are pursuing goals autonomously. Furthermore, the consideration of capabilities, like *perceiving* and *acting* in the environment, as well as *learning* and *adaptivity*, is in accordance with our vision of robots that are able to operate in a dynamic environment. The other extended attributes *mobility* and *character* might be beneficial for agents in certain applications, but are not crucial in the sense of the agent understanding in this thesis. However, *continuity* is intuitively given for all kind of robots as well as *mobility* for all mobile robots, which are relevant for this work, too.

The definition of Weiss, Braubach, and Giorgini, 1999 includes already *interactive* as an important capability, which links to the concept of multi-agent systems. Multi-robot systems are often considered as multi-agent systems, which just extend the idea of a single monolithic agent to a system consisting of several agents. In a multi-agent system, agents are interacting with each other without having full knowledge about the internal states of others (Jennings, Sycara, and Wooldridge, 1998). Such distributed systems are communicating with each other and coordinate their tasks in order to cooperatively achieve their goals. Here, we take only into account such fully-cooperative multi-agent, respectively multi-robot systems, because currently considered applications of multi-robot systems are still limited in the number of robots and these are deployed together for achieving a joint goal. Nevertheless, research is also considering self-interested agents within multi-agent systems, which are only interested in maximising their individual reward (Panait and Luke, 2005).

5.2. Emergence and Swarm Intelligence

The research about emergence has a long history and has diverse scientific roots, for instance in cybernetics, solid state or condensed matter physics, evolutionary biology, artificial intelligence, and artificial life. Its aim is to develop tools, methodologies, and solutions that allow to understand and to reproduce processes that lead to emergence (Serugendo, Gleizes, and Karageorgos, 2006).

The phenomenon *emergence* exhibits a beneficial utilisation of arising global behaviours from distributed agents, which can often be observed in nature. As an example, consider the observable collective intelligence of social insects, like ants and termites, which is used to find the shortest paths, foraging or to build temperature-balancing hives (Bonabeau, Dorigo, and Theraulaz, 1999; Deneubourg et al., 1990b). Other higher developed animals are as well showing similar kind of collective intelligence, a popular example are fish schools, which increase the protection from predators for the individual (Viscido, Parrish, and Grünbaum, 2004). In the mentioned examples, the observable beneficial global behaviour does not rely on individuals that control or coordinate the group.

Emergence can be described as global (macro level) behaviour, patterns and properties that are arising from the interactions between local parts of the system (micro level). This is commonly constraint in the way that the macro level structure needs to be novel. This implies that on the micro level, there is no knowledge about the global or macro level goal, nor there are intentions to control the global behaviour of the entire system. Similar definition can be found in the literature, see Ali, Zimmer, and Elstob (1998), Camazine et al. (2003), De Wolf and Holvoet (2005), and Goldstein (1999).

Moreover, De Wolf and Holvoet (2005) classifies the research into four schools of research, namely complex adaptive systems theory (Kauffman, 1995), Nonlinear dynamical systems theory (Newman, 1996), synergetics (Haken, 1984), and far-from-equilibrium thermodynamics (Nicolis, 1989). A compiled history as well as analysis of different definitions can also be found in Serugendo, Gleizes, and Karageorgos (2006).

Whereas emergence describes the observable phenomenon in general, the often interchangeable used term *swarm intelligence* refers to the process that tries to achieve collective intelligence. Bonabeau, Dorigo, and Theraulaz (1999, p.7) defines swarm intelligence as ‘any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of social insect colonies and other animal societies’.

The fascination of emergence and swarm intelligence derives from the nature of the concept itself. Systems that exhibit emergence can be characterised as *simple*, *robust*, and *adaptive* (Bonabeau, Dorigo, and Theraulaz, 1999; Serugendo, Gleizes, and Karageorgos, 2006). Emergence fosters a simplification of the implementation because the individual entity or agent needs only a straightforward behaviour algorithm in order to play its part

5. Foundational Concepts

in a global system. The latter can be exceptionally complex. Additionally, robustness is increased, since an emergent solution does not depend on a central single point of failure and agents can usually be replaced, removed, or added without changing the global behaviour. Finally, an emergent system is adaptive on the global system level due to the several independent entities that can flexibly adapt and thus easily master changes that occur in an uncertain environment. However, the agent's behaviour algorithm is static on the micro level and not supposed to be adapted.

5.3. Swarm Robotics

The idea of adaptive multi-robot systems that exhibit self-organisation as well as decision-making and planning is related to the research field of *swarm robotics* where the transition in between swarm robotics and multi-robot systems is fluent and not always consistent in the literature. The remainder of this section is clarifying the term of swarm robotics as well as other related terms like *swarm engineering*.

In Chapter 2, we argued that robustness and adaptability are characteristics that are particularly interesting for the domain of (mobile) multi-robot systems—after all, multi-robot systems are supposed to operate in extremely dynamic and highly uncertain environments. Practical examples for challenging application areas are e.g. precision farming, autonomous driving, traffic surveillance, nature conservation, and disaster rescue operations. All of these environments have one thing in common, that is: multi-robot systems have to achieve their goals autonomously, in a timely manner. In doing so, robot systems have to adapt to ever-changing conditions. The concept of emergence can significantly contribute to more effective multi-robot systems, especially in terms of scalability, robustness, and flexibility.

Furthermore, a system of several probably simpler robots can be easier adjusted to varying problem complexities, e.g. by simply adding or removing agents. Multi-robot systems, which implement or adapt the concept of emergent behaviour, are commonly referred to as *swarm robotic systems* (Şahin, 2005). Another definition from Dorigo et al. (2004, p.239) is ‘Swarm robotics consists in the application of swarm intelligence to the control of robotic swarms, emphasising decentralisation of the control, limited communication abilities among robots, use of local information, emergence of global be-

haviour and robustness’. Commonly, swarm robotic systems are classified as multi-robot systems consisting of simple robot individuals with limited decision-making capabilities (Mokarizadeh et al., 2009). A disaster rescue system with several aerial and ground robots capable of executing complex tasks individually is usually considered a multi-robot system, whereas several simple robots with limited individual capabilities, e.g. the robots are only able to move around and execute one primitive task, are understood as swarm robotic system.

The concept of a *swarm* is significantly based on the idea of homogeneity (Şahin, 2005). Nevertheless, concepts of *heterogeneous swarms* can be found in robotics as well. Existing examples commonly employ different specialised entities (Dorigo et al., 2013; Ducatelle, Di Caro, and Gambardella, 2010).

Swarm engineering is a subdiscipline of swarm robotics that deals with ‘the design of predictable, controllable swarms with well-defined global goals and provable minimal conditions’ (Kazadi, 2000). Swarm engineering was first formally defined by Winfield, Harper, and Nembrini (2004) as a combination of swarm intelligence and dependable systems. In particular, this includes procedures for modelling, designing, realising, verifying, validating, operating, and maintaining swarm robotics systems (Brambilla et al., 2013).

The particular domain of controlling robot swarms has gained much attention over the last years. One rather popular sub-branch of this domain is human-robot swarm interaction (Naghsh et al., 2008). *Human-robot swarm interaction* is commonly understood as direct steering of, or influencing the swarm directly, in order to control its motions. This can be done by controlling special leader or predator entities directly, being either virtual or real, which are influencing the remaining swarm members (Bashyal and Venayagamoorthy, 2008; Goodrich et al., 2011). One refers to *assistive swarming* (Penders et al., 2011), whenever the human being is integrated into the swarm as the steering leader or predator. Although these approaches are beneficial for direct interaction during a mission, they are not directly supporting the application’s goals during design and development. Furthermore, these approaches mostly rely on behaviour rules that are specified at design time in a bottom-up approach, see, e.g. Krupke et al. (2015).

Kloetzer and Belta (2006) argue that in contrast to single robot systems, the behaviour of swarm robots has to be specified more qualitatively. Following Kloetzer and Belta

5. Foundational Concepts

(2006), a swarm is naturally described by features like shape, size, and position of the region that is occupied, while the exact position or trajectory of each robot is not important.

Nevertheless, for this dissertation, we follow the understanding of a swarm robot system being an ensemble of simple robot individuals, whereas a multi-robot system consists of potentially more complex multi-purpose robots. Anyhow, we consider the particular characteristics like scalability, robustness, and flexibility of robot swarms as desirable as well for multi-robot systems. For this reason, we will explore how these concepts can be connected throughout this thesis.

5.4. Self-Organisation and Self-Adaptation

Very close to biologically inspired concepts of emergence and swarm intelligence is the idea of self-organisation. The terms emergence and self-organisation are frequently confused. The history of the term, as well as the distinction between both concepts, are comprehensively discussed by De Wolf and Holvoet (2005), who define the concept of self-organisation as follows:

‘Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control’ (De Wolf and Holvoet, 2005, p. 7).

Nevertheless, the term self-organisation is also used as the process that leads to the state of emergence (Goldstein, 1999; Noël and Zambonelli, 2015).

Additionally, Serugendo, Gleizes, and Karageorgos (2005) distinguish between strong self-organising systems that do not have central explicit external or internal control and weak self-organising systems that might have some internal central control instance.

Considering self-organising and emergent systems as distinct concepts, they still have one thing in common, that is: There is no explicit external control whatsoever. Nevertheless, the external influence on self-organised systems is more explicitly researched in the domain of *guided self-organisation* (Prokopenko, 2009). In this context, the external influence is distinguished in specific and non-specific, referring to a specific influence for direct control on the spatial, temporal, or functional structure and to non-specific influence if the system still decides on its own how it reacts upon the external stimulus. In

consequence, Prokopenko (2009) defines the guidance of self-organisation as a possible limitation of the scope or extent of the structures/functions, or specification of the rate of internal dynamics, or selection of a subset of possible options that the dynamics may take. Furthermore, Ay, Der, and Prokopenko (2012) propose external rewards, problem-specific error functions, and assumptions about the symmetries of the desired behaviour as possible strategies for guided self-organisation.

Following De Wolf and Holvoet (2005), the main difference between self-organisation and emergence is that in the case of the former, individual entities can be aware of the system's intended global behaviour. In consequence, self-organisation can be seen as a weak form of emergence. The intuitive and regularly used approach to realise self-organisation is applying the concept of feedback loops. Here, parts of the system are monitoring the state, analysing it in reference to the intended behaviour and triggering appropriate responses. This approach is also used for *single entity systems*. In this case, the concept is referred to as *self-adaptation* (Brun et al., 2009; Lemos et al., 2013). However, if a decentralised system containing several entities exhibits adaptive behaviour to external changes this is as well considered self-adaptation. Self-adaptation in relation to software in general is defined as follows:

'Self-adaptive software modifies its own behaviour in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.'(Oreizy et al., 1999, p. 55)

For this dissertation, the reader should rely on the presented understanding of (De Wolf and Holvoet, 2005) that self-organisation is adaptation of multi-agent systems targeting a system goal that is known to all agents. In the remainder of this work, self-organisation will always be used for the capabilities of multi-agent or multi-robot systems that provide the discussed adaptive characteristics commonly assigned to emergent, robot swarms, and self-organised systems. Moreover, the term self-adaptation is used as adaptation on the single agent level instead of adaptation of agent ensembles in case of self-organisation.

5.5. Common Characteristics of Emergence, Swarm Intelligence, Self-Adaptation, and Self-Organisation

While the particular characteristics of emergence, swarm intelligence, self-adaptation, and self-organisation may differ, the general objective of all of these concepts is exceptionally coherent, that is: Having ensembles of robust (multi-)agent systems that maintain their structure and feature a high level of adaptation. Here, the structure can either be understood in the organisational sense in case of multiple systems or as the integrity of a single agent.

A system, which enjoys these characteristics, e.g. appropriate adaptation capabilities that remain within predefined bounds (also applied in the context of *adjustable autonomy* (Sierhuis et al., 2003)), would lead to a simplified design because not all possible states and state transitions have to be specified in advance. In fact, these ideas are in compliance with the visions of the promoters of autonomic computing (Kephart and Chess, 2003) and organic computing (Schmeck, 2005), respectively. Both concepts consider systems that promote self-adaptation as major property, facilitating the development of software that can manage itself at runtime. Here, autonomic computing concentrates on network and infrastructure management, whereas organic computing has a broader focus on distributed and ubiquitous systems in general. Furthermore, self-adaptation is considered to be the foundation of other envisioned self-* properties, such as self-configuration, self-optimisation, self-healing, and self-protection (Kephart and Chess, 2003).

Continuing this thought, it would no longer be necessary to exactly specify the low-level system behaviour in all possible situations that might occur, but rather, leaving the system with a certain degree of freedom to allow for autonomous reaction and adaptation to new situations in an intelligent way.

Still, not answered are the questions, how such systems can be developed from a goal-driven perspective, especially how this can be integrated into general-purpose robots that exhibit a high level of autonomy with advanced decision-making and planning capabilities. These rather general questions are the foundation for the research questions that have been already formulated in Chapter 3. Particularly, we want to investigate how a goal-driven approach based on decision-making and task-level planning can be

made more adaptive and how far this supports operation in dynamic environments. Furthermore, we research if the combination of the above-described approach with self-organisation is able to improve the robustness of coordination in multi-robot systems.

Before we review work from all above-mentioned fields in more detail to create a foundation for the later on developed approach, we illustrate possible applications, especially for multi-robot systems, in the following chapter.

6. Self-Organisation Mechanisms and their Application

In the motivation of this dissertation, several advantages of multi-robot systems are listed, for instance, parallelism, the combination of forces, distributed detection and actions, robustness, scale-up capabilities, exploitation of heterogeneous properties and overall flexibility.

These advantages are especially valid if a multi-robot system is controlled and organised in a distributed manner through the application of self-organisation algorithms. Such self-organisation algorithms have been applied in many multi-agent domains, like combinatorial optimisation, communication networks, and robotics (Bonabeau, Dorigo, and Theraulaz, 1999). The algorithms, respectively mechanisms, are used for various purposes, like motion control, information sharing, and decision-making. In the following, common mechanisms and patterns used for implicit coordination are described and referenced with their purpose and particular application in robotics to illustrate the wide range of possible applications. The mechanisms are classified according to the patterns identified by (Fernandez-Marquez et al., 2012), whose work will later also be discussed in more detail in Chapter 8.

All patterns described in the following are targeting multi-agent systems and allow for decentralised and robust solutions, especially for large-scale systems. In consequence, the particular implementations rely on local interactions and reasoning. The following brief descriptions refer to the problems addressed by the patterns and provided solution concepts.

6. *Self-Organisation Mechanisms and their Application*

Spreading Information diffusion to share local knowledge amongst the agents to provide and estimate a broader view of the global system state in a distributed manner.

Aggregation Fusion of information shared amongst the agents to obtain meaningful information and reduce the amount of information available to avoid an overload of communication and individual systems.

Evaporation Reduction and elimination of available information over time to model temporal problems and enable adaptation based on the most recent information available.

Repulsion Applying artificial repulsion forces for collision avoidance and coordination of uniform distributions.

Gradients Combined aggregation and spreading of information extended with additional data about the distance and direction of the sender to increase the global knowledge about the system for an individual agent.

Digital Pheromone Gradients are placed physically or virtually in the world in the inspiration of natural pheromones enabling indirect communication, also called stigmergy communication. The information often evaporates over time to enable adaptation to dynamic changes.

Gossip Decentralised decision-making to reach an agreement progressively in a group of agents by applying spreading and aggregation.

Ant Foraging Collaborative search and exploration to find the shortest paths for the exploitation of a resource.

Chemotaxis Gradient-based motion coordination considering sources and boundaries of events. Agents locally sense gradients, respectively potential fields and react according to this information.

Morphogenesis Using spatial positions of agents in the system to determine particular behaviour.

Quorum Sensing Gradient-density-based, decentralised, and collective decision-making applying thresholds in a local neighbourhood.

Flocking Extension of repulsion, also applying alignment and velocity for dynamic motion coordination and pattern formation.

An overview of mechanisms applications and literature references is given in Table 6.1. The classification follows above described common patterns.

Table 6.1.: Classification of self-organisation mechanisms to patterns and applications in multi-robot systems.

Pattern	Application
<i>Spreading</i>	<ul style="list-style-type: none"> • Formation & role assignment (Iocchi et al., 2003) • Navigation (Ducatelle et al., 2009a) • Distributed coordination (Kudelski et al., 2012) • Task assignment (Kudelski et al., 2012)
<i>Aggregation</i>	<ul style="list-style-type: none"> • Object position estimation (Stroupe, Martin, and Balch, 2001) • Cooperative robot localisation (Song, Tsai, and Huang, 2008) • Mobile robot navigation (Kam, Zhu, and Kalata, 1997)
<i>Evaporation</i>	<ul style="list-style-type: none"> • Patrolling (Portugal and Rocha, 2011) • Collaborative search (Fernandez-Marquez, Serugendo, and Montagna, 2011)
<i>Repulsion</i>	<ul style="list-style-type: none"> • Pattern formation (Cheng, Cheng, and Nagpal, 2005; Varga et al., 2015) • Collision avoidance (Fernandez-Marquez et al., 2012)
<i>Gradients</i>	<ul style="list-style-type: none"> • Robot navigation (Parunak, Purcell, and O’Connell, 2002) • Guidance (Yang et al., 2015a)
<i>Digital Pheromone</i>	<ul style="list-style-type: none"> • Path planning (Parunak, Purcell, and O’Connell, 2002) • Coverage (Ranjbar-Sahraei, Weiss, and Nakisae, 2012) • Patrolling (Chu et al., 2007) • Collective exploration (Brambilla et al., 2013)
<i>Gossip</i>	<ul style="list-style-type: none"> • Task allocation (Ducatelle et al., 2009b) • (Distributed) consensus (Habibi et al., 2015) • Routing (Franceschelli et al., 2013)
<i>Ant Foraging</i>	<ul style="list-style-type: none"> • Foraging (Kumar and Sahin, 2003) • Path planning (Reshamwala and Vinchurkar, 2013) • Target navigation (Behera and Sasidharan, 2011)
<i>Chemotaxis</i>	<ul style="list-style-type: none"> • Pattern formation (Barnes, Fields, and Valavanis, 2009) • Path planning (Semertzidou et al., 2016) • Target search (Yang et al., 2015a)
<i>Morphogenesis</i>	<ul style="list-style-type: none"> • Shape formation (Mamei, Vasirani, and Zambonelli, 2004) • Motion selection (Nagasaka et al., 2001) • Motion Control (Song and Kumar, 2002)
<i>Quorum Sensing</i>	<ul style="list-style-type: none"> • Distributed consensus (Brambilla et al., 2013) • Synchronization (Bechon and Slotine, 2012) • Control, e.g. population (Zhao et al., 2015)
<i>Flocking</i>	<ul style="list-style-type: none"> • Formation control (Lei and Li, 2008) • Area exploration (Kumar et al., 2012)

The described patterns and related mechanism application references indicate that there are many relationships and dependencies between the algorithms. This was already

discovered by Fernandez-Marquez et al., 2012 and is further discussed in Chapter 8. Furthermore, these different application examples of such specific algorithms lead to the question of how this can be reused and applied for multi-robot systems in general.

In the following chapters, available works are presented and analysed that aim to develop specific self-adaptation and self-organisation mechanisms, provide methodologies and engineering techniques for the development process of such systems or present complete frameworks addressing specific problems in the related fields. In doing so, we discuss the limitations of these approaches but also elaborate on their advantages and disadvantages. This is done to identify problems, opportunities, and challenges that can build a foundation for the approach developed in this dissertation.

Moreover, in Part III and IV the above-presented classification and relationships of self-organisation mechanisms are revisited and used as a foundation for the integration into the decision-making and planning approach developed in this dissertation.

7. Related Surveys about Emergence, Self-Organisation and Self-Adaptation

The field of emergence, self-organisation, and self-adaptation has already been surveyed from different perspectives. In the following, the existing literature is summarised.

The survey from Lemos et al. (2013) presents state-of-the-art methods for engineering self-adaptive systems. The authors distinguish between design space, processes, centralised and decentralised control, and practical runtime verification and validation. Lemos et al. understand *design space* as which developers need to decide during the development of a self-adaptive system and see the main challenge of creating a systematic understanding of the options for self-adaptive control during the design process. The concept of *Processes* refers to influences of self-adaptive systems to the software system life-cycle. Following Lemos et al., activities are not bound to the traditional development-time but are shifted to runtime. In the area of control architectures, different possibilities with advantages and disadvantages are presented. It is emphasised that adaptation control is always realised with feedback loops, which comprises Monitoring,

Analysis, Planning and Execution (MAPE) in different combinations. Planning is albeit only considered regarding the control of the adaptation process and not with respect to the goal-oriented behaviour of the system.

The formulated question in this context is under what circumstances and for what systems the different patterns or architectures of control are applicable. For verification and validation, the open challenge is to design and deploy certifiable methods that are able to deal with the explosion of the state-space. Since they are focusing on very general aspects of engineering, their perspective is more on the abstract process of developing self-adaptive systems and less on how such systems can be realised in practice, e.g. in the context of multi-robot systems.

Another survey paper focuses on the importance of feedback-loops in self-adaptive systems (Brun et al., 2009). Brun et al. (2009) present several existing approaches from different fields like control theory, nature, and software—like the autonomic computing initiative. A generic feedback loop is proposed as well. This feedback loop acts as a refinement of the sense-plan-act mechanism that is commonly used in multi-agent systems and highlights that software engineering needs to develop an own understanding of feedback loops. Moreover, they introduce some specific challenges regarding the concepts of modelling, maintenance, middleware support and verification, and validation. In particular, Brun et al. (2009) substantiate the need to create reference libraries and architectures of control-loop types and mechanisms of control-loop interactions as well as to create appropriate middleware solutions with possibilities of validation and verification. We generally agree with this argumentation and see similar requirements for self-organising multi-robot systems, e.g. providing middleware support, respectively framework support, for fostering code reuse, which we are addressing in this work. Existing frameworks and approaches are discussed in Chapter 9 and Chapter 11.

The survey from Serugendo, Gleizes, and Karageorgos (2006) is about self-organisation and emergence and the differences and relations of these concepts and the arising challenges. The paper is focused on mechanisms and definitions rather than on particular engineering techniques or frameworks.

7. *Related Surveys about Emergence, Self-Organisation and Self-Adaptation*

Focusing on multi-agent systems, the authors classified existing mechanisms of realising self-organisation into five classes:

- Using direct interactions between agents using basic principles such as broadcast and localisation,
- using indirect interactions between agents using the environment (stigmergy),
- using reinforcement of agent behaviours,
- using cooperation behaviour of individual agents,
- using a generic architecture.

Furthermore, Serugendo, Gleizes, and Karageorgos (2006) formulate the central question, how individual agents can be programmed to exhibit self-organised behaviour as a whole. For this reason, their highlighted major challenge is developing means to define global goals and to design local behaviours so that the global behaviour emerges.

The field of swarm engineering is surveyed by Brambilla et al., 2013. The authors review on the one hand side methods for designing and analysis and on the other hand, collective behaviours that have been studied using swarm robots. It is argued that the most studied fields are design and realisation as well as verification and validation, which are still not sufficiently solved. Moreover, Brambilla et al. (2013) claim that other topics, such as requirements analysis, maintenance, and performance measurement, have not yet gained much attention. The authors are also coming to the conclusion that there is still no formal or precise method to design individual-level behaviours that produce the desired collective behaviour. The existing automatic design methods are further classified into the categories of evolutionary robotics and multi-robot reinforcement learning. However, the authors do not link to the problem of developing systems that are pursuing their goals and exhibiting self-organised behaviour at the same time.

The referenced surveys analysed various important aspects such as engineering of self-adaptive systems, feedback-loops for control of self-adaptive systems, classification of self-organisation mechanisms, and methods for designing and analysing of swarms. Nevertheless, the questions remain open how this can be realised in practice and combined with existing means of control such as decision-making and planning. The next

chapter continues the foundation of the existing surveys. In contrast to the recited surveys, this chapter is starting from a common perspective, revisiting self-adaptation and self-organisation architectures and methodologies in the different related fields. It also focuses on existing frameworks and middlewares that have been proposed in the related work. Furthermore, we go into detail on how the actual adaptive self-organising mechanism is going to be realised and how it could be integrated into the decision-making and planning capabilities of intelligent general-purpose robotic systems.

8. Self-Adaptation and Self-Organisation Software Development Methodologies

In addition to the presented results of the survey from Lemos et al. (2013) about software engineering and self-adaptation in general, other authors have developed different concepts.

The software engineering process PosoMAS is introduced and compared to other existing Agent-oriented Software Engineering (AOSE) methodologies in Steghöfer et al. (2014). The process practices are embedded into the risk-value life cycle of the Open Unified Process (OpenUP). It is targeting open, self-organising systems and it contains different practices for the design of agent architecture, agent organisation, agent interaction, system architecture. The core practices are goal-driven requirements determination, pattern-driven MAS design, using existing architectural, behavioural and interaction patterns from reference architectures, an iterative, evolutionary agent design, model-driven observer synthesis, trust-based interaction design and agent organisation design. The *organisation design*, as well as the lifecycle phase *design system dynamics*, are the aspects where self-organisation and its consequences have to be considered by the user of the PosoMAS process. Unfortunately, there are no details about how this has to be done in (Steghöfer et al., 2014). The process only provides placeholders for the consideration of self-organisation during system design.

In contrast to PosoMAS, more support is provided by the earlier and continuously

8. Self-Adaptation and Self-Organisation Software Development Methodologies

developed agent-oriented methodology ADELFE that provides a model-driven software development process for developing adaptive cooperative multi-agent systems (Bonjean et al., 2014; Picard and Gleizes, 2004). ADELFE is supported by a special notation named Agent Unified Modelling Language (AUML), which is based on Unified Modelling Language (UML), as well as some tools and libraries. Moreover, the process itself is separated into the stages of preliminary requirements, final requirements, analysis, design, implementation, and tests.

The authors propose that global behaviour is determined with testing and simulation but the methodology is not bound to this approach. ADELFE has still a very coarse view on the problem of developing self-adaptive and self-organising systems, since the actual complexity of how to create the required behaviours, mechanisms, and algorithms is not addressed, even though the process provides more guidance with tool and library support than the PosoMAS approach.

Ideas from ADELFE are also integrated into the TROPOS4AS extension (Morandini et al., 2009) of the agent-oriented software engineering methodology TROPOS (Bresciani et al., 2004). TROPOS focuses on requirement analysis and models the system top-down in several analysis steps. The entire design process is supported by the TROPOS modelling language. Of particular interest are goals and how they are decomposed and delegated to the individual agents. TROPOS4AS extends the TROPOS goal models to allow for the description of self-adaptive systems. This is achieved by additional annotations that allow to express environment specific conditions, goal creation, and failure handling. TROPOS4AS incorporates ideas from ADELFE to model agent organisations with cooperation rules.

Gershenson (2007) presents a self-organisation engineering methodology introducing the *mediator* as a central concept. The mediator is understood as an adaptation controller that is reducing the friction between parts of the system in order to fulfil the overall system goals. Reducing friction means increasing positive synergies and reducing inhibitions between the system parts. The mediator or adaptation controller is supposed to adjust the behaviours of parts leading to the global goal. The controller could be using methods like evolutionary algorithms or reinforcement learning. Furthermore, the author proposes a process consisting of the stages representation, modelling, simulation, application, and evaluation with iterative cycles between adjacent stages. The author

states that this approach is neither top-down nor bottom-up. Based on the additional degrees of freedom of self-organising systems, the author declares that they cannot be tightly controlled since they have their own goals; instead, they should be steered. In contrast to the above-presented approaches, the methodology of Gershenson is providing guidance for a suitable architecture of self-organising systems. However, the support is still on a high level of abstraction, and it remains unclear how the user could determine or select appropriate mechanisms, mediators, or controllers.

Another high-level architecture is presented in the context of the Organic Computing initiative Branke et al. (2006). The proposed abstract observer-controller architecture is created for the realisation of systems with self-* properties. The observer collects data from the system and computes some indicators characterising the global state and the dynamics of the system. It may use methods like classification in order to characterise the situation. The controller compares situation parameters with the goal, which was defined by the user, and decides whether an intervention is required and what action would be most appropriate. Additionally, the controller is able to influence the local decision rules of the system agents, the system structure, the number of agents, or the environment. Moreover, the controller interferes only when necessary and affects only some system parameters. The controller does not control single agents in detail. Decisions (mapping from situation to action) are based on learning. Learning can be facilitated by simulation. The presented architecture is only defining a centralised feedback loop for the system and abstracts from the intended behaviour but how to define the behaviour is still an open issue. Moreover, the proposed learning approach requires the occurrence of errors. These can be problematic in cases where errors would affect the operability of a system.

In Noël and Zambonelli (2015), the authors propose an abstract approach based on component decomposition for engineering application-specific self-organising systems. The authors argue that the identification of elements and the assignment to roles build a design bridge between the problem and the emergent behaviour, because the selected decomposition limits possible macro-level behaviours that can be executed during run-time.

Contrary to the surveys we presented in Chapter 7, which have highlighted the importance of formal methods with validation and verification and contrary to the software engineering approaches that we discussed above, Edmonds (2005) argues that in order

8. *Self-Adaptation and Self-Organisation Software Development Methodologies*

to create adaptive, self-organising systems, experimental methods—as used in classic science—are required. The reason for this is that a traditional engineering approach supposedly cannot deal with unexpected changes during future runtime. In fact, the author argues that there is no general systematic or effective method that can generate or find a program to meet a given formal specification and determine whether it meets that specification. Instead, Edmonds argues that systems should be accompanied by an open hypothesis database, which collects hypotheses that already have been tested—including all relevant test-conditions. Albeit, it is not mentioned how such a system could be realised in concrete applications.

De Wolf and Holvoet (2007) focus more on the mechanisms themselves, describe the properties of some core self-organisation mechanisms as design patterns, and provide a guide when they can be applied and how they should be selected to reach a targeted goal. In particular, they mention digital pheromones, gradient fields, market-based control, tags, and tokens. However, only gradient fields and market-based control are discussed in more detail. Although the paper is a helpful guide for developers of self-organising systems, it leaves open how the actual implementation or class design could be realised.

In (Fernandez-Marquez et al., 2012), the work of De Wolf and Holvoet (2007) is continued by analysing, classifying, and describing a set of bio-inspired self-organising mechanisms. Here, the authors classified mechanisms and their relations into three layers of basic, composed and higher-level patterns. In that sense, the composed layer is created by a combination of basic mechanisms and the higher-level patterns show different options of exploiting the basic and composed mechanisms. On the basic level, they identified the basic patterns of spreading, aggregation, evaporation, and repulsion that build the foundation for a realisation of all composed and higher-level mechanisms. The created catalogue of patterns is intended to be used as a base for a more modular design and implementation of self-organising systems.

The idea of reusable design patterns is also incorporated by Reina, Dorigo, and Trianni (2014), who first introduced cognitive design patterns. However, cognitive design patterns are not focused on self-organisation or self-adaptation. Instead, the concept considers mechanisms for collective cognition in distributed multi-agent system in general. In (Reina, Dorigo, and Trianni, 2014), the authors present an in-depth study of a collective decision-making mechanism in order to use it as a cognitive design pattern

independent of its application in the future.

The review of existing methodologies shows that many approaches tackle the problem of designing self-adaptive or self-organising systems on a very coarse process and architecture level and only identify the particular points where the designer would have to integrate related considerations. Nevertheless, this is providing some support for system designers and developers on the initial stage, this seems not sufficient without considering how the actual problems are addressed on an implementation level, which we foresee as the bigger challenge. On the other hand, some related work provides valuable foundations that will be reconsidered later in the concept of this dissertation. In particular, De Wolf and Holvoet (2007), Edmonds (2005), and Fernandez-Marquez et al. (2012) are relevant for this work, which are more specific regarding the actual challenges of guiding the development of such systems in practice. Additionally, we will also look again on the ideas of Edmonds (2005), who supports a general paradigm shift that focuses on experimental methods instead of concentrating on provable correctness. Especially, De Wolf and Holvoet (2007) and Fernandez-Marquez et al. (2012) provide reusable design patterns that can be applied to new applications, which are foundational for the later presented integration of self-organisation into the developed decision-making and planning framework.

The support of an actual implementation in a more specific and application-oriented perspective is reviewed in the next chapter, where we discuss specific middleware solutions and frameworks.

9. Middleware Solutions and Frameworks

Several middleware solutions, frameworks, and infrastructures for the development of self-adaptive and self-organising systems with different perspectives have been proposed. All have in mind to foster and support the development of adaptive systems that are able to deal with uncertainty for either single systems or multiple system entities. The specialities and similarities are discussed in the following.

Hernandez-Sosa et al. (2005) present an adaptation framework inside their component-

9. *Middleware Solutions and Frameworks*

based robotic framework with the focus on resources used by modules. Thus, their focus is purely adaptive Quality of Service (QoS), in order to keep the system robust and stable, even in high load situations, and not the general behaviour of the system under normal execution conditions.

Another self-adaptation framework with a focus on resource usage is ReFrESH, which targets all kinds of embedded systems. This includes hardware and software (like robotic system) and is realised with an embedded virtual machine running on Real Time Operating System (RTOS) (Cui et al., 2014). Different from Hernandez-Sosa et al. (2005) the focus is especially on self-adaptation mechanisms for fault detection and not QoS. Particularly, the goal of the framework is an automatic reconfiguration of the system with other or additional components. The evaluation of the current state is tailored to the detection of abnormal resource usage. Nevertheless, the adaptation does not include learning and it is strongly focused on the described problem domain.

Rainbow is a more general architecture framework for self-adaptive systems (Garlan et al., 2004) that is not limited to resource usage adaptation like former approaches. It focuses on single systems and does not consider multi-agent system environments. The core idea is the separation of the adaptation layer in order to foster reuse of adaptation mechanisms. Furthermore, their approach requires a definition of the adaptation mechanisms during design time based on preprogrammed adaptation operators and strategies. This seems feasible for very simple adaptations like parameter tuning but does not satisfy the vision of systems adapting autonomously in complex versatile environments.

In the dissertation of Krupitzer, 2018 the author presents a software engineering workflow that is realised in a prototype Integrated Development Environment (IDE) for developing self-adaptive systems. The focus of the presented Framework for Engineering Self-Adaptive Systems (FESAS) is to increase the reuse of created artefacts for improving the development speed and decreasing the error rate. Particularly, the author introduces self-improvement through meta-adaptation, which is understood as a high-level template for adaptation. The adaptation itself is realised through the exchange of MAPE components, their interaction patterns, or parameter adjustments. A core requirement of the approach is the separation of concerns to increase reusability. Here, available components are held available in the repository of the FESAS middleware. Due to its focus on software engineering aspects, the overall approach stays on an abstract software com-

ponent level that provides guidance for the structure of a self-adaptive system as well as skeletons that would have to be filled for an application and adaptation mechanism implementation. Nevertheless, different scenario specific implementations are presented that are using rule engines or specific planners. The work of Krupitzer, 2018 is situated somewhere in between methodology focused approaches that have been discussed in the previous section and more concrete frameworks analysed in this section. We placed the reference here because of the realised prototype middleware implementation.

Differing from the resource-focused frameworks and rather inspired by Rainbow and SodekoVS is a more general architecture framework, supporting software engineering of self-organising systems (Sudeikat et al., 2009). The SodekoVS library provides a catalogue of mechanism patterns as reusable components that provide systematic problem-oriented descriptions of mechanisms in an abstract, reusable format. This facilitates the selection and combination of mechanisms. Their multi-layer architecture consists of a topmost application layer including standard application functionalities, which is potentially linked to agent implementations for application-specific parts. The coordination layer placed below consists of the agents as well as a substrate, which can contain one or more coordination media. Mechanism instances are encapsulated in distinct coordination media and are interfaced by coordination components that allow to modify agent states. However, the presented architecture does not mention how decision-making and planning could be combined with self-organisation mechanisms.

The application-independent description of (inter-)agent coordination patterns is supported by the domain specific language MASDynamics that allows to map interrelations of agent activity to detailed agent design models (Sudeikat and Renz, 2009). However, the language describes the intended macroscopic behaviour only on a very abstract level. Particularly, the macroscopic behaviour is just defined by group count, roles and group membership count. The coordination pattern can be parametrised and configured based on agent events and internal state representations that can be passed through the coordination media. Nevertheless, it is still required to determine the actual mechanism manually. Furthermore, the authors propose a process for self-organisation engineering with the stages of requirements, analysis, design, implementation and test (using simulation), similar to the approach of Gershenson (2007) (see Chapter 8), but without explicit iterations. The focus of the framework lies on the engineering structure with-

9. Middleware Solutions and Frameworks

out considering the development of coordination mechanisms themselves nor the explicit integration with other higher-level decision-making and planning capabilities.

In analogy to the Rainbow and SodekoVS frameworks Preisler, Vilenica, and Renz (2013) developed a framework that separates the coordination of self-organising MAS from the application logic. Similar to SodekoVS, the so-called coordination space represents a separate and explicit layer in the architecture. Their goal was improving the reuse and exchange of coordination mechanisms. The coordination itself can be distributed over several nodes by using information forwarding with mechanisms like publish and subscribe. Further, the coordination is written in an external coordination model also using the domain specific language MASDynamics (Sudeikat and Renz, 2009) with mechanisms described in a declarative fashion. This language allows for a more structured description of coordination using concepts of roles and links between agents. Nevertheless, the presented framework does not provide a mechanism for deducing or selecting the mechanism itself from given goals. Additionally, the authors are not linking the coordination components with the agent's decision-making and planning, like Sudeikat et al. (2009).

jSwarm is a middleware for cyber-physical systems in general with an explicit focus on swarm robotics (Graff, Richling, and Werner, 2013, 2014). It allows for centralised sequential programming with spatial-temporal constraints with a concentration on the motion in the space of robot systems. The application code is analysed, scheduled, and distributed according to a centrally computed dependency graph amongst the system entities. This enables the replacement of individual robot motion by an application migration between different robots for increasing the system performance, while considering spatial-temporal constraints. The used service-oriented architecture hides heterogeneity and diversity and abstracts from particular resources. How the specific swarm behaviour based on the constraints for individual entities is designed is not further considered. In fact, jSwarm provides an abstract infrastructure for a centralised controlled distributed swarm robot system. Even though it fosters adaptability by reassigning robot applications to other robots, the focus is more on performance optimisation and less on enabling more robust and adaptable distributed systems.

The framework Tuples on the Air (TOTA) mainly focuses on the aspect of information propagation in a decentralised self-organising system (Mamei, Vasirani, and Zambonelli,

2005). Their approach from the field of sensor networks is inspired by biological morphogen gradients (cells react based on thresholds that change while propagating) and does not need global perception, distance, and direction sensing. TOTA provides abstractions and mechanism to support the creation of distributed overlay data structures spread across a mobile network. It is composed of a dynamic ad-hoc wireless network of possibly mobile nodes, each capable of locally storing tuples of information and letting them diffuse through the network. The consideration of how influences propagate is essential for large-scale real-life applications and not considered by most of the other frameworks.

Another more application-specific work comes from the field of modular robotics, describing robots consisting of several independent modules. The paper from Yu and Nagpal, 2009 shows how distributed constraints can be used to build adaptive modular robots. Their framework abstracts from simple sensors and actuators and uses distributed constraints combined with neighbourhood sensing for distributed adaptation. Open questions are how the required constraints or control laws (sensor feedback functions) can be developed to achieve the entire system goal, how several goals are going to be addressed in parallel, and how these distributed constraints can be combined with the robots' decision-making and planning.

Tuple Centres Spread over the Network (TuCSoN) is a platform for the development of self-organising systems (Viroli, Casadei, and Omicini, 2009). It is based on the linda tuple-space model and provides Java agents with tuple centres (tuple spaces enhanced with a reactive, rule-based programming model), which can be used to implement probabilistic and timed coordination rules in a declarative style. It is designed around the formulated requirements for self-organised coordination. In particular, *topology* describes that each agent is directly connected to a small set of neighbourhood locations. *Locality* defines the local interaction between agents and the coordination media and between two coordination media. The requirement *On-line character* says that coordination is triggered by interaction or by elapsed time. *Time* ensures that coordination rules generally timed. The last requirement *probability* defines that coordination is always non-deterministic.

The rules in TuCSoN are represented with templates as identifiers for the matching mechanisms. In case of several matching templates, the selection is randomised. The re-

9. *Middleware Solutions and Frameworks*

activity is realised with reaction tuples that are triggered through formulated conditions of source, target, status, time, and goals.

Altogether, TuCSoN has a much stronger focus on the coordination mechanisms themselves and allows for the implementation of data-centred self-organisation coordination algorithms. Nevertheless, it does not include support for goal-driven development. In consequence, the developer still needs to figure out on its own, how to move, transform or copy data in order to achieve a certain pattern. Moreover, the authors do not combine their approach with other application-specific decision-making and planning.

A subset of the described patterns in (Fernandez-Marquez et al., 2012), see Chapter 8, is implemented in the execution model BIO-CORE (Fernandez-Marquez, Serugendo, and Montagna, 2011), which provides basic bio-inspired services, namely the basic patterns evaporation, aggregation and spreading as well as the gradient pattern. BIO-CORE consists of three main parts: a shared data space that allows to exchange data, basic bio-inspired services implementing basic bio-inspired mechanisms, and interfaces providing primitives for the agents to interact with the core.

Buzz is a domain specific language (DSL) that provides a middleware for the simplified implementation of swarm robot applications (Pinciroli and Beltrame, 2016). In Buzz, a set of robots (swarm) is a first level object that can be used for group task assignment and set operations (intersection, union, difference, and negation). Furthermore, Buzz has capabilities for neighbourhood operations (queries, filtering, and virtual stigmergy) and information sharing. The language runs on an own virtual machine. Nevertheless, the realised mixed bottom-up and top-down approach does only provide an environment for developers wherein self-organisation and cognitive algorithms can be implemented. A disadvantage of Buzz is that developers need to leave their well-known ecosystem with tools and programming languages for learning a new language that is limited in expressiveness.

The work presented in this chapter shows various directions, the approaches are either very specific to a particular domain or problem, like resource sharing or information propagation, or are general frameworks that simplify the implementation of abstract concepts, like presented in Chapter 8. The important problem of implementing the actual mechanisms and algorithms in general is only addressed in TuCSoN (Viroli, Casadei, and Omicini, 2009) and (Fernandez-Marquez, Serugendo, and Montagna, 2011). However, it

is still open how a developer could solve a particular problem from a goal-oriented perspective, and how the coordination and adaptation part can be linked with the agents' decision-making and planning. In the next chapter, we go into detail and analyse works that explicitly focus on the support of designing and implementing the required mechanisms for adaptive systems.

10. Mechanism Design and Implementation

We already highlighted that, besides the architecture and the engineering process, the mechanism itself plays a key role for self-organisation or self-adaptation. The challenge of implementing these mechanisms is also raised in several above-mentioned references (Kephart and Chess, 2003; Schmeck, 2005; Serugendo, Gleizes, and Karageorgos, 2006). Facilitating the development of these mechanisms is a crucial requirement for a suitable framework. Thus, in this section, we focus on existing means of development.

The power and simplicity of coordination mechanisms are proven in nature, and existing phenomena are extensively studied in the field of swarm intelligence (Bonabeau, Dorigo, and Theraulaz, 1999). Using similar approaches led to a number of promising results, especially for swarm robotics. For instance, Masar (2013) created an algorithm for swarm exploration and surveillance based on a combination of a swarm particle algorithm, using the popular force field paradigm with ant colony algorithms. Other concepts are based on behaviour descriptions (Balch and Arkin, 1998), rigid virtual structures (Ren and Beard, 2004) and graphs (Desai, Ostrowski, and Kumar, 2001). In comparison to the numerous force field approaches, the latter methods are less flexible and are computationally more expensive.

Unfortunately, all these approaches have the disadvantage that they are inherently problem and application-specific and that they were created in a bottom-up manner by means of trial and error testing. None of the approaches was inferred or selected from a goal description, respectively using top-down engineering.

We classified existing work that deals with design and implementation of self-organisation and self-adaptation mechanisms into the categories of evolution, learning, and simula-

10. Mechanism Design and Implementation

tion approaches, functional languages, declarative and application-specific approaches. These classes are discussed in the following sections and completed with a consideration of the mechanisms analysis, verification, and validation.

10.1. Evolution, Learning and Simulation

Some attempts of determining mechanisms of self-adaptation and self-organisation are in consensus with the biological inspiration and are based on the concepts of evolution and learning. They are either implemented in a simulation or directly shaping the agents' behaviour online during execution.

Das et al. (1995) have used genetic algorithms to automatically generate application-specific synchronisation strategies. Even though the presented approach still depends on a centralised timer for the calculation of updates in the distributed entities, it defines a promising direction. The particular challenge is the formulation of an adequate utility function and evaluation environment for a general application of this approach.

Matarić (1995) describes an approach of controlling group behaviour of robots by synthesising a particular complex behaviour based on a set of primitive behaviours. On that account, unsupervised reinforcement learning is used on a basic behaviour set to hide low-level control details to steer the global group behaviour towards the global goal. The authors also shaped the reinforcement function by partitioning the goal into sub-goals for providing more immediate feedback. In experiments, the authors have been able to implement a foraging task based on the empirically derived set of basic behaviours of avoidance, following, aggregation, dispersion, homing, and wandering. Open questions are, how the initial behaviour set is going to be selected, how such a system would be initialised in order to avoid possible harmful state conditions in the early stages of the reinforcement learning process, how the goals are partitioned, and how this might be combined with individual tasks of the robots.

Evolutionary robotics (Nolfi and Floreano, 2000) is a special domain of robotics that mainly researches how to automatically design robot controllers. The evolutionary idea is as well applied in swarm robotics (Kernbach et al., 2008). Depending on the particular approach the researchers use algorithms from machine learning, like artificial neural networks (Dorigo et al., 2004; Groß et al., 2006), combined with evolutionary-programming,

genetic programming (Groß et al., 2006) combined with physics simulations, before generated controllers are deployed to the real robots.

Francesca et al. (2015) are working on a general-purpose solution that generates the individual controllers of robots in a swarm. Therefore, the authors evaluated the performance of different approaches to design distributed control software for robot swarms in two empirical studies including several tasks. The evaluation of manual, manual with constraints and three automated control generation approaches is done using simple e-puck robots. Here, the solution *vanilla* is using F-Race optimisation, *chocolate* is using an iterated F-Race optimisation, and for comparison, *EvoStick* uses an Artificial Neural Network (ANN) approach without hidden layers that directly connects sensors and actors. Additionally, the simulation environment ARGoS is applied to determine the fitness of particular configurations against a utility function. The AutoMoDe approaches *vanilla* and *chocolate* are generating a probabilistic finite state machine by combining and fine-tuning pre-existing parametric modules. In (Francesca et al., 2015) these approaches are compared against human designers, which are allowed to freely design the control algorithms as well as in another setting, designing controllers that are constraint with the same limitations as the AutoMoDe approaches. The final results show that *chocolate* was able to outperform the human designers in the given controlled experimental environment. The presented result is promising, but it is still not clear if it can be generalised and applied to more complex scenarios. Especially the definition of a proper utility function can be a challenging task.

10.2. Functional Languages

The approaches presented in this section have developed different functional languages in order to program the behaviour of distributed systems. Broadly stated, using formalised functional languages has the advantage that it is simpler to prove the correctness of the program.

An early attempt is presented by Klavins (2004) with a functional language CCL for distributed systems based on predicates. The language is formalised and could be generally proven for determining the correctness of a program, but this is not yet developed. Furthermore, there is no functionality that would allow deducing single entity behaviour

10. Mechanism Design and Implementation

from a global mission description, requiring a specific implementation of each agent.

The functional language of Beal and Bachrach (2006) allows for programming sensor networks in a top-down fashion. Here the meta-code is compiled to bytecode that is going to be executed on a platform-specific virtual machine on the nodes. They are able to compile the global behaviour description, which is written in the language Proto, into locally executed code that produces the intended emergent phenomena. The validation of the result is accomplished with simulation. Nevertheless, the focus seems more on aggregation features for abstracting communication between the nodes, whereas the actually required rules still need to be developed in a manual way and bottom-up fashion.

The programming language Protoswarm is an extension of Proto. It allows programming robot swarms moving in space as single continuous spatial computer (Bachrach, McLurkin, and Grue, 2008). This approach provides an abstraction that enables global programming of a decentralised system. A disadvantage of this approach is that it requires homogeneous swarms with all agents exhibiting the same behaviour. This makes it very difficult to combine these behaviours with other individual tasks of the agent. Furthermore, Protoswarm has no explicit support for verification and validation.

10.3. Application-specific Approaches

A simplification of the problem domain can be achieved by concentrating on the application-specific requirements. Existing approaches are discussed in the following paragraphs.

The origami-inspired programming language Origami Shape Language (OSL) is able to translate its instructions into gradient rules that lead to self-assembly of the programmed geometric structure for a large set of identical agents (Nagpal, 2002). It relies on a set of simple primitives that are all related to gradient diffusion and neighbourhood gradient sensing. It is difficult to apply the folding language to real-life problems since it is hard to imagine how to fold the geometric structure in order to reach a desired shape. Furthermore, the approach also requires some kind of global knowledge, because each agent needs to know if it belongs to a certain area.

Similar to the previous paper, Joshi et al. (2014) are also addressing the specific application of self-assembly. The authors present a robotic system that allows to build a desired structure in a decentralised fashion with automatically generated rules for

the single agents. The agents perform independent local sensing and coordinate their activity through stigmergy of the perceived structure. In their solution, all robots have a representation of the intended global target and use qualitative stigmergy, meaning actions are triggered by qualitatively different stimuli, such as distinct arrangements of building material. In order to derive the local rules (movement guidelines) for the single robots, an offline compilation step is recursively searching through the state space to identify paths without cycles that are meeting the requirements. This is possible since the state space is reduced by using some global constraints, like all robots are starting from the same start point, a fixed movement direction, and a fixed robot behaviour routine.

Another work focused on modular robots, consisting of several independent agents, shows a generalised distributed consensus framework that allows for collective self-adaptation by using simplified local constraints (Yu and Nagpal, 2011). The authors mathematically proved the convergence to the intended goal state from all initial states and determined factors that are influencing the convergence speed, namely number of agents, task complexity, agent failure and agent reactivity. The authors used their framework to implement different self-adaptive modular robots, for instance, a gripper and an automatically balancing bridge. In their discussion, they stated several limitations and future research directions. In particular, their approach is only working for a single consensus state, thus, fulfilling a goal state of two independent sensory information would not be possible. Furthermore, tasks are assigned in advance, and dynamic task selection is not available. Another difficulty is the determination of the local constraints from the global objective function that could be addressed in future research by implementing a task-compiling framework for automatic constraint generation. Finally, the authors propose that future research could benefit from a mixed strategy of decentralised and centralised control.

10.4. Declarative Approaches

A different and more general direction is taken from other researchers investigating the applicability of declarative approaches. In this context, logic programming is used to describe global goals and restrictions of the system and allow to deduce the rule set of

10. Mechanism Design and Implementation

individual agents.

Kloetzer and Belta (2006) developed a hierarchical framework for defining swarm motion behaviour. It is based on linear temporal logic, constraints, and predicates. In particular, the authors abstract a large continuous geometric state space that needs to be fulfilled by the swarm entities. The abstraction is achieved by describing control constraints with mean and variance. Furthermore, the linear temporal logic formulas are automatically mapped to provably correct robot control laws. A disadvantage of this approach is that it requires global knowledge of the system state.

In the field of modular-robots, there is a promising programming language called Meld (Ashley-Rollman et al., 2007). In the fashion of logical programming languages, Meld uses a collection of facts and a set of production rules for combining existing facts. Execution facts are combined to satisfy given rules and to produce new facts that—in turn—can satisfy additional rules. This forward chaining process is executed until all provable facts were proven. However, the rules meeting the application goals have to be determined manually by the developer.

An additional perspective is given by another paper presenting the Locally Distributed Predicate (LDP) language. The LDP language was designed to develop distributed modules in a robot ensemble, so-called modular robot programming, from a global perspective (De Rosa et al., 2008). A LDP program is a collection of LDPs with associated actions that are triggered on any agent that matches the predicate. Their reactive programming approach does not consider even simple control structures and parallelisation, therefore, the realisation of more complex tasks becomes very challenging. Furthermore, their approach has not been evaluated with regard to self-organisation.

A property-driven design approach for swarm engineering is introduced by Brambilla et al. (2012). In this iterative top-down process, the requirements of the system are defined with logic formulas in the language PRISM. Based on the formal description, a macroscopic model is generated and verified with a model checker. Next, the verified model is used to guide the implementation of the system using a simulator. Finally, the system is tested on real robot hardware. While this approach can help to create and implement a model of a swarm with desired properties, it becomes problematic to create more complex models with robot-to-robot interaction, time, and spatial aspects that cannot be easily expressed with logic formulas.

Montagna et al. (2013) presents a domain specific solution addressing the field of pervasive services. The introduced framework for self-organising and self-adaptive systems is based on Live Semantic Annotation (LSA). Live Semantic Annotations are actually predicates with local predicates for individual agents as well as for the environment. Predicates are applied in inspiration from chemical formulas, where a specific input leads to a certain output. The global behaviour is enacted by defining rules in the LSA-space that are called eco-laws. All Live Semantic Annotations in a distributed space are updated based on the context and the eco-laws, which are defined and represented as the set of LSA stored in a given locality. They identified a set of basic rules to model mechanisms: *evolution, decay, diffusion, contextualisation, composition, synthesis*. This set is not complete, but in their opinion is sufficient for supporting low-level patterns. Furthermore, the authors developed the Resource Description Framework (RDF) as a language for structuring LSA and coding eco-laws. RDF relies on the SPARQL Protocol and RDF Query Language (SPARQL)/SPARQL/Update (SPARUL) query languages. The whole approach is tailored to the service domain with a comparable, less complex environment that is less prone to uncertainty and noise. Applying this solution, for instance to the field of multi-robot systems, would require modelling the complex real-world environment with LSA, which seems to be a very challenging task.

Service Component Ensemble Language (SCEL) is a formal language based on knowledge stored in tuple spaces (Nicola et al., 2015). The interaction (behaviours) with the tuple space can be guarded with policies. Furthermore, the language itself is minimal and does not include higher-level control constructs, but allows for semantic verification using model checking. In addition, the extensions Policed SCEL (PSCEL) adds direct support for the policy language Formal Access Control Policy Language (FACPL) and Stochastic Extension of SCEL (StocS) enables SCEL semantics. Due to its minimal characteristic, the expressiveness of SCEL and its extensions is limited, but decision-making can be integrated using external reasoners, but this is not part of the formal language yet.

10.5. Analysis, Verification and Validation

In connection with the question of how we can develop appropriate adaptation or self-organisation mechanisms is the challenge of analysing, verifying, and validating the created results. Due to the emergent characteristics and additional degrees of freedom, it needs to be ensured that the solution does not exhibit undesired, malicious, or dangerous behaviour. This is particularly important to make adaptive systems widely acceptable for users (Schmeck, 2005).

The best option of generating respective guarantees is using mathematical proofs, as they are achievable with logic programming and formal languages, like Meld (Ashley-Rollman et al., 2007) and SCEL (De Nicola et al., 2013), shown in Chapter 10.

In the discussed paper of De Rosa et al. (2008), the authors refer to logic programming languages for distributed systems like Meld. They argue that these tools are powerful because programs written in them have certain provable properties, but this provability limits the expressive range of the languages. A similar perspective is described by Edmonds and Bryson (2004), who state that self-organisation performance cannot be evaluated by purely formal methods. This argumentation is based on the assumption that reasoning alone cannot process all the information required to predict the behaviour of a complex system (see Chapter 8). Edmonds and Bryson (2004) argued that verification of code, as a pure design methodology, is only suitable for elementary problems and does not work for larger problems. This is proven with the halting problem of Turing. They propose that especially because of the large amount of time spent on debugging applications, we should better spend time on validation through experimentation. This is called experimental methodology like it is done in classical science like chemistry. In fact, they recommend that modules like agents should be accompanied by a set of testable hypothesis and data sets that have been tested with the modules.

Even the authors of Meld argue that representing the state space can be problematic. Moreover, dealing with a continuous state space, including probabilities and temporal constraints, can be difficult. For example expressing a behaviour that is able to react to perception of a fuzzy sensor, like a computer vision system, and expressing a response with a certain intensity or speed cannot easily be expressed with logic programming.

A different perspective is taken by De Wolf, Samaey, and Holvoet (2005). They show

an analysis of self-organising emergent application behaviour based on numerical analysis. It is executed on discrete simulation steps instead of mathematical equations. This allows for more reliable and valuable results in comparison to just observing simulation results. A critical point is how to identify the essential macroscopic properties and their related variables that quantify the property because these properties are specific for each application. Furthermore, De Wolf, Samaey, and Holvoet (2005) are referencing Wegner (1997), who showed that pure formal descriptions of the macroscopic behaviour of complex interacting systems are impossible in general.

To sum up, a pure formal verification of correctness is only partly applicable, since it relies on a complete description of the state space and hence is less useful for unknown, uncertain conditions. Other approaches using simulations and experimental results are limited as well, but could possibly complement a formal verification.

11. Planning and Decision-Making in Robotics

Self-organisation is often applied in swarm robotic systems to model collective behaviour, such as collective decision-making and task allocation (Brambilla et al., 2013). However, existing approaches entirely focus on the global perspective of the system without considering how this is integrated with the individual obligations of a robot, which might not be directly related to the global collective behaviour. To incorporate self-adaptation and self-organisation into the individual behaviour control of mobile robots in order to benefit from the advantages of those concepts it is necessary to integrate and combine them with existing approaches for individual decision-making and planning. This is necessary since especially intelligent general-purpose robots still have to pursue their mission of solving particular problems while they need further capabilities for self-adaptation and self-organisation in dynamic environments. In the following, we evaluate existing works regarding the applicability and support of dynamic environments.

As discussed in Hrabia, Masuch, and Albayrak (2015), adaptability in general, and

11. Planning and Decision-Making in Robotics

fast and flexible decision-making and planning in particular, are crucial capabilities for autonomous agents. The widely applied Robot Operating System (ROS) proposed by Quigley et al. (2009) and the many existing third-party packages illustrate the current state-of-the-art in applied robotics research. In the ROS community, the most commonly used approaches for task-level or behaviour control are rule-based and reactive. A popular package is *SMACH* that allows the user to build hierarchical and concurrent state machines (Bohren and Cousins, 2010). The approach of Nguyen et al., 2013 further extends *SMACH* with a system called ROS Commander (ROSCo) for rapid creation of Hierarchical State Machines through different user interfaces. The realised interfaces allow users of diverse experience levels to model robot behaviours on different abstraction levels in a fashion of visual programming. Particularly, the approach focuses on the service robotics domain in the home environment. However, Nguyen et al., 2013 are also defining a set of general parametrisable states that build the foundation for several higher-level tasks in the home environment. A more recent extension from Foukarakis et al., 2014 in the project *HOBBIT* is extending the *SMACH* approach, too. Here, the authors extract decisions about the transition to the next states from the particular state implementation to an external decision layer that is implemented with the Decision Making Specification Language (DMSL) (Savidis, Antona, and Stephanidis, 2005). Even though the authors propose that their solution is increasing the adaptation capabilities of a robot, their approach is only separating the design and implementation of decisions from the implementation of the state. The adaptation through the decision layer is still using static, and a priori defined transitions. Nevertheless, their approach potentially allows splitting the work of system-designers working on the high-level decisions and system-engineers that are working on the state realisation with detailed operational requirements like how to move.

All kind of (hierarchical) state machine based approaches have the problem that a decision or reaction can only be given if a state transition was already modelled in advance. New requirements, system goals, or actions have to be manually integrated into the existing state model, and potentially induce comprehensive changes in the overall structure. This is in contrast to goal-driven approaches that allow to dynamically adjust the target state of the system through adding, removing, or updating goals. The more task-oriented behaviour trees, available in the *pi_trees* package (Goebel, 2013), are

another popular alternative to HSM approaches. Even though behaviour trees allow for more reusable behaviour implementations and some higher-level abstractions, by using special operators like sequencer and selector, the path of execution is still preprogrammed and is likely to fail in dynamic environments. The reason is that it is difficult to design such system without missing possible useful dynamic transitions.

More flexible is the Belief, Desire, and Intention (BDI)-based implementation *CogniTao* (CogniTeam, 2015) available in the *decision_making* package that also includes modules for the creation of finite and hierarchical state machines. CogniTao is targeting high-level control of multi-robot systems in dynamic environments. In CogniTao the user defines behaviours, called plans, as decision-graphs with start and end conditions. Behaviours are executed and selected by protocols. The selection and decision process is synchronised through a team of robots. The concept is more suitable for uncertain environments because the execution sequence is not fixed, and the selection of behaviours (plans) is based on conditions. Nevertheless, it is still difficult to define mission or maintenance goals, and there exist only simple protocols for plan selection.

Outside of the ROS community, we can find other reactive approaches, for instance, the subsumption architecture (Brooks, 1986) that is using a hierarchy of controllers. The controllers generate output signals and are able to suppress or influence the signals of lower levels. Finally, signals are combined and mapped to particular actions. The system is able to deal with uncertain situations since it always creates reactions, but it is still hard-wired, and can address neither dynamic goals nor sequential task dependencies.

Influenced by the concept of artificial neural networks is the behaviour network architecture introduced by Maes, 1989. Here, the behaviours are dynamically connected with each other based on their preconditions and effects, which enables dynamic state transitions, receptively a stateless action model. The executed behaviour is dynamically selected based on a utility function. The utility function is calculating the positive influence on a goal for each behaviour, called activation, which is spread through the network. Different extensions of this approach introduce learning capabilities (Decugis and Ferber, 1998; Jung, 1998) and multi-robot application (Jung, 1998) or concurrency (Allgeuer and Behnke, 2013) and non-binary preconditions (Allgeuer and Behnke, 2013; Decugis and Ferber, 1998). The hierarchical version of Allgeuer and Behnke is available as a ROS package but simplified in the way that it only relies on pre-computed static

inhibitions of conflicting behaviours. A general issue is that network parameters need to be properly tuned towards a specific application to avoid the risk of getting stuck in cycles since it is difficult to express a required execution order.

Classical symbolic planners in the tradition of STRIPS (Fikes and Nilsson, 1972), like Hierarchical Task Networks (HTNs) (Breuer et al., 2012) or further advanced implementations, like graph-based approaches (Blum and Furst, 1997) or heuristic planners (Bonet and Geffner, 2001; Hoffmann, 2001) are very good at directing towards a goal and determining a possible execution order. ROSPlan is a generic symbolic task planning framework, which follows the further developed STRIPS approach (Cashmore et al., 2015). In fact, ROSPlan is a Planning Domain Definition Language (PDDL) dispatching framework that allows to map PDDL actions to ROS actions. It also includes an Web Ontology Language (OWL)-based knowledge base, which is used to automatically generate the problem file, and an observer that determines situations requiring replanning. However, the model of the domain has to be provided in PDDL, and the mapping has to be verified manually. Moreover, ROSPlan monitors the plan execution and uses its filter module to manage eventually needed replanning. The support of multi-robot systems is only limited, even though the ROS foundation enables planning across multiple robots, it is not possible to execute multiple actions in parallel.

Another deliberative system for ROS is the skill-based task planning system SkiROS (Rovida et al., 2017; Rovida and Krüger, 2015). It is focusing on applications in automation and supports the usage of multiple robots. The framework SkiROS enables a goal-oriented mission execution based on provided skills and their constraints. In SkiROS, skills are always compound tasks. The implemented solution has a three-layer architecture with a deliberative, executive and behaviour layer. Whereas the behaviour layer is the part of the implementation that has to be provided by the user but is supported by several interfaces and base classes. The framework implementation is based on common standards as Web Ontology Language Description Logics (OWL-DL), SPARQL, PDDL, and common ROS features. The actual planning is also realised with an integrated PDDL-based planner. In detail, the PDDL interface requires PDDL compatibility to version 2.1, all required PDDL-files are generated automatically. SkiROS includes a knowledge base that holds the semantic world model as an ontology. The ontology is expressed with OWL and has to be provided by the user, similar to ROSPlan. In contrast

to ROSPlan, the filtering and monitoring of the plan execution are not handled in one central component, but in SkiROS, every skill checks the execution conditions before it is started. If conditions are violated, replanning is triggered automatically.

In Rovida, Grossmann, and Krüger, 2017 an extension to SkiROS is presented that can be used as a post-processing step after the PDDL-based symbolic planning to optimise the execution order of primitive tasks. For this purpose, the authors developed an approach where execution procedures and the planning domain are specified at the same time with an extended Behaviour Tree (eBT). eBT is a joint model of the behaviour tree and HTN paradigms. Unfortunately, the published implementation is not yet fully integrated into SkiROS and does not have a direct connection to a PDDL planner.

Pure PDDL-based symbolic planners like used by SkiROS and ROSPlan are not considering probabilities because only the extension of PDDL named Probabilistic Planning Domain Definition Language (PPDDL) (Younes et al., 2005) and Relational Dynamic Influence Diagram Language (RDDL) (Sanner, 2010) are supporting it. In particular, PPDDL is considering a fully-observable state space as a Markov Decision Process (MDP) while RDDL also supports Partially-Observable Markov Decision Processes. Applying POMDPs and MDPs in ROS is possible with a combination of solvers like the Multi-Agent Decision Process (MADP) toolbox (Oliehoek et al., 2017) and the ROS package `markov_decision_making`¹. This ROS package enables abstract representations of states, actions, and observations to model a problem and execute the policy that was calculated with an interfaced solver.

However, all symbolic planners have problems under uncertain conditions because of high computational costs and eventual replanning that has to calculate alternative decisions in case of unreachable goals or unexpected world changes. Especially MDP and even more POMDP planners that are considering probabilities have high computational complexity due to the state space explosion induced by the additional probability dimensions and the need to trace historical decisions (Kaelbling, Littman, and Cassandra, 1998). A general problem with planning-based approaches is that what to do if there is currently no plan available that will lead to the desired goal.

In order to combine the advantages of both reactive behaviour approaches, being fast and more flexible and symbolic planning, being more goal-directed, some researchers

¹http://wiki.ros.org/markovdecision_making

have proposed hybrid architectures for single robot systems. An early attempt was given by Hertzberg et al., 1998, who proposed a low-level behaviour controller using two differential equations, one for creating the actuator control signal and one controlling the activation. The integration of a special term in the activation equation enabled external influences from an operator like a human or a symbolic planner. Here, the reactive behaviour layer is still operational on its own, and the model for the planner has to be provided independently. Additionally, Hertzberg et al., 1998 are using Action Description Language (ADL) (Pednault, 1989) as representation language.

A more recent hybrid approach from Lee and Cho, 2014 models each high-level task as an independent behaviour network that is selected by a planner. This architecture is strongly goal-directed but loses flexibility for dynamic adaptation since behaviours of different networks cannot be combined with each other.

In general, hybrid decision-making and planning systems allow to balance the advantages and disadvantages of both approaches in a beneficial way. They combine reactive behaviour-based approaches for short-term fast response with symbolic planning for the determination of the correct sequence of actions in long-term. Such architecture provides a suitable foundation for a robot that adapts to unexpected and dynamic changes in its environment. The support of dynamic state transitions as well as relying on a reactive agent model are also enabling a suitable foundation for an integration of self-adaptation and self-organisation capabilities. On this view, especially behaviour-network-based solutions on the foundation of (Maes, 1989) are promising for the reactive part of the architecture because they are not hard-wired and allow for dynamic state transitions. The literature review shows many fruitful ideas and extension of behaviour-network-based concepts, such as non-binary preconditions (Hertzberg et al., 1998), learning capabilities (Jung, 1998), ROS-support (Allgeuer and Behnke, 2013), and the integration with symbolic planning (Lee and Cho, 2014). Nevertheless, these ideas are not yet integrated into one solution, and even though not all of them are available for the ROS community. Furthermore, extending the concept towards multi-robot applications and enabling additional learning capabilities are open issues. On that account, also any kind of adaptive coordination is not yet considered. Here, multi-robot coordination, if it is available at all, is only considered in a static a priori defined centralised fashion enabling the distribution of tasks amongst the robots. The following chapter will elaborate this in more

detail, but will also broaden the focus with respect to backgrounds as well as relevant related work about task decomposition, allocation, and delegation.

12. Task Allocation, Decomposition, and Delegation

In the previous chapters, we already discussed work that is addressing self-adaptation, self-organisation, as well as decision-making and planning in order to foster adaptability of autonomous multi-agent systems.

Additional essential aspects that allow to improve the adaptation capabilities of a multi-agent system and which are closely related to decision-making and planning are concerned about tasks, their partition, and distribution in a multi-agent system. Here, the decomposition of complex tasks, its allocation, and delegation give additional degrees of freedom and increase possibilities for a dynamic adaptation during the runtime of the system. For instance, a large complex task that is handled by a decision-making and planning component could be fulfilled in parallel or in alternative ways by multiple agents after it has been decomposed, allocated and delegated in contrast to a simple direct allocation to one agent. Furthermore, this may also allow to explicitly coordinate certain tasks that would not have been possible for single agents.

Even though this is also partially addressed by some self-organisation algorithms (Ducatelle et al., 2009b) because distributed task decomposition and assignment are core topics of self-organisation. If self-organisation is used the coordination of tasks including the decomposition, such as exploring a larger environment in chunks, it is coordinated implicitly and highly distributed. Nevertheless, a combination of decision-making and planning with self-organisation needs also to consider a combination of implicitly and explicitly coordinated tasks. In consequence, we will introduce allocation, decomposition, and delegation of tasks as well as relevant approaches in the remainder of this chapter.

12.1. Decomposition

Decomposition is the process of splitting larger tasks or goals into smaller sub-goals, respectively sub-tasks. A common representation of task decomposition is a task-tree.

Following the definition from Zlot and Stentz, 2005 a task tree is a rooted set of task nodes. These task nodes are connected by directed edges in a graph that specify parent-child relationships. Each successive level of the tree represents a further refinement of a task. The root task node describes the task in the most abstract way, while the lowest level contains the most primitive or atomic tasks.

A task decomposition can be done on a single agent as well as on a multi-agent level. Planning and scheduling is also task decomposition because such systems try to select and arrange a set of actions in order to fulfil a given goal or higher-level task.

In order to realise task decompositions, a popular approach is the Task Description Language (TDL) (Simmons and Apfelbaum, 1998). TDL is a language for describing task trees. Here, complex goals are refined by sub-goals, which again can be refined. Additionally, the language allows to describe dependencies between sub-goals like necessary execution orders. The actually used decomposition is dynamically determined during runtime. Nevertheless, the complete decomposition in TDL has to be modelled a priori by a system designer.

Realising a dynamic decomposition based on a priori defined task trees with certain conditions is a common approach. Zlot and Stentz, 2005 extends the idea with additional conditions for logical disjunction and conjunction of tasks, while Doherty et al., 2016 adds concurrency, repetition, and fulfilment checks.

A decomposition can be calculated statically in a central instance of a system (Lemaire, Alami, and Lacroix, 2004), in a distributed way in individual agents or in a mixed approach where agents share and combine their decompositions (Zlot and Stentz, 2005) with a central instance or other distributed agents. The distributed decomposition in (Doherty et al., 2016) does also enable a further decomposition of each agent individually.

After the decomposition of a task, we obtain sub-tasks or sub-goals. If such sub-tasks are not processed directly by a single agent, it leads to the question of which agent is taking care of which task. This question is discussed in the following section.

12.2. Task Allocation and Delegation

Korsah, Stentz, and Dias, 2013 define task allocation as the general problem of determining which robot should execute which task in order to achieve the overall system goals. As already mentioned, this is one core aspect of coordinating a multi-robot system. If such coordination emerges from local interactions between the agents, we talk about one type of self-organisation algorithms. Nevertheless, the coordination can also be achieved with explicit coordination that can be executed centrally or in a decentralised way.

Task delegation is closely connected to task allocation. It describes the assignment of a task from one agent to the other. Here, the agent that gets the task assigned becomes responsible for the task.

In most cases, the allocation and delegation of tasks are targeting for an optimal solution, usually in terms of efficiency with respect to the goal fulfilment. Efficiency is often measured in time but it can also consider other limited resources.

A common approach for task allocations is applying a central instance that takes care of the calculations. Popular examples are the Hungarian algorithm (Mills-Tettey, Stentz, and Dias, 2007) or direct auctions like combinatorial auctions (Rassenti, Smith, and Bulfin, 1982).

The Hungarian algorithm tries to distribute a set of known goals to agents such that the costs for the individuals are minimised. Unfortunately, the approach does not address scenarios where agents can take care of multiple goals.

Combinatorial auctions are a group of algorithms for finding cost-efficient allocations that do not rely on multiple individual auctions for multiple goals and instead uses auctions for a set of goals. Here, the auctioneer is responsible for calculating the most optimal solution considering that goals can be part of several goal packages.

Other common approaches are contract-net protocol based (Smith, 1980) solutions. In the decentralised implementation M+ (Botelho and Alami, 1999) each agent sends an offer to the best matching goal, for instance, the best utility, as a broadcast to all agents. If the agent does not receive a better offer for the same goal from other agents, it considers the goal delegated to itself and pursues it. A disadvantage of this approach is a high communication overhead and the sensitivity against lost messages, which could result in multiple agents pursuing the same goal.

Other approaches extend the idea of the contract-net protocol with subcontracting that allows to further delegate sub-goals of a goal to other agents, which is especially useful if it is combined with task decomposition.

13. Reinforcement Learning

Autonomous learning of agents can further foster the self-adaptation capabilities of agents. Moreover, it has already been mentioned in Section 10.1 for the implementation of scenario-specific self-organisation. Additionally, learning can also be the foundation for decision-making that is mostly based on experience. Such additional learning features potentially allow to further relieve the system designer or engineer from the burden of designing complex models of the system, which have to consider all future situations the system will have to handle.

In this regard, the machine learning class of reinforcement algorithms is particularly interesting because it has proven astonishing performance in various fields, such as board games (Silver et al., 2017), logistics (Rabe and Dross, 2015), and robotics (Peters, Vijayakumar, and Schaal, 2003). Furthermore, such RL algorithms enable continuous online learning and adaptation of a system during its execution. For this reason, the remainder of this chapter provides general background as well as a discussion about relevant related work.

The main idea behind RL is that systems learn what to do in a certain situation by exploring the environment in a trial-and-error fashion while recording the world state at this time and the result (reward) of the evaluation against a given utility function, to finally learn over time from experience.

In contrast to *supervised learning*, RL does not rely on existing datasets of labelled data that is used for training the system, e.g. as commonly done in pattern matching (Hrabia et al., 2019). RL is also different from *unsupervised learning* because it is not trying to discover structures in unlabelled data, as for instance used for classification. Moreover, the problem of how to deal with the trade-off between exploration and exploitation, describing the challenge of the algorithm if it should prefer to evaluate new possible actions or if it should take the best option of its already learned policy, is a core property

of RL and not considered in *supervised learning* and *unsupervised learning*.

13.1. Problem Classification

In order to select the most suitable RL method, it is necessary to classify the learning problem. First, the environment state can be either discrete or continuous. Discrete problems are having a finite set of states such as board games like chess or Go. Continuous problems can be found in all kind of (real-) time-dependent applications such as robotics with sensors measuring the real-world environment. Continuous problems are more challenging due to the extended state space.

Secondly, we can distinguish stochastic and deterministic problems. In deterministic problems, an action will always lead to the same transition between two states, in a stochastic problem the transition might lead to different states. This consideration also includes stochastic properties induced by other agents, e.g. they might react not always the same way, or a generally uncertain and dynamic environment. In stochastic problems, each action must be tried many times to gain a reliable estimate of its expected reward.

Thirdly, the environment can be either fully-observable or partly-observable. Here, the agent is either able to observe all states in the environment, e.g. in chess or some information is not visible for him, like in Texas Hold'em Poker. Here, the challenge is to consider also the influences of only partly observable states.

Finally, we can differentiate problems with just a single agent or with multiple agents. Additionally, multiple agents can either collaborate to achieve their common and individual goals, or the agents can compete with each other. Problems with multiple agents are more challenging due to the increased search space.

The classification of the problem does not directly lead to the selection of the one most optimal algorithm, but it allows selecting something suitable based on the experience from the related work. For instance, discrete problems are more often addressed with value-function methods and continuous problems with policy gradient methods. Both methods are explained in the following section.

13.2. Reinforcement Learning Methods

The already mentioned methods of value-action and policy gradient define the general classes of RL.

The *value function method* maps state-action pairs to gained rewards and is optimised to maximise the reward. The value function method defines the problem as MDP; hence, the formalisation consists of states, actions, and transitions. A specific state includes all information that is required for further calculations. The value function itself aggregates the reward gained of multiple decision rounds. The value is more important than the current reward because an action might not gain the optimal reward at the moment but might lead to the optimal aggregated reward over several decision steps. A value function is optimal if its rewards for each possible state action pair and policy are maximised. An optimal value function results in an optimal policy.

A popular approach of optimising the value function is Q-Learning (Watkins and Dayan, 1992). The introduction of Q-Learning marks the beginning of modern RL. The special attribute of Q-Learning is the fact that it is able to find the optimal policy for every finite MDP without requiring a model of the environment. Q-Learning uses an *action-value method* that describes for every action and state the expected reward. On this foundation, the optimal policy can be achieved by selecting the maximal reward for every state over time. In consequence, the value is defined by the weighted sum of all expected future rewards. The weighted sum is calculated with a discount factor, which determines the importance of future against current rewards.

The *policy gradient method* optimises the selection of actions to maximise the reward based on world observation. In consequence, policy methods directly map observations to actions. This is in contrast to the above-discussed value function methods like Q-Learning, which are learning the assessment of actions in certain states. Hence, a policy method learns a particular policy, which is in contrast to a value function that is used to create a policy. The advantage of policy methods is that they potentially converge faster and more effective, especially in the case of high dimensional search spaces. Furthermore, policy methods allow to learn stochastic policies. The disadvantages of policy methods are that they are prone to get stuck in local optima and that solutions potentially have high variance, and that there is no guarantee for converging in finite time.

13.3. Comparing Value Function and Policy Gradient Methods

Policy methods are classified into the groups of policy with and without gradient. Methods without gradient are for instance applying hill climbing (Kimura, Yamamura, and Kobayashi, 1995) and simplex (Kohl and Stone, 2004), and genetic (Moriarty, Schultz, and Grefenstette, 1999) algorithms. In the following, we focus on policy gradient methods because they are often more efficient and are able to exploit the sequential structure of the decision problem.

Several approaches for gradient methods are known in the literature. Deterministic model-based methods are one option that rely on environment information provided by the developer. The application of a model allows for faster convergence. However, apart from toy examples, it is difficult to model the world in appropriate detail, especially if the environment is dynamic. For this reason, model-free approaches are more suitable for robotic applications. A comprehensive survey about applicable policy gradient methods in robotics can be found in (Peters and Schaal, 2006).

Common approaches for policy gradient methods are *finite difference* and *likelihood ratio* methods. Finite difference methods provide a black-box approach to compute gradients of arbitrary policies, even though a policy is not differentiable. However, the approach is noisy and comparable inefficient. Finite difference methods are especially suitable for deterministic systems (Kimura, Yamamura, and Kobayashi, 1995). Likelihood ratio methods are based on conditional probabilities. They are able to converge faster in general and especially for stochastic problems. The disadvantage is the more difficult implementation in comparison to the finite difference method.

13.3. Comparing Value Function and Policy Gradient Methods

Both *value function* and *policy gradient* methods have advantages and disadvantages. Selecting the most suitable specific algorithm for an application is not distinct and always depends on the actual problem. Nevertheless, it is possible to provide general guidance if either value functions or policy gradient methods should be taken into account.

Advantages of policy gradient methods are that they usually converge faster, can be applied on stochastic problems, and expert knowledge can easily be integrated through a model. Policy gradient methods are often applied to continuous problem spaces, e.g. (Kohl and Stone, 2004). As mentioned before, the major disadvantage is that they are

13. Reinforcement Learning

prone to get stuck in local optima and cannot guarantee to converge in finite time.

Value functions methods are especially applicable for problems with a limited action space, such as board games like chess. Moreover, approaches based on Q-Learning are able to find the optimal solution in finite time. Furthermore, it is especially interesting that value function based approaches are also able to handle high-dimensional and continuous input spaces in a dynamic environment as they can be found e.g. in Atari computer games (Bellemare et al., 2013). The application of value function methods in such environments becomes possible through the approximation of complex input states, which can be achieved with ANN (Mnih et al., 2013).

13.4. Deep Q-Network

The value function-based approach Q-Learning is a powerful algorithm and performs well in many toy examples. However, it lacks generality and is difficult to apply to real-world scenarios. The reason is that Q-Learning usually stores its transitions in a two-dimensional action-state-space table, which in fact resembles dynamic programming. Q-Learning is selecting the most appropriate action based on the states it has seen before. First, this does not scale well because all experience has to be stored. Secondly, this does not work for continuous values that are often including noise and which are often not repeated as exactly the same values. As already mentioned above, there are implementations that approximate the complex action-state-space table input with ANN, for instance, the Deep Q-Network (DQN) approach (Mnih et al., 2013) replaces the two-dimensional table of Q-Learning that way. In this approach, the ANN is used to estimate the action-value function. The input for the ANN is the current state, while the output is the corresponding Q-value for each of the actions. In particular, Mnih et al., 2013 used a Convolutional Neural Network (CNN), which is a class of ANN often used for computer vision tasks to abstract from pixels to patterns. The DQN approach received a lot of attention because it allowed to create an artificial intelligence that was playing Atari video games on the level of human experts without any prior knowledge about the game rules.

Nevertheless, the DQN approach is not limited to CNN, but instead, the most suitable ANN architecture can be selected with respect to the input values and application

scenario. In consequence, also fully connected networks for general-purpose learning can be chosen.

Furthermore, various extensions and special considerations improve the applicability and performance of the DQN approach. Extensions of particular importance are presented in the following.

Experience replay enables continuous training from stored experience. This is achieved by storing taken actions and their results as transition tuples of {State, Action, Reward, Next-State} and using batches of these tuples as an additional information source during the training instead of just training with the most recent result. Usually, the batch of experience data is randomly selected from the set of stored transition tuples. A common approach is also to limit the storage, also called replay memory, of experience to a specific amount and to replace the oldest available data with the most recent one continuously in a First In - First Out (FIFO) approach.

The advantage of *experience replay* is that it stabilises the network and avoids unwanted correlation between states. The reason is that after a certain state, only a limited number of other states can follow, which is avoided by shuffling the data used for training with the *experience replay* approach.

In order to stabilise the learning mechanisms, an additional ANN – *Target Network* – can be used to generate the Q-Values (Lillicrap et al., 2015). This *Target Network* is used to calculate the loss for all actions during the training of the network. This approach delays and desynchronises the update of the weights in the main ANN of the DQN.

Another extension is the Double DQN (DDQN) approach that avoids overfitting of the bias by differentiating between selection and evaluation of actions (Hasselt, 2010). The pure DQN is prone to overfit the bias due to a maximisation step in the Loss function. In this step, a selection of the most rewarding action is taken place based on the ANN, but the same ANN is used to evaluate this action with the Loss function and to train the network. The DDQN decouples this process through the introduction of an additional ANN that results in improved performance.

Other existing extensions such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) allow improving the applicability in continuous action space.

13.5. Exploration vs. Exploitation

A general challenge of all RL-based approaches is the selection of the current action. Here, either the algorithm can select the most optimal action on the foundation of the currently known policy or it can decide to select an option with unknown or not yet stabilised reward to further explore the search space to potentially gain more reward in the future. This trade-off challenge is commonly known as the question of *exploration vs. exploitation*. In general, both strategies are reasonable, but they have to be balanced. Exploration supports an increased reward in the long-term, while exploitation achieves the best reward in short-term. Different algorithms can be applied to implement the balancing between exploration and exploitation. Subsequently, we present common action-value methods and the softmax action approach.

A simple *action-value method* uses a parameter that defines the percentage or ratio of actions that are selected to further explore the search space. An example configuration could use, for instance, 10% of the action selection to explore, while in 90% of the time, the action is chosen that maximises the reward.

This approach is easy to implement but has the clear disadvantage that the ratio of exploration has to be defined by the system designer, which would have to be determined empirically for each learning problem domain. Furthermore, the probability of selecting a better or worse action during the exploration will stay the same.

An extension of this method is called *epsilon-greedy* algorithm. The epsilon-greedy algorithm is reducing the probability (ϵ parameter) of exploration over time to favour exploration in the beginning of the search and exploitation in a later stage with an already trained system.

In contrast to the action-value method, the *softmax method* has the advantage of eliminating the uniform distribution of probabilities between the selection of better or worse actions during exploration. This is achieved by considering an additional factor that describes the expected value of an unknown action. During the action selection, a *softmax function* is used to favour potentially better actions. However, this approach requires additional information about the expected value of unknown actions, which have to be provided by the system designer. Commonly, the softmax action selection is also known as *Boltzmann Exploration*.

14. Analysis Summary: Opportunities and Challenges

The aim of this part is to identify ways, mechanisms, and approaches that allow to develop systems that comprise large numbers of constituting agents in complex and uncertain environments and to make these systems more robust and to increase their ability to adapt. Explicitly in focus is one particular application domain, namely multi-robot systems.

Despite the fact that there is some progress and interesting work done in the community, there are still many open questions that remain open. Table 14.1 allows for an aggregated comparison of all self-adaptation, self-organisation, and swarm robotics approaches that were mentioned in Chapter 8 to 10. The comparison shows that existing approaches become increasingly specific (that is domain-specific and application-specific) the closer one looks into the actual mechanisms and algorithms that are required to exhibit adaptive and cooperative behaviour. Hence, we observe that—on a methodical level—all solutions are domain independent. Methodologies that can be classified as *more specific* are very close to the evolutionary emerged natural role models of self-organising systems that have been developed with trial and error, respectively in a bottom-up manner, over centuries. For instance, they apply learning, simulation, and experimentation or put existing natural algorithms into reusable patterns.

All mechanism designs and implemented solutions that do not apply mechanisms in an ad-hoc and bottom-up manner are confined to a specific domain and are sometimes even centralised. The complete centralisation is exceptionally cumbersome whenever the system scales up. This is mainly due to possible communication and computation bottlenecks as well as having a single point of failures that limits performance and robustness for larger systems.

Available middleware solutions and frameworks are also often limited by their rather centralised or abstract approach or focus on particular problems, like system performance or resource management. Furthermore, none of the more concrete approaches tries to address the problem of creating an adaptive, self-organising system in a goal-oriented manner. A general-purpose approach to develop the required mechanisms is still missing,

14. Analysis Summary: Opportunities and Challenges

Table 14.1.: Comparison of the properties of reviewed approaches from self-adaptation, self-organisation and swarm robotics discussed in Chapter 8 to 10.

Reference [SolutionName]	Domain	Centralised/ Decentralised	Specific Focus	Top-down/ Bottom-up	Approach
Methodologies					
Steghöfer et al. (2014), [PosoMAS]	general	not specified	no	n/a	software engineering process, simulation
Picard and Gleizes (2004), [ADELFE]	general	not specified	no	n/a	software engineering process
Genshenson (2007)	general	not specified	no	n/a	software engineering process
Branke et al. (2006), [Organic Computing]	general	Hybrid	no	n/a	learning, feedback-loop architecture, simulation
Edmonds (2005)	general	not specified	no	n/a	software engineering process, experimental methods
De Wolf and Holvoet (2007)	general	not specified	no	n/a	self-organisation algorithm guide
Fernandez-Marquez et al. (2012)	general	not specified	no	n/a	self-organisation algorithm guide, incl. combinations
Noël and Zambonelli (2015)	general	not specified	no	n/a	component decomposition
Middlewares and Frameworks					
Hernandez-Sosa et al. (2005), [CoolBot]	robots	centralised	QoS	n/a	component-based framework
Cui et al. (2014), [ReFRESH]	embedded systems	centralised	fault-detection	n/a	special reconfiguration layer, built into RTOS
Garhan et al. (2004), [Rainbow]	general	centralised	no	n/a	separated adaptation layer
Krupitzer (2018), [FESAS]	general	application-specific	reusability	top-down	meta-adaptation of components, MAPE
Sudeikat et al. (2009), [SodekoVS]	general	decentralised	no	n/a	separated adaptation layer, domain specific language
Presler, Vilenca, and Renz (2013), [JadeX]	general	decentralised	no	n/a	separated adaptation layer, domain specific language
Graff, Rüdiger, and Werner (2014), [Swarm]	robot swarms	centralised	performance, scheduling, information propagation	mixed	precomputed dependency graph
Mamei, Vassirani, and Zambonelli (2005), [TOTA]	sensor networks	decentralised	information propagation	bottom-up	spatial-temporal constraints
Yu and Nagpal (2009)	modular robots	centralised	no	bottom-up	control laws
Vinoli, Casadei, and Onicini (2009), [TWCSON]	general	decentralised	no	bottom-up	probabilistic and timed coordination rules on tuples
Fernandez-Marquez, Serigendo, and Montagna (2011), [BIO-CORE]	general	decentralised	information propagation	mixed	shared data space, pattern services
Pinciroli and Beltrame (2016), [BUZZ]	robot swarms	centralised	no	mixed	domain specific language, task assignment
Mechanism Design and Implementation					
Das et al. (1995)	cellular automata	centralised	synchronisation	top-down	genetic algorithms
Mataric (1995)	robot teams	decentralised	motion	mixed	set of primitive behaviours, reinforcement function
Klavins (2004), [CCL]	general	decentralised	no	bottom-up	predicate-based functional language
Beal and Bachrach (2006)	sensor-networks	decentralised	no	bottom-up	functional language, simulation, VM abstraction
Bachrach, McClunin, and Grue (2008), [Protoswarm]	robot swarms	decentralised	motion	top-down	functional language, homogeneous swarms
Nicola et al. (2015), [SCEL]	general	centralised	general assembly	top-down	formal language, tuple space, model checking
Nagpal (2002), [OSL]	robot self-assembly	centralised	assembly	top-down	special origami-inspired language, gradient rules
Joshi et al. (2014)	robot self-assembly	decentralised	no	bottom-up	offline compilation, special application constraints
Yu and Nagpal (2011)	modular robots	decentralised	no	bottom-up	generalised distributed consensus
Kloetzer and Belta (2006)	robot swarms	centralised	motion	top-down	mean and variance of geometric state
Ashley-Rohman et al. (2007), [Meld]	modular robots	decentralised	no	top-down	logic programming language
De Rosa et al. (2008), [LDP]	modular robots	centralised	no	top-down	reactive predicate language
Brambilla et al. (2012)	robot swarms	centralised	no	top-down	property-driven design, formal language
Montagna et al. (2013), [USA]	pervasive services	decentralised	no	mixed	predicate rules, basic rules to model mechanisms
Francesca et al. (2015), [AutoMoDe]	swarm robots	centralised	no	top-down	evolutionary programming, simulation, optimization

even though some promising solutions for simple robot swarms with limited capabilities exists (Francesca et al., 2015). We do not know yet, how such kind of system can be designed and developed from a goal-oriented perspective in a way that an intended global behaviour is deduced to individual agent behaviours. Furthermore, as yet, it is unclear how the required self-organisation mechanisms can be automatically determined from the application's perspective. Even though declarative approaches, such as (Ashley-Rollman et al., 2007; Brambilla et al., 2012; Montagna et al., 2013; Nicola et al., 2015), allow for some deduction and validation of the mechanisms, they lack expressiveness and cannot be used to model uncertain, complex environments.

Existing architectures dictate that the engineer is implementing the actual agent behaviours from a bottom-up perspective or the approach is very limited to a particular use-case and limited in its expressiveness. A framework architecture that allows for a goal-driven approach using a top-down, or mixed bottom-up and top-down process is not yet available.

Moreover, when it comes to multi-robot systems, the target domain of this work, existing solutions are focused either on a particular application, e.g. robot self-assembly, where modular robots are centralised and inflexible, or their capability is limited to encompass the conceptualisation of the motion of a swarm system. The extensive analysis of existing literature did not discover works that deal with the problem on how self-organisation and self-adaptation can be combined with existing and indispensably required task-level decision-making and planning of general-purpose robotic systems that are not just focusing on the self-adaptation or self-organisation task itself.

Existing robotics planning and decision-making approaches discussed in Chapter 11 are related to each other in Table 14.2. Due to inherently different concepts, only approaches based on symbolic planning, behaviour networks, and hybrid approaches are able to provide a goal-oriented application. The other approaches derive from HSM, whereas they rely on predefined static transitions between modelled states and do not encompass any consideration of goals. This is especially problematic if the final application needs to be controlled by a user who wants to adjust the system goals during runtime. Especially extensions of presented concepts with adaptive learning capabilities, as introduced in Chapter 13, or implicit coordination mechanisms for multi-robot systems that can be used in a goal-directed fashion are open issues. This underlines the

14. Analysis Summary: Opportunities and Challenges

relevance of the proposed research questions *R1* and *R2* again.

Avoiding problems of centralised management and control also requires a decentralisation by introducing additional hierarchy levels into behaviour network, symbolic planning, and hybrid behaviour planning approaches. Similarly, considerations of further aspects of multi-robot decision-making such as task decomposition, allocation, and delegation are not yet done. Particularly, these aspects are also able to improve behaviour adaptability on the task-level in a multi-robot environment.

Table 14.2.: Comparison of robot planning and decision-making approaches.
[Y=Yes, N=No, P=Partially, M=Manually]

Reference [Solution]	Goal-Oriented	Learning	Hierarchies	Standard Description	Non-binary Conditions	Parallel Behaviour Execution	ROS Integration	Multi-Robot Support
Maes (1989) [MASM]	Y	N	N	N	N	N	N	N
Decugis and Ferber (1998)	Y	P	Y	N	N	N	N	N
Hertzberg et al. (1998)	Y	N	N	ADL	Y	N	N	N
Jung (1998) [ABBA]	Y	Y	N	N	N	N	N	M
Bohren and Cousins (2010) [SMACH]	N	N	Y	N	Y	Y	Y	P
Breuer et al. (2012) [HDL,JSHOP2]	N	N	Y	OWL-DL	Y	N	N	N
Goebel (2013) [pi_trees]	N	N	Y	N	Y	Y	Y	P
Allgeuer and Behnke (2013)	N	N	N	N	Y	Y	Y	N
Foukarakis et al. (2014) [HOBBIT]	N	N	Y	DMSL	Y	Y	Y	P
Lee and Cho (2014)	Y	N	N	N	N	N	N	N
Cashmore et al. (2015) [ROSPlan]	Y	N	N	PDDL, OWL	Y	N	Y	P
CogniTeam (2015) [CogniTAO]	N	N	Y	N	Y	Y	Y	Y
Rovida et al. (2017) [SkiROS]	Y	N	N	PDDL, OWL-DL	Y	N	Y	Y
Oliehoek et al. (2017) [MDM]	N	Y	Y	N	Y	Y	Y	Y

Nevertheless, the analysed literature shows a number of common ideas and concepts in order to further the spirit of more reusable, respectively less application-specific, implementations, and simplified development of adaptive single or multi-agent systems. The most important ideas and concepts are:

- CC1** The implementation of coordination and adaptation are separated from the application behaviour and have been made explicit, forming own layers or components in the architecture.
- CC2** Adaptability is realised with feedback loops that can be integrated on different levels into the architecture, for instance providing global, local, or group feedback.
- CC3** Feedback loops require monitoring and influencing the system state that is commonly realised in accordance with the MAPE paradigm.
- CC4** The development of actual coordination mechanisms is only supported on an infrastructure level and specialised solutions using evolutionary approaches, logic, or predicate programming are having limitations in expressiveness and modelling of the dynamic world.
- CC5** Verification and validation are crucial, but existing solutions that use either logic proofs or simulation have limitations in terms of expressiveness or applicability.
- CC6** Decision-making and planning either is following a HSM-derived approach or targets a goal-orientation.
- CC7** Only the most recent decision-making and planning approaches in the robotics domain are inherently considering multi-robot systems.

In that sense, we still face the following open challenges:

- OC1** Automatically selecting or determining adaptation or organisation mechanisms based on the mission goals.
- OC2** Maintaining complex runtime behaviour in regard to given boundaries and application scope.
- OC3** Integrating and combining adaptation as well as organisation mechanisms with higher-level decision-making and planning components in a goal-oriented fashion.
- OC4** Incorporating explicit task partitioning and distribution as an additional degree of adaptation in a multi-agent system.

14. Analysis Summary: Opportunities and Challenges

In order to foster the application of adaptive multi-robot systems, there are numerous opportunities to combine and to extend existing ideas into an applicable framework, leading to new valuable insights. Particularly, addressing the encountered open challenges will directly enable to answer the formulated research questions regarding the possible facilitation of the operation of multi-robot systems operating in uncertain environments (*Q1*) while exhibiting a robust coordination (*Q2*). In the following Part III, an abstract concept for a framework is presented that incorporates the points analysed in this chapter. The detailed concept of individual components and their implementation will then be presented later in Part IV.

Part III.

Concept

In this part, the abstract concept of the solution presented in this dissertation is developed. For this reason, the next Chapter 15 deduces specific objectives for the approach of developing a new framework for combined self-adaptation, self-organisation, decision-making, and planning. Particularly, the general approach and why it is required to develop such a framework to answer the research questions has been motivated and formulated in Chapter 3. The presented concept incorporates the analysis of the related work that has been done in Part II. Subsequently, in Chapter 16, the methodology for adopting the formulated objectives is argued. Finally, Chapter 17 develops the abstract architecture of the approach, which is then further detailed for the individual components in Part IV.

The foundation of this part is given by the conference paper (Hrabia, 2016) that won the *Best Doctoral Symposium Paper Award* in the joined symposium of the 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO2016) and the International Conference on Cloud and Autonomic Computing (ICCAC 2016). Nevertheless, all chapters of this part have been further elaborated and complemented with the new Chapter 17.

15. Objectives

In Part I, it has been motivated that complex tasks for robots may require the application of multiple robots that perform their tasks within a dynamic and uncertain environment. It is the hypothesis of this work that the particular demand for mobile multi-robot systems, which put emphasis on robustness and adaptivity rather than on behaving in an optimal fashion, can be addressed with the application of self-adaptation and self-organisation. However, as discussed in Chapter 14, existing solutions have many limitations. As we have explained, they are either too much abstracting from particular adaptation methods and are only providing methodologies or architectures, or are very application-specific and not applicable for a broader purpose or limited in their expressiveness to simplify the declaration and validation.

Fostering of more goal-oriented approaches combined with the automatic selection

or determination of required adaptation mechanisms is crucial for the development of adaptive multi-robot systems because it relieves designers and engineers from anticipating in advance all conditions, interactions, and side effects during the development. A particular goal-oriented approach would especially allow to describe the system more intuitively from a top-down application perspective instead of a tedious bottom-up parameter tuning or rule tweaking of manually developed adaptation mechanisms. Nevertheless, such adaptive goal-oriented systems are achieving more autonomy in operation, but still have to operate in specified boundaries in order to prevent malicious or unintended behaviour (Hrabia, Masuch, and Albayrak, 2015).

Even though a general-purpose solution for all kind of multi-agent systems operating in uncertain environments is preferable, it seems not feasible to address all the different underlying conditions in one framework, at least not in one implementation. The reason is that robotics and for instance communication systems have different computation capabilities, required response rates and are running in distinct software ecosystems, e.g. on top of a specific Linux operating system or directly on a microcontroller. In contrast to the existing robotic approaches, which have been analysed in the previous part and that are only addressing specific subdomains like modular robotics, a general application in the domain of mobile robotics is targeted in this dissertation. Widening the scope increases the possible reuse of implemented concepts and algorithms and closes a gap in the research landscape of robotics. The domain of mobile robotics is considered as a particularly challenging domain due to the permanent interaction with the uncertain and complex real-world environment that is sensed through imperfect sensors (Thrun, 2002). Nevertheless, the approach that is developed in the following could also be transferred to other domains like pervasive services, but this would potentially require minor adjustments to address special characteristics of the target domain like incorporating certain runtime requirements.

In the context of task-level robot behaviour control, existing approaches are applying decision-making and planning in order to solve particular tasks. However, the focus is mostly on single robots or centralised coordination of multiple robots. A consideration or integration of self-adaptation or self-organisation into decision-making and planning is missing so far, as we have shown in the state-of-the-art analysis in Chapter 11. This directly links to the formulated research questions *Q1* and *Q2*, which are concerned

15. Objectives

about fostering self-adaptation in task-level decision-making and planning as well as the combination with self-organisation approaches. In order to answer the research questions, it is required to develop and implement a corresponding framework. This also supports the necessary practical evaluation of an implemented solution in order to gain additional insights from tests and analyses in practice. Furthermore, this allows at the same time to address the challenges deduced from the related work analysis, which is summarised in Chapter 14. In the following, we formulate objectives for the approach and relate them to the determined challenges before we add more specific requirements in the next chapter.

First, it is the goal to develop a reusable framework implementation that is easy to integrate into existing applications in order to make the research results transferable as well as to enable others to evaluate the presented approach in their potentially different scenarios. Secondly, self-adaptation capabilities have to be explicitly considered within the decision-making and planning, which directly connects to *Research Question Q1* as well as *OC3*. Here, it is the particular challenge to develop a foundation that allows to integrate different adaptation concepts and algorithms seamlessly. Thirdly, higher-level decision-making needs to be combined with implicit coordination through self-organisation to benefit at the same time from goal-oriented problem solving of complex tasks as well as decentralised and robust coordination, which addresses challenge *OC3* and *Research Question Q2*. Likewise, a goal-driven approach provides a suitable control for the user and allows to respect the defined application scope (*OC2*). Following a goal-driven approach is also offering a foundation for an automated selection of mechanisms as required for *OC1*. Additionally, to foster adjustable autonomy, we have to be able to define outer boundaries for the system and its parameters of freedom, which corresponds to *OC2*. Furthermore, we are evaluating our approach in real-world applications in order to prove that the implementation supports self-adaptation and self-organisation not only on a conceptual level. Moreover, the combination of general decision-making and planning with self-adaptation and self-organisation allows for a complete application perspective from system design to execution instead of focusing solely on very specific self-adaptation and self-organisation algorithms, as it is commonly done in swarm robotics research. Linked to this point is also the necessity of having one common, coherent, and integrated software design to avoid discontinuities induced by,

e.g. switching programming languages, frameworks etc.

In that sense, the following summary of objectives for the particular realisation is deduced to answer the proposed research questions formulated in Chapter 3 as well as the encountered challenges listed in Chapter 14.

- O1** Developing a new reusable task-level decision-making and planning framework for mobile robots in dynamic and uncertain environments.
- O2** Supporting self-adaptation through algorithms and conceptual integration.
- O3** Supporting self-organisation with an automatic selection of suitable mechanisms.
- O4** Enabling the realisation of multi-robot applications including means for task partitioning and distribution.
- O5** Providing capabilities for describing the system goals and boundaries wherein the system is supposed to operate autonomously and adaptively.
- O6** Aiming for coherent, integrated design and implementation of adaptation capabilities within general decision-making and planning.

The next chapter describes the methodology that has been developed to address the aforementioned objectives.

16. Methodology

In this chapter, the methodology is presented that has been developed to address the objectives described in the former section. Especially, it will be discussed on what existing work the new approach builds.

For the purposes of illustrating how the developed concept fosters multi-robot application in practice, a complex scenario as the following is considered.

Multiple Unmanned Aerial Vehicles are used for security protection and surveillance of a large industrial plant. The UAVs have to patrol in the area, cover as much of the area as possible, inspect and follow suspicious objects while maintaining their own safety

16. Methodology

as well as keeping away of certain critical facilities. Moreover, the number of UAV and their particular roles are dynamic because of charging or broken entities. Even more, the connectivity is not always guaranteed. Additionally, it is necessary to optimise the operation planning for individual UAV to cope with the limited capabilities of the UAV in terms of available battery, respectively available flight time.

This scenario is challenging due to the required coordination amongst the UAVs, the dynamic setup with a changing number of robots, several independent and conflicting tasks, and a highly dynamic environment. The coordination could especially benefit from an implicit approach to minimise the dependencies between system entities and enable a seamless adjustment of the number of used robots without struggling with limited connectivity and a single point of failure in a centralised approach.

The described scenario can definitely be implemented application-specific using a standard approach like HSM with static state transitions and a manually tuned self-organisation mechanism for implicitly coordinated surveillance and coverage. However, this is prone to errors and would need to be adjusted bottom-up on every single change, like introducing additional tasks, system goals, or dependencies. The problem is that the HSM approach relies on states and static transitions in between the states, which would have to be adjusted either internally by each state implementation or by an external entity like a human or external system component. HSMs are not allowing for external steering from a top-down perspective from the conceptual point of view. For instance, the already discussed advanced HSM approach of Foukarakis et al., 2014 is actually only separating the decision-layer from the state implementation in a distinct component, which is potentially used for adaptation. However, the decision-layer still has to be implemented by system-designers.

In contrast, an alternative goal-directed solution based on centralised planning would also have problems because of possibly incomplete information during connection loss and the growing state space with an increasing number of UAVs. Furthermore, a planning-based approach cannot take a reasonable decision if currently no plan can be found that directly leads to the goal, which is often the case as the system designer will never be able to describe all possibilities in advance. The solution of this thesis, aiming for a more robust and automatically adapting methodology, is presented in the following. Thus, a resulting system is potentially able to adapt to dynamic changes and

cope robustly with uncertain conditions.

In the first place, a more applicable solution can be supported by a more abstract definition of behaviour runtime conditions to simplify the system modelling and increase the reuse of implemented behaviours. Here, formal methods are beneficial and required in the first stages—this was e.g. demonstrated by Gershenson (2007). Moreover, formal methods such as declarative programming facilitate to define the scope of the systems and to deduce certain guarantees. In consequence, a declarative programming layer allows to model the outer boundaries of the system and scope of adjustable autonomy, which is in line with the *Objective O5*. Following this spirit, a high-level declarative programming layer is used for describing the general behaviour of the system, its outer boundaries, and its goals. Furthermore, this solution has been selected because a declarative implementation allows to determine possible state transition dynamically during runtime and in consequence, makes the system more adaptive, which supports *Objective O2*.

From a user point of view, declarative programming is about defining building blocks of the application and their requirements, this stands in contrast to traditional programming models that also define the particular sequence of execution. To provide appropriate goal pursuance, it is beneficial to combine the declarative system model with a planning system to calculate goal-oriented action sequences instead of following all or randomly selected directions/rules in an undirected forward chaining process, like e.g. in Meld (Ashley-Rollman et al., 2007) and TuCSon (Viroli, Casadei, and Omicini, 2009). This approach directly addresses *Objective O1*. Besides, a planning layer potentially enables managing individual optimisations regarding an optimal utilisation of robot capabilities. Moreover, a planner already increases the ability to adapt to unforeseen states because it allows to combine existing building blocks dynamically during runtime. Additionally, using a goal-oriented approach with a planner on top of a declarative model is also avoiding the requirement of defining comprehensively all negative rules/conditions to prevent that the system turns into malicious states (*O5*), because the system would favour the direction of the goal, even if a condition was missing, instead of exploring edge cases.

In accordance with *Objective O1*, task-level decision-making and planning represents the highest abstraction level for behaviour within a robot and already foster adapta-

16. Methodology

tion through the dynamic selection of behaviours that are most suitable for the given situation. Due to the fact that this thesis aims for further increased adaptation capabilities, we consider task-level decision-making and planning modules as a suitable starting point for the targeted framework. The idea is to incorporate other concepts like specific self-adaptation (*O2*) and self-organisation (*O3*) means in the second step. Furthermore, such planning and decision-making component enables the combination with established methods of designing and developing robot behaviour in order to increase the reuse of existing code supporting the achievement of *Objective O1*. In particular, symbolic planning allows to use a declarative programming style, which only describes available system capabilities and its boundaries, to automatically determine the necessary actions to undertake for achieving the mission of the system, respectively to adapt to the environment under consideration of the given goals. Nevertheless, an important point about using a planner is the monitoring of plan execution and managing potentially necessary re-planning. An alternative one shot planning would not support dynamic adaptation to requirements as well as environmental changes during the runtime.

Since declarative programming is limited in its expressiveness, it is going to be accompanied by a low-level behaviour representation that is able to deal with dynamic environments, temporal constraints, and probabilities. A suitable behaviour representation can be realised with behaviour networks that are goal-driven and enable dynamic state transitions in contrast to static transitions in HSM-based approaches like shown in Chapter 11. Here, especially the dynamic transitions between behaviours that do not have to be defined a priori help to provide a larger degree of freedom for adaptation. Additionally, a combination with the former mentioned symbolic planning in a hybrid approach allows to further improve the goal-directedness in difficult scenarios like dead ends (Lee and Cho, 2014; Nebel and Babovich-Lierler, 2004). Such hybrid implementation makes it possible to build systems with fast responses but which are still pursuing their goals. Moreover, in Chapter 11, it was argued that an integration of self-adaptation and self-organisation capabilities into decision-making and planning can benefit from dynamic state transitions as well as they are relying on a reactive agent model. This can be explained with the reactive nature of self-adaptation and self-organisation algorithms that are based on locally available information. In consequence, decision-making and planning approaches that are incorporating concepts from behaviour networks are an ap-

appropriate foundation for the targeted integration of self-organisation and self-adaptation, which enables accomplishing *Objective O2* and *Objective O3*.

On the one hand, the behaviour network part strongly supports the idea of a self-adaptive robot system on a conceptual level (*O2*) by being

- performing well in dynamic and partially-observable environments under the restriction that actions, which were taken at one point in time, will not block decision paths in the future;
- light-weight in terms of computational complexity, guaranteeing fast responses;
- opportunistic and trying to perform the best-suited action at any time, even if a symbolic planner cannot handle the situation and does not find a suitable plan, in order to reach a different beneficial state that provides more planning options.

On the other hand, the extension with a symbolic planner in a hybrid concept considers critical execution orders to avoid dead ends in the decision path that are difficult to handle with behaviour networks only, while providing additional goal pursuance as already discussed before. Particularly, the long-term consideration of the symbolic planner helps to avoid oscillations and local maxima in the decision path, which could prevent the system from reaching the actual system goal. At the same time, the underlying behaviour network keeps flexibility on the lower level with the use of dynamic state transition and would still be able to take reasonable decisions even though the current situation does not allow to calculate a sequence for reaching the goals. Moreover, the symbolic planning allows to create certain guarantees and boundaries for the underlying adaptation algorithms, which is matching *Objective O5*.

Furthermore, it is the concept to apply a calculated plan for guiding the behaviour network in long-term considerations instead of forcing a strict operation order to increase the responsiveness and adaptivity of the system. This approach is a generalisation of the so-called *operator coupling* of Hertzberg et al., 1998. The external guidance of the behaviour network by a symbolic planner also addresses the general issue of behaviour networks to get stuck in local optima. Behaviour networks can oscillate in local optima if their parameters are not properly tuned towards a specific application since it is difficult to express a required execution order in the behaviour network itself.

16. Methodology

The realisation of symbolic planning is realised on top of standardised planning approaches supporting the common PDDL (Mcdermott et al., 1998). Using PDDL provides a domain independent, expressive, and declarative mission description. PDDL is the most popular planning language and standard in the international planning competition (Vallati et al., 2015). In consequence, various open source planners are freely available. Hence, relying on such standardised description allows to exchange and experiment with different existing symbolic planners. Nevertheless, it is not the intention to use the planner as single one-shot solution calculator, and instead follow the often-applied MAPE paradigm with continuous monitoring and replanning to achieve adaptation to unexpected events, which is in accordance with *CC3*. The MAPE paradigm has proven its potential in various solutions that have been discussed in the analysis of the related work in Part II. This approach of frequent on-demand replanning in an uncertain environment is similar to the RDDDL planner implementation of Yoon, Fern, and Givan (2007), which is also extending a classical PDDL planner with a relational layer to dynamically convert probabilistic problems to several deterministic problems. This underlines the suitability of the presented approach to deal with uncertain environments (*O1*). However, Yoon, Fern, and Givan are entirely focusing on the domains of the International Probabilistic Planning Competition that neglects requirements of the integration into robotics in practice, such as computational limitations. Similarly, other probabilistic planning, such as MDPs, POMDPs, and related languages like PPDDL, are not applied to circumvent the additional computation complexity.

The existing literature, see Chapter 11 about behaviour networks and hybrid behaviour networks, include many ideas and extensions of the initial concepts. Nevertheless, all presented behaviour network variations have certain limitations and are not yet integrated into one solution, which limits their applicability in practice as well as their potential for adaptation. In consequence, it is the goal of this dissertation to combine and further extend these ideas in a common approach to further increase the applicability in dynamic environments (*O1*). A comparison matrix of features of different discussed and relevant approaches is shown in Table 16.1, which is a subset of what has been presented and discussed in Table 14.2 in Chapter 14. The contained approaches are selected because they are following a goal-oriented approach, which is the case for all discussed symbolic-planning, behaviour network and hybrid behaviour approaches.

Like already mentioned in Chapter 14 with *OC3*, extending the existing concepts in the direction of improved learning capabilities and implicit coordination mechanisms for multi-robot systems that can be used in a goal-directed fashion are open issues. In the same respect, introducing additional hierarchy levels into behaviour network, symbolic planning, and hybrid behaviour planning approaches reduces problems of centralised management and control. Moreover, improving adaptability on the task-level in a multi-robot environment can benefit from the consideration of further aspects of multi-robot decision-making such as task decomposition, allocation, and delegation, which is in line with *Objective O4*.

Table 16.1.: Comparison of symbolic planning frameworks, behaviour networks and hybrid planning approaches.
[Y=Yes, N=No, P=Partially, M=Manually]

Reference [Solution]	Behaviour Network			Symbolic Planning			General			
	Included	Learning	Hierarchies	Generating plan descriptions	Included	Standard Planner	Non-binary Preconditions	Parallel Behaviour Execution	ROS Integration	Multi-Robot Support
Maes (1989) [MASM]	Y	N	N	N	N	N	N	N	N	N
Decugis and Ferber (1998)	Y	P	Y	N	N	N	N	N	N	N
Jung (1998) [ABBA]	Y	Y	N	N	N	N	N	N	N	M
Hertzberg et al. (1998)	Y	N	N	N	Y	ADL	Y	N	N	N
Allgeuer and Behnke (2013)	Y	N	N	N	N	N	Y	Y	Y	N
Lee and Cho (2014)	Y	N	N	N	Y	N	N	N	N	N
Cashmore et al. (2015) [ROSPan]	N	N	N	P	Y	PDDL	Y	N	Y	P
Rovida et al. (2017) [SkiROS]	N	N	N	P	Y	PDDL	Y	N	Y	Y

In order to realise implicit coordination by applying self-organisation algorithms in a goal-oriented fashion (*O3* and *OC1*) on top of the hybrid behaviour network, it is beneficial to avoid the usage of distinct implementation approaches, e.g. relying on different

16. Methodology

programming languages, environments, or domain models. For this reason, as well as to address *Objective O6*, we are aiming for a coherent implementation of self-organisation within the behaviour model to avoid discontinuities. In this respect, the particular mechanism implementation itself is also considered consisting of one or multiple behaviours within a behaviour network. Furthermore, a coherently used domain model for general task-level decision-making as well as self-organisation mechanism realisation is implemented in an object-oriented manner in Python, which fosters reuse of code, simplifies user extensions as well as customisations, which is again supporting *Objective O1*. Likewise, using Python enables a vast flexibility in terms of the programming paradigm used by the system user at the end (Koepke, 2011).

Furthermore, it is crucial to support the determination of suitable self-organisation mechanism and configurations in a goal-oriented manner (*O3*) to decouple the potentially reusable implementations from the particular scenario (*CC1*). Hence, the definition of the desired implicit coordination strategy is expressed by the system designer by providing additional self-organisation goals. An appropriate configuration of behaviour sets for a provided self-organisation goal could be determined automatically through metaheuristic, like evolutionary algorithms, reinforcement learning, and simulation, like it is applied in evolutionary robotics, see also Section 10.1. Nevertheless, it is the objective to narrow this by automatically applying expert knowledge about the application and combination of existing mechanisms as presented in (De Wolf and Holvoet, 2007; Fernandez-Marquez et al., 2012) and following the idea of a hypothesis database from Edmonds, 2005. Furthermore, collected mechanisms are modelled in a declarative fashion following the MASDynamics approach of Sudeikat and Renz, 2009. Moreover, applying an initial setup selected from such an expert knowledge library has the advantage of avoiding unstable initial states, which is crucial for real-life applications. Such an initial setup is then further refined by a learning component, which adjusts the behaviour and mechanisms through reinforcement in order to deal with new situations in accordance with the goal description.

Driven by our background and our particular interest in multi-robot systems, it is necessary to equip the developed framework with capabilities to integrate with existing infrastructures as well as with a convenient abstraction from common sensor and actuator interfaces. For this reason, the integration into a widespread framework, like the

Robot Operating System (Quigley et al., 2009) is a major requirement for the presented approach and also fosters the reusability of the entire approach (*O1*). Furthermore, taking advantage of an existing ecosystem increases the interoperability with other robotic software components. Likewise, building on top of ROS also allows to create a solution that can be evaluated in practical scenarios and adapted to existing applications in the domain of mobile robots. An alternative solution is to utilise common not domain-dependent standards like web-services to increase applicability and interoperability as proposed by Mokarizadeh et al., 2009. We neglected this approach in order to minimise the communication overhead, due to the fact that ROS is currently the de facto standard robotics middleware. Furthermore, it is possible to achieve a further increased interoperability by an automated generation of web interfaces from ROS communication means as we have shown in the EffFeu project (Hrabia et al., 2019).

In summary, this dissertation has following detailed objectives for the realisation of the hybrid planning approach consisting of a behaviour network and symbolic planner.

- R1** Integrate various behaviour network concepts into one extended solution.
- R2** Extending the hybrid behaviour network concept with more advanced learning capabilities.
- R3** Allowing for multiple hierarchy, respectively abstraction levels in the decision-making.
- R4** Integrating implicit coordination through self-organisation in a goal-oriented manner.
- R5** Enabling an automated selection of self-organisation mechanism based on expert knowledge.
- R6** Explicitly addressing automated task decomposition, allocation and delegation.
- R7** Using the standardised planning interface PDDL.
- R8** Implementing one coherent object-oriented domain model.
- R9** Applying the ROS middleware and Python programming language for the realisation.

Aside from the connection to the objectives that have been formulated in the last chapter, we also want to relate the requirements directly to the research questions of this dissertation. In detail, *R1*, *R2*, and *R3* are directly required to answer *Research Question Q1* because this provides the foundation for a decision-making and planning system with a focus on adaptation. Likewise, *R4* and *R5* are necessary to realise a goal-oriented integration of self-organisation for robust coordination within the framework as required to address *Research Question Q2*. Here, *R6* further extends *R4* and *R5* to provide additional means of adaptation in the context of multi-robot systems, which supports the approach examined in *Q2*. Finally, *R7*, *R8*, and *R9* are additional requirements that foster the evaluation of the system in practice.

In the next chapter, follows the description of the high-level component architecture deduced from the methodology presented in this chapter.

17. Architecture

From a task-level perspective, robotic software systems architectures consist of different abstraction layers. A three to four-tiered architecture has been proven a reasonable approach to realise single and multi-robot systems with decision-making and planning capabilities (Alami et al., 1998; Gat, 1998; Kramer and Magee, 2007; Rovida and Krüger, 2015; Simmons and Apfelbaum, 1998). All mentioned approaches share similar abstraction layers. They have a low-level software layer that directly interacts with the physical world by controlling actuators and collecting sensory information. This layer takes care of real-time control and provides atomic behaviours respectively primitives of a robot; it is often named control layer (Gat, 1998) or behaviour layer (Simmons and Apfelbaum, 1998). Sometimes this layer is again separated into two layers, resulting in four layers overall, with a specific layer abstracting from the hardware devices as in (Rovida and Krüger, 2015). On the highest level of the architecture, it is defined on an abstract level how system goals can be achieved; this layer can be specified as planning layer (Simmons and Apfelbaum, 1998) or deliberation layer (Gat, 1998). A layer in between is responsible to mediate between the high abstraction level with discrete states and the

continuous level of the primitives of the lower layers. Such an executive layer (Simmons and Apfelbaum, 1998) or decision-making layer (Gat, 1998) translates the abstract tasks into particular atomic commands, executes the commands, and monitors their execution. A summary of these best-practice architectural approaches is depicted in Figure 17.1.

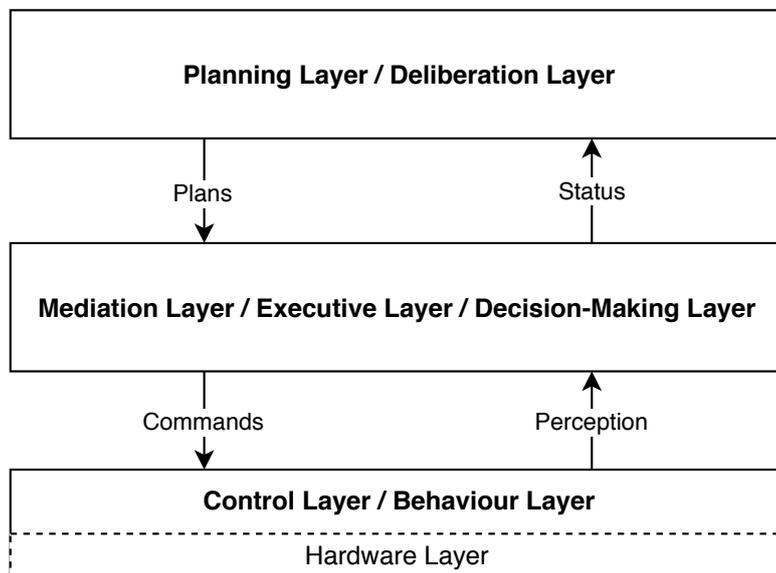


Figure 17.1.: Combined three to four tiered architecture for realising robotic systems summarising (Alami et al., 1998; Gat, 1998; Kramer and Magee, 2007; Rovida and Krüger, 2015; Simmons and Apfelbaum, 1998).

The following presented architecture builds on the established abstraction levels that foster the reuse of existing modules and separation of concerns. Notably, the presented solution incorporates a planning layer, mediation layer, and behaviour layer. The architecture for a single agent is depicted in Figure 17.2.

The most important element of the architecture is the *core* component. This *core* component provides the foundation for the entire framework and implements the actual decision-making and planning with the common abstractions of a planning layer, mediation layer and behaviour layer. These abstractions are as well corresponding with the components required for the realisation of a hybrid behaviour network approach as considered by *R1*. In our case the planning layer is represented by a *Symbolic Planner*, the mediation layer is formed by a manager component a *Reinforcement Learner* component (*R2*), and a *Delegation* component, while the behaviour layer is composed by the behaviour network with included behaviour implementations. The symbolic planner is

17. Architecture

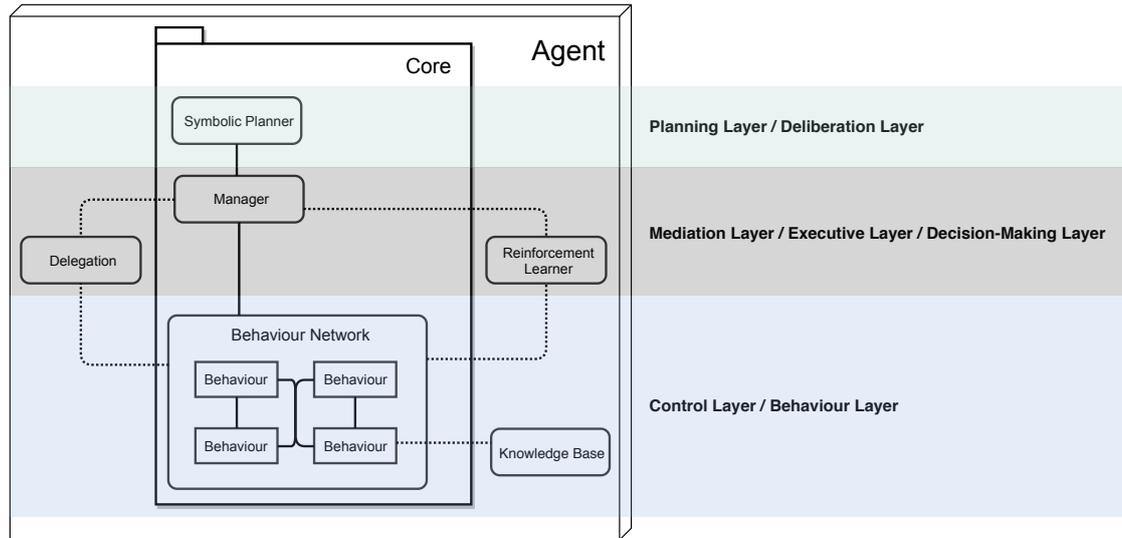


Figure 17.2.: System architecture and architecture layers of a single agent.

realised on top of the standardised PDDL interface ($R7$). The behaviour (network) layer consists of several behaviours and additional optional components such as the *Knowledge Base*. Here, a *behaviour* is implemented by the system engineer on the foundation of a provided base class that takes care of the overall integration. A behaviour itself corresponds to an atomic action, task, or operation an agent is capable of executing that has an influence on the real or virtual environment. Due to the fact that the granularity of a behaviour is not predefined, the expressiveness is not limited. In consequence, a concrete behaviour implementation can range from setting a certain virtual variable to something more complex as picking up a bottle from a fridge, which would include own specific controls like dedicated motion planning. The role of the manager component is inspired by the concept of the mediator as proposed by Gershenson, 2007, considering it as an adaptation controller in respect to the given goals. A minimal agent realisation could be implemented only with the *core* and its included components.

The multi-robot architecture also incorporating hierarchies and self-organisation aspects is further explained below based on the overview diagram visualised in Figure 17.3. The architecture provides a mixed strategy of centralised high-level planning to achieve the application goals with decentralised robot agents that are autonomously operating within their given boundaries.

The architecture is widely modular and allows to create complex hierarchies of agents

and task abstraction levels in a multi-robot setup ($R3$). In consequence, an individual behaviour of a behaviour network may consist as well of another *core* with own lower-level behaviours. Such a nested core enables the creation of static task decompositions similar to hierarchical states in a HSM or task trees. Moreover, entire agents can be represented as behaviours in a higher-level agent to implement more centralised multi-agent decision-making, which allows to resemble a hybrid architecture similar to the approach of Lee and Cho, 2014.

The *Knowledge Base* component is responsible for hosting, sharing, and persistence of arbitrary information, and it can be used on all abstraction levels. Moreover, multiple distributed instances can be used to increase the flexibility and robustness.

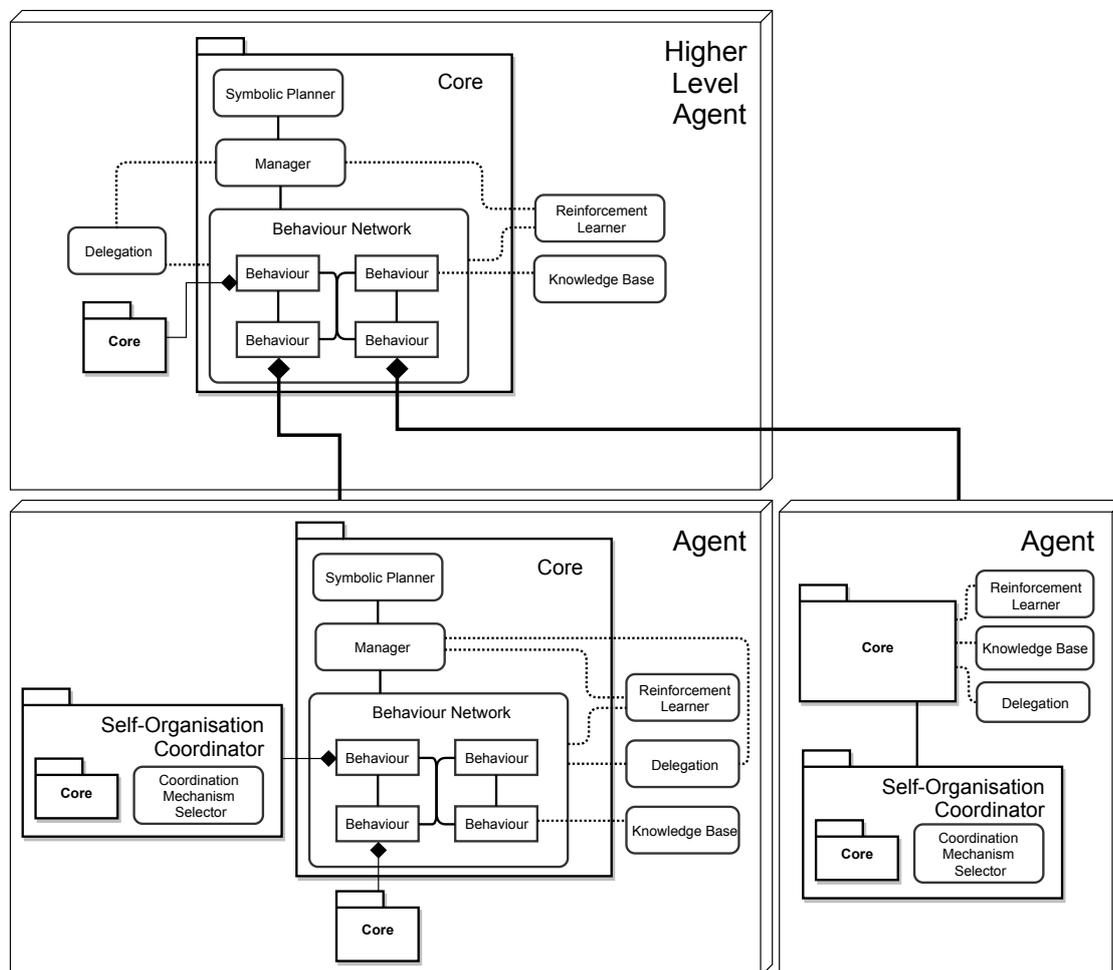


Figure 17.3.: System architecture with multiple agents on different hierarchy/abstraction levels.

17. Architecture

Each *core* executes an own independent decision-making and planning cycle that is enabled if the component is activated in case of nested *core* components. Therefore, robots remain independent of the higher-level components and are still able to operate in case of interrupted or unavailable communication. Higher-level agents provide initially or from time to time sub-goals to lower level agents. The different agents, as well as levels of hierarchy, are communicating through the ROS communication middleware (RQ9). Furthermore, the manager component contained in each core is also continuously monitoring and adjusting the currently targeted goals in a separated feedback loop. Particularly, the manager tries to achieve as many goals as possible, considering their individual priorities. Details about the realised algorithm for the goal selection are later given in Section 18.2. In consequence, the presented architecture contains at least three levels of feedback loops for adaptation if we do not have a nested agent architecture, which would result in additional feedback loops for each agent (*CC2*). The highest level is given by the just mentioned goal monitoring and selection in each manager component. The second level is the continuously executed decision-making and planning cycle in each agent that is based on the behaviour-network model. Additionally, the third level is the low-level control loop potentially implemented in a concrete behaviour implementation. This can also be related to the reference architecture for engineering adaptive software DYNAMICO (Villegas et al., 2013). Here, our goal monitoring and selection corresponds to the control objectives feedback loop (CO-FL) of DYNAMICO. Subsequently, the decision-making and planning feedback loop resembles the adaptation feedback loop (A-FL). However, due to DYNAMICO being an abstract software engineering architecture it is not considering the lowest level of adaptation, which is often required to control hardware actuators, taking place in the feedback loop of a particular behaviour as we have it in our proposed architecture. Furthermore, our presented architecture is neglecting the monitoring feedback loop (M-FL) of DYNAMICO, which is supposed to adapt the system with respect to changed context requirements. The reason is that in the particularly focused scenarios of this dissertation, which considers a concrete set of robots collaboratively working on a mission, external context adaptation would correspond to things like automatically deploying different or additional robots as well as complementary infrastructure. Usually, such context changes would require in practice human intervention and therefore are out of the scope of this dissertation.

Nevertheless, if a comparable adaptation on the context level as formulated in DYNAMICO is desired, it would be possible to implement this as a special high-level agent in a multi-level architecture exploiting the modularity of our approach supporting several nested decision-making levels.

An important point is that the *core* components of the higher-level agent are only providing the outer boundary with an initial configuration, including constraints and sub-goals for the operation of the lower-level agent. Outer boundary describes the scope of the operation for the adjustable autonomy. The particular configuration could also be dynamically updated from time to time, but this is not mandatory. An update could e.g. modify conditions as a result of a behaviour execution. In general, the architecture allows to create mixed implementations of decentralised and more centralised decision-making and planning, which allows the system designer to control the level of dependency between the agents. For example, a more centralised design fosters planning for global optimality while a more decentralised design increases the robustness against failures of individuals. The promising direction of such a mixed strategy is also mentioned as future research direction in (Yu and Nagpal, 2011).

Since the architecture is targeting multi-robot applications, it requires coordination amongst the robots to achieve certain group level behaviour. Coordination between agents can be achieved either by using the *Delegation* component to decompose, assign, and delegate tasks to other agents explicitly (*R6*) or by applying the *Self-Organisation Coordinator* to realise implicit coordination based on self-organisation algorithms (*R4*). The *Self-Organisation Coordinator* automatically selects appropriate self-organisation mechanisms dependent on the current goals of an agent (*R5*). Particularly, the *core* is also controlling the *Self-Organisation Coordinator* based on its higher-level goals (*R4*). The latter component is responsible for the determination of an appropriate coordination mechanism for the multi-robot system that is supporting the intended goals. Here, the specific mechanisms are again realised with a *core* component within the *Self-Organisation Coordinator*. Reusing the same concepts for decision-making, planning, and control supports especially *O6* and *R8*. Based on the selected mechanism, an appropriate nested *core* component is instantiated and configured.

If either implicit or explicit coordination or even a combination of both is used, can be freely decided by a system designer. Both approaches have advantages and dis-

17. Architecture

advantages: Implicit coordination based on self-organisation is highly distributed and therefore very robust against failures of individuals. Moreover, it can also be scaled up easily to a large number of agents. Explicit coordination with the *Delegation* component is especially useful for complex goals that consist of several other sub-goals and the distributed processing of dependent tasks across multiple agents. However, combining both approaches in a single robotics framework is inherently unique and provides maximum flexibility and adaptation capabilities to the particular application scenario.

In general, the behaviour network of an individual robot allows for an integration of lower-level behaviour actions, which have been developed in a bottom-up approach, into the global system context that is controlled top-down using goals and boundary conditions (*O6*). Thus, the presented approach brings together the global view on multi-robot applications with the locally available behaviours and mechanisms.

The manager component, which is instantiated at various places of the architecture, is composing a behaviour network using the information provided by behaviours and goals, like preconditions and effects. Moreover, the manager is applying the MAPE paradigm (*CC3*). In consequence, it is monitoring and supervising the plan execution by interpreting the deduced plan and influencing the behaviours accordingly.

The *Reinforcement Learner* component allows to include the experience of the agent into the decision-making and planning (*R2*). It is an optional component that can be used by the manager to learn from past decisions that have been taken based on the symbolic planner and the behaviour network. The *Reinforcement Learner* applies Reinforcement Learning algorithms to learn over time in order to be able to adapt to the dynamic environment. The component implementation itself is strongly decoupled from the hybrid behaviour planner domain that is formed by the symbolic planner, manager, and behaviour network. Here, a general interface that allows to insert experience over time and get out learned decisions based on rewards enables the reuse of this component also for other modules of the framework, or to easily replace the learning architecture with an alternative e.g. domain-specific implementation.

The separated *Self-Organisation Coordinator* is inspired by many existing approaches that are using a comparable separation of concerns following *CC1* (Garlan et al., 2004; Preisler, Vilenica, and Renz, 2013; Sudeikat et al., 2009). This component is responsible for the integration of self-organisation coordination mechanisms, while the self-

adaptation is realised within the *core* itself.

The *Coordination Mechanism Selector* component as part of the *Self-Organisation Coordinator* allows for an automatic selection of mechanisms appropriate for a given goal (*R5*). This selection is based on expert knowledge, experience, and existing design patterns. In particular, it is the concept to implement the pattern collection in a reusable fashion that has been proposed by Fernandez-Marquez et al., 2012. The challenge here is to find an appropriate generalisation from expert knowledge and the mapping to goals based on the properties of self-organisation patterns. Especially, the expert knowledge allows for an initial structure on how specific mechanisms are related to each other and how they can be classified. The assessment of specific configuration setups allows to create a hypothesis database following the idea of Edmonds, 2005. Additionally, it is also considered to further extend this approach with experience-based learning as proposed from other researchers, too (Beal and Bachrach, 2006; Branke et al., 2006; Matarić, 1995). Especially, the *Reinforcement Learner* could be applied here due to its generic interface to tune the existing expert knowledge over time using experience.

All together, the presented approach is allowing for a mixed top-down and bottom-up strategy in the application development. While basic behaviours for individual agents are developed in a traditional bottom-up manner, the actual application including a suitable combination and configuration of behaviours is going to be determined automatically by the presented system with respect to the application goals and boundaries (*O5*).

Getting back to the initially presented example scenario. This solution would allow the developer to realise the required multi-robot system by providing the high-level goals (e.g. patrol and cover environment, follow suspicious objects) and boundaries (e.g. avoiding critical facilities) as well as implementing low-level behaviours (e.g. fly to, follow object) with their corresponding preconditions (e.g. distance to team-mate, battery level) and expected effects. This initial configuration is then further tuned automatically and can be dynamically adjusted after deployment by incorporating other conditions and behaviours or adding and removing robot agents on demand.

In the next part, the particular implementations, concepts, and algorithms of the individual components of this architecture are going to be explained in more detail. Due to the reason that some later conceptual extension like the *Coordination Mechanism Selector*, the *Reinforcement Learner* and the *Delegation* component require implementation

17. Architecture

backgrounds about the *core*, corresponding conceptual considerations are presented in detail in the following part instead of being outlined in this part of the dissertation.

Part IV.

**Detailed Concepts and
Implementation**

In the following chapters, the components of the framework architecture, which have been introduced in Chapter 17, are developed and explained in more detail. Particular chapters, corresponding to components of the framework, are covering both the detailed concept and insights about the implementation.

The entire framework received the name *ROS Hybrid Behaviour Planner (RHBP)*, in consequence, this name will be used in the following to refer to the approach presented and developed in this dissertation. The name indicates the strong relationship to the ROS framework, even though the overall conceptual approach is independent and general applicable, many parts of the implementation are very tightly integrated into ROS in order to improve the applicability in the corresponding popular robotics ecosystem (*R9*). In detail, the name component *Hybrid Planner* refers to the complete ensemble of a symbolic planner and behaviour network that forms the core of the implementation as introduced in the architecture, see Chapter 17. In detail, this *core* component of the framework, the hybrid behaviour planner, is presented in the following Chapter 18 and addresses especially the requirements *R1*, *R3*, *R7*, *R8*, and *R9*. This component provides the foundation for all following chapters that build on top of the *core*. Subsequently, Chapter 19 describes the extension *Self-Organisation Coordinator* that enables the incorporation of self-organisation mechanisms into decision-making and planning in order to realise *R4* and *R5* while at the same time it applies the already existing features of the *core* to realise the mechanisms themselves (*R8* and *O6*). In Chapter 20, the component *Reinforcement Learner* is elaborated in more detail, which extends the hybrid behaviour planning approach with advanced learning capabilities (*R2*). This component allows to increase the self-adaptation capabilities of the system by applying machine learning to shape future decisions based on experience. Finally, Chapter 21 is about the RHBP extension that allows to apply automated delegation of tasks including decomposition and allocation of tasks within a group of multiple agents (*R6*). The increased decoupling of task delegations in hierarchical decision models introduced by the last extension allows to further increase the flexibility and adaptation means in multi-robot systems.

The three main contributions of this part presented in Chapter 18, Chapter 19, and Chapter 20 are based on the conference papers (Hrabia, Wypler, and Albayrak, 2017), (Hrabia, Lehmann, and Albayrak, 2019), and (Hrabia, Kaiser, and Albayrak, 2017). However, all chapters have been revised, updated, and extended to reflect the latest

development stage of the RHBP framework.

18. Hybrid Behaviour Planner Core

In the discussion of the abstract architecture of RHBP in Chapter 17, it has been stated that the decision-making and planning is the most essential part of the architecture. In particular, decision-making and planning are represented by the hybrid approach explained further in this Chapter. The reason for this being the *Core* of the architecture is that this part is required to implement a minimal agent within RHBP. Furthermore, this part comprises all the presented common abstraction layers of a robotics architecture, namely a planning layer, a mediation layer, and a behaviour layer. In detail, the foundation is provided by the behaviour layer of our hybrid approach that is formed by a behaviour-network. The model behind the behaviour-network is introduced in the following Section 18.1. In an application, this model provides the means for implementing the scenario-specific behaviour layer in a declarative fashion. This behaviour layer is then connected with the planning layer, which is based on a symbolic planner, see Section 18.2, through the mediation layer. The mediation layer itself is represented by the manager component, which is taking care of the overall decision-making lifecycle. The manager component implements the algorithms outlined in Section 18.1.2.

18.1. Behaviour-Network Base

A behaviour-network-based approach has been selected for various reasons that we have already discussed in the overall concept in Part III, such as being fast, goal-oriented, opportunistic, and independent of statically defined state transitions. Especially, the characteristic of enabling a declarative description of a system while focusing on its capabilities and limitations without defining the actual solution provides a lot of potential for adaptation. The following Subsection 18.1.1 concentrates on the developed model and abstract components that are later on used as a base by the system engineer to realise the particular implementation. In contrast to traditional sequential programming

18. Hybrid Behaviour Planner Core

techniques, the presented approach enables to separate the implementation of what the system is able to do and how the system is going to use its features to address a certain goal. This becomes especially interesting as the how is automatically deduced and does not have to be specified by the system engineer.

18.1.1. Model

In Chapter 16, a comparison of symbolic planning frameworks, behaviour networks, and hybrid planning approaches has been shown, see especially Table 16.1 for an overview. The target of this work is to incorporate the advantages of the different existing ideas, which have not yet been combined, as well as additional extensions in one common behaviour network layer. All details of the implemented model and its components are presented below.

The behaviour network layer is mostly inspired by the concepts of Jung, 1998 and Maes, 1989, but incorporates other recent ideas from Allgeuer and Behnke, 2013, in particular supporting concurrent behaviour execution, non-binary preconditions and effects. Furthermore, new ideas for enabling hierarchical structures and sharing knowledge are explained in specific sections later in this chapter.

The main components of the network are behaviours. Behaviours are representing competences of an agent (Maes, 1989) and are also referred to as tasks, actions, or skills (Rovida and Krüger, 2015). Behaviours are able to interact with the environment by sensing and acting. The behaviour itself is an abstract model that can be freely modelled by the system designer, important is that behaviours have an influence on their environment. On which logical abstraction level a behaviour is influencing the environment depends on the concrete application scenario and the abstraction level the system designer wants to realise.

Behaviours use *condition* objects composed of an *activator* and *sensors* to model their environmental runtime requirements, see Figure 18.1. Simple condition objects are just container objects that combine activator and sensors. Furthermore, the condition classes *Conjunction*, *Disjunction*, and *Negation* allow for combining multiple condition objects in order to define arbitrary complex logical expressions. Details about *sensors* and *activators* are provided further below.

Behaviours also have a priority property that describes the importance of a behaviour

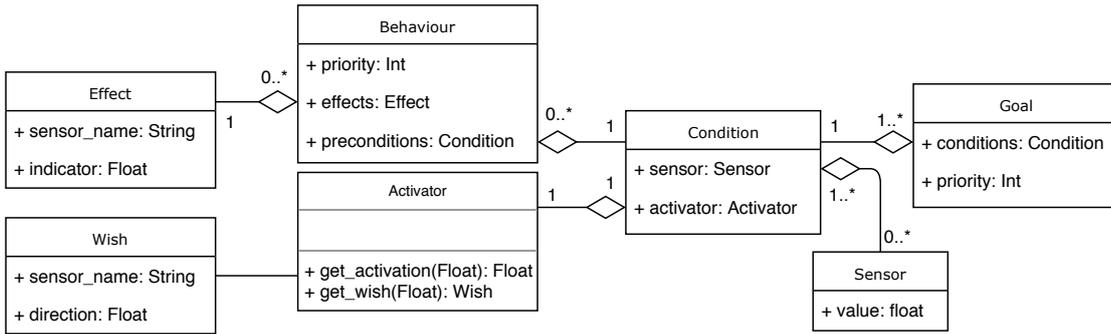


Figure 18.1.: Behaviour network components and their relationship as UML class diagram.

in comparison to other behaviours, especially similar or conflicting behaviours. The priority is applied to resolve conflicts in case of opposing effects. In case of a conflict of behaviours with the same priority, the one with a higher activation value is prioritised. The actual network of behaviours is created from the dependencies encoded in wishes based on modelled preconditions and behaviour effects.

A *wish* of a behaviour expresses the satisfaction with the world state (current sensor values). It is related to a sensor and uses a real value $[-1, 1]$ to indicate both the strength and direction of a desired change, 0 indicates complete satisfaction. Greater values express a stronger desire, by convention, negative values correspond to a decrease, positive values to an increase. In the case of Boolean values -1 is used if a predicate should become false and 1 if it should become true. Wishes are the consequence of every behaviours' or goals' conditions, or in other words, they describe the difference between the current world state and what would make a condition become true. However, wishes are internal objects that are used in various calculations but normally not exposed to the user of the framework.

Effects model the expected influence on available sensors (the environment) of behaviours similar to wishes. The effect is defined by a tuple of sensor name and indicator. The indicator describes similarly to wishes both the strength and direction of the assumed sensor change. In contrast to wishes, effects are not bound to the value range of $[-1, 1]$. Instead, effects can have arbitrary real values. This can be used to specify the influence on the sensor values more fine-grained. Such specification makes the application of the planner within the hybrid behaviour network more powerful because it

18. Hybrid Behaviour Planner Core

allows to determine in detail how often a behaviour has to be executed to reach a certain stage instead of just giving the information that it has influence in a certain direction. An example would be a charging behaviour that has to be executed five times, which is determined based on the charging rate and the desired battery level. However, in many scenarios, it is also sufficient to use only the value range of $[-1, 1]$ to indicate the direction of change, especially in the case when the specific influence is difficult to express or unknown. Moreover, the system is re-evaluating its decisions frequently, which allows for quick adaptation, hence the direction of change influenced by the next decision (selected behaviour) is often more critical.

Goals are defined by the user or any other external component and describe the desired state of the system. In contrast to wishes, they describe the desired absolute state instead of the required change to reach the desired world state. The desired state modelled with goals is as well defined with condition objects. The implementation of goals is similar to behaviours except that they do not have an execution state or model effects on the network, respectively sensors of the network. Therefore, goals incorporate conditions that allow for the determination of their satisfaction and express wishes exactly like behaviours do. Furthermore, goals are either permanent and remain active in the system as maintenance goals, or are achievement goals that are deactivated after one-time fulfilment.

Sensors model the source of information for the behaviour network, Jung, 1998 named them feature detectors. Particularly, they buffer and provide the latest sensor measurements. The type of the sensor value is arbitrary, but to form a valid condition, a matching combination of sensors and activator must be used. Besides mapping raw sensor devices to the behaviour network, other system internal values, or intermediate computation steps can be modelled and distributed, too. A common approach is to aggregate and normalise sensor values within the sensor instance to a single dimensional real value representation that can most easily be combined with default activator types afterwards. Sensor values of arbitrary type are mapped into the behaviour network by *activators*.

Activators provide heuristical functions that allow to express a desired condition and to apply a sensor value for the decision-making. For this reason, activators compute a utility score (precondition satisfaction) from sensor values using an internal mapping function, see next section for more details.

The separation of sensor and activator fosters the reuse of code and allows for the abstract integration of algorithms using more complex activation functions like used in Section 19.1 for the incorporation of self-organisation algorithms. Moreover, the implementation already comes with several basic activator types for expressing Boolean, numeric equality, string equality, numeric thresholds, and numeric linear mappings of one-dimensional sensors. In detail, the Boolean activator compares a Boolean value to a desired value of true/false. Similarly, the equal activator checks if a numeric sensor value is equal to a given number and the string activator compares equality against a given reference string. The numeric threshold activator compares a numeric sensor value to a threshold; here the activation can either depend on the value being equal, larger, or lower than the given threshold. In addition, the greedy activator maximises or minimises the current sensor value towards a given offset. In consequence, the greedy activator is never satisfied and always directing into the given direction through the wishes that are generated on its foundation. The linear activator takes a numeric value and computes a linear slope of activation within a given value range. Similar activators for other mathematical functions such as logarithm, sigmoid, and exponential functions can easily be implemented in the same way.

Multi-dimensional types can be integrated either by custom activators that provide a normalisation function, if the normalisation is not already handled in the sensor implementation, or by splitting dimensions into multiple one-dimensional sensors. In most cases, a normalisation within a custom sensor implementation is preferable as this increases the possible reuse of code due to a clear separation of concerns, which simplifies the application of existing activators.

18.1.2. Decision-Making

The behaviour network is used to select the most suitable action at a time. The taken decision is re-evaluated over time as part of the lifecycle of the manager component, see Chapter 17 for details. The realised core lifecycle corresponds to the common MAPE loop (monitor, analyse, plan, execute). Here, the *monitoring* is covered by the sensor components that provide the input for the system and the *execution* is realised by the specific behaviour implementation of the selected and executed behaviour. Whereas the actual decision-making is accomplished with the behaviour network implementation that

18. Hybrid Behaviour Planner Core

resembles the stages *analyse* and *plan* of MAPE.

The key characteristics and capabilities of a behaviour network are defined by the way activation is computed from sensor readings and the behaviour/goal interaction. Behaviours are selected for execution based on a utility function that determines a real number behaviour score, called activation. There are multiple sources of activation, with negative values corresponding to inhibition. If the total activation of behaviours passes the execution threshold and all preconditions are fulfilled, the planner selects behaviours for execution. The behaviour network calculation is repeated in a fixed frequency that can be adjusted according to the application requirements. A suitable frequency should be chosen based on the desired response time for behaviour changes. Furthermore, it is limited by the update frequency of the attached sensors because it would not make sense to recalculate the decision-making cycle if the perceived world state has not changed.

The positive execution threshold is coupled to the activity of the network and defines the required activation of a behaviour to be selected for execution. It is decreased after every iteration without a running behaviour by the *Activation-Threshold-Decay* parameter (by default 20%) and increased by the same amount every time a behaviour is started in order to support the desired opportunistic habits. Moreover, this approach supports the continued execution of already executed behaviours but also enables a fast adaptation in case the running behaviours are not applicable anymore.

The formulas discussed in the remainder of this section are making use of the following definitions and symbols.

C = precondition; B = behaviour; P = plan; G = goal

a_{last} = last total network activation; s = satisfaction

a = activation; w = wish; e = effect; β = weight parameter

n = number of elements sharing a property

i_B = behaviour index in plan sequence

i_P = current plan element index

p_{goal} = goal priority weight

p_{range} = goal priority range

At every decision-making iteration, all 7 activation sources (18.1 to 18.9) are summed to a temporary value called activation step for every behaviour. After the activation step has been computed for every behaviour, it is added to the current activation of the behaviour reduced by an activation decay factor of 0.9 by default. The decay reduces the activation that had been accumulated over time if the behaviour does not fit the situation anymore and prevents the activation value from becoming indefinitely large. Next, we describe the individual activation sources one by one.

The fulfilment of *preconditions*, modelled as a combination of sensors and an activator, is called satisfaction (real $[0,1]$), see Equation 18.1. The overall behaviour satisfaction is the product of all precondition satisfactions.

$$a_{prec} = \prod_{C|C \in B} s \cdot \beta_{prec} \quad (18.1)$$

A *predecessor* (Equation 18.2) is a behaviour that fulfils a wish of another behaviour. If the product of the effect of behaviour A and wish of behaviour B for a particular sensor is greater than 0 then A is a predecessor of B . A behaviour gets activation from all its executable (preconditions fulfilled) predecessors for every wish.

$$a_{pred} = \sum_{B|B \in pred} \frac{e \cdot w \cdot a_B \cdot \beta_{pred}}{a_{last} \cdot n_w} \quad (18.2)$$

The *successor* (Equation 18.3) describes the reverse relationship to a predecessor. If the (pre-) conditions of behaviour or goal B are fulfilled by behaviour A , then B is the successor of A .

$$a_{succ} = \sum_{B|B \in succ} \frac{e \cdot w \cdot a_B \cdot \beta_{succ}}{a_{last} \cdot n_e} \quad (18.3)$$

If the product of the effect of behaviour A on one sensor and the wish of behaviour B for the same sensor is smaller than 0 then the behaviour A is a *conflictor* of B , see Equation 18.4. Behaviour A is as well a *conflictor* if it has an effect on a wish

18. Hybrid Behaviour Planner Core

with an indicator value of 0 of another behaviour B because it would undo a satisfied precondition of B .

$$a_{conf} = \sum_{B|B \in conf} \left(-1 \cdot \frac{(1 - |w|) \cdot |e| \cdot a_B \cdot \beta_{conf}}{a_{last} \cdot n_{conf}} \right) \quad (18.4)$$

The activation by goal a_{goal} is defined similarly to the activation from successors a_{succ} except that not the indicator of wishes of successors but those of goals are considered. Additionally, instead of factorising with the activation of the particular behaviour the factorisation uses the normalised goal priority (Equation 18.6), and that another weight term goal weight β_{goal} is used instead of the successor weight term β_{succ} , see Equation 18.7.

$$p_{range} = \max(G_{Prio}) - \min(G_{Prio}) \quad (18.5)$$

$$p_{goal} = \begin{cases} \frac{G_{Prio}}{p_{range}} + \left(2 - \frac{\max(G_{Prio})}{p_{range}}\right) & \text{if } p_{range} \neq 0 \\ 1 & \text{otherwise} \end{cases} \quad (18.6)$$

$$a_{goal} = \sum_{G|G \in succ} \frac{e \cdot w \cdot p_{goal} \cdot \beta_{goal}}{a_{last} \cdot n_w} \quad (18.7)$$

Inhibition by a goal is expressed with Equation 18.8 to model undo or counteracting of maintenance goals. The weight term influences the characteristic of the system to strive after goals. Additionally, both goal related activation calculations (Equation 18.7 and 18.8) are weighted by a normalised priority influence of the considered goal. The goal priority weight p_{goal} is calculated as a relative fraction in the range $[1, 2]$ that depends on the currently available goal priority range.

$$a_{confgoal} = \sum_{G|G \in conf} \left(-1 \cdot \frac{(1 - |w|) \cdot |e| \cdot p_{goal} \cdot \beta_{conf}}{a_{last} \cdot n_{conf}} \right) \quad (18.8)$$

The behaviour network is influenced by the symbolic planner with the activation term a_{plan} , see Equation 18.9. It depends on the position of the first concurrence of the

behaviour in the symbolic plan using 1-based indices and the plan weight term β_{plan} . This direct influence is similar to operator coupling terms, as presented in (Hertzberg et al., 1998).

$$a_{plan} = \begin{cases} \frac{1}{i_B - i_P + 1} \cdot \beta_{plan} & \text{if } B \in P \wedge i_B \geq i_P \wedge i \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (18.9)$$

The particular influence of the activation sources, respectively weights to the decision-making characteristics of the system are described in the following. Here, the meaning of larger and smaller weight values has to be always considered with respect to other weights.

The activation from preconditions only depends on the environment. It is entirely independent of other behaviours and their activation. The activation from preconditions describes the level of fulfilment of the preconditions of a behaviour. Furthermore, the corresponding *situation weight* allows to strive for fast reactions to environmental changes, more opportunistic and egocentric decisions with larger values in comparison to other behaviour-dependent activation weights. Too large values can be problematic if a certain order of goals needs to be guaranteed.

The *predecessor weight* can be used to adjust the influence of executable predecessors on their successors. It supports to spread opportunities from the bottom up (from the perception in the direction of the goals) and activates behaviours whose preconditions are most likely fulfilled in the future.

The *successor weight* accounts for the positive influence of a behaviour to its successors independent of a goal. The successor weight influences the thrustfulness of a network path. Due to only positive effects being considered too large values might lead to the execution of behaviours that are perhaps regretted in the future.

The inhibition sources for behaviours and goals as well as the corresponding *conflictor weight* are used to prevent or reduce undesirable situations. A large conflictor weight results in more cautious and slow decisions, while too large values might result in decisions leading to a dead end of unpopular behaviours and prevents to reach certain goals.

The *goal weight* emphasises a goal-driven character of decision-making with larger values, while it encounters the risk of being too opportunistic so that it favours goal-fulfilling

18. Hybrid Behaviour Planner Core

but otherwise mission-breaking behaviours, especially if several potentially conflicting goals are enabled at the same time.

It is important to remark that goal weight and successor weight both support activation flow towards the goals, although they are very different. The goal weight acts only on behaviours directly contributing to (or conflicting with) goals, while the successor weight also affects intermediate behaviours.

As already stated, the symbolic planner and corresponding *planner weight* are used to guide the behaviour network component with the determined sequential plan of the particular PDDL planner. A larger value results in a more dominant influence, while a smaller value will give only little guidance. Applying the planner allows to support a certain order of behaviour but relies on the existence of plans, which is not always possible or feasible in time in dynamic environments.

Additionally, an optional eighth activation source for the incorporation of RL will be introduced in Section 20.2. This activation source is omitted here for simplification and because it is not mandatory for realising and using the core concept.

After behaviour execution, the activation value is reset to 0 if the behaviour is stopped because of preconditions that became invalid or an exceeded execution timeout. To avoid oscillations of behaviour activations, the activation value is not reset if the behaviour is stopped because of having lower activation than the current activation threshold or having too many behaviours executed in parallel. Behaviours are not expected to finish instantaneously, and multiple behaviours are allowed to run concurrently if they do not have conflicting effects. Additionally, it is configurable how many behaviours are allowed to be executed in parallel in case several behaviours fulfil their preconditions and are above the activation threshold. The limitation of the number of parallel-executed behaviours can be applied in certain scenarios, such as simulations where agents are only capable of doing one thing at a time. If the number of executable behaviours in terms of preconditions and activation is larger than the allowed numbers of in parallel-executed behaviours, the behaviours are selected in decreasing order starting from the highest activation value.

18.2. Symbolic Planner Extension

As already stated in the previous section, one term (Equation 18.9) in the activation calculation is influenced by the symbolic planner, applying the index position of the particular behaviour in the planned execution sequence. In order to allow for a quick replacement of the planner, we based our interface on the widely used PDDL standard in version 2.1. Hence, the majority of existing planners can be used. In particular, the planner requires support of deterministic problem-solving in a *STRIPS model*, *Negated predicates* for translating Boolean preconditions, *Numeric fluents*, and *equality* for modelling non-Boolean sensor values. *Conditional effects* are optionally required because they are allowed in the behaviour layer but are not essential, and most planning scenarios can be designed to avoid them.

For our implementation, we developed a ROS Python wrapper for the Metric-FF planner (Hoffmann, 2002), a version of FF (Hoffmann, 2001) extended by numerical fluents and conditional effects. It meets all above-listed requirements, and due to its heuristic nature, favours fast results over optimality. The wrapper is responsible for appropriate result interpretation, execution handling, and exception handling. To avoid that the entire decision-making is blocking because of planning errors or unsolvable problems, the planner is executed in a separated process.

The mapping and translation between the domain PDDL and the resulting plan is part of the *manager* component. The PDDL generation on entity level is done automatically by the *behaviour*, *activator*, and *goal* objects themselves through a defined service interface. This includes the conversion of complex data types of sensors (like locations or other multi-dimensional data) to single dimensional fluents while maintaining as much of the original meaning as possible. Moreover, the manager monitors time constraints defined in behaviours, replans in case of timeouts, unexpected or missing behaviour influence, newly available behaviours, or if the behaviour network execution order deviates from the proposed plan. This ensures that replanning is only executed if really necessary and keeps as much freedom as possible for the behaviour network layer for fast response and adaptation.

The manager also handles multiple existing goals of a mission by selecting appropriate goals at the right time depending on available information, e.g. if goals cannot be reached

currently. Since goals can have priorities, the manager tries to find a plan that reaches as many high-priority goals as possible. For that, it uses an elimination algorithm that first tries to achieve the highest priority goal together with as many other goals as possible and repeats this process with lower priority goals until it is able to find a valid plan.

18.3. ROS-Integration

All components of the RHBP are based on the ROS messaging architecture and are using ROS services and topics for communication. Figure 18.2 visualises the ROS architecture of a RHBP core instance with corresponding nodes and the communication between them.

Every component of the behaviour network, like a behaviour with its conditions and sensors, is automatically registered to the manager node and reports its current status accordingly. The application-specific implementation is simplified through provided interfaces and base classes for all behaviour network components that are extended by the application developer and completed by filling function hooks, like start and stop of a behaviour. Particularly, the class constructor automatically uses registration methods and announces available components to the manager.

The ROS sensor integration is inspired by Allgeuer and Behnke, 2013 and implemented using the concept of virtual sensors. This means sensors are subscribed to ROS topics and updated by the offered publish-subscribe system. Moreover, certain behaviour network components like the special publisher-goals are automatically generating ROS topics to publish their satisfaction that can also be used outside the actual RHBP environment. The communication of the current states of behaviours and goals is realised with services that are triggered from the manager component to be able to check the connectivity as well as to synchronise the decision-making together with the transfer of the pure status information.

All displayed knowledge base components will be explained in the following Section 18.4, they are already depicted here in Figure 18.2 to provide a complete picture of the core components in one figure.

For each registered component, a proxy object is instantiated in the manager to serve as a data source for the actual planning process where the activation is computed based

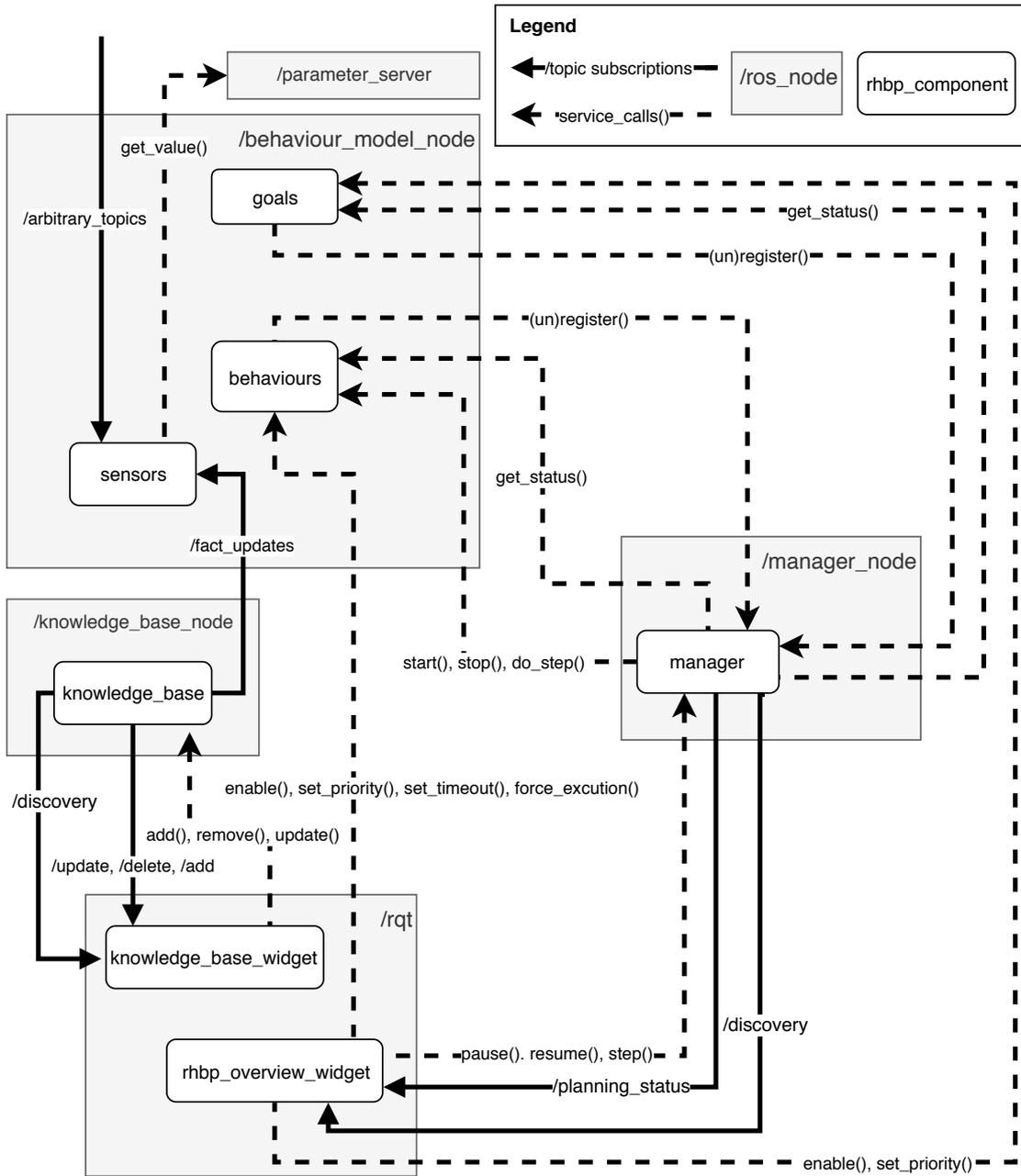


Figure 18.2.: RHPB core ROS node and communication architecture. Direction of the edges correspond to the initiated direction of the data flow.

18. Hybrid Behaviour Planner Core

on the relationships arising from the reported wishes and effects. Besides the status service offered by behaviours and goals, there are a number of management services available to influence the execution, for instance, to start, stop, activate, deactivate and prioritise. Moreover, each manager is also broadcasting its particular ID (namespace and name) to a specific ROS topic to simplify discovery of RHBP managers by other components such as debugging and monitoring tools.

Due to the distributed ROS architecture, the whole system works even across the physical boundaries of individual robots and computers on a distributed system. It is important to note, especially with respect to Figure 18.2, that not only the manager and behaviour model can be distributed over multiple ROS nodes. Likewise, individual behaviours can be situated on different nodes. A setup with behaviours distributed over multiple nodes is exceptionally recommended if certain behaviours are creating high computational load in order to parallel the processing and avoid bottlenecks. Particularly, we followed this pattern in the project discussed later in Section 23.1.

Additionally, parameters and constants can be conveniently set using ROS mechanisms even at runtime, for instance, by using the provided visual `rqt` monitoring and control Graphical User Interface (GUI) that is visualised in Figure 18.3. `rqt` is a Qt-based framework for GUI development for ROS. It allows to freely configure combinations of visual widget plugins in one or many dockable windows. `rqt` is shipped with several included widget plugins and can be used to configure application-specific monitoring and control GUIs. The developed RHBP plugin enables introspection into the decision-making and planning process for the developer. In particular, the interface lists all currently registered behaviours and goals with the current status like conditions satisfaction, influenced sensors, current wishes, currently valid symbolic plan, current activation and activation subcomponents. Moreover, this plugin supports the stepwise execution of the decision-making as well as manually starting and stopping behaviours independent of the manager activation for debugging purposes. The plugin is always connecting to one specific manager but makes use of the previously described discovery mechanisms to enable convenient switching between different manager instances.

Furthermore, RHBP comes with generic sensor implementations that directly support the extraction of single data fields from arbitrary topic types in order to enable direct integration of existing sensors by just configuring the topic name. For more complex

The screenshot displays the ROS RHPB monitoring interface, titled "Default - rqt". It is divided into several panels:

- GLOBALS:** Contains configuration for the planner prefix (agentA1), activation threshold (3.81), and activation threshold decay (0.80). It also shows the current step (1188) and running behaviours (recharge_backup).
- BEHAVIOUS:** Shows the configuration for the "goto_location" behavior, including its activation status, force start, activation time, satisfaction, ready threshold, and execution parameters.
- Logger Level:** A panel for configuring log levels for various nodes and loggers. The "Levels" column shows the current log level for each, such as "Debug" for "rosgraph" and "Info" for "rosgraph.network".
- Process Monitor:** A table showing the status of various processes, including their PID, CPU usage, and memory usage.

Node	PID	CPU %	Mem %
/rqt_gui_py_node_6637	6637	25.20	1.66
/rqt_gui_cpp_node_6637	6637	25.20	1.66
/rosout	6523	0.00	0.03
/rhbp_agent_agentA6	6553	22.10	0.64
/rhbp_agent_agentA5	6551	22.10	0.65
/rhbp_agent_agentA4	6549	23.20	0.64
/rhbp_agent_agentA3	6547	24.30	0.64
/rhbp_agent_agentA2	6545	27.40	0.64
/rhbp_agent_agentA1	6543	25.30	0.65
/knowledgeBase_tub_contest	6541	6.30	0.41
/graphhopper_node	6540	0.00	0.40
/bridge_node_agentA6	6552	1.10	0.39
/bridge_node_agentA5	6550	0.00	0.39
/bridge_node_agentA4	6548	1.10	0.38
/bridge_node_agentA3	6546	1.10	0.39
/bridge_node_agentA2	6544	1.10	0.39
/bridge_node_agentA1	6542	3.20	0.40

Figure 18.3.: Graphical RHPB Monitoring and control in ROS: Custom widget plugin for RHPB planner status monitoring on the left, dynamic log level configuration on the right top corner, and dynamic log level filtering on the right bottom corner.

sensor types, the user has to extend a reduction function in a template that reduces one or more sensors to a single dimensional value. Moreover, it is possible to directly get information from the ROS parameter server with another specialised sensor base class.

Additionally, the *DynamicSensor* base class allows to subscribe to ROS topics in an adaptive manner by defining a pattern of topic name and type to which the sensor connects. The default implementation returns the last valid message received from all matching topics, but this can be adjusted in an inheriting class if other reduction patterns in case of several matching patterns are desired. The *DynamicSensor* enables a self-adaptation to changing robot configurations that might be induced by replaced components or failures. Here, the component enables the dynamic hand over to another suitable ROS topic in case of several applicable instances.

Activators for some common ROS types, such as Pose messages, are provided as well in the default RHBP scope of supply, and additional extensions for other common types could easily be developed.

In order to enable the described target architecture with multiple independent RHBP instances on individual robots in a distributed multi-robot system, several RHBP can also coexist in one ROS environment using individual namespaces.

In order to illustrate how the application of RHBP using its object-oriented API in practice would look like, Listing 18.1 shows an example code snippet. This indicates that the user mostly has to fill the hooks of particular behaviour implementations that are inheriting from the RHBP base class. Aside from implementing behaviours, provided RHBP classes are directly used to model the behaviour network dependencies with conditions, effects, and goals. All communication, registration, and management is embedded into the base class implementations.

18.4. Knowledge Base: Sharing information in a hybrid behaviour network

Another missing point in existing behaviour network architectures and implementations, which also includes the hybrid architectures, are concepts for information sharing amongst multiple entities. This is not surprising considering the fact that only Jung, 1998 is targeting multi-agent systems. Nevertheless, a concrete concept is also missing

Listing 18.1: Example of a RHBP model definition in Python.

```

class ExampleBehaviour(BehaviourBase):
    def start(self):
        # do something
        # ...
    def do_step(self):
        # called every decision-making cycle
        # ...
    def stop(self):
        # stop doing something
        # ...

# creating a sensor instance bound to a RDS topic,
# types are automatically inferred
example_sensor = TopicSensor(name="sensor_name", topic="/example")

# creating an instance of a specific activator, here linear interpolation
example_activator = LinearActivator(zeroActivationValue=1.0,
                                     fullActivationValue=10.0)

# forming compound conditions
example_cond = Negation(Condition(example_sensor, example_activator))

# creating a goal instance, condition(s) describe the target state
example_goal = GoalBase(name="goal_name", permanent=True,
                        conditions=[example_cond], priority=0)

# creating an instance of the behaviour
example_behaviour = ExampleBehaviour(name="behaviour_name")

# adding a precondition to a behaviour
# another_example_cond is omitted here for simplicity
example_behaviour.add_precondition(another_example_cond)

#defining a sensor effect, indicator describes effect strength and direction
example_effect = Effect(sensor_name=example_sensor.name, indicator=1.0,
                        sensor_type=float)

# effect is bound to the behaviour
example_behaviour.add_effect(example_effect)

```

in the solution of Jung, 1998 because the presented ABBA architecture is only providing the abstract concept of feature detectors. How individual agents exchange information is not considered.

The PDDL-based deliberative planning approaches (Cashmore et al., 2015; Rovida et al., 2017) are considering the problem of information sharing to some extent in their centralised solutions. ROSPlan from Cashmore et al. is using a completely centralised approach with one central OWL knowledge base that has to be filled by the user of the framework based on a certain set of interfaces. The framework is using the interfaces to create an ontology that is used to determine when replanning is required. The SkiROS approach (Rovida et al., 2017) has similarities to ROSPlan. It is also using one central knowledge base, which is called World Model, but it partitions the knowledge into continuous, discrete, and semantic data. Nevertheless, neither ROSPlan nor SkiROS are considering information sharing on an agent level, it is only used as a central input component for centralised planning.

RHBP is in contrast to described approaches also directly aiming for decentralised multi-robot systems. In such cases, sharing information amongst agents can foster the cooperation and simplifies the creation of solutions for distributed problem solving, as shown in practice in (Albayrak and Krallmann, 1992). In consequence, it is crucial to allow for a straightforward implementation in RHBP, too. Examples for scenarios that require sharing information are, e.g. search and rescue or exploration scenarios, where the agents need to exchange information about which places have already been visited or which items have been found.

The solution from Albayrak and Krallmann, 1992 is applying a blackboard architecture (Hayes-Roth, 1985). Despite its age, the blackboard architecture is still used (Perico et al., 2018; Tzafestas, 2018) and provides a useful pattern for the implementation of adaptive intelligent systems.

For RHBP, we followed the blackboard approach and integrated a component that is called *knowledge base*. The RHBP knowledge base allows to store arbitrary knowledge facts in a shared tuple space. Moreover, the implementation gives complete freedom of the instantiation. It is possible to have one central knowledge base, use individual knowledge bases per agent, or using different knowledge bases for certain agent groups. Likewise, an agent can also access several knowledge bases at the same time. From the

18.4. Knowledge Base: Sharing information in a hybrid behaviour network

implementation point of view, each knowledge base is a ROS node that is identified by its name and namespace. All agents that know the name are able to store and retrieve information from the knowledge base. Furthermore, all knowledge bases are continuously broadcasting their name on a specific ROS topic for simplified discovery, which follows the same concept as the discovery of the manager component.

The implementation of the communication is also applying common ROS concepts such as the publish-subscribe pattern. This pattern is mainly used to implement filtering and notification of information. Here, the client agent is able to register a search pattern with placeholders for being updated in case desired tuples are updated, added, or deleted. The corresponding updates are shared over automatically created ROS topics. Here, a distinct topic for each type of changed information is created, which corresponds to one topic per pattern for: update, add, and delete. This becomes especially useful if multiple agents are interested in the same search pattern, which would be mapped to the same topics, allowing for notification broadcasts. The communication means are already visualised in Figure 18.2 in the previous section. On the client site, tuple facts can be stored in a local tuple cache to guarantee fast access to all required information. The local tuple cache is also beneficial in situations with interrupted communication and increases the robustness of the information exchange. All writing access is implemented with ROS services that are conveniently abstracted in an own knowledge base client library. The client library supports reading, writing, updating, and tuple existence checks. The tuple space implementation is based on the Linda tuple space (Gelernter, Carriero, and Chandran, 1985) Python implementation `lindypy`¹, which has been further extended to enable additional search modes.

RHBP is as well supporting the information sharing through the ROS parameter server with specific behaviours and sensors for storing and retrieving information. This alternative solution can also be used to implement blackboard architectures. Nevertheless, the knowledge base implementation has the advantage of being much more efficient due to the publish-subscribe-pattern. In order to access information from the parameter server, every information piece has to be polled. Furthermore, the parameter server is less flexible, only one central instance can be used, and it does not support filtering with search patterns.

¹<https://pypi.org/project/lindypy/>

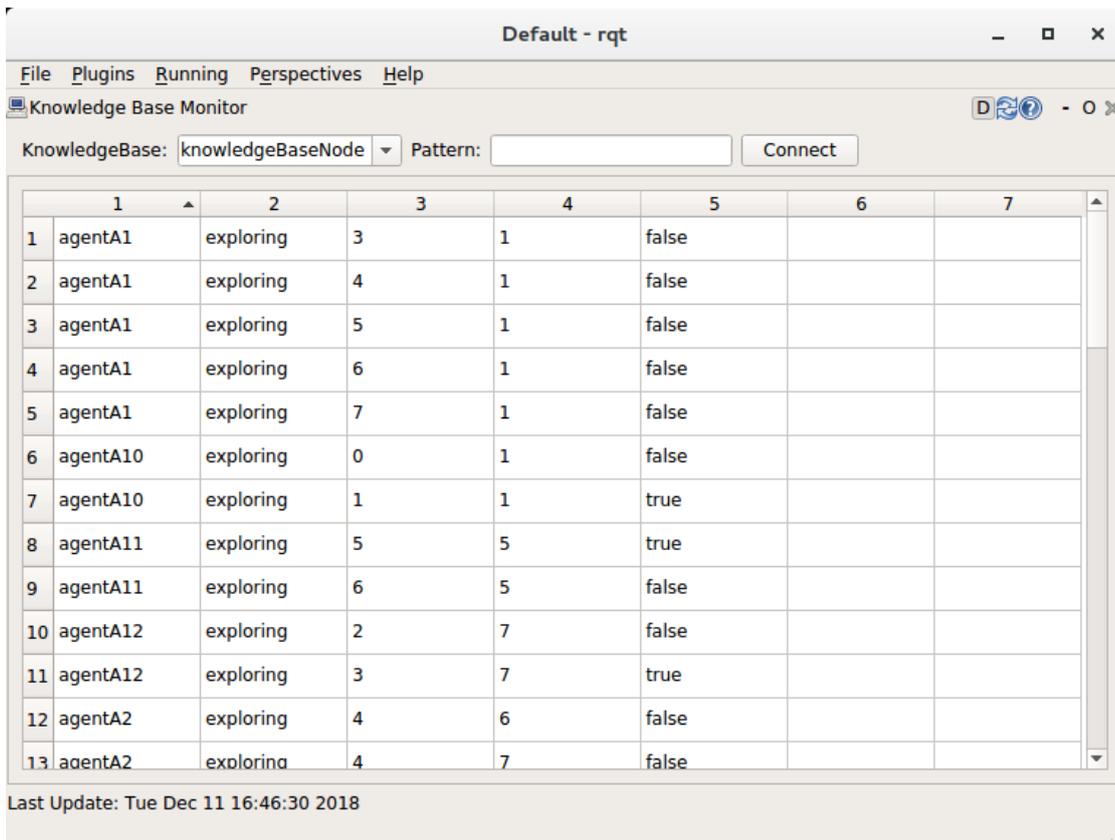
18. Hybrid Behaviour Planner Core

The general knowledge base concept and implementation are entirely independent of RHBP and can also be used together with other ROS packages. For this reason, the implementation is also shipped as a separated ROS package. The RHBP integration is realised with specific classes in a RHBP extension package named *rhbp_utils*. This package includes special sensor and behaviour implementations for retrieving and manipulating knowledge in a knowledge base. This allows the direct integration of shared knowledge in the existing behaviour model concept with conditions and activators. The provided behaviour and sensor classes are also providing a base for application-specific implementations. Currently, the library contains a *KnowledgeUpdateBehaviour* that updates a certain fact once the behaviour has been activated, as well as various sensor implementation that allow to check for fact existence, retrieving specific facts matching a pattern or the number of matching facts.

The knowledge base implementation is also accompanied by a GUI that allows to monitor, inspect, and manipulate the facts during execution. The GUI is implemented as a rqt plugin widget to conveniently integrate into the ROS ecosystem, the widget is depicted in Figure 18.4. The widget is implemented based on the described publish-subscribe pattern, too. Moreover, it also supports the configuration of patterns to filter the visualised information.

All in all, the RHBP approach for knowledge sharing with the knowledge base component is modular in terms of distribution and decentralisation, tightly integrated into ROS and provides as the first behaviour-network based approach means for sharing knowledge over multiple agents running a behaviour model for task-level decision-making. Moreover, the knowledge base component allows to extend the basic behaviour network decision-making lifecycle of RHBP core, which is corresponding to the MAPE loop concept, to the extended Monitoring, Analysis, Planning, Execution and Knowledge (MAPE-K) concept (Brun et al., 2009; Kephart and Chess, 2003). In the MAPE-K concept, a knowledge component is required and is used to share data amongst the monitor, analyse, plan and execution components. In this respect, the presented knowledge base integration concept is exactly enabling the proposed process where the monitoring components create the knowledge, which corresponds to the knowledge sensors in RHBP. Later in the lifecycle, other RHBP components like the behaviours as execution component are also able to modify the stored knowledge.

18.4. Knowledge Base: Sharing information in a hybrid behaviour network



The screenshot shows a window titled "Default - rqt" with a menu bar (File, Plugins, Running, Perspectives, Help) and a title bar "Knowledge Base Monitor". Below the title bar, there is a "KnowledgeBase:" dropdown menu set to "knowledgeBaseNode", a "Pattern:" text input field, and a "Connect" button. The main area contains a table with 7 columns and 13 rows. The columns are labeled 1 through 7. The table data is as follows:

	1	2	3	4	5	6	7
1	agentA1	exploring	3	1	false		
2	agentA1	exploring	4	1	false		
3	agentA1	exploring	5	1	false		
4	agentA1	exploring	6	1	false		
5	agentA1	exploring	7	1	false		
6	agentA10	exploring	0	1	false		
7	agentA10	exploring	1	1	true		
8	agentA11	exploring	5	5	true		
9	agentA11	exploring	6	5	false		
10	agentA12	exploring	2	7	false		
11	agentA12	exploring	3	7	true		
12	agentA2	exploring	4	6	false		
13	agentA2	exploring	4	7	false		

At the bottom of the window, it says "Last Update: Tue Dec 11 16:46:30 2018".

Figure 18.4.: Graphical knowledge base monitoring, inspection and control in ROS. Shown is a live view of the application discussed in 23.2.2.

18.5. Cores of Cores: Hybrid Behaviour Networks in a Hierarchy

The presented overall RHBP architecture in Chapter 17 has already introduced the concept of multiple decision-making and planning cores that can operate independently, or that can be nested within each other. The idea behind such decision-making hierarchies is to provide an option for structuring the implemented behaviour models in several logical levels. Such a structure can be used to control multiple robots from a higher-level centralised perspective, to partition several robots into groups, or to provide additional guidance for the decision-making of individual agents. The guidance comes from the implicit reduction of the degree of freedom in decision-making because certain decisions are only possible if the agent is executing a nested core with a nested behaviour model, hence not all behaviour can be chosen. Furthermore, enabling nested decision-making and planning in RHBP provides means for a priori definitions of task decompositions. In this case, multiple nested decision-making and planning cores can be used to statically decompose a more complex task into atomic behaviour implementations.

The concept of nested cores is realised with so-called *NetworkBehaviours*, not to be confused with the chosen concept of behaviour networks. *NetworkBehaviours* are a special type of behaviours that directly inherit from the behaviour base class of RHBP. In contrast to normal behaviour implementations in RHBP, the *NetworkBehaviours* are not directly executing any actions that have an influence on the environment. Instead, *NetworkBehaviours* are triggering a nested decision-making and planning process to select suitable behaviours from their encapsulated behaviour model to achieve the targeted effects.

In detail, the implementation of the nested decision-making and planning is realised with an additional manager instance, including a symbolic planner within each *NetworkBehaviour*. This way, the *NetworkBehaviour* becomes an additional and mostly independent RHBP core. A dependency exists only in the decision-making lifecycle. The manager of the *NetworkBehaviour* is running its decision-making and planning lifecycle every time the *NetworkBehaviour* is activated by its surrounding behaviour model. In the end, the decision of the *NetworkBehaviour*'s manager is leading to one or multiple activated sub-behaviours that are executing a certain action. Furthermore, an already

nested NetworkBehaviour can also contain other NetworkBehaviours, too. This allows to create arbitrary complex behaviour model trees that are only limited by the computational resources. All the manager instances are differentiated by names following the ROS namespace concept. The integration of the NetworkBehaviour on a class level is visualised in Figure 18.5. The diagram shows that NetworkBehaviours are directly inheriting from common RHPB behaviours and that they have a composition relationship to the manager class because a NetworkBehaviour without internal manager would not be operational.

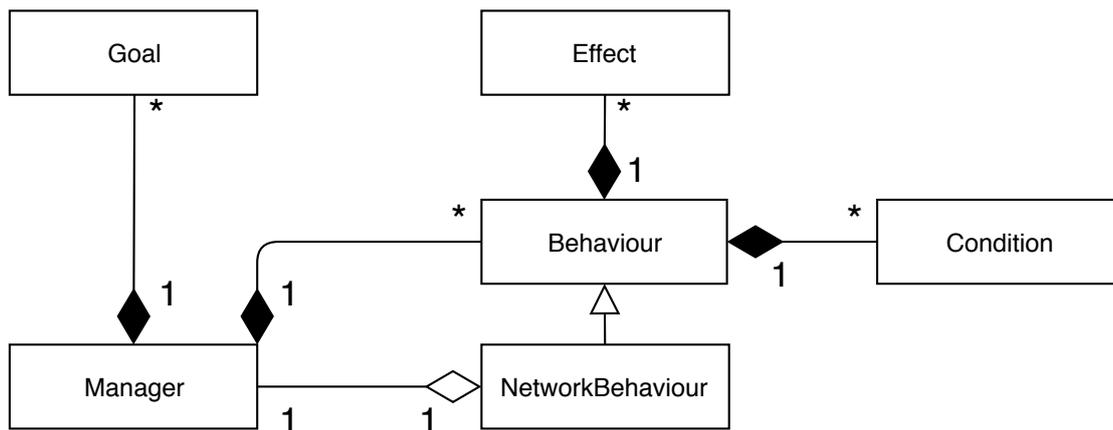


Figure 18.5.: UML Class diagram illustrating the architectural integration of NetworkBehaviours in the RHPB framework.

A NetworkBehaviour can be stopped like any other behaviour, which for instance, occurs to resolve conflicts. In RHPB interrupting running behaviours is only possible if a behaviour is marked as interruptible (by default every behaviour is interruptible), in case of the NetworkBehaviour either all registered or all currently running nested behaviours are checked if they are interruptible. If one behaviour is not interruptible, the entire NetworkBehaviour is considered as not interruptible.

The modelled effects of a NetworkBehaviour are representing sub-goals for the nested behaviour model because this is what the surrounding behaviour model is expecting from the particular NetworkBehaviour. However, two different situations have to be considered here. First, the NetworkBehaviour with its nested behaviours and all surrounding behaviours are working in the same domain level with the same sensors and level of abstraction. Secondly, the modelled effects of NetworkBehaviours are only im-

18. Hybrid Behaviour Planner Core

PLICITLY mapped to the effects of the nested behaviour due to different abstraction levels in the modelling. An example of a behaviour model with comparable different abstraction levels is a very low-level motor controller within the NetworkBehaviour and the usage of Global Navigation Satellite System (GNSS) coordinates for navigation in the surrounding model. The implemented solution is applicable for both scenarios of closely connected and very different domain abstraction levels. Even more, for the first scenario, the NetworkBehaviour implementation supports the automatic generation of nested sub-goal objects derived from the modelled effects. In the second scenario, the system designer has to manually model the effect of the NetworkBehaviour as well as the goals of the nested behaviour model. Furthermore, it is also possible to have a hybrid model that makes use of automatically generated sub-goals as well as using additionally modelled goals.

Due to the reason that an effect is often considered to be reached repeatedly during the time of the execution over several decision steps, an additional activator type has been developed. The activator is named *GreedyActivator*, already mentioned in Section 18.1.1, and allows to formulate conditions for the nested behaviour model goals that are never reached because the activator would always strive for a higher or lower value depending on the current sensor value of the condition. Furthermore, in contrast to the alternative solution of just using very large effect values, the GreedyActivator approach avoids the calculation of very long plans in the nested planner.

Preconditions of behaviours in the nested behaviour model and the preconditions of the NetworkBehaviour model are considered independently. This allows to formulate conditions that enable or disable the decomposition through a NetworkBehaviour, while the nested behaviour can independently be checked for being executable depending on environmental conditions.

In general, the NetworkBehaviour concept is also fostering the reuse of specific behaviour models because they allow to model a separated behaviour-goal ensemble that fulfils a certain sub-goal. The reuse is especially supported by the tight ROS integration that helps to easily (re-)connect different components with each other through the topic interfaces, which can be easily remapped using standard ROS features.

18.6. Summary Decision-Making and Planning Core

In this chapter, we presented in detail the so-called *Core* of the RHBP framework. The realised concept further enhances the concept of hybrid behaviour networks for decision-making and planning of agents into various directions. The solution combines characteristics of both traditional symbolic planning and reactive decision-making to enable goal-directed, self-adaptive task-level decision-making and planning. In particular, it is the first approach of its kind that has a thorough ROS integration, allows for highly distributed computation, provides a structured integration of knowledge sharing by the knowledge base component, and means for creating different decision-making abstraction levels or hierarchies through the so-called *NetworkBehaviours*.

19. Combining Self-Organisation and Hybrid Behaviour Planning

19.1. Self-Organisation Framework

Integrating self-organisation into the planning and decision-making core of the framework RHBP, which has been presented before in Section 18.1, first requires structuring and classifying the various available self-organisation mechanisms and patterns in a reusable fashion. This has been accomplished by the realisation of a modular and reusable framework for self-organisation that is presented in more detail below. This part of the implementation is based on ROS but is entirely independent of the RHBP and can be used generally.

A modified version of the bio-inspired design patterns of self-organisation mechanisms developed in Fernandez-Marquez et al. (2012) constitutes the basis of our self-organisation framework and is reprinted in Figure 19.1.

A detailed discussion can be found in Chapter 8. An advantage of both the original design patterns and our adapted version is their modular character and the modelled relationships between the patterns. Thus, existing patterns or pattern components can

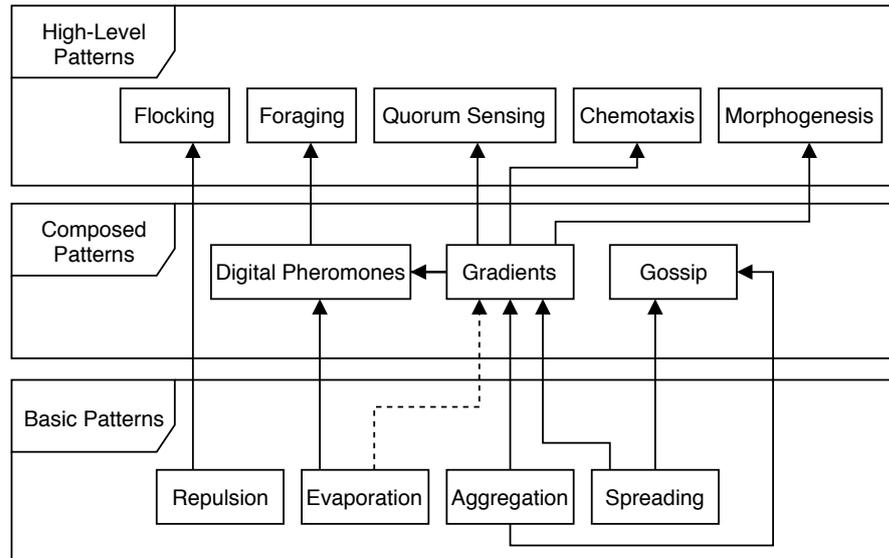


Figure 19.1.: Design patterns for self-organisation and their relationships after Fernandez-Marquez et al., 2012. Arrows indicate how patterns are composed. A dashed arrow is an optional connection.

be used as the basis for the realisation of new ones. In contrast to the design patterns presented in Fernandez-Marquez et al. (2012), our adapted version bases all advanced patterns on the gradient pattern, which we determined as the common ground amongst the various patterns. The concentration on one core pattern allows for a simplified and more general implementation. Furthermore, the patterns were regrouped to categorise them on their purpose instead of their complexity, as shown in Figure 19.2. The individual patterns have already been introduced in Chapter 6.

The bio-inspired design patterns are categorised in three groups, namely *Basic Functionality Patterns*, *Movement Patterns* and *Decision Patterns*. *Basic Functionality Patterns* provide required functionalities for the other pattern categories. Apart from spreading, these patterns do not lead to actions that are executed by the agents themselves. *Movement Patterns* lead to the movement of the agents, e.g. enabling robots to base their movement on a potential field. Finally, *Decision Patterns* enable collective decisions.

The central *Basic Functionality Pattern* is the *Gradients* pattern as all *Movement Patterns* and *Decision Patterns* can be implemented based on its foundation. Gradients are situated information, which are subject to *Spreading*, *Aggregation*, and possibly

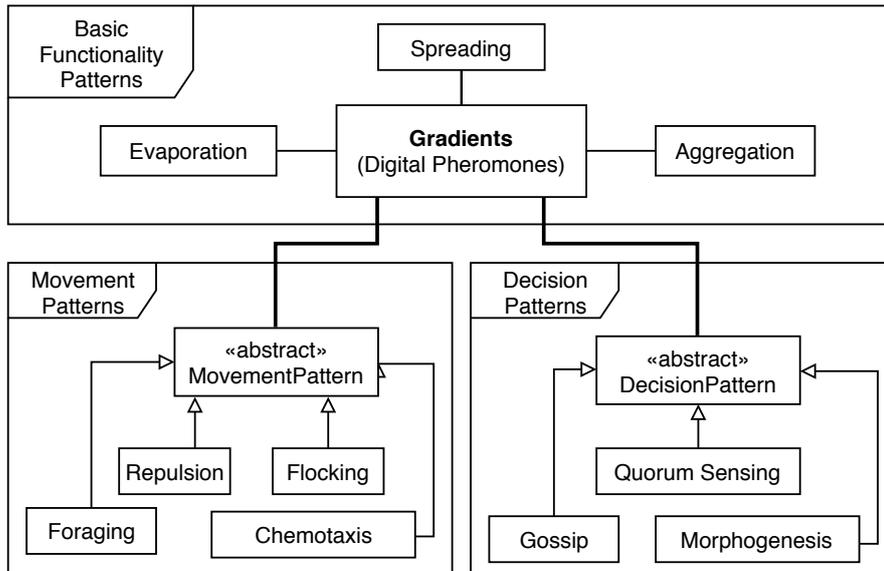


Figure 19.2.: Framework of bio-inspired self-organisation design pattern dependencies in a UML class diagram style visualisation.

Evaporation. Gradients are similar to pheromones and force spread in a magnetic field, but in contrast to the natural role models, virtual gradients are usually forwarded point-to-point locally by agents within a system for propagation and are not diffusing through the environment on their own (Nagpal, 2004). Using the gradient pattern as core pattern is possible because the gradient pattern includes already most of the required capabilities for other patterns such as spatio-temporal attributes describing where the data is located and when it was created.

Gradients can either be deposited in the (virtual) environment or be attached to a moving entity like an agent. Moreover, gradients are either spread by agents themselves or distributed in the environment through proxy components. However, we enhanced the gradient pattern to contain all information required by movement and decision patterns. A visualisation of a gradient is depicted in Figure 19.3. The meaning of the particular elements will be explained later, together with the corresponding elements in Listing 19.1.

Gradient data is sent, stored, and manipulated by a component we have named self-organisation buffer (*SoBuffer*). This component is part of the ROS package *so_data*, which also contains abstract implementations of above-mentioned patterns. The self-organisation framework, with its core the *SoBuffer*, was realised completely within the

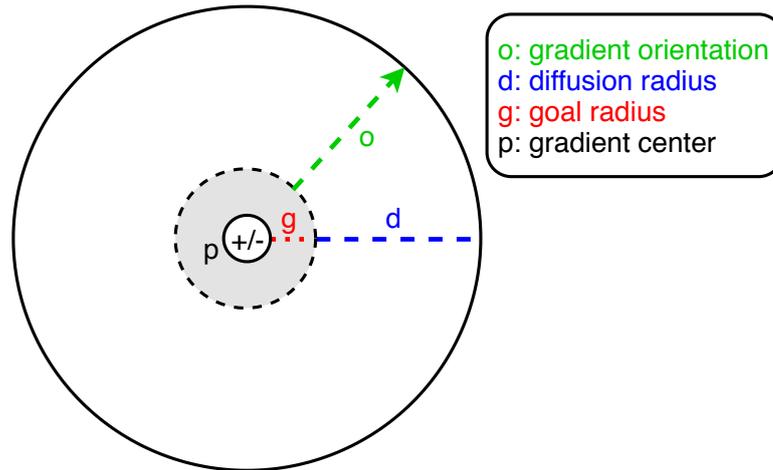


Figure 19.3.: Visualised gradient with attractive/repulsive center, goal radius, diffusion radius, and orientation.

ROS. Next, to being easy to extend and to apply, it provides hardware abstraction and an established messaging communication infrastructure. The latter aspect is useful in particular to realise the *Spreading* pattern. The implementation relies on the publisher-subscribe design pattern, which is a core concept of ROS, and it is inspired by the information sharing and filtering architecture of the popular tf package (Foote, 2013). In detail, the messages are exchanged and forwarded amongst the agents through a dedicated ROS topic named `\so_data`. Here, all information is exchanged in a distributed fashion, and all processing based on this data is done locally by each agent using an own instance of the SoBuffer. The data structure of the used message, which is named *SoMessage*, is shown in Listing 19.1.

In addition to the *gradient center*, each gradient includes a *goal radius* and a *diffusion radius* that define the range in which the information can be sensed by others. This resembles the diffusion parameter presented by Fernandez-Marquez et al. (2012) to specify the range of the gradient information. Other attributes enable to define gradient direction and heading, namely *gradient orientation* and the *initial direction vector*. An *attraction* value describes whether the gradient is attractive or repulsive. The *payload* field allows for the integration of application- or pattern-specific information. In *payload*, any additional information is stored as key-value pairs. The *frame ID* included in the ROS header allows to differentiate gradient data while *parent frame ID* provides a reference to the sender of the gradient, e.g. a robot or environment component. Addi-

Listing 19.1: The gradient data message (*SoMessage*) structure in pseudocode.

```

frame ID
time stamp
parent frame ID

gradient center
gradient orientation
(initial) direction vector

attraction value
diffusion radius
goal radius

evaporation factor
evaporation time
evaporation timestamp

boolean is_moving

gradient payload dictionary

```

tional fields in the ROS message enable to specify whether the gradient is subject to evaporation, namely *evaporation factor*, *evaporation time* and *evaporation time stamp*.

Moreover, gradients can be either static or moving, which is defined by the *boolean* value *is_moving*. Static gradients are deposited at a particular place in the environment while moving gradients are attached to a moving entity, e.g. a robot, and change their position with it. Depending on the pattern, moving, static or both gradient types are used in the mechanism calculations where they are handled in the same way. The storing process differs though. A configurable number of gradient messages is stored for moving gradients based on their frame ID and parent ID, which reference the origin of the gradient. On this foundation, it is possible to calculate additional information such as the velocity of the moving entity the gradient is attached to. Furthermore, a special *pose frame* can be referenced in the SoBuffer instance to indicate which gradients provide data about the poses of the agent itself. Differentiating own gradient data from other sensed gradient data based on pose frame and the specified ID allows to store the own gradient data of an agent separately. In consequence, calculations requiring the current

19. Combining Self-Organisation and Hybrid Behaviour Planning

position of the agent are facilitated. Static gradients are aggregated and stored based on their frame ID. In contrast to moving gradient data, not all received information is stored individually. Instead, the data is directly aggregated using functions such as the average of the gradients at one position.

Both *Aggregation*, or information fusion, and *Evaporation*, which reduces the relevance of information over time, are applied on the gradient data by the SoBuffer. Particularly, aggregation allows to reduce the amount of data present in the system, while evaporation enables reducing the relevance of information over time. Each gradient has individual evaporation attributes, namely frequency and strength, attached to it, which specify the rate at which evaporation is executed over the data point by the SoBuffer. In addition, the SoBuffer can aggregate the received gradient data based on their purpose. Hence, gradients that are used for different tasks can be aggregated differently or used without aggregation. For example, gradient data can be aggregated based on its location or its sender. The implementation applies the publish-subscribe design pattern, whereby spread gradient information (*SoMessage*) is received, collected and filtered on the receiver-side (*SoBuffer*). This enables the combination of individual and decentralised mechanisms within one information space.

The above-described extended gradient implementation contains all the required information for various patterns, which can be utilised in the calculations of different self-organisation mechanisms. In the next section, specific pattern implementations applying the *SoBuffer* are described to illustrate that the gradient pattern provides a rich foundation for the implementation of all movement and decision patterns within this framework.

19.2. Self-Organisation Pattern Implementations

In the following, we consider a self-organisation mechanism as an implementation of an abstract self-organisation pattern. All Basic Functionality Patterns were realised based on Fernandez-Marquez et al. (2012).

The *Digital Pheromone* pattern is a special case of the Gradients pattern. Digital Pheromones are evaporating gradients that are deposited in the environment (Fernandez-Marquez et al., 2012). In consequence, this pattern can be implemented with static

gradient messages using only the diffusion radius, no goal radius, and an evaporation factor below 1 to enable evaporation.

Advanced patterns can request different subsets of the stored gradient data from the SoBuffer to base their calculations on those. Movement Patterns, and Decision Patterns are integrated as mechanisms in the self-organisation framework. Both are based on abstract classes that provide a common basis for the mechanism implementations. There-with, new mechanisms can be straightforwardly implemented using these blueprints. In the remainder of this section, we will first describe the realised movement patterns in Subsection 19.2.1, before we provide additional details about the implemented decision patterns in Subsection 19.2.2

19.2.1. Movement Patterns

Each movement mechanism determines a movement vector an agent can follow. All four movement patterns depicted in Figure 19.2 are integrated into the self-organisation framework. For each movement pattern, one or more mechanisms are realised, which allow to employ the pattern in different scenarios. The basis for the implementation of all movement patterns is provided by an abstract movement pattern class that serves as a starting point with a consistent structure and some common functionality. In contrast to decision patterns, movement patterns are less application-specific and can be reused more easily. Depending on the particular robot type, only the actual motion behaviour has to be customised, respectively selected to meet the movement characteristics such as differential driving or flying.

Chemotaxis refers to the ability of an organism to follow the gradient of a diffusing substance (Nagpal, 2004). The *Chemotaxis* pattern enables motion coordination based on such gradients. Two different algorithms to calculate movement vectors based on gradient fields were implemented to realise this pattern. Firstly, a general approach for attractive and repulsive gradient calculations as in Balch and Hybinette (2000) is integrated. Secondly, a more sophisticated gradient calculation was integrated, which allows agents to reach an attractive gradient even if it is overlapped by a repulsive gradient following the formulas by Ge and Cui (2000). In particular, attractive gradients have an attraction of zero within the goal radius while the attraction increases linearly within the diffusion radius until it reaches its maximum outside the gradient reach.

19. Combining Self-Organisation and Hybrid Behaviour Planning

Differently, repulsive gradients have an infinite repulsion within the goal radius while the repulsion decreases linearly to zero within the diffusion radius and remains zero outside the gradient reach. Therewith, attraction and repulsion values are determined based on the shortest distance between the agent and the goal radius of a particular gradient. Depending on the particular scenario, different mechanism implementations can be used to follow, for instance the nearest gradient, the overall minimum, or maximum gradient, alternatively the minimum or maximum gradient within reach, or to follow all gradients.

The *Repulsion* pattern enables agents to reach a uniform distribution and to avoid collisions (Fernandez-Marquez et al., 2012). Two mechanism implementations are provided to realise this pattern. In one mechanism, the repulsive gradient formula of Balch and Hybinette (2000) is utilised while the second mechanism applies the repulsion formula presented in Fernandez-Marquez et al. (2012). The approach from Balch and Hybinette (2000) has the disadvantage that it only enables collision avoidance and no distribution patterns.

Flocking allows motion coordination and pattern formation in swarms (Fernandez-Marquez et al., 2012). A mechanism based on Reynolds (1999) is provided, and also a more complex version applying the gradient-based formulas of Olfati-Saber (2006) is integrated into the framework. Both implementations use a list of moving gradients, representing the poses of agents, within view distance for their calculations. The version of Olfati-Saber (2006) relies on the velocities of the agents instead of their movement vectors as done by Reynolds (1999).

Moreover, the *Ant Foraging* pattern is part of the presented self-organisation framework. Foraging is a pattern for collaborative search, which allows to explore and exploit an environment (Dorigo, Birattari, and Stutzle, 2006). In the beginning, agents either are following an existing pheromone trail or are exploring the environment randomly until they localise a resource. The exploitation of resources is achieved by following pheromone trails. The pattern requires several mechanisms to be realised that are based on Fernandez-Marquez et al. (2012) as well as on Deneubourg et al. (1990a). Notably, the realisation consists of a scenario specific *ForagingDecision* pattern to decide for exploration or exploitation based on probabilities, and three movement mechanisms for exploration, exploitation, and returning. Even though ant foraging contains a decision mechanism, it is classified as a movement pattern due to the main aspect of realising

motion of agents based on pheromones exhibiting exploration and exploitation. The particular implementation reuses the already presented chemotaxis mechanism from Ge and Cui (2000) for exploration and returning. Moreover, ant foraging makes use of the SoBuffer feature of filtering the gradients not only by view distance but also by view angle to consider the heading of the agents.

19.2.2. Decision Patterns

Decisions made by a mechanism are based on gradient data provided through the SoBuffer. The gradients are mapped to agents with the parent frame ID. This allows all agents to differentiate whether gradients were sent by themselves or by their neighbours. Even though the implementation of decision mechanisms is highly dependent on the specific application, several common features were identified and are made available in abstract classes as a foundation for specific implementations.

The abstract class for decision mechanisms, which implements the decision patterns, includes two common methods. One method determines the current value and state of an agent and depends on the pattern. The other method spreads these values as a gradient message and is universal for all patterns.

Quorum Sensing allows collective decision-making based on a required threshold number of agents. It can be implemented in a general way following Fernandez-Marquez et al. (2012). Here, the state of an agent is set based on the fulfilment of a threshold value, but the state is usually not spread to the neighbours. It is the only mechanism of the decision patterns that can be implemented independently of a particular scenario. The other two decision patterns, namely *Morphogenesis* and *Gossip*, are highly dependent on the use case.

The Morphogenesis pattern allows to determine the agent's behaviour based on its spatial position (Fernandez-Marquez et al., 2012). For this purpose, agents spread morphogenetic gradients, and others calculate their relative positions to these gradients. Subsequently, this relative position is used within the decision to determine the agent's state and behave according to it. How many morphogenetic gradients are spread and which states are considered depends on the particular application scenario.

Gossip enables to obtain shared agreements between all agents (Fernandez-Marquez et al., 2012). The common principle of the gossip pattern is the aggregation of infor-

mation received from neighbours and own information, followed by a resspreading of the aggregate. For both patterns, sample mechanisms are implemented to exemplify their feasibility. The sample Gossip mechanism determines the maximum spread value while the sample Morphogenesis mechanism determines the barycentre of a group of agents using the algorithm proposed by Mamei, Vasirani, and Zambonelli (2004).

19.3. Combining Decision-Making, Planning and Self-Organisation

The combination of decision-making, planning, and self-organisation is realised by integrating the self-organisation framework presented in Section 19.1 into the RHBP core that is introduced in Chapter 18. The following introduced extension provides necessary components to use the presented self-organisation mechanisms within the hybrid planning structure of the RHBP. The behaviour network layer of the RHBP is based on the dependencies of behaviour preconditions, effects, and desired world states formulated with goal conditions. These dependencies are used for the activation calculation within the RHBP, serving as a heuristic estimation for decision-making and providing a well-matching foundation for the integration of self-organisation. This is because the normalisation of conditions through the application of different utility functions (activation functions) enables the straightforward integration of non-discrete relationships as we find them in self-organisation mechanisms.

Figure 19.4 illustrates both the self-organisation framework presented in Section 19.1 and the integration into the RHBP using components provided by the extension. Three extended component types have been implemented to enable the use of self-organisation mechanisms, namely *sensor*, *condition*, and *behaviour*.

The *GradientSensor* is a central component to enable the integration of self-organisation into the RHBP. It is a complex sensor type that senses gradient-based information provided by the self-organisation framework by invoking their common methods. Specifically, it provides access to the SoBuffer of the *so_data* package and thereby allows to sense movement vectors, gradient values, and agent states. Movement vectors are calculated by the movement mechanisms. For some mechanism implementations, it is possible to sense a vector leading to the goal gradient, too. The sensing of values and states of

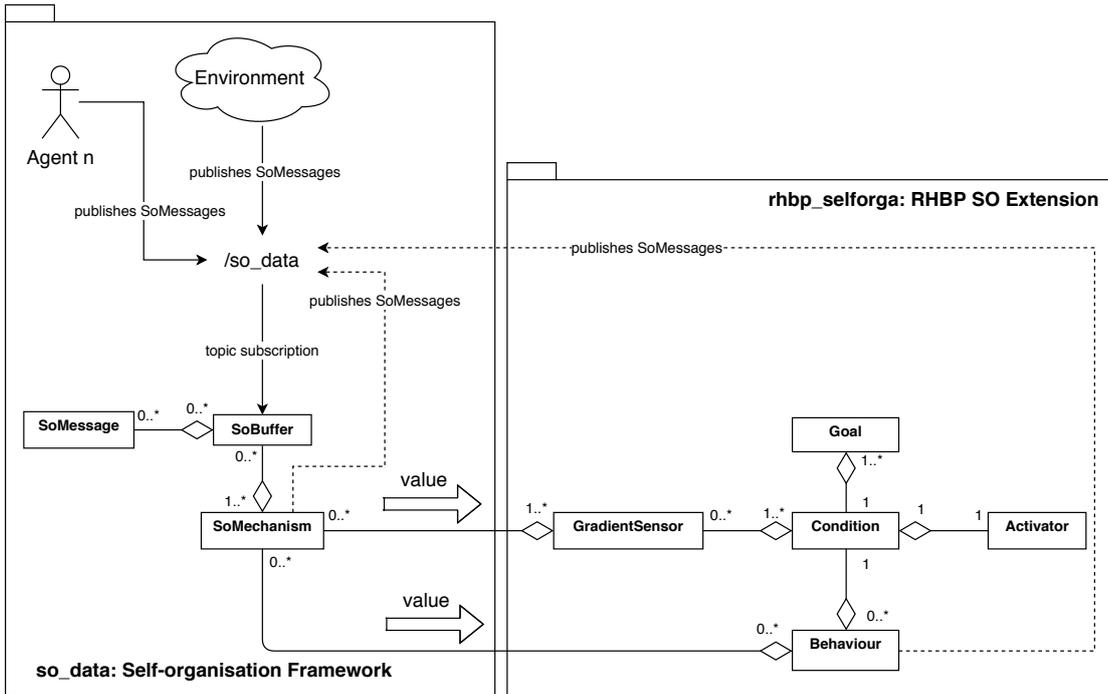


Figure 19.4.: Architecture for the integration of self-organisation into the RHPB. Diagram is a mimicking an UML package/class diagram extended with information about the data flow of gradient data. Lines with directed arrows indicate data flow. Dotted lines are optional connections. Big arrows nearby aggregation connection illustrate exchanged data through this connection.

19. Combining Self-Organisation and Hybrid Behaviour Planning

agents are related to the decision mechanisms.

Figure 19.4 illustrates the usage of the GradientSensor in a *Condition*, which normalises the sensor values and maps them to activation levels using the standard activators provided by the RHBP core implementation, for instance, Boolean, threshold and linear functions. Several special self-organisation conditions are provided by the extension to allow the modelling of different application scenarios. For example, conditions related to movement mechanisms determine activation levels based on the presence of a potential field or the length of the movement vector. Sample conditions for mechanisms related to decision patterns lead to activation when the state or value of the robot has changed. The provided conditions are not exhaustive and can be extended as required.

The main components for the execution of self-organisation within the RHBP are behaviours. Several behaviours that execute the self-organisation mechanisms provided by the framework are part of the RHBP self-organisation extension. Both movement mechanisms and decision mechanisms are implemented based on abstract classes. Thus, common methods exist for the different mechanisms that can be reused by the specific behaviours to conduct self-organisation.

The *Move Behaviour* executes movement mechanisms within the RHBP by invoking their common method *move()*. The method returns a movement vector that will be transformed into a steering command which matches the robot type. Currently, the extension provides a Move Behaviour that transforms the three-dimensional movement vector to a linear velocity in the x-direction and an angular velocity around the z-axis. Thus, it is suitable for all differential drive robots. However, providing additional behaviours for other robot types would only require implementing the mentioned conversion from a movement vector to the particular steering command.

The *Decision Behaviour* executes decision mechanisms within the RHBP by invoking their common method *spread()*. This method determines the value and state of an agent and spreads those values in a gradient message. The distribution of an agent's value and state is a core aspect for the realisation of decision patterns as each agent determines its own value and state based on its neighbours' data.

Several additional behaviours were integrated into the extension to realise behaviours that are not common for all mechanisms. For example, the ant foraging pattern requires that the state of the agents is set to specific values in several cases. Hence, a special

19.3. Combining Decision-Making, Planning and Self-Organisation

RHBP behaviour allows to set the state related to a mechanism to a predefined value.

Listing 19.2 illustrates how the presented self-organisation extension of RHBP based on the `so_data` package is applied in the context of the definition of a particular behaviour model. In the example is illustrated how the created `SoBuffer` instance is used at the same time in the `GradientSensor` as well as the `Move Behaviour`, which calculates the motion commands based on the current gradient vector. The example does omit that the created conditions and behaviours could also be combined directly with non-self-organisation related conditions, behaviours, and goals as shown before in Listing 18.1, due to the fact that they are implemented on top of the same object-oriented domain model.

Listing 19.2: Python example of the RHBP self-organisation extension applied in a simplified behaviour model definition.

```
# creating an instance of SoBuffer with a specific view range,  
# also several buffers can be used in parallel  
so_buffer = SoBuffer(id='robot_id', view_distance=2.0)  
  
# creating an instance of a particular mechanism implementation,  
# here the repulsion mechanism from Fernandez Marquez et al. is created  
mechanism = RepulsionFernandez(so_buffer)  
  
# creating a GradientSensor to integrate gradient sensing into the model  
gradient_sensor = GradientSensor('sensor_name', mechanism)  
  
# creating a Boolean condition that is true if a gradient is sensed  
so_condition = VectorBoolCondition(gradient_sensor, BooleanActivator())  
  
# creating an instance of a self-organisation enabled movement behaviour  
move_behaviour = MoveBehaviour(name="behaviour_name", mechanism)  
  
# adding the so_condition as a precondition to the behaviour  
move_behaviour.add_precondition(so_condition)  
  
#defining a sensor effect, indicator describes effect strength and direction  
example_effect = Effect(so_condition, indicator=-1.0, sensor_type=bool)  
  
# effect is bound to the behaviour  
move_behaviour.add_effect(example_effect)
```

19.4. Selecting Self-Organisation Mechanisms

Selecting an appropriate self-organisation mechanism for the intended system behaviour is a challenging task, which is discussed intensively in Part II. Usually, system designers have to choose a suitable coordination mechanism during design time to let multi-robot systems collaborate in the desired fashion to fulfil an intended task. However, the suitability of a chosen coordination mechanism might change during task execution as the environment or system capabilities might change. Hence, the *Coordination Mechanism Selector (CMS)* was realised, which provides a foundation to determine the most suitable self-organisation mechanism in a given situation based on expert knowledge or experience and a self-organisation goal that indicates the task of the agent. Thus, system designers are relieved from the task to select a self-organisation mechanism during design time, and it is possible to improve the adaptation capabilities of the resulting system.

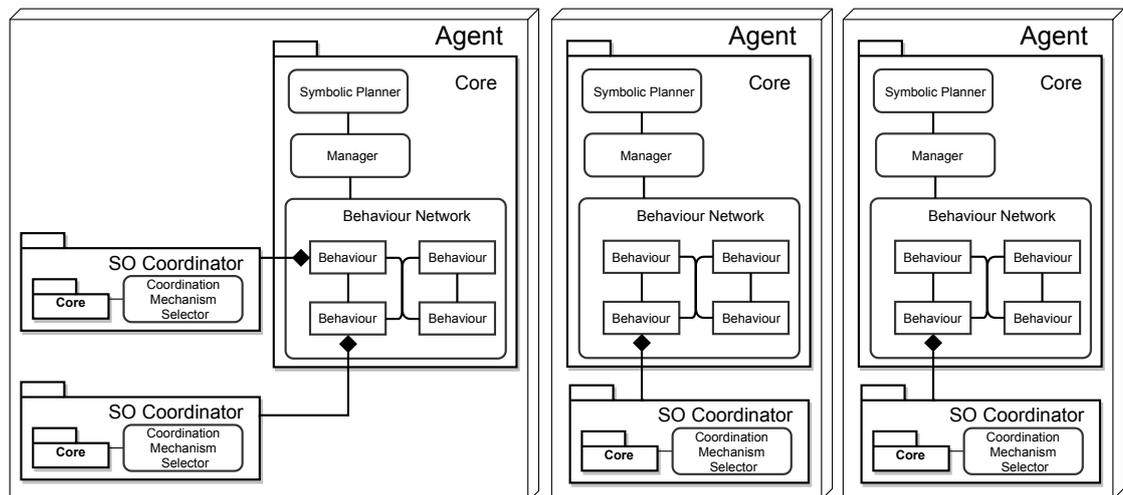


Figure 19.5.: Integration of the Coordination Mechanism Selector in the structure of the RHBP. Each SO Coordinator is always bound to an individual agent in a decentralised fashion.

The *Self-Organisation Coordinator (SO Coordinator)* consists of two components, namely its own RHBP *core* instance and the Coordination Mechanism Selector, as illustrated in Figure 19.5. Self-organisation mechanisms can be encapsulated by the Self-Organisation Coordinator as all components required for the realisation of a self-organisation mechanism, e.g. behaviours and goals, are assigned to its own *core* in-

stance. Thus, a higher-level *core* instance can treat the self-organisation mechanism as one behaviour, in the form of the Self-Organisation Coordinator, no matter how many components are required for its realisation. Hence, its own RHBP *core* instance is used to monitor and control the particular self-organisation mechanism realised within our framework. In detail, this is realised by further extending the *NetworkBehaviour* base class, see Section 18.5, and incorporate the self-organisation specific extensions. It is also possible to integrate or combine several self-organisation mechanisms by using multiple Self-Organisation Coordinator instances per behaviour network. This architecture helps to separate the application-specific modelling using the RHBP from a generic self-organisation mechanism implementation and makes the mechanism implementation exchangeable and reusable.

As illustrated in Figure 19.6, the *Coordination Mechanism Selector* consists of four components, namely *Expert Knowledge*, *Expert Strategy* as particular instance of the *Decision Making Strategy* interface, *SO Specification*, and *Self-Organisation Components* (*SO Components*). Additionally, each Self-Organisation Coordinator requires a self-organisation goal (*SO Goal*), which indicates the task of the multi-robot system, as input.

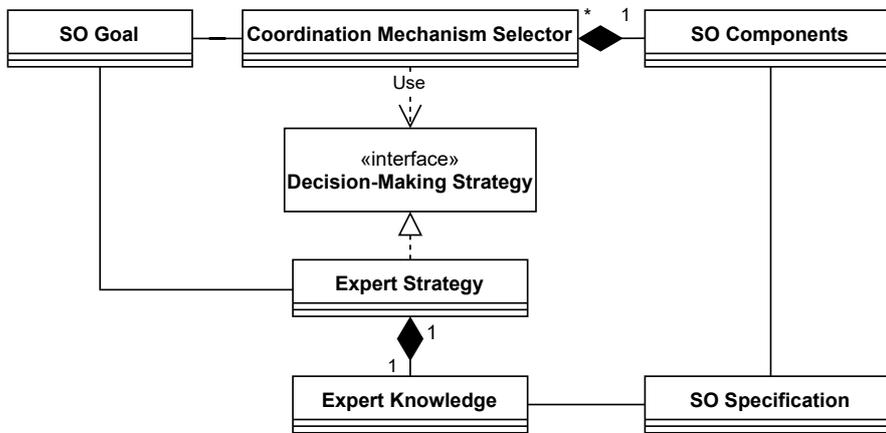


Figure 19.6.: Architecture of the Coordination Mechanism Selector.

The Expert Knowledge maps self-organisation goals (*SO Goal*) to a list of suitable mechanism options. Each option consists of a *configuration key*, a *score*, and *parameters*. The configuration key is an indicator for a self-organisation specification (SO Specification) that can be used to fulfil a self-organisation goal. The indicated self-organisation

19. Combining Self-Organisation and Hybrid Behaviour Planning

specification are specified in a reusable library. Each specification includes RHBP components and parameters as presented in Sections 19.1 and 19.3. The parameters of the specification is replaced with the parameters included in the option to adjust the setting based on the self-organisation goal. The score specified in each option rates the feasibility of the self-organisation goal using the given self-organisation specification. This allows to create a repository of suitable self-organisation specifications that have been evaluated with respect to a given system goal. This concept is inspired by the proposed hypothesis database of Edmonds, 2005.

The *Decision-Making Strategy* is the central interface component of the *Coordination Mechanism Selector*. A particular implementation has to determine a suitable self-organisation specification based on a self-organisation goal. The specific realisation *Expert Strategy* is determining the specification on the foundation of the provided *Expert Knowledge*. Moreover, it might be used to adjust the score included in the options of the Expert Knowledge, e.g., using online learning (Mataric, 1995) or evolutionary approaches (Francesca et al., 2015). Therewith, the decision-making process can incorporate experience, which is essential to determine the most suitable self-organisation mechanism in a dynamic environment. As the environmental conditions might change during task execution, the particular Decision-Making Strategy re-evaluates its decision and adjusts the self-organisation mechanism during task execution if required. In the current development stage, the Decision-Making Strategy selects the option with the maximum score, which is an empirically determined value for a given self-organisation goal.

After determining a configuration key for a self-organisation specification and its parameter adjustments, the Self-Organisation Components (*SO Components*) factory creates the specified RHBP components. These components are associated with the *core* instance being part of the Self-Organisation Coordinator and are therewith encapsulated. All subcomponents of the Coordination Mechanism Selector can be replaced or enhanced straightforwardly due to its modular structure.

In Listing 19.3, a simplified Python code example is shown that applies the SO Coordinator together with other RHBP components to define a behaviour model. In detail, the SO Coordinator instance is instructed (SO Goal) to instantiate a nested behaviour model that exhibits foraging. Moreover, the example illustrates that the SO Coordinator

19.5. Summary Combining Decision-Making and Planning with Self-Organisation

for foraging acts like a normal behaviour because it is based on the NetworkBehaviour concept, which enables to combine it seamlessly with other RHBP components. How the particular foraging mechanism is realised is separated and internally determined with the concept presented above in this section. The implementation itself is then applying the framework as presented in the previous section, for instance using the SoBuffer and GradientSensors.

Listing 19.3: Python example of the application of the SO Coordinator within a RHBP behaviour model definition.

```
# Creating a SO Coordinator instance that aims for exhibiting foraging
foraging = SoCoordinator(name='behaviour_name', id='robot_id',
                        params={'pose_frame': 'robot_pose_frame',
                              'motion_topic': 'robot_motion_topic'},
                        so_goal='foraging')

# creating a sensor instance bound to a ROS topic
example_sensor = TopicSensor(name="sensor_name", topic="/example")

# creating an instance of a specific activator, here targeting a threshold
example_activator = ThresholdActivator(thresholdValue=1.0)

# creating a condition
example_cond = Condition(example_sensor, example_activator)

# adding a precondition to the foraging, which acts as a normal behaviour
foraging.add_precondition(example_cond)

#defining a high-level effect for the self-organised foraging behaviour
example_effect = Effect(sensor_name=example_sensor.name, indicator=1.0,
                      sensor_type=float)

# effect is bound to the behaviour
foraging.add_effect(example_effect)
```

19.5. Summary Combining Decision-Making and Planning with Self-Organisation

In this chapter, we presented our solution for an integration of self-organisation algorithms into the decision-making and planning of general-purpose robots. In detail,

the presented approach relies on the developed library *so_data* for expressing arbitrary self-organisation mechanisms as abstract patterns based on a further extended gradient pattern as its primary representation. Furthermore, we described how RHBP enables a complete integration of our self-organisation library due to its dynamic state transitions and heuristic calculations for decision-making. The combined approach of implementing self-organisation based on *so_data* and general-purpose decision-making and planning with RHBP allows for a goal-directed application of self-organisation and provides a foundation for an automated selection of suitable mechanisms with the *Self-Organisation Coordinator*.

20. Increasing Adaptability with Reinforcement Learning

Extending the already adaptive and opportunistic properties of RHBP with additional machine learning features allows to further increase the self-adaptation capabilities of the entire approach. Such additional learning features allow to further relieve the system designer or engineer from the burden of designing a model of the system that considers all future situations the system will have to handle.

In particular, the class of RL algorithms is interesting because they enable continuous online learning and adaptation of a system during its execution. Common concepts and necessary backgrounds are provided in Chapter 13. In the remainder of the chapter, we go into detail about the integration of RL into RHBP as well as the particular realisation.

20.1. Integration Concept

We already discussed different methods that are used to implement RL in Chapter 13. In this context, we especially highlighted that the selection of algorithms depends on the particular learning problem, see Section 13.1.

The RHBP framework is a general-purpose framework for multi-robot systems that operate in partially-observable and dynamic environments. The goal of integrating RL

into RHBP is to improve the self-adaptation capabilities of the robots in general. For this reason, the RL implementation has to be flexible with respect to the actual learning problem and should avoid limitations to certain problem domains.

Following Sutton and Barto, 1998 a RL problem is a Markov decision process that is defined by the environment of the agent – *sensation*, the possible *actions*, and the *goals*. This can be directly mapped to the first class objects sensors, behaviours, and goals of the RHBP *core*. Here, a particularly important point is the goal-directedness of RHBP because the goal components can be used to automatically formulate required utility functions for the integration of RL.

Additionally, robotic environments in real-life are usually continuous and only partially-observable. This results in a very large search space. For this reason, a traditional Q-Learning approach is not applicable from the performance point of view because it would be difficult to store all the state, action, reward tuples. To address this challenge, ANN-based approaches can be used to approximate the continuous input state. Additionally, ANNs have the advantage that the policy application of an already trained model with forward propagation is very fast in contrast to the search in large Q-Learning tables. Such a fast selection of the action based on the policy is especially important in real-time environments that are common for robotics applications.

An additional requirement in the context of RHBP is a non-intrusive integration to keep the entire solution configurable and flexible for different application scenarios with an adjustable level of influence from the learning component.

The RHBP *core* implementation contains several possible starting points for the integration of additional RL capabilities. Several existing options with advantages and disadvantages are discussed in the following.

Learning effects The behaviour model requires the definition of behaviour effects. Effects describe a likely influence of the behaviour on certain sensors, respectively the sensed environment. Instead of modelling the effects, it would be possible to learn the effects through experience by observing the sensor changes after the execution of behaviours. This would enable to either start without any modelled effects or to continuously refine the effects of the robot model over time. Nevertheless, there exist some problems that would have to be handled. First, it is difficult to determine if the sensed environment changes are induced by the behaviour itself,

20. *Increasing Adaptability with Reinforcement Learning*

independent environmental changes, or results of former interaction that changed the environment to some extent. Secondly, the RHBP allows for the execution of parallel behaviours, which complicates the mapping between behaviours and effects.

Learning activation thresholds The behaviour network of the RHBP applies various threshold values, such as the global activation threshold that defines when behaviours are executed, thresholds in the activators, and thresholds that describe the level of fulfilment of preconditions (satisfaction). Learning thresholds would simplify the modelling and tuning of the system, but it is challenging to be implemented in practice. The activator and satisfaction thresholds are having a strong influence on the entire system stability and directly reflect the safety constraints of the system designer, automatically adjusting these values could result in very unpredictable results. Learning the global activation threshold would adjust the reactivity instead of using the current static statistical calculation. This is difficult because it cannot be directly expressed in a utility function and might lead to unexpected oscillations.

Learning activation weights Activation weights define the specific influence of the seven activation/inhibition sources (precondition activation, predecessors activation, successors activation, goals activation, goal inhibition, plan activation, conflicting behaviour inhibition), which are aggregated to the activation of a behaviour. These weights are used to define how opportunistic the system behaves and how strongly it pursues its goals. Continuous learning of these weights could relieve from manual tuning, even though the symbolic planner integration already minimises this issue. Nevertheless, this is particularly challenging because we cannot apply goals to generate a utility function as adjusting the weights has a direct influence on the capability of achieving goals at all.

Learning an activation source A less intrusive integration of learning can be achieved by integrating an additional eighth activation source that works similarly as the activation that is calculated from the planner results. Here, the learning system calculates an activation value independently of the behaviours that expresses the preference from former experience, which is then fed into the overall activation

calculation. An advantage of this solution is that the system designer can easily adjust with an additional activation weight how much influence the learning capability is having with respect to the other activation sources. Furthermore, the learning influence would not be able to overwrite safety constraints that are modelled in preconditions and effects.

The discussed advantages and disadvantages lead to the decision of integrating RL into the RHBP following the last option of learning an activation source. The clear advantage of this solution is that it avoids a completely unpredictable behaviour of the robot because the defined behaviour and goal model is still evaluated and used for decision-making. Similar to the integration of the symbolic planner, the RL component is rather working as an experience guide and not an ultimate decision-maker as in existing approaches like presented by Hester, Quinlan, and Stone, 2012. The resulting less intrusive integration is also supporting the concept of adjustable autonomy (Sierhuis et al., 2003) with policies controlling the scope of autonomous adaptation. Learning an activation source avoids the problem of all other options that potentially wrongly learned coherences could gain exclusive control of the system.

20.2. Algorithmic Integration

In the previous section, it has been argued that a suitable integration of RL is achievable with the integration of an additional activation source. This integration approach enables adjustable autonomy with respect to the learning component. Furthermore, it results in a solution that is similar to the planner integration more guiding the system with the learned model instead of having a solution that is fully relying on it. In the following subsections, we are iteratively developing the extension of the RHBP core activation algorithm. First, we start with the introduction of the additional activation source. Secondly, we discuss how this has to be extended to enable exploration strategies. Thirdly, we describe how rewards are automatically determined.

20.2.1. Activation Algorithm

In order to integrate RL into the existing activation algorithm, it is necessary to extend the formulas of the 7 core activation sources that have been discussed in Section 18.1.2.

20. Increasing Adaptability with Reinforcement Learning

The Equation 20.3 shows the additional activation formula that integrates the RL result. Here, the minimal and maximal reinforcement value overall currently executable behaviours, which is determined by the RL component, is used to normalise the activation within a step. By default, the highest learned reinforcement value is mapped to an activation of 1 and the lowest reinforcement (suppression) to -1. Finally, the RL activation is weighted with the weight β_{rl} that allows to configure the influence of the component.

a = activation; B = behaviour;

β_{rl} = weight parameter; AB = set of all behaviours

$rl(B)$ = Reinforcement of a B

max_a = max activation bound;

min_a = min activation bound;

$$min_{rl} = \min_{\forall b \in AB} rl(b) \quad (20.1)$$

$$max_{rl} = \max_{\forall b \in AB} rl(b) \quad (20.2)$$

$$a_{rl} = (max_a - min_a) \cdot \frac{rl(B) - min_{rl}}{max_{rl} - min_{rl}} \cdot \beta_{rl} \quad (20.3)$$

An advantage of the combination of RHBP and RL is that the declarative and model-based approach of RHBP can be used to guide a model-free RL approach. More specific, the defined conditions of a behaviour model can be used to direct the learning by eliminating selectable actions to avoid impossible, unwanted, or dangerous system behaviour. Moreover, such conditions also enable a faster converging because the system can benefit from the domain knowledge that was already implemented by the system designer. In consequence, the system does not have to learn itself actions that are not applicable in certain situations. Hence, using the defined conditions of the RHBP behaviour model is practically reducing the search space of the learning algorithm. In order to integrate conditions into the learning process, the model is trained with automatically generated additional high negative rewards for all behaviours in situations that are not executable due to their preconditions.

Equation 20.4 shows an adapted version of the initial Equation 20.3 that sets the

activation to the minimal activation value if a behaviour is not executable due to its preconditions. s_T describes the percentage of precondition fulfilment that makes a behaviour executable.

$$s = \text{satisfaction}; s_T = \text{satisfaction threshold of B}$$

$$a_{rl} = \beta_{rl} \cdot \begin{cases} (max_a - min_a) \cdot \frac{rl(B) - min_{rl}}{max_{rl} - min_{rl}} & s \geq s_T \\ min_a & \text{otherwise} \end{cases} \quad (20.4)$$

20.2.2. Exploration Strategy

A particular challenge in this work is how to enable exploration in the scope defined by the RHBP behaviour model. Following Equation 20.5 shows the finally implemented extension of Equation 20.4 that enables exploration independent of the actually used exploration strategy.

$$a_{rl} = \beta_{rl} \cdot \begin{cases} max_a & exploring() = \text{True} \\ (max_a - min_a) \cdot \frac{rl(B) - min_{rl}}{max_{rl} - min_{rl}} & s \geq s_T \\ min_a & \text{otherwise} \end{cases} \quad (20.5)$$

Here, the *exploring()* function returns True if an exploration should take place, which results in the maximal activation for the specific behaviour, while all other behaviours get the configured minimal activation value from RL. Nevertheless, the finally executed behaviour is selected by the manager component of the RHBP core by combining all activation sources to an overall behaviour activation; hence, an exploration may not be executed even though the learning component initiated an exploration. The not guaranteed execution of an exploration operation is limiting the exploration strategy to some extent but is also ensuring that the system is not exploring behaviour out of the scope defined by the system designer.

The implemented exploration strategy is following the action-value method, see also Section 13.5. The action-value method has been favoured over a more sophisticated Boltzmann Exploration to avoid the dependency on another exploration model or a

20. Increasing Adaptability with Reinforcement Learning

further increased dependency upon the RHBP behaviour model.

In particular, the epsilon-greedy algorithm has been implemented, which is depicted in Equation 20.6.

$$\text{exploring}() = \text{random}() < \epsilon \quad (20.6)$$

The ϵ is reduced over time until a minimum, which is configurable by the system designer. In general, the minimum will be close to zero.

Additionally, the implemented exploration strategy supports a pre-training stage. The pre-training stage is only randomly selecting behaviours to widen the search at the beginning of the learning. Using a pre-training stage can be especially useful if an application is first tested in simulation before it is deployed on real robots to first shape the model safely in simulation with a broad exploration before it is tuned on the real system.

20.2.3. Reward

Defining the reward for actions in RL is always challenging because there does not often exist an unambiguous reward, and several possibilities are available. In most cases, the system designer is engineering the most suitable solution corresponding to the desired state of the system. In RHBP, the desired world state is already defined by the *Goal* components that describe the desired state with conditions. The fulfilment of conditions is already quantified to floating-point values between 0 and 1 as well as the priority of the goals with positive numbers ascending from zero. In consequence, the difference between fulfilment and priority information can directly be used to deduce an abstract reward for each possible world state. In particular, the delta of the goal fulfilment change since the action has been executed is multiplied with the normalised goal priority and summed up across all goals. The goal priority normalisation is detailed in Equation 20.8, whereas the combination with the goal fulfilment is given in Equation 20.9.

$G = \text{goal}; B = \text{behaviour}$

$f = \text{goal fulfilment}; p_{\text{goal}} = \text{goal priority weight}$

$$p_{\text{range}} = \max(G_{\text{Prio}}) - \min(G_{\text{Prio}}) \quad (20.7)$$

$$p_{\text{goal}} = \begin{cases} \frac{G_{\text{Prio}}}{p_{\text{range}}} + (2 - \frac{\max(G_{\text{Prio}})}{p_{\text{range}}}) & \text{if } p_{\text{range}} \neq 0 \\ 1 & \text{otherwise} \end{cases} \quad (20.8)$$

$$\text{reward}(B)_{\text{goal}} = \sum_G (f_{\text{current}} - f_{\text{last}}) \cdot p_{\text{goal}} \quad (20.9)$$

20.3. Reinforcement Learning Extension Architecture

This section focuses on the architecture and realised components for the integration of the algorithms of the previous section into RHBP.

The architecture of the *Reinforcement Learner* component that is described in this chapter is illustrated in Figure 20.1. The figure is a refinement of the overall system architecture shown in Figure 17.3 but is focusing on the core aspects for a single agent without again showing other optional components like the *Self-Organisation Coordinator* and the *Delegation*.

The architecture shows that the *Reinforcement Learner* component consists of a *RL Activation Algorithm*, the actual *RL Algorithm*, the *Exploration Algorithm*, the *RL Transformer*, and the *RL-Parameter-Extension*. Particularly, the *Reinforcement Learner* is a modular extension to the RHBP core, whereas the RHBP core itself contains the *Manager* component that is handling the overall decision-making and planning lifecycle. Here, the decision is taken based on the activation algorithm, which we extend to incorporate RL through an additional activation source, see before in Section 20.2.1, in the *RL Activation Algorithm* component. The *RL Activation Algorithm* is an extended version of the common *Activation Algorithm* that was introduced in Section 18.1.2. A configuration parameter allows to configure the used activation algorithm individually for each manager component in multi-agent configurations or nested behaviour models. The *RL Activation Algorithm* is using a service interface to train the actual RL algorithm implementation or request the reinforcement for behaviours from a separated

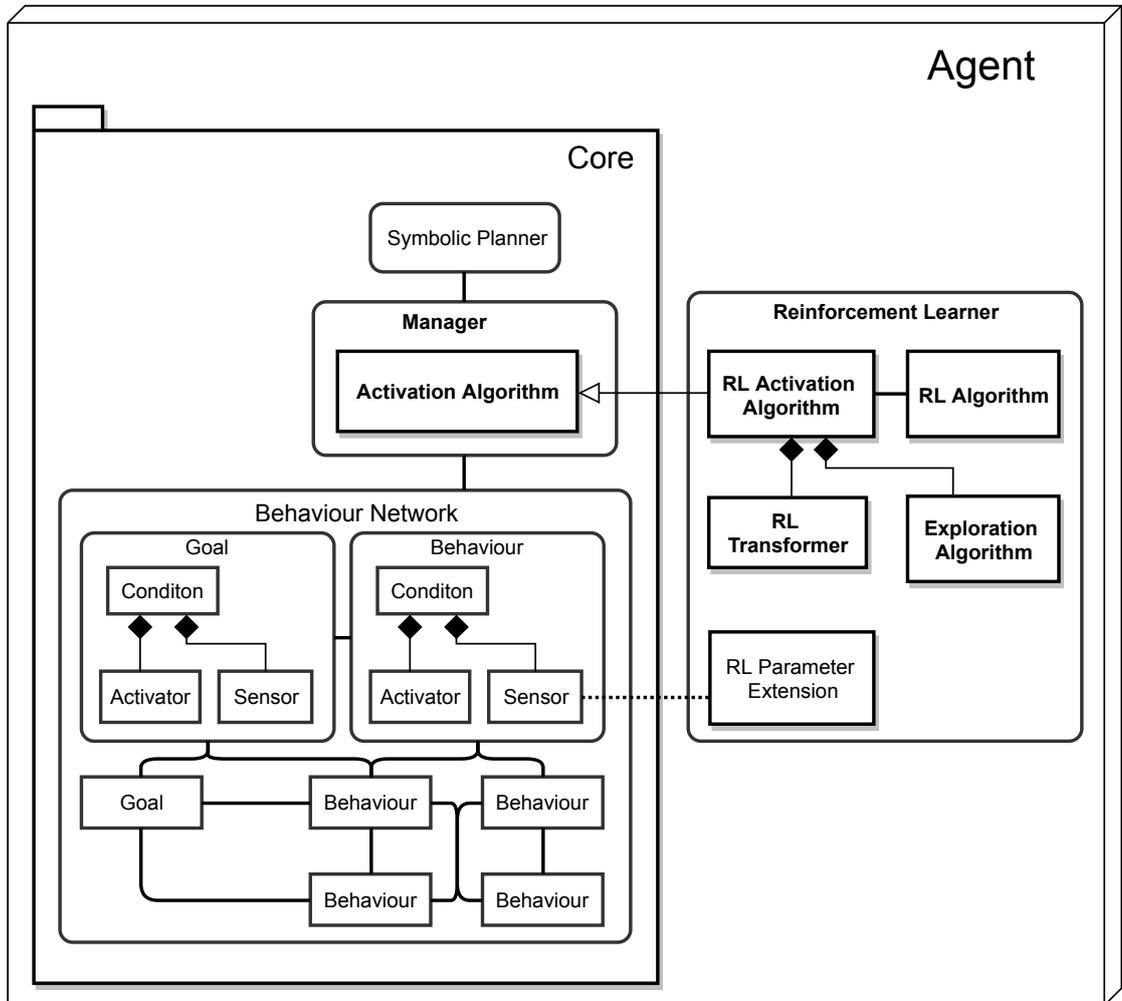


Figure 20.1.: Details about the Reinforcement Learner component within a simplified system architecture for a single agent.

RL Algorithm component. This component can either be encapsulated within the *RL Activation Algorithm* or executed as an external process. The latter option has the advantage that high computation load during the training updates is separated from the operational decision-making. The actual implementation of a *RL Algorithm* is entirely independent of RHBP and the addressed scenario. It is only receiving the input states with environment states, rewards, and possible actions as normalised and abstracted vectors. Information about the environment state and the action state of the system is necessary to calculate the reinforcement of a particular behaviour and to train the underlying ANN. Furthermore, the reward of an executed behaviour has to be determined. The conversion of all this information to abstract vectors is managed by the *RL Transformer* component.

The *RL Transformer* component is determining the *environment state* directly through sensor components in RHBP, which are linked to the (pre-)conditions of behaviours and goals. Sensor values have different data types, hence conversion is eventually necessary. Simple data types, such as Boolean, floats, and integers can directly be mapped. In particular, Booleans are only having two possible values, while floats and integers have an unambiguous rank order. Other data types like strings or integers encoding other nominal information have to be encoded to avoid unintended interpretations of the algorithms during learning. In the case of nominal values, it is necessary that the system designer defines a suitable encoding strategy or mapping. The additional encoding strategy can be defined in specific sensor extension modules named *RL Parameter Extension*. The *RL Parameter Extension* can be optionally connected to all sensor components. It allows to configure a one-hot encoding, which maps all nominal values to a vector of 0 or 1 with only the current value set to 1. An example could be a traffic light sensor encoding the state red, yellow, green in 1-3, which would result, e.g. in the vector [0, 0, 1] for green. This encoding avoids including any unintended ranking within the numbers. Nevertheless, a disadvantage is that the value range has to be known in advance. Furthermore, the *RL Parameter Extension* allows configuring if a specific sensor should be considered for learning at all.

Additional information about the environment is encoded in the RHBP model with preconditions of behaviours. These preconditions limit the problem scope and help the system designer to avoid unintended or dangerous behaviour of the robot. However, this

20. Increasing Adaptability with Reinforcement Learning

additional expert information about the environment is also used to potentially increase the learning speed by avoiding learning certain world properties that are already known in advance. The integration of the conditions is using the calculated wishes for each condition instead of the satisfaction information. The advantage of using *Wishes* is that the satisfaction is potentially less expressive because they are already further processed with *Activators*, which potentially simplifies the information. An example could be a condition with a *ThresholdActivator* and a distance sensor that defines the minimum distance between a robot and a human. Here, the satisfaction would discretise the information to just true or false, respectively 0 and 1, while the wish provides the full information about what distance change is needed to fulfil the condition.

The *action state* includes the executed and available actions, which are directly represented by the behaviours within the RHBP model. Every behaviour represents an action, and the executed behaviour represents the last executed action. Special consideration is necessary to handle the feature of RHBP to execute multiple behaviours in parallel, which is usually not considered in RL problems. One option would be to additionally learn the combination of behaviours, but this is cumbersome because it would largely increase the action space. The selected and more appropriate solution is to learn the model for each behaviour independently to implicitly favour combinations of behaviours because all would benefit from the same reward. Nevertheless, this approach has the disadvantage that combinations are not explicitly represented.

Instead of implementing the exploration algorithm directly within the *RL Algorithm*, it is decoupled into an own component named *Exploration Algorithm* as part of the *RL Activation Algorithm*. This modularisation allows to easily replace or select different exploration strategies. Moreover, it makes the *RL Algorithm* completely independent of the runtime of the system by abstracting information such as the number of learning iterations. The specific exploration strategy implementation corresponds to the formulas described in Section 20.2.2. Here, we first determine if the algorithm is triggering an exploration to avoid an unnecessary feed-forward calculation of reinforcement for a behaviour.

20.4. Reinforcement Learning Algorithm Implementation

As we already discussed, the optimal general-purpose RL method does not exist, rather the method has to be carefully selected for the given scenario considering the individual requirements and properties. Applying RL in the RHBP framework is intuitively indicating properties that are common for most robotic applications. The common properties are a sufficient handling of continuous input and output values. Continuous input values are required to handle arbitrary sensor information including various such different types as Booleans, integers and floats. Commonly the output values are also continuous to interface motor controllers, velocities, or angles. In fact, the last point with continuous output values is different for the integration into RHBP because we do not want direct end-to-end learning. Instead, RL is used to select behaviour implementations and not specific parameter values. In the RHBP approach, the behaviours are taking care of the calculation of continuous output if necessary. In consequence, the learning output values are discrete due to the limited behaviour set. Furthermore, in most robot scenarios, which have been evaluated so far with RHBP, the number of possible behaviours is rather limited. The number of behaviours is commonly below 20 behaviours, probably never having more than 100 behaviours even for very complex scenarios.

In order to address continuous input values with RL, an implementation based on ANN is a suitable choice because it allows to approximate the continuous value space efficiently. As we discussed earlier, the literature shows that for discrete output values value function algorithms are a preferable choice. Nevertheless, a particular value function implementation has to be able to handle integer and float inputs, which is not the case for all implementations. Furthermore, we want to calculate the positive and negative reinforcement for all executable behaviours and not just a discrete selection of one specific behaviour like many value function algorithms. Moreover, the implementation needs to deal with delayed rewards as it is the case for most robotic applications because it usually needs some time (several decision-making steps) until a taken decision shows its specific reward.

Handling delayed rewards is an intrinsic characteristic of all RL algorithms. Nevertheless, some approaches are able to cope with them more efficiently.

The mentioned properties and requirements are well covered by value-function-based

20. Increasing Adaptability with Reinforcement Learning

DQN approaches that have already been used in various research projects for complex tasks (Mnih et al., 2013; Silver et al., 2016). DQN is based on ANN and hence is able to handle arbitrary input values. Only nominal information such as provided by strings has to be encoded. Even though the above-mentioned research projects are using CNN architectures, DQN is not limited to them. Instead, the most suitable ANN architecture can be selected with respect to the input values and application scenario. In consequence, also fully-connected networks for general-purpose learning can be chosen to make DQN applicable for the integration into RHBP, too.

More details about the DQN approach in general, are given in the Section 13.4. This includes the presentation of several extensions that further improve the performance and applicability of DQN in the context of this dissertation. Particularly, the extension called DDQN is important as a foundation for the implementation of RL in this dissertation to avoid overfitting of the bias by differentiating between selection and evaluation of actions (Hasselt, 2010). Other existing extensions such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) are not relevant because they improve the applicability in a continuous action space, which is limited and discrete in the RHBP framework.

The following sections discuss in detail how the selected DDQN approach is integrated into RHBP.

20.4.1. Adaptation of the DDQN approach

The background and ideas of the selected DQN approach, respectively the extended version DDQN, which have been presented before, have to be adjusted to enable the integration into the RHBP framework. First, we focus on the components that are used for the initialisation. Secondly, the functions for applying the algorithm are described.

Initialisation

The base of the algorithm is a Q-Network implemented on the foundation of an ANN, another ANN to realise the *Target Network*, and an *experience buffer*. Both ANNs are identical during the initialisation. Depending on the configuration, they have an input and output layer that are connected by one or more hidden layers. This architecture can be configured depending on the scenario. The number of neurons on the input layer

corresponds to the number of encoded sensor values from the perception of the agents that describe the state of the environment. The number of neurons on the output layer corresponds to the number of available behaviours in the RHBP framework.

Functions

The Q-Network of the implemented DDQN is used in two ways. First, the feed-forward function is used to determine the activation (reinforcement) of behaviours/actions for a particular state. In most RL approaches, only the action with the highest activation is returned, but the integration into RHBP requires getting activation values for all behaviours. Secondly, another method is used to train the network. Training is triggered in configurable intervals independent of the RHBP decision-making cycle. The input for training is batches of randomly selected tuples {State, Action, Reward, Next-State} that are taken from the Experience Buffer, which continuously records the experience in a FIFO replay memory of configurable size. The Q-Network and the Target Network are trained simultaneously but independent of each other.

Updating the Q-Network is based on the expected future reward, which is determined with the following calculation.

1. Calculation of the Target-Q-Value with $\text{Target-Q-Value} = \text{Reward} + \text{Discount-Factor} \cdot \text{Double-Q-Value}$
2. The Double-Q-Value is calculated from the Q-Value of the Target Network for the expected next action (Next-State). The Double-Q-Value is used to update the Q-Network.
3. The loss function for the network is defined as the Squared Error between Target-Q-Value and the Q-Value output from the network. This loss function is minimised (depending on the configuration Adam optimiser or Gradient descent) to let the network converge to an optimal solution.
4. In the next step, the Target Network is trained and updated depending on the variable τ . τ describes the influence of a training step, which results in a smoother network update in small steps to avoid instability in the training process.

20. Increasing Adaptability with Reinforcement Learning

The described process and underlying network architecture are visualised in Figure 20.2. Many parameters such as the size of the experience buffer, batch size, training interval factor τ , training discount factor, number of hidden layers and cells are configurable to be customisable for the particular scenario. In contrast to the shown architecture, the less complex DQN approach would calculate the Target-Q-Value with the same network that is also used to determine the next expected action.

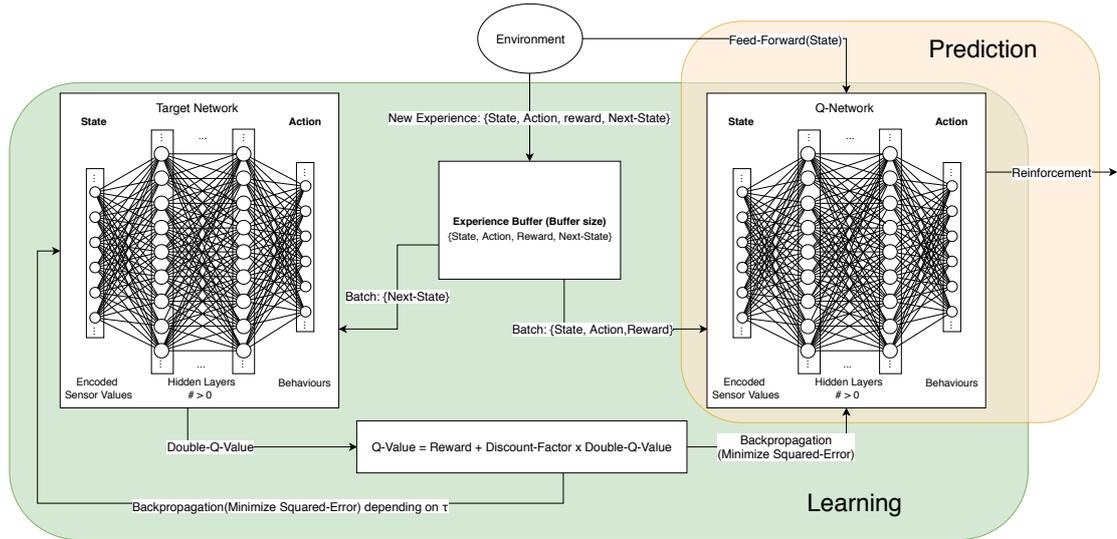


Figure 20.2.: DDQN learning and prediction architecture and process.

The implementation of the DDQN makes use of the popular open source machine learning framework TensorFlow to realise the ANN architecture. TensorFlow (Abadi et al., 2016) was selected because of its reasonable performance, flexibility to easily adjust it for specific use cases, and the available Python API that simplifies the integration into RHBP. Furthermore, TensorFlow can be executed on a Central Processing Unit (CPU) as well as a Graphics Processing Unit (GPU), which makes it applicable for robots of various types. Although other frameworks like Neon provide higher performance for specific use-cases, they lack flexibility (Abadi et al., 2016). TensorFlow provides a high-level API to define a *computational graph* with nodes corresponding to specific operations like matrix-multiplication. Nodes can have several inputs and outputs while the data flows alongside the edges of the graph. This approach allows to easily customise ANN architectures as well as to distribute the calculation to several devices.

20.4.2. Parametrisation

All machine learning approaches have in common that the particular result heavily depends on the chosen configuration. The time until the algorithm converges as well as the final result depends on various parameters. Most parameters are only influencing the required time, respectively the number of steps, until convergence. However, some parameters are also influencing the result the algorithm is selecting as the optimal solution. In the following subsection, we discuss various important parameters and their influence on the implemented RL approach. The parameters are classified into the groups: exploration, neural network parameters, and DDQN parameters.

Exploration

The implemented epsilon-greedy algorithm with pre-train phase for exploration has two important properties, the epsilon, and the duration of the pre-train phase.

The epsilon ϵ defines the probability of selecting a random behaviour instead of returning the reinforcement for the behaviours from the feed-forward calculation. Here, the epsilon is usually reduced over time to explore the action space more intensively in the beginning and reduce the tendency for exploration in a later stage when the learned model already stabilised. The rate of reduction over time, as well as the initial and final epsilon values, are configurable. A common configuration is to start with $\epsilon = 1$ to purely explore in the beginning. The final epsilon value is selected based on the specific application scenario. In a static learning problem without uncertainty and a dynamic environment, the final epsilon can be reduced until $\epsilon = 0$, which would result in no exploration at all after a certain time. Due to the fact that the RHBP framework is targeting (multi-)robot systems, scenarios will not be static. For this reason, a default final epsilon of $\epsilon = 0.1$ or $\epsilon = 0.05$ is a more suitable configuration to enable also the exploration and adaptation to changes in the environment in a later stage, even though the algorithm already converged.

The rate of reduction for the epsilon value depends purely upon the problem domain and has to be determined empirically. A general rule is that a more complex learning problem requires a slower reduction, whereas a less complex problem benefits from a faster reduction because of the smaller search space that would have to be explored.

20. Increasing Adaptability with Reinforcement Learning

The pre-train phase is optional in general and only influences the time to convergence, but some learning scenarios can benefit from this pre-filling of the experience buffer before the learning is actually started. Enabling a pre-train phase of a certain duration is especially recommended for problems that can be first tested in simulation and which have many input values (sensors) and large action space. Using a pre-train phase of a specific number of steps helps to stabilise the learning process. If the pre-train phase is configured too long, the learning process will be slowed down.

Neural Network Parameters

A very important parameter to influence the learning process of a neural network is the learning rate. The learning rate defines the rate of adjustment of the neural network weights with respect to new training data. A small learning rate results in minor adjustments of the weights, whereas a higher learning rate favours the faster adaptation of the weights during training. The selection of the learning rate depends again on the application scenario. If the scenario and environment are uncertain and dynamic, a higher learning rate supports faster adaptation. The disadvantage of a higher learning rate is that the learned model is less stable due to the possible massive changes of the network weights.

Furthermore, the chosen optimiser has an influence on the time to convergence for a neural network. The optimiser calculates the gradients of the cost function based on the learning rate, the cost function, and the weights. Here, the gradients define the direction of change of the weights during the learning process. Common optimisers are the stochastic gradient descent optimiser and the Adam optimiser, which is an extension of the gradient descent. The Adam optimiser requires less parameter tuning but requires more computation in comparison to the simpler gradient descent approach. Which optimiser works best in a certain scenario, has to be determined empirically.

Another configuration parameter for neural networks is the selected network architecture that can strongly affect the learning result. The literature does not provide clear answers or rules on how the optimal architecture can be chosen. A common approach is either to empirically determine a suitable architecture for a scenario or to use architectures that worked well in similar problems in the past. In general, the architecture is described by the number of layers, the number of neurons per layer and the type of

layers. A general recommendation is to first start with a simple network (smaller amount of hidden layers), which reduces the computation requirements, and increase the number of hidden layers if the achieved results are not sufficient. Due to the fact that RHBP is aiming for general-purpose learning, independent of special input sensors like images, the default configuration uses fully-connected layers and not CNN, which are common in computer vision tasks. The number of neurons for the input layer is given by the number of sensors and their encoding in the RHBP behaviour model, which are enabled to be used for learning, while the number of output neurons is defined by the number of behaviours in the system. The number of neurons for the hidden layers is configured by the user and should be in between the number of input and output neurons.

DDQN Parameters

Additionally to the neural network parameters, the implemented DDQN is based on, the approach has the extra parameters, buffer size, batch size, training interval and discount factor.

The *buffer size* defines the size of the experience buffer that corresponds to the number of past transitions that are stored in the FIFO buffer. A smaller buffer would only keep recent transitions that might result in getting stuck in local optima, which can be avoided by a larger buffer size. If the buffer size is configured too large, the learning process can be slowed down because of repeated training with transitions that did not direct in the right direction. Determining the optimal size is difficult as well and mostly depends on the complexity of the learning problem, which is defined by the number of input and output states, respectively the number of input sensors and behaviours.

The *batch size* parameter configures the number of transition tuples that are used for training in one training step. In general, a larger batch size should correspond to a faster convergence, but it can also destabilise the learning process due to larger changes (jumps) of the network weights in every training step. In consequence, it is possible that the algorithm is not able to converge to the most optimal solution. If the batch size is too small, the training is slowed down but is less prone to get stuck in local optima.

The *training interval* is the configuration for the training frequency in RHBP decision steps. Here, it is important to consider the dependence on the batch size, which should be larger than the training interval to use each transition multiple times on average in

the training. The training interval also depends on the problem scenario. If each action has a strong influence on the environment, the interval should be chosen smaller. If the training interval is too small, the algorithm is more prone to get stuck in local optima. Moreover, the training interval has an impact on the computational performance of the system and other components. If the interval is higher, the training is performed more often, which would require more computational resources if the same batch size is used.

The most important parameter for the DDQN is the *discount factor*; it determines the importance of future rewards in comparison to direct rewards. In consequence, the discount factor heavily affects which solution is considered as the optimal solution the algorithm is converging to over time. If the discount factor is chosen too high, the solution will oscillate around the optimum. If the discount factor is too small, the convergence will slow down. It is also possible to dynamically decrease the discount factor over time, similar to the epsilon of the epsilon-greedy algorithm in the exploration. Such a dynamic adjustment would result in first favouring actions with short-term reward because long-term rewards might not be known yet, whereas in a later stage with increased experience, the long-term reward would be favoured.

20.5. Reinforcement Learning Integration Summary

In general, decision-making and planning require a priori definitions of capabilities, rules, decision models, or world knowledge. Similarly, the RHBP approach, discussed in this dissertation, relies on a declaratively described behaviour model. Due to the challenge of handling the uncertainty of robot applications in dynamic and uncontrolled environments, such definitions or descriptions are always incomplete, hence the possible adaptation capabilities are limited.

The approach described in this chapter extends the RHBP solution consistently with reinforcement learning means through the integration of an additional eighth activation source. The presented concept guarantees through its strict separation of concerns that a developer can easily adjust and configure the specific reinforcement learning setup to meet the particular scenario requirements.

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

Decomposition is the capability of breaking down a complex goal into multiple less complex sub-goals. The RHBP core concept with the hybrid approach of a behaviour network guided by a planner allows already to dynamically decompose modelled goals into sequences of suitable behaviours, which can also be considered as some kind of sub-goal. This is providing a first level of adaptation through decomposition because it allows to dynamically select suitable behaviours considering the current context of the agent. At this point, the system is also able to select the most suitable behaviour from several alternatives.

Furthermore, considering the concept of *NetworkBehaviours*, which allows to encapsulate behaviour network models within one behaviour on a higher-level (see Section 18.5), as another kind of sub goals, we have another level of adaptation through decomposition. Here, a higher-level goal is decomposed into higher-level behaviours that consist of other behaviours and goals. While the set of behaviours, goals, effects, and conditions is statically defined, the particular decomposition into selected behaviours, which are used to achieve the given goal, is dynamically determined. The goal for nested behaviour networks is defined by the effects of the *NetworkBehaviours*.

Nevertheless, the decomposition with *NetworkBehaviours* has limitations as it is only working in a centralised RHBP model, e.g. a central model that controls multiple agents, and relies on a statically modelled behaviour network set with its precondition and effect dependencies. This can be especially cumbersome in practice with multi-robot systems because it limits parallelisms. Moreover, it is problematic if robots do not have persistent connectivity or in systems that have agents which are not working together continuously, e.g. if robots are added and removed during runtime.

The task decomposition, as described above, can be potentially enhanced by an automatic task decomposition that decomposes given goals dynamically in sub-goals independent of predefined behaviour sets. Such automatic task decomposition is especially beneficial and desired in situations where enabled goals are not achievable by one agent with its behaviour capabilities on its own. Here, it would be necessary to combine the

capabilities of several agents with suitable sub-goals for the individuals to achieve the common goal.

To fulfil a common goal that is decomposed into several sub-goals, it is necessary to allocate and delegate the decomposed tasks to other agents. An automated allocation and delegation of the decomposed goals is the foundation for an explicit coordination amongst agents. This is in contrast to the implicit coordination that is realised with the self-organisation extension of Chapter 19. Both approaches, explicit and implicit coordination, potentially increase the adaptability of a multi-agent system through the distribution of tasks to several identities, which makes the entire system more robust against failures of individuals. Nevertheless, implicit coordination with self-organisation algorithms cannot easily handle complex goals. This is especially the case if achieving a goal requires the cooperation of multiple agents over several independent decomposition levels. The term *multiple decomposition levels* means that goals require other sub-goals being fulfilled, with sub-goals that have to be decomposed again into lower-level sub-goals etc.

In this chapter, we will further discuss the implementation of the *Delegation* component that has been developed in order to extend the RHBP framework with an automated and dynamic task decomposition, allocation, and delegation for multiple (heterogeneous) agents. Due to the exceptionally dynamic nature of robot environments, the solution is especially considering robustness against failures of individuals as well as unreliable communication channels.

In the next section, we will look at the requirements that have been considered for the implementation of the framework extension.

21.1. Delegation Component Concept

Task decomposition is breaking down given complex and potentially not directly resolvable goals into sub-goals. In the following, we discuss possibilities and select the most suitable concepts, first for the decomposition, next for the allocation, and last for the delegation.

The state-of-the-art in task decomposition is usually focusing on a priori modelled decompositions. A priori modelling is limiting the system to decompositions the sys-

tem designer has considered beforehand, which results in limited adaptation capabilities. Especially in the multi-robot domain, we can expect a great variety of goals that are dynamically created. Even more, if we consider future robots working autonomously in uncertain real-life environments. In consequence, it is necessary to enable a dynamic decomposition of goals within the system that considers the currently available capabilities and assigned obligations.

In many scenarios, it is possible that several decompositions are available within in a multi-agent system. Selecting the best fitting decomposition is a challenge and needs to be addressed. Various different parameters for the selection of the decomposition with the best utility are imaginable, such as time, resource costs, and reliability. In most cases, selecting the most efficient solution that completes the goal in the shortest time is a reasonable approach. Nevertheless, it is necessary to keep in mind that selecting the decomposition might result in high computational costs, especially if many decompositions are available.

Another question is if the calculation and selection of the decomposition are done in a central system component that requests all necessary information or if a decentralised solution based on individual assessments is favoured. The central solution has the advantage of a more complete view on the problem, which potentially allows for a more optimal solution. However, this advantage comes with higher computational costs at a single point, including reduced failure safety, and the increased amount of exchanged information. A decentralised solution has the advantage of being more reliable against failures and reduced computational costs due to the implicit partitioning of the problem that is analysed by the individual agents considering only their local view.

The targeted environment of systems that apply RHBP are working in uncertain and complex environments. For this reason, a decentralised solution that does not require extensive a priori modelling of the decomposition is chosen. Moreover, the selection of the most suitable decomposition is based on configurable utility functions to support various different use cases.

The aim of the allocation is to fulfil a set of given goals within a multi-agent system. In the ideal case, all goals are fulfilled, if this is not possible either as many goals as possible are fulfilled or a certain set with respect to given priorities. Commonly, it is also a target to fulfil the goals, e.g. in the most efficient way with respect to the duration

or another different utility function.

In order to increase the efficiency of allocation in a multi-agent system, we need knowledge from the individual agents about their estimated costs to fulfil a goal as well if they are capable of achieving it in general. Costs can be the duration or other used resources. However, determining the cost is challenging, especially because it also depends on the decomposition and other allocated goals. The local estimation of costs is always an estimation of unknown quality because it is based on current knowledge and plans calculated in an uncertain domain, which has to cope with unforeseen changes. Nevertheless, we aim for a local estimation of goal costs with a reasonable quality considering the necessary computational costs. The dependency on the decomposition results from different possible sets of sub-goals that are potentially allowing to distribute and allocate to different agents with different costs. In consequence, the particular implementation is considering how far decomposition and allocation are combined.

After a suitable allocation to agents is determined, it is necessary to delegate the sub-goals to other agents finally. Here, we especially aim for robustness in terms of communication to foster the information exchange about new delegations, fulfilled delegations and failed delegations.

In general, the implementation aims for a solution that is able to handle frequent changes in a dynamic environment to increase the adaptability of the entire approach. However, it is not the goal to advance the state-of-the-art in multi-agent task allocation on the algorithmic level. Instead, it is the aspiration to show how suitable existing approaches can be integrated comprehensively into the existing RHPB framework.

21.2. Decomposition

We already discussed that the symbolic planner in RHPB is already doing a task decomposition on a first level for a single behaviour model. In order to apply it as well in a decentralised multi-robot scenario, respectively in a situation with multiple independent and distributed behaviour models, some extensions are necessary. In the current situation, behaviours represent the local capabilities of an agent. To extend this, it is necessary to determine which external behaviours are needed in order to solve the given problem.

The most dynamic approach would be to explore the search space with a symbolic planner and try to identify all edges (behaviours) between the states in a state graph that are missing or would provide an alternative path to find a solution. Unfortunately, this is a very challenging question because there is eventually an infinite number of behaviours that could expand the search space in the right direction. Furthermore, this could also lead to unintended potentially dangerous solutions that we want to avoid in a system that strives for adjustable autonomy. Moreover, such an overcomplete search requires much computation time, and none of the existing PDDL-based planners allows such insights.

Even though we are discussing the decomposition and the delegation independently in this chapter as far as possible, the overall approach combines the decomposition with the delegation by directly making use of the decomposition for the delegation similar as (Zlot and Stentz, 2005). An alternative approach would be to consider the decomposition independently first and allocate and delegate the atomic tasks in a second step. While such an approach is generally possible, it has the drawback of leading to very suboptimal solutions because the consideration of what is done is separated from when it is done. Here, we follow Zlot and Stentz (2005, p.1515) who states that ‘[...] it is not possible to know how to efficiently allocate complex tasks among the robots if it is not known how these tasks will be decomposed. Similarly, it is not possible to optimally decompose complex tasks before deciding which robots are to execute them (or their subtasks).’

Due to the reason that we want to guide the autonomy of our system, we need to provide some instructions about where decomposition and delegation are beneficial and desirable. For this reason, we introduce the concept of *DelegationBehaviours*. A *DelegationBehaviour* marks certain effects in a behaviour model as externally delegable. On that foundation, it is possible to model a part that can be dynamically executed by other agents, or in other words to model an externally operated sub-goal. This completely abstracts from how the required effects are achieved as well as how many agents are necessary to achieve it. The *DelegationBehaviour* provides an entry point for the automated allocation and delegation of goals. From the perspective of the behaviour model, the *DelegationBehaviour* operates like any other behaviour. For that reason, the *DelegationBehaviour* is considered in the symbolic planning and decision-making in the same way as any other behaviour. Similar to the base behaviour concept, a *Delegation-*

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

Behaviour can have preconditions to control the situations when it is allowed to be used, hence to control when decomposition can take place.

The introduction of the *DelegationBehaviour* provides more freedom for dynamically allocating and delegating task by reducing the amount of a priori modelled information in the system as well as the amount of information that is exchanged in comparison with task trees. Nevertheless, this approach is also limited in the adaptability in comparison to a computationally expensive complete free decomposition based on all possible behaviour effects because it is necessary to model a *DelegationBehaviour* explicitly. On the other hand, the explicitly modelled effects do also minimise the complexity of the decomposition by only enabling certain aspects to be decomposable and delegable, which is also reducing the required communication. Furthermore, this approach gives additional control about the system behaviour to the system designer, fostering the idea of adjustable autonomy.

Due to the fact that multiple *DelegationBehaviour* are allowed in a behaviour model, and each agent can have them, it is possible to have several levels of delegation as well as alternating behaviour execution between multiple robots. An alternating behaviour execution refers to situations where both agent's goal fulfilment is mutually dependent on one another.

The *DelegationBehaviour* is also allowing a dynamic adaptation during runtime with the known means of RHBP such as plan monitoring, replanning, and reinforcement learning, always based on the currently available sensor information.

While the concept of the *DelegationBehaviour* is a designated hook for a delegation within a behaviour network, another component called *DelegableBehaviour* provides means for an optional delegation. The *DelegableBehaviour* is a *DelegationBehaviour* that is also capable of achieving modelled effects itself locally, but in contrast to a normal behaviour, it is comparing its own local costs with alternative solutions from other agents using features of the *DelegationBehaviour* to find the most optimal solution.

Additionally, the component *DelegationGoal* can be used to inject goals from outside into a running multi-agent system. The *DelegationGoal* is not bound to a particular agent or behaviour network, it is a component that allows to use the delegation process to automatically allocate goals to the most suitable agent in a decentralised way. For instance, the *DelegationGoal* can be used to integrate an external user interface, which is

allocating goals to a group of agents instead of adding the goal to a particular behaviour network instance of a single agent.

In the next section, we discuss how the sub-goals represented by *DelegationBehaviours*, *DelegableBehaviours*, and *DelegationGoals* are allocated and delegated to other agents in a distributed fashion.

21.3. Allocation

An efficient allocation is a two-stage process. In the first stage, it is the target to determine an agent that can complete a goal most efficiently with respect to a certain utility. In the second stage, the actual goal is allocated and delegated to a particular agent.

21.3.1. Selecting an Agent

In order to determine the most suitable agent, it is necessary to find the costs for all available agents. To realise the cost estimation in a distributed manner, we apply a first-price sealed-bid auction algorithm (Dias et al., 2006), which is suitable because we consider only one sub-goal at a time and assume that the agents in our multi-robot system are cooperating trustfully.

Auction process

An agent with a *DelegationBehaviour* becomes an auctioneer after the *DelegationBehaviour* has been activated by the surrounding behaviour network. Likewise, the instantiation of a *DelegationGoal* provides means for starting and controlling auctions externally. The auction process is illustrated in Figure 21.1 from the perspective of the auctioneer agent. Here, the auction is initiated with a broadcast message, the so-called Call for Proposal (CFP). The CFP includes all necessary information about the sub-goal, which can then be used by other agents to estimate if they could help to achieve this goal by calculating their costs respectively bids. Bids are replied to the auctioneer, which waits for a configurable time to close the auction. Alternatively, it is possible to configure the amount of minimally replied proposals to wait for. If the agent is not able

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

to support the goal, it is not replying to minimise the amount of exchanged messages. The participating agent is not storing any information about the auction and would also reply to other parallel auctions during the auction time. After the auction time is exceeded, which is calculated in decision steps or returned proposals, the auctioneer is selecting the bid with the lowest costs and informs the bidding agent with a *PRECOM* message that it would be the desired delegate. If no bid was received during the auction time, the agent repeats the CFP. The *PRECOM* includes the same information as the CFP plus the successful bid of the agent. In consequence, the auction process on the site of the participating agent is completely stateless, which increases the robustness of the solution. Based on the information in the *PRECOM*, the agent is estimating its costs again because the environment might have changed in the meanwhile. Next, the agent acknowledges the *PRECOM* to the auctioneer if the costs are smaller or equal to the initial bid. If the costs are higher, the agent replies with a new bid to give the auctioneer the opportunity to select a different agent with lower costs. After the *PRECOM* is acknowledged, the agents start the delegation process.

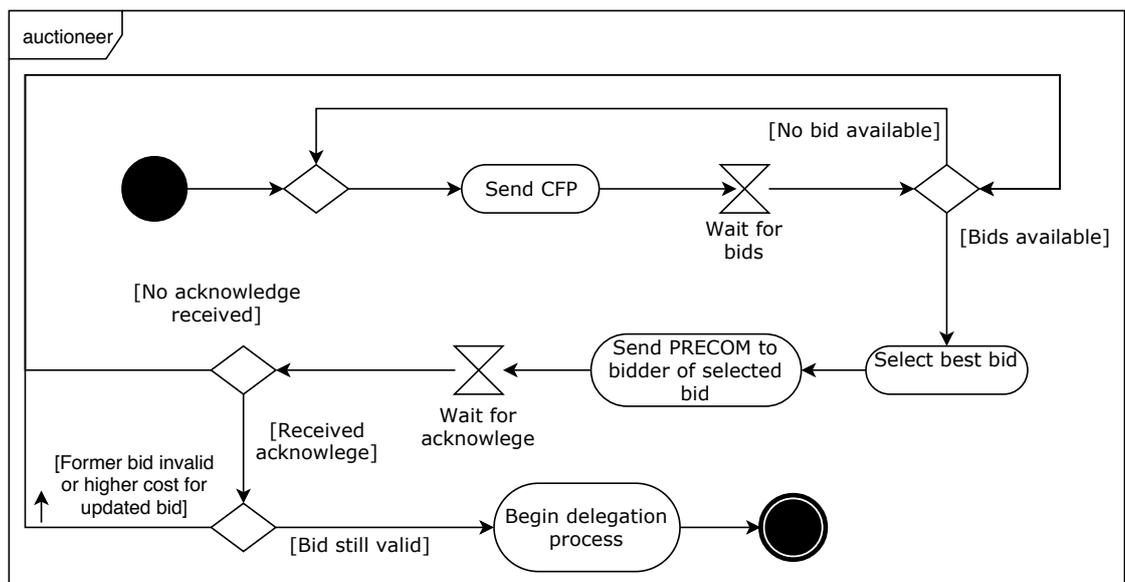


Figure 21.1.: The process of the auctioneer visualised as UML activity diagram. *Wait for bids* considers either auction time or the number of returned proposals depending on the configuration.

The perspective of the participating agent is visualised in Figure 21.2. The diagram

shows that the participating agent is using its planning component to estimate the costs of achieving the broadcasted sub-goal. The particular implementation is using the symbolic planner within the RHBP framework to estimate the general feasibility and the required costs. More details about the cost estimation will be provided in the following subsection.

Cost estimation

The foundation for local cost estimations and creation of a bid for the *PROPOSE* message are planning results determined with the symbolic PDDL planner of RHBP.

The information sent with the CFP or *PRECOM* message is used to create temporary goal conditions that are fed into the PDDL planner together with the domain knowledge that is given by the behaviours of the agent and its current goals. The planning process is executed independently and in parallel in an own process to avoid interference with the decision-making and planning of the RHBP core. In fact, the planner is used to calculate two different plans, one plan that tries to achieve only the sub-goal from the CFP or *PRECOM* and one that tries to achieve already existing goals together with the sub-goal. The first plan is considered as the *simple plan*, the second plan is named *full plan*. Additionally, the currently followed plan of the agent, the *base plan*, is evaluated as well.

If the *simple plan* or *full plan* cannot be achieved with a plan, the proposed sub-goal is considered as not fulfillable, and the agent is normally not replying with a *PROPOSE* or is cancelling its participation in a *PRECOM-REPLY*. Nevertheless, an alternative configuration allows for replying with proposals of infinite costs to enable monitoring the number of participating agents.

The three different planning results are combined in Equation 21.1 to provide an overall cost estimation considering different cost factors reflecting influences like the number of involved agents, the additional amount of required work, and nested delegations. Depending on the scenario, it is possible to balance the specific cost factors with the weights f_x with $x \in \{tu, wp, aw, ad, ca, cn\}$. Here, $f_x > 0$ increases the influence of a weight of a cost factor, with $f_x = 0$ a factor is neutral, and $0 > f_x > -1$ can be used to reduce the influence of a factor. The different weights make the cost estimation highly configurable.

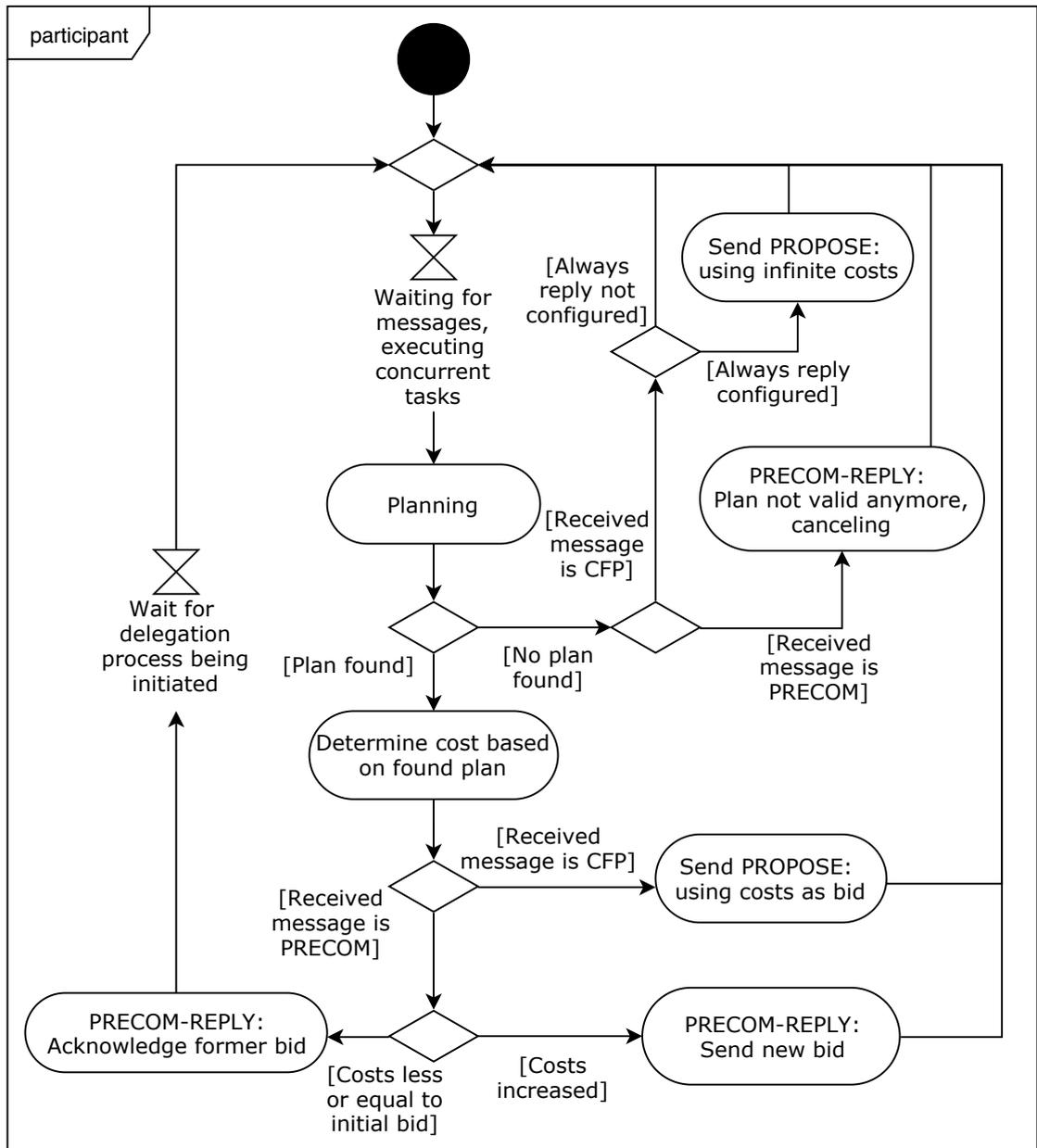


Figure 21.2.: The process of the auction participant visualised as UML activity diagram.

$$\begin{aligned}
cost &= steps_{fullplan} & (21.1) \\
&\cdot \left(1 + f_{tu} \cdot \frac{t_{curr}}{t_{max}}\right) \\
&\cdot \left(1 + f_{wp} \cdot \frac{steps_{simpleplan}}{steps_{fullplan}}\right) \\
&\cdot \left(1 + f_{aw} \cdot \frac{steps_{baseplan}}{steps_{fullplan}}\right) \\
&\cdot \left(1 + f_{ad} \cdot newdelegation \cdot \frac{d_{curr}}{d_{max}}\right) \\
&\cdot \left(1 + f_{ca} \cdot \frac{steps_{delegations}}{steps_{simpleplan}}\right) \\
&\cdot \left(1 + f_{cn} \cdot newcontractor \cdot \left(1 - \frac{c_{curr}}{(d_{curr} + 1)}\right)\right)
\end{aligned}$$

The main cost is always given by the number of steps for the $steps_{fullplan}$, all other factors are used to tune the result towards a desired direction.

In detail, the first factor, *Task-Limit-Utilisation-Factor* $\left(1 + f_{tu} \cdot \frac{t_{curr}}{t_{max}}\right)$, adjusts the costs regarding the limit of maximum accepted external sub-goals. t_{curr} is the number of maximum accepted goals; t_{curr} the number of currently accepted goals.

The *Workload-Proportional-Factor* $\left(1 + f_{wp} \cdot \frac{steps_{simpleplan}}{steps_{fullplan}}\right)$, models the fraction of required work (number of steps) for the new sub-goal within the *full plan*.

The *Additional-Workload-Factor* $\left(1 + f_{aw} \cdot \frac{steps_{baseplan}}{steps_{fullplan}}\right)$ determines the fraction of additional work induced by the new sub-goal within the *full plan*. If many additional steps are needed to fulfil the new sub-goal, which is integrated into the existing plan, the costs increase.

The *Additional-Delegation-Factor* $\left(1 + f_{ad} \cdot newdelegation \cdot \frac{d_{curr}}{d_{max}}\right)$ can be used to adjust the costs according to additionally required delegation dependencies. It allows to configure if additional dependencies induced by additional nested delegations are desirable or not. Here, d_{curr} is the current delegation depth, d_{max} the configured maximum delegation depth and *newdelegation*, which is 1 if at least one *DelegationBehaviour* or *DelegableBehaviour* is part of the *simple plan*, otherwise *newdelegation* = 0.

The *Cooperation-Amount-Factor* $\left(1 + f_{ca} \cdot \frac{steps_{delegations}}{steps_{simpleplan}}\right)$ reflects the costs by the number of involved agents, hence it determines if spreading the labour or minimising the number of involved agents should be favoured. $steps_{delegations}$ is the number of *Delega-*

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

tionBehaviour or *DelegableBehaviour* steps within the simple plan. The difference to the *Additional-Delegation-Factor* is that it is only considering delegations on one delegation level, while *Additional-Delegation-Factor* considers multiple nested delegations.

The last factor, the *Contractor-Number-Factor* $(1 + f_{cn} \cdot newcontractor \cdot (1 - \frac{c_{curr}}{d_{curr} + 1}))$, models the costs induced by involving additional agents in the delegation in reference to the current delegation depth d_{curr} . In consequence, the costs would stay the same for additional delegation within the same group of agents but would increase for newly involved agents depending on the delegation depth. d_{curr} is the number of currently involved agents, $newcontractor = 1$ if the agent is not yet involved in the delegation, otherwise $newcontractor = 0$.

21.3.2. Delegation

After the auction process is finished and an agent is selected for the execution of a sub-goal based on the estimated costs of each agent, the agent acknowledges its delegation with the *PRECOM-REPLY* message. Next, the actual delegation of the sub-goal is processed. At this point in time, the auctioneer becomes the contractee, and the auction participating agents becomes the contractor. Now, the contractor is responsible for fulfilling the delegated goal.

The actual delegation makes use of existing RHBP functionality. The contractee is just remotely registering a new goal at the manager instance of the contractor. This allows the contractee to check the current state as well as the progress. Additionally, the contractor is using a *FAILURE* message to notify the contractee in case it is no longer capable of fulfilling the goal, for instance, due to changed environmental conditions. Another feature to monitor the delegation and improve the robustness is the *CHECK* message, which is continuously sent from the contractee to the contractor to check if it is still responsive and working. If the *CHECK* is not successful for a configurable amount of time, the contractee considers the delegation as failed and assumes the delegated sub-goal will not be fulfilled. In such a case, the complete allocation and delegation process has to be repeated.

21.3.3. Communication

The implemented communication protocol is aiming for robust communication that is able to work with agent failures, lost messages, and timeouts. The implementation is based on the common communication features of the ROS framework.

ROS communication is based on the Transmission Control Protocol (TCP), which ensures the retransmission and session handling on the network package level. In our implementation, we apply ROS services for all direct bidirectional communication to have a guaranteed response or to be able to react to timeouts. *PRECOM* and *PRECOM-REPLY* are implemented as ROS services because they are crucial for the stability of the system. Moreover, they are combined in one service utilising the ROS Service-Request and Service-Response to speed up the communication by reducing the overhead.

The communication involving multiple agents is implemented with ROS topics. In particular, the CFP is implemented with ROS topics because it is a broadcast that does not require a direct answer and which would not destabilise the system if it gets lost. In the worst case, a lost CFP would only result in a less optimal or repeated delegation. The *PROPOSE* is also implemented as a ROS service, even though the guaranteed response is not required but it simplifies the implementation by avoiding to create new topics for each new auction, where only two agents would be involved in the communication.

Multiple auctions and auction participations can be operated in parallel because the related messages can be distinguished with the transferred auction and goal IDs.

Table 21.1 summarises all the messages mentioned in the auction process description of Section 21.3.1, like CFP, *PROPOSE*, *PRECOM*, *PRECOM-REPLY*. Furthermore, the table includes additional messages that are used during the delegation process after the most suitable agent has been determined, such as *SEND-GOAL*, *REVOKE-GOAL*, *FAILURE*, and *CHECK*. Because all communication during the actual delegation process is important for the stability of the system, all related communication is implemented on the foundation of ROS services. The *CHECK* implementation is a special case, here we are only using the first part (handshake) of the ROS service process that checks if the service is accessible to minimise the communication between the agents.

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

Table 21.1.: Message types with descriptions, most relevant content parameters, and ROS realisation.

Message type	Description	Parameters	ROS realisation
CFP	Call for proposal: Requesting bids for a goal	Target conditions, auction ID	Topic
PROPOSE	Proposed costs as reply to specific CFP	Bid, auction ID	Service
PRECOM	Tentative accept of a PROPOSE, request for acknowledgement	Target conditions, accepted bid, auction ID	Service-Request
PRECOM-REPLY	Acknowledgement / Cancellation of a tentative PRECOM	Acknowledgement / Cancellation, updated bid	Service-Response
SEND-GOAL	Delegation begin	Goal	RHBP-Service
REVOKE-GOAL	Delegation end, revoked delegation	Goal ID	RHBP-Service
FAILURE	Goal not responding timeout	Goal ID	Service
CHECK	Connection and response test message	-	WaitForService() function

21.4. Task Delegation Component Architecture

The delegation component consists of two modules. First, the *delegation_module* that covers a RHBP independent implementation of our first-price sealed-bid auction algorithm and the corresponding communication means. The *delegation_module* is implemented on the foundation of the ROS framework and could also be used stand-alone in combination with other ROS task planning frameworks. Secondly, the *rhbp_delegation* module consists of a RHBP specific extension that is implemented on the foundation of the *delegation_module* and the RHBP core API. An overview of the architecture is depicted in Figure 21.3, additional details are explained in the following subsections.

21.4.1. Delegation Module

The *DelegationManager* forms the core of the *delegation_module*, this class handles the entire auction process, including the ROS communication, as well as the high-level management of the following delegation. This class is used for the implementation of the

21.4. Task Delegation Component Architecture

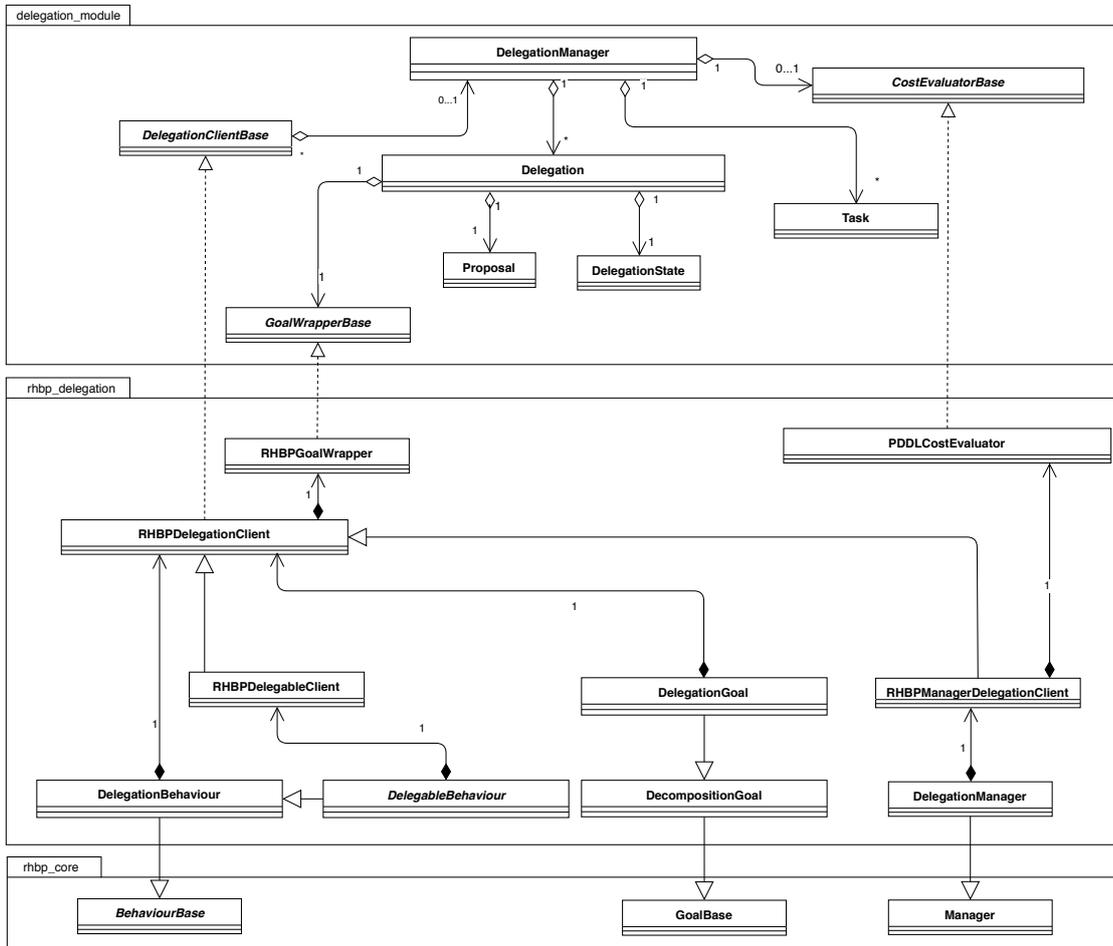


Figure 21.3.: Object-oriented delegation module and RHPB integration architecture visualised as UML class diagram with additional package scope.

21. Task Decomposition, Allocation and Delegation in a Multi-Robot System

auctioneer as well as the auction participant. In order to reduce redundancy and increase the performance, only one *DelegationManager* has to be instantiated per ROS node.

The particular delegation management is implemented in the *Delegation* class, which are also holding the bids of the agent's proposals. Already delegated goals are managed as *Task* objects by the *DelegationManager*.

The *CostEvaluatorBase* and the *GoalWrapperBase* are abstract classes that have to be extended specifically for the used task planning framework. *CostEvaluatorBase* is covering all cost estimations for creating bids of a certain goal. The *GoalWrapperBase* represents the abstract goal understanding of the delegation_module and enables a seamless integration into various frameworks.

The interface for implementing the contractee and contractor is provided by the abstract class *DelegationClientBase*. It allows to initiate new auctions and is the interface for all agents that are participating in auctions. The particular *DelegationClient* implementation is also managing currently allocated goals and has to be used to inform the *DelegationManager* about finished or cancelled delegations. Furthermore, the *DelegationClient* is responsible for the registration of a framework-specific *CostEvaluator*.

21.4.2. RHBP Decomposition

The module *rhbp_decomposition* covers all specific extensions of *delegation_module* that are necessary to integrate the delegation component into RHBP.

The costs for an announced delegation within an auction are determined with the *PDDLCostEvaluator*. It implements the cost function with its configurable weights that has been explained in Section 21.3.1. The *PDDLCostEvaluator* is making use of an interface in the RHBP manager that allows to create plans with the symbolic PDDL planner of the framework independent of the current operation but using current environment information and already allocated goals.

The *RHBPDelegationClient* integrates the *DelegationClientBase* into RHBP. It includes all common *DelegationClient* RHBP-specific code and is used to create an instance of the *DelegationManager* if no instance is available on the ROS node. Moreover, it is registering itself at the *DelegationManager* and creates the *RHBPGoalWrapper*. The *RHBPDelegableClient* is a special version of the *RHBPDelegationClient* that allows to consider also costs of the behaviour itself, to enable the optional local execution of

the behaviour instead of the explicit delegation. The *RHBPMangerDelegationClient* is another extension of the common *RHBPDDelegationClient* that provides the interface to the RHBP manager component. In particular, it is responsible for registering the *PDDLCostEvaluator* for PDDL-based cost estimations.

Furthermore, the default RHBP manager component is extended to the *DelegationManager*. It is necessary to always use this special version of the manager in a scenario where delegations are used. The *DelegationManager* instance is used to create a *RHBPMangerDelegationClient* instance that is necessary for getting access to the manager information. In particular, this information is used to monitor the goal fulfilment and to measure the delegation progress based on the decision-making steps of the manager. The information about goals like finished, cancelled, and current progress are crucial information for the *DelegationClient* implementation, which is integrated by the *DelegationManager*. An alternative implementation could also integrate the *DelegationManager* through ROS service hooks, which would result in a more decoupled implementation, but this would induce more communication overhead.

The *DelegationBehaviour* extends the RHBP behaviour base class and functions as a normal behaviour. *DelegationBehaviour* enables the integration of the delegation into the RHBP behaviour model. It initiates and processes the allocation and delegation process once it is activated. In order to provide additional freedom for modelling, it is able to handle different types of effects that are either considered as targeted sub-goal for the delegation or which are used only locally in the behaviour model to describe, e.g. effects on a different logical level. If a *DelegationBehaviour* is stopped by its surrounding behaviour networks, respectively the corresponding manager, all currently running delegations are stopped as well. The *DelegationBehaviour* is stopping itself in case the delegated sub-goal is fulfilled. As already mentioned, the *DelegableBehaviour* is a special version of the *DelegationBehaviour* that is also executable itself with designated costs but which is trying to find a potentially more efficient solution through delegation first.

The concept behind the *DelegationGoal* was also already explained before, it can be used to inject new goals for the delegation from external systems that are not directly modelled or considered in RHBP. Due to the fact that the *DelegationGoal* is not running in the *RHBP* decision-making lifecycle, the included delegation process has to be triggered externally. For this reason, the external system has to start manually the

delegation as well as it has to update continuously the module to indicate elapsed time similar to the step-function in the RHBP manager. Nevertheless, the targeted conditions for a goal have to be modelled with common RHBP domain model objects using conditions, sensors, and activators. After the delegation process is finished, the *DelegationGoal* is registering the required RHBP goals in the manager of the agent that won the auction.

21.5. Summary Decomposition, Allocation, and Delegation

The presented solution extends RHBP with an explicitly coordinated task decomposition, allocation, and delegation. The implemented delegation module is tightly integrated into RHBP through additional behaviour types. Particularly, the new behaviour types *DelegationBehaviour* and *DelegableBehaviour* function as objects for modelling an (optional) decomposition as well as hooks for initiating the actual allocation and delegation. Furthermore, the *DelegationGoal* allows for injecting goals from external components, such as other agent behaviours or user interfaces. The realised modelling enables a reasonable control of the decomposition by system designers and avoids a complex and complete modelling in advance. In detail, the allocation itself is based on the first-price sealed-bid auction algorithm, which enables distributed decompositions in other agents and the distributed calculation of costs for achieving goals. Moreover, the implemented cost function is providing a generic foundation for adjustments towards a specific scenario. The function can easily be configured using several weight factors. All in all, the implementation focuses on a robust communication with minimised communication efforts to address the requirements of dynamic and uncertain multi-robot environments.

Part V.

Application and Evaluation

The realised RHBP framework consists of various building blocks separated into several modules and specific functions. This part evaluates the realised RHBP framework in various use-cases and example applications. First, Chapter 22 focuses on diverse robot simulation experiments with different simulation engines for testing general or specific functionalities of the implementation. Secondly, in Chapter 23 we have a closer look on more complex applications realised with RHBP in different research projects that proves the applicability in practice. Furthermore, the complex applications provide examples of possible integration architectures in comprehensive software ensembles. Thirdly, Chapter 24 is giving a classification of RHBP against related frameworks.

Most of the presented evaluations are based on the corresponding publications that have introduced the particular component or application scenario but have been comprehensively revised and extended, which includes, for instance, new or recreated figures, additional details and comparisons, as well as updated evaluation results using a further elaborated framework version. In detail, Chapter 22.1 and Chapter 22.2 are based on (Hrabia, Wypler, and Albayrak, 2017), the foundation for Chapter 22.3 is (Hrabia, Kaiser, and Albayrak, 2017). The Chapter 23.1 about the project SpaceBot Cup is based on (Hrabia et al., 2017) and Chapter 23.3 is substantiated by (Hrabia et al., 2018a, 2019) but extended with an additional experiment incorporating the delegation extension of RHBP. Finally, the Chapter 23.2 covering the two years of contest participation has its foundation in (Hrabia et al., 2018b) for the sections discussing the participation in 2017 as well as in (Hrabia et al., 2020) and (Hrabia, Ettliger, and Hessler, 2020) for the sections related to the two teams that participated in 2018.

22. Experiments

The experiments presented in this chapter use rather simple simulation environments with limited complexity to test the core functionalities of the RHBP framework in an isolated fashion within a controlled environment. This allows it to check and analyse the implementation without interferences and issues induced by the environment.

First, we compare our solution against its ancestor in Section 22.1 by implementing

the same artificial test scenario as Maes, 1989 has used for evaluating her purely reactive behaviour system. This experiment is a first proof-of-concept of the RHBP core module and tests its capability for general-purpose decision-making and planning. In detail, this experiment is already showing the successful realisation of the requirements *R1*, *R7*, and *R8*, which reflect the formerly proposed objectives *O1*, *O5*, and *O6*. Furthermore, the experiment shows the superior performance of the further developed hybrid approach against the original purely reactive behaviour network of Maes. Next, Section 22.2 explores the applicability of RHBP for lower-level controls and is the first experiment that considers multi-robot control to also support above mentioned requirements and objectives in such an environment (*O4*). Subsequently, Section 22.3 illustrates the application of the RHBP self-organisation extension with another simulated multi-robot example scenario to prove the feasibility of combining decision-making and planning with self-organisation, which directly validates the realisation of the requirements *R4* and *R5* that are based on *Objective O3*. Lastly, Section 22.4 confirms the expedience of the RL integration in a revisited and extended version of the Maes' scenario already used in Section 22.1. This underlines the achievement of *Requirement R2* and the corresponding *Objective O2*. Due to the fact that all experiments make use of ROS and Python, they all support the *Requirement R9*.

22.1. Maes Automation Scenario

The original evaluation scenario from Maes, 1989 is a simulated artificial automation scenario. We revisit this scenario in an initial proof-of-concept to test the feasibility of our approach in terms of general-purpose decision-making and planning. The scenario consists of a robot with two arms, which has the goal of sanding a board and spray-painting itself. After the robot has spray-painted itself, it is no longer operational, which is a precondition to sanding the board. Hence, the robot needs to sand the board before it spray-paints itself to fulfil the complete mission. Additionally, there is a vice at the workplace to place in the board. Furthermore, a board, a sander, and a sprayer are reachable by the robot's arms and have to be grasped to execute certain actions. Possible plans for solving the task would be to pick up the board and sander, sand the board, put down either of it, pick up the sprayer in the free hand and spray-paint itself.

22. Experiments

Alternatively, the robot could use the vice to avoid putting the sander or board down. It is important to note that the robot must not spray-paint itself before it has sanded the board, although this action would directly lead to the fulfilment of one goal. Hence, the system has to handle conflicting goals that require a particular order of execution.

In order to keep our solution comparable, the same behaviours have been implemented as in the original scenario. These are *PickUpSprayer*, *PickUpSander*, *PickUpBoard*, *PutDownSprayer*, *PutDownSander*, *PutDownBoard*, *PlaceBoardInVice*, *SandBoardInHand*, *SandBoardInVice*, and *SprayPaintSelf*. The simulation is discrete with all behaviours executing instantly and completing their action within one decision-making cycle. Behaviours are only having an effect on the environment state if their particular preconditions are met, such as having a board in hand to be able to put it down. If a precondition is violated the system does not fail, only execution will take more time. However, the state representation is slightly different from Maes regarding the point that we model the used hand for picking an item explicitly. Although one particular goal of the original scenario was to show the resource management abilities of the planner, Maes' state representation makes this challenge relatively simple for the planner by having two indistinguishable hands. Thus, the system does not have to track which of the two hands can carry out the correct operation. Maes' simplification allows *a* hand to put down an object as long as it is currently located in *any* of the two hands. Our extended scenario implementation also applies the new RHBP features of numeric sensor values and conditional effects in order to minimise the state space in comparison to pure Boolean state representations and less abstracted behaviours. To track the location of objects, its position is encoded in an integer value: 0 means that the object is not in any hand, 1 means that it is grabbed by the left hand, and -1 means that it is located in the right hand. These values determine the conditional effect that putting down an object has on the hand-occupancy sensors. The modelled RHBP solution with behaviours, goals and corresponding preconditions and effects is visualised in Figure 22.1. The diagram visualises how the particular RHBP components are connected with each other and how the behaviour network is spanned between behaviours, goals, and sensor through the connection of effects and preconditions. In the remainder of this part, we will always use the same diagram type to allow for comparison amongst the different scenarios and experiments. The diagram type is explicitly developed for RHBP and called behaviour

model diagram.

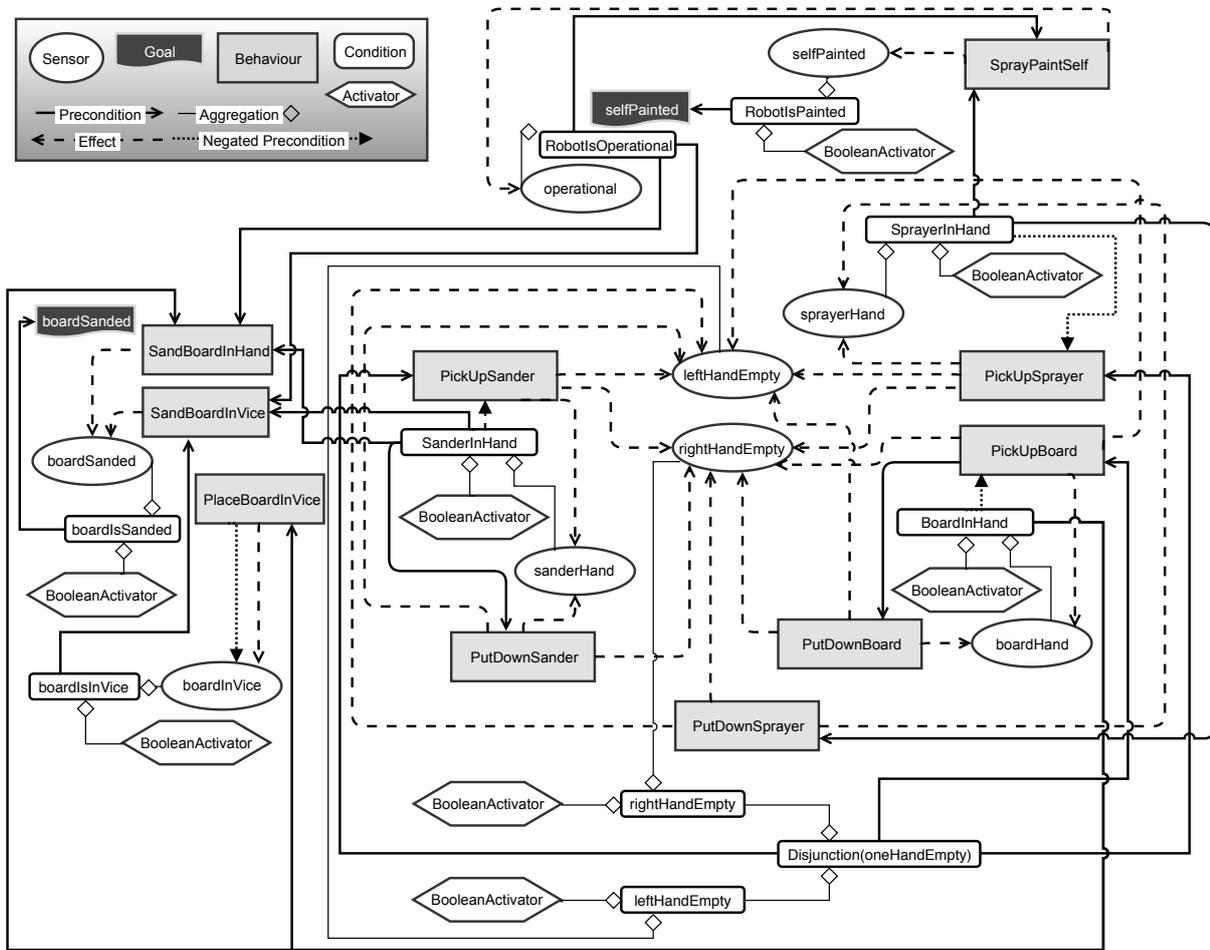


Figure 22.1.: RHBP behaviour model of the Maes' scenario.

Figure 22.2 shows the activation of each behaviour at the end of the planning step before action selection is performed. Behaviours above the black activation threshold may be selected for execution if they are active, their preconditions are fulfilled, and they do not conflict with any other already running behaviour. At the end of their execution, the activation is reset to zero. In this particular scenario, all behaviours are finishing instantaneous so that their activation is reset before the next planning iteration starts. This is clearly visible in the plot as a sudden decrease of activation and marks the point of activation clearly. This results in a behaviour sequence of *PickUpBoard*, *PickUpSander*, *SandBoardInHand*, *PutDownBoard*, *PickUpSprayer*, and *SprayPaintSelf*.

22. Experiments

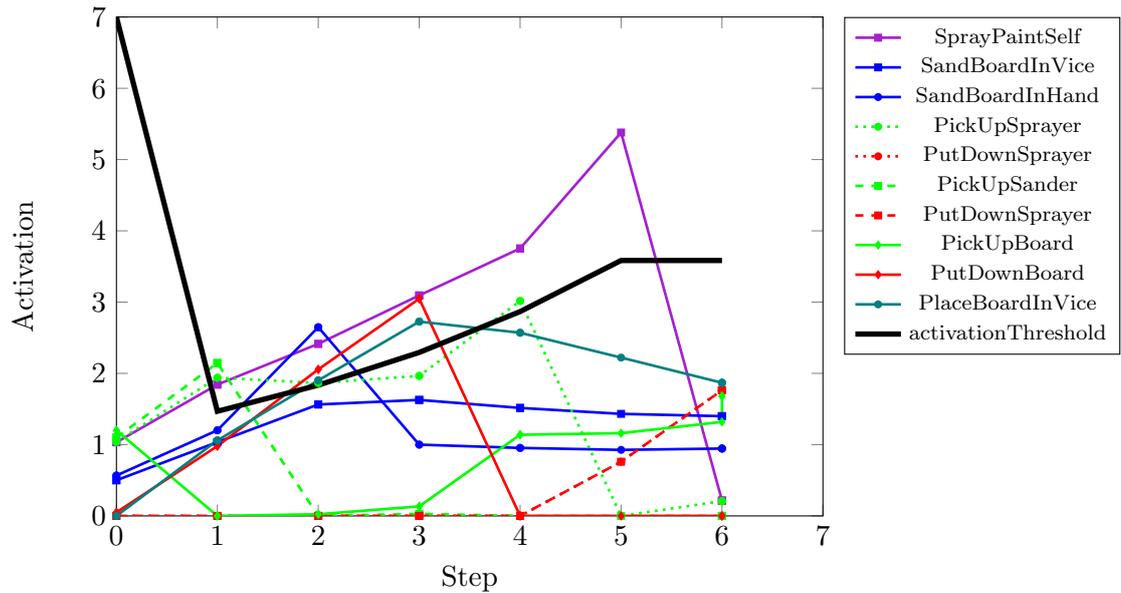


Figure 22.2.: RHPB activation plot of the hybridly planned scenario from Maes, 1989.

Our hybrid implementation using the same behaviour domain model with the additional used-hand limitation outperforms Maes' with 7 instead of 19 required planning, respectively decision-making steps. Moreover, we encountered that our solution was able to solve the problem without any parameter tuning of the behaviour layer, in contrast to Maes' implementation, due to the goal-directing influence of the symbolic planning layer. The experiment with activated planner uses weight 1.0 for all weight configurations. For comparison, our solution using only the behaviour network layer and tuned weight parameters requires 18 iterations instead of 19 in the original implementation. The corresponding diagram is also shown in Figure 22.3. This small improvement about one decision-making step is probably grounded in the further improved activation formulas of our implementation. In particular, the empirically tuned configuration without the symbolic planner uses plan weight = 0.0 to deactivate the planner and situation weight = 0.2, predecessor weight = 0.5, successor weight = 0.92, confictor weight = 0.4, goal weight = 0.05.

This very first proof-of-concept of the RHPB core component, which is focusing on decision-making and planning, showed that our approach is capable of solving a reasonable complex task-level planning problem for a single agent. Particularly, we successfully tested that the approach can handle multiple conflicting goals as well as alternative so-

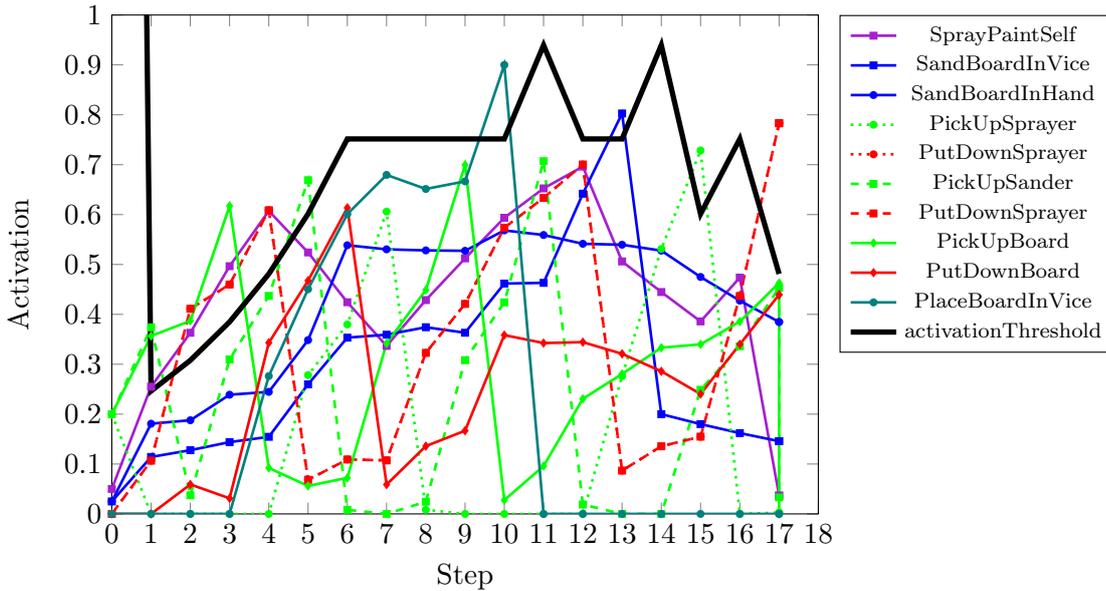


Figure 22.3.: RHPB activation plot of the scenario from Maes, 1989 with behaviour network only using empirically tuned weights. *activationThreshold* is also starting at 7, diagram is clipped to improve readability.

lution paths. Additionally, we determined that the hybrid approach solves the problem without parameter tuning as required for a pure behaviour-network-based approach. Furthermore, the presented RHPB solution has been able to outperform its ancestor in terms of planning efficiency resulting in $> 63\%$ fewer decision steps.

22.2. Planning and Decision-Making in a Multi-Robot Simulation

The following experiment was the first test of RHPB's multi-robot control capabilities as well as study regarding the suitability for lower-level behaviour control decisions. In particular, we tested the RHPB in a multi-robot scenario using the well-known turtlesim simulation of the ROS tutorial package. The simulation allows to control the motion of turtle robots with differential-drive velocity commands. These commands are applied for a short amount of time allowing for simulation in a stepwise execution. In particular, we implemented a simple pathfinding scenario, which included random start and two target positions for each robot and a constraint of avoiding collisions with each other.

22. Experiments

For that, we extended the turtlesim simulation with a simple collision detection of the nearest neighbour robot and additional visual representations (e.g. to highlight collision sensing range of turtles and target positions)¹.

In order to test the basic multi-robot support, we implemented a centralised approach with one RHBP core instance (behaviour network and symbolic planner) managing all robot behaviours. As an example of a specific realisation, the implemented and used behaviour modules for each robot are described below. All behaviours and sensors are instantiated for each robot, configured with the particular name and corresponding topics. The two target position goals are shared amongst all robots, while every robot has an individual goal to avoid collisions. The implemented model is also visualised in Figure 22.4.

- *TargetDistance-Sensor* extends the provided sensor wrapper *RawTopicSensor* to subscribe to the Pose-Topic of the robot and calculate the distance to a given target. *RawTopicSensor* simply provides any sensor type to the connected activator without any type conversion, thus requiring a matching activator implementation or scenario specific extension.
- *TeamMateDistance-Sensor* is an instance of the *AggregationSensor* that subscribes to the Neighbour-Pose topic, which provides the position of the closest team-mate position in a perception radius, and the Pose-Topic of the robot to determine their Euclidean distance.
- *LinearActivator* of RHBP's is used to compute activation based relative to the distance.
- *Move-Behaviour* is a custom behaviour implementation that calculates the necessary velocity command based on the current distance and publishes it on the Twist-Topic of the turtle robot. *Move-Behaviour* is instantiated in two configurations, first as *MoveToLeftBottom-Behaviour* and second as *MoveRightTop-Behaviour* leading to the two target positions with effects on the *TargetDistance-Sensor*.
- *MoveTeam-Behaviour* is a specialisation of *Move-Behaviour* implementing a simple

¹Source code available: https://github.com/cehberlin/ros_tutorials

22.2. Planning and Decision-Making in a Multi-Robot Simulation

collision avoidance behaviour (rotating away counter-clockwise from the detected neighbour and moving with a velocity proportional to the neighbour distance) in order to keep the distance to other robots with effect on *TeamMateDistance-Sensor*.

- Two common acquisition goals are instantiated that use a condition formulated with the *TargetDistance-Sensor* and the *LinearActivator* to express the target positions.
- A maintenance goal instance applies the *LinearActivator* and *TeamMateDistance-Sensor* to express the collision avoidance.

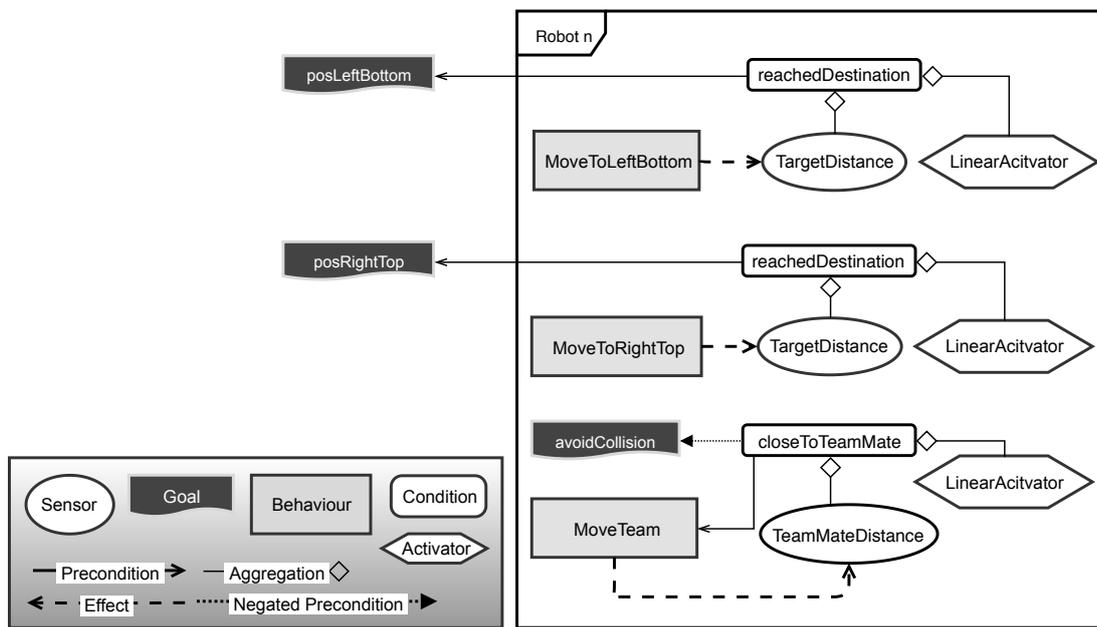


Figure 22.4.: RHBP behaviour model of the multi-robot turtle experiment. The scope *Robot n* is initialised for each robot.

It is important to consider that the above approach is one possibility of modelling the problem with RHBP. Alternatively, the collision avoidance could also be realised by using two single dimensional real sensors providing distance and orientation of the closest team-mate and instead of using two goals, the same could have been expressed with a different precondition configuration.

The modelled application scenario was tested on 10 randomly generated start configurations for 5 robots. The execution was monitored, and the required number of plan-

22. Experiments

ning and decision-making steps have been recorded until all goals have been achieved. Whereby the two target positions stayed the same for all evaluation runs. Moreover, the selected target position in the lower-left and top-right corner of the quadratic environment represent conflicting goals for the robots as they are exactly mirroring each other on the diagonal axes of the environment. The decision-making cycle of RHBP was configured to 1Hz to simplify monitoring and tracking. Additionally, the weight configuration without the planner uses plan weight = 0.0 to deactivate the planner, and in the other case plan weight = 1.0 to activate the planner. All other weights (situation weight, predecessor weight, successor weight, conflictor weight, goal weight) are set to the default weight of 1.0.

Figure 22.5 shows scenario 7 as an example, including the start configuration and the two resulting final situations using only our Behaviour Network layer and the full hybrid solution of RHBP. The given example illustrates the advantage of the hybrid planning architecture as the planner helps to generate more efficient coordination amongst the robots and leads to a more optimal solution.

Table 22.1 summarises the results of the comparison between the experiment executions using RHBP in behaviour network only mode and in the full hybrid planner configuration using the behaviour network together with the PDDL planner. The aggregated results show that the configuration using the planner requires 40% less planning cycles to resolve the scenario, which is directly related to execution time in our simple simulation. Only start configuration 2 needed more planning steps in the hybrid planning configuration. Here, the plan led to a collision avoidance situation that was difficult to resolve and could not be considered by the planner in advance due to the simplified model of the world.

The experiment shows that the combination of a behaviour network with a symbolic planner leads to more efficient solutions and the integration of a planner helps to coordinate multiple robots. However, the experiments also show that centralised planning of multiple robots using RHBP is limited in the way that the currently used symbolic planner (Metric-FF) does not support parallel behaviour execution. For that reason, our planned extension has to consider parallel behaviour execution not only on the behaviour network level to further improve the results. Furthermore, the multi-robot simulation experiment proved that our solution is also capable of lower level robot control by im-

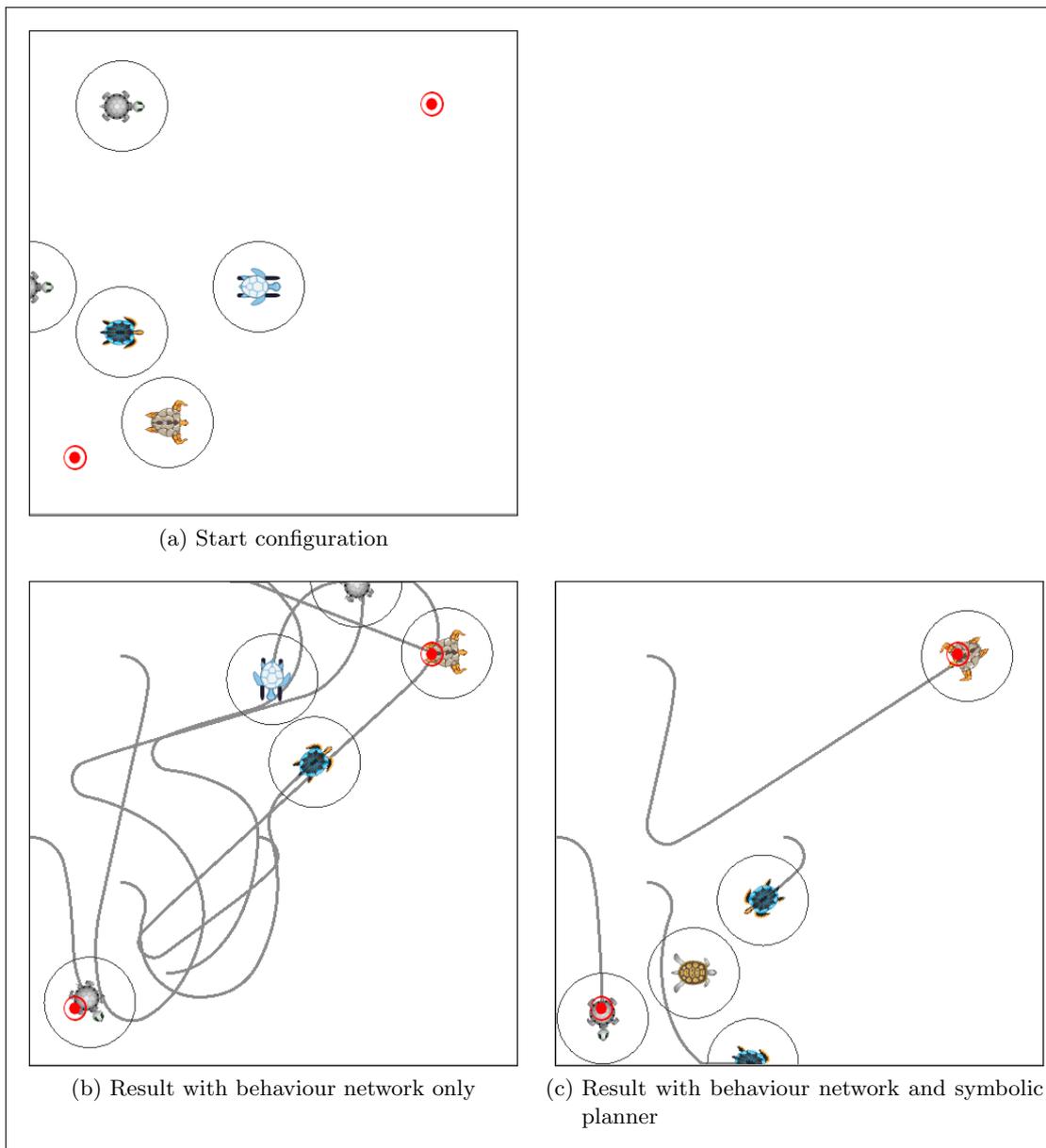


Figure 22.5.: Turtlesim multi-robot example scenario 7 with start configuration (a) and final results comparing behaviour network only (b) with complete hybrid planner approach (c).

Red points highlight the target positions. Circles around robots indicate the collision detection range. Grey lines mark the robot trajectories.

22. Experiments

Table 22.1.: Comparison of a RHBP behaviour network layer only configuration against the hybrid-planning configuration in a multi-robot path finding scenario.

Scenario	Decision steps	
	Hybrid Planner	Behaviour Network
1	24	26
2	87	48
3	54	95
4	24	39
5	37	44
6	45	70
7	34	137
8	10	56
9	35	93
10	55	71
Sum	405	679
Mean	40,5	67,9
Median	36	63
STD	20,3	31,4

plementing behaviours that directly steer robot motion, besides validating the current multi-robot planning and decision-making capabilities.

22.3. Multi-Robot Experiment Self-Organisations with Decision-Making and Planning

The experiment presented in this section illustrates the beneficial combination of self-organisation with decision-making and planning in a multi-robot scenario. Likewise, the experiment is a proof-of-concept of the realised RHBP self-organisation extension.

The example scenario considered in this section comprises multiple robots that have to maintain an unknown open space by keeping it clean and managing the recycling and dumping process of found garbage items. Particularly, the recycling and dumping process requires that once garbage is found, first it needs to be transported to a recycling station before all leftovers are transported to the dump station. Moreover, the robots have to patrol the environment repeatedly over time. Likewise, robots have to make sure that they avoid collisions with each other.

22.3. Multi-Robot Experiment Self-Organisations with Decision-Making and Planning

The cleaning process with several dependent stages is different from typical swarm robot experiments that focus on achieving one certain stable state in a decentralised manner. We have integrated this additional complexity in order to illustrate the beneficial combination of self-organisation with more complex decision-making and planning. However, we still keep this simulated experiment comparably simple to improve transparency.

For the simulation, we use a further extended version of the basic turtlesim simulation, commonly known from the ROS beginner tutorials, and which was already used in Section 22.2. This simulation allows to control multiple differential wheeled robots in an empty space. Our version² extends the original implementation with capabilities of sensing other robots, allowing to configure a torus environment without borders, adding various visualisation options to draw additional elements into the world, and replacing the turtle robots with cleaning robots. To simulate the garbage items and their detection we make use of the SoBuffer module of the so_data package to randomly spread garbage gradients with a special identifier in the environment, which then can be sensed with the corresponding gradient sensors. A real-life application would need an additional intermediary component that translates the corresponding sensor perception to virtual gradients. The simulation environment in a particular start configuration and during execution is shown in Figure 22.6.

The modelled RHBP solution with behaviours, goals and corresponding preconditions and effects is visualised in Figure 22.7. Each robot instantiates this model independently resulting in a decentralised solution with coordination and interaction amongst the robots only carried out through the simulation environment by sensing each other as well as exchanging information through the virtual gradient space of the SoBuffers. Three goals formulate the target conditions for the robots, *PatrolEnvironment* expressing the need for a repeated cleaning process in the environment, *GarbageCleaned* modelling the need to clean garbage items once they are found, and *AvoidCollision* to keep the robots in a safe distance of each other. Particularly, the garbage recycling and dumping state is tracked by influencing special ROS topics in the related behaviours and accessing them through so-called *KnowledgeSensors*. Here, the picking of garbage is implemented by sending gradient information that result in the deletion of garbage gradients at this

²https://github.com/cehberlin/ros_tutorials/tree/clean_robots

22. Experiments

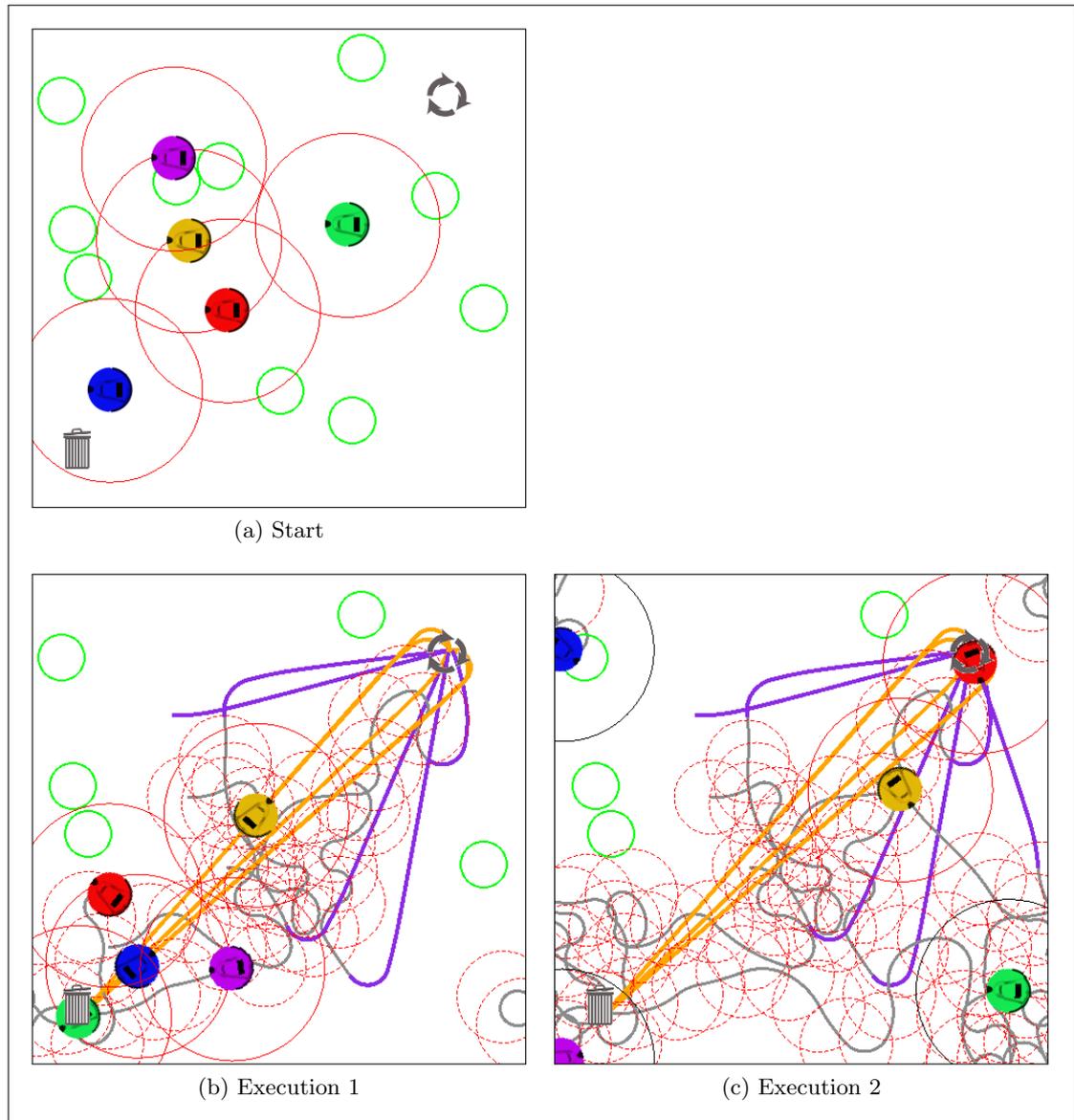


Figure 22.6.: Visualised self-organisation simulation scenario. Green circles are garbage items; Red solid circles around robots denote the sensor range. Red dotted circles show virtual pheromones, the size corresponds to the evaporation stage (smaller=older). Garbage bin and recycling symbol visualise dump and recycling station. Grey lines mark robot trajectories; Purple lines mark trajectories after garbage was collected; Orange lines mark trajectories after garbage was recycled.

22.3. Multi-Robot Experiment Self-Organisations with Decision-Making and Planning

position. Additionally, the behaviours *MoveToDump* and *MoveToRecycling* are using distance conditions formulated with *LinearActivators*, which provide higher activation for a larger distance to the target, and *Pose* sensors for the current robot position.

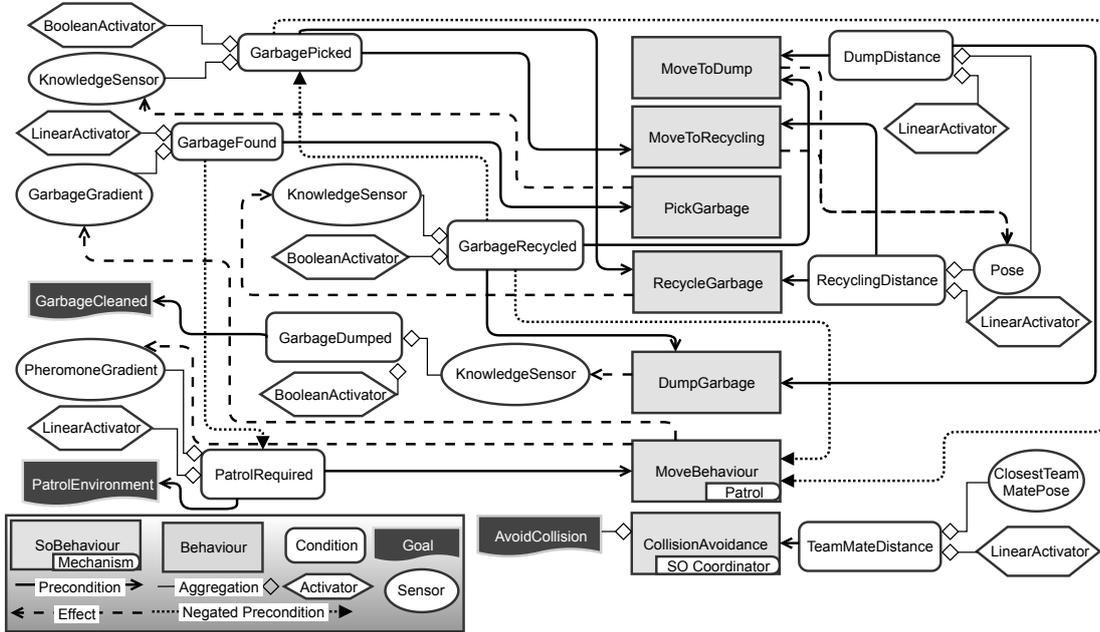


Figure 22.7.: Model of the RHPB solution for an individual robot of the experiment illustrating the relationships with preconditions, behaviours and effects.

So far, this part of the model is expressed with RHPB core components. Nevertheless, the exploration and patrolling apply self-organisation with a patrol mechanism that is based on the virtual pheromone pattern. In our implementation, each robot spreads evaporating gradients at its current position while moving through the environment and calculates the movement vector using a repulsion pattern to push itself away from the gradient field. Both together results in robot motion that prefers the motion into unknown space or space that has not been visited for a longer time period. All robots share these virtual pheromones through the SoBuffer library communication infrastructure; thus, they are able to coordinate in a self-organised manner.

In the shown model, we have two self-organisation mechanisms for combined exploration-patrol and collision avoidance. For collision avoidance, we apply the *So Coordinator* that automatically selects a suitable mechanism based on the given self-organisation goal *AvoidCollision* and the available scores in our database. The different character-

22. Experiments

istic of the goal is also indicated by the visualised aggregation relationship instead of a link to a condition. In contrast to collision avoidance, patrolling is manually selected by integrating the *Patrol* mechanism directly. The direct selection of a mechanism has the disadvantage of making a later exchange of the mechanism more difficult. However, we have taken this approach here to illustrate different possible usage styles of our framework, although the specific *Patrol* mechanisms could be replaced easily by a *SO Coordinator* instance with a corresponding self-organisation goal.

The scenario visualised in Figure 22.6 with the model of Figure 22.7 has been tested with 5 robots and 10 garbage items randomly positioned in 5 different start configurations (scenarios). The positions of recycling and dump station have not been altered between the trials. Moreover, we have manually manipulated the expert knowledge scores to force the *SO Coordinator* to run all 5 scenarios once with the collision avoidance mechanisms based on the repulsion pattern from Fernandez-Marquez et al., 2012 and once with the algorithm from Balch and Hybinette, 2000. Both patterns rely on robot pose gradients to allow the robots to determine the repulsive forces from each other.

Table 22.2.: Experiment results of a comparison between different available self-organisation patterns for collision avoidance.

	Fernandez-Marquez et al.		Balch and Hybinette	
Scenario	Duration in s	Collisions/s	Duration in s	Collisions/s
1	270.39	2.02	89.84	2.65
2	1484.70	1.42	177.26	1.42
3	1162.18	1.60	471.90	1.55
4	952.40	1.22	326.14	2.08
5	363.93	2.25	267.02	3.11
Mean	846.72	1.70	266.43	2.16
Median	952.40	1.60	267.02	2.08
STD	465.39	0.38	130.32	0.64

The experiment results are listed in Table 22.2 and show different characteristics for both applied mechanisms. We see that the runs with the mechanism from Balch and Hybinette clearly outperform runs with the mechanism from Fernandez-Marquez et al. in terms of execution time (duration) for completing the mission. However, runs with the Fernandez-Marquez et al. mechanism are having fewer collisions per time. For the number of collisions over time, it has to be considered that one collision might be counted

several times, depending on the sensor frequency of the simulation, if the robots stay for a moment in the collision pose.

The obtained results of this experiment could now be used to create a score for the mechanism configurations in our expert knowledge library. However, we have not yet fine-tuned the parameters of the mechanisms, which might influence the results. Thus, it would be useful to repeat the experiment with other parameter configurations. Nevertheless, the presented results illustrate the application of our work and highlight the importance of an inexpensive exchange of self-organisation mechanisms and their configuration within a decision-making and planning framework. Moreover, it also indicates that our SO coordinator concept is providing a suitable foundation for the integration of experience and expert knowledge with self-organisation pattern.

22.4. Learning and Self-Adaptation in the Maes Automation Scenario

In order to evaluate our RL integration approach, we reuse the experimental setup that has initially been introduced by Maes, 1989 and that was also used as a first proof-of-concept of RHBP in Section 22.1. Please refer back to this section for a description of the scenario.

To analyse if our extension of RHBP with RL improves the self-adaptation capabilities, the original experiment is extended in two ways. First, we repeatedly execute the mission to be able to learn from experience. Secondly, every 150 simulation episodes, the mission constraints are interchanged. Particularly, the required order of fulfilling the goals is interchanged. In the beginning, we follow the original scenario from Section 22.1, before we change that sanding the board makes the system non-operational, which results in the required task execution order of first spray-painting the robot before sanding the board.

In contrast to Section 22.1, the RHBP model in this experiment only contains goals, behaviours, and their preconditions, respectively target conditions. We do not model the effects of behaviours a priori to test if our learning-based approach allows us to infer behaviour effects implicitly from experience. The simplified model does not allow to complete the mission successfully stand-alone without learning, because neither the be-

22. Experiments

haviour network decision-making layer nor the planner is able to connect the behaviours with each other without modelled effects.

For the experiment, we have chosen the following configuration. Exploration with epsilon-greedy is starting with $\epsilon = 0.5$ and ending with $\epsilon = 0.09$ over 100 annealing steps. Moreover, the RL process is running continuously, enabling online learning. The ANN is configured with 3 hidden layers having 16 neurons each, which are a multiple of 2 in between the number of inputs (23 due to the one-hot encoding of sensors) and outputs (10 behaviours), Discount-Factor = 0.01, and training interval factor $\tau = 0.01$. Furthermore, we have not applied a pre-training stage but used the experience buffer with a size of 140 and a batch size of 36. Training is executed in cycles of 12 decision steps. The activation weights for the RHBP decision-making are set to 0.5 for each activation component (predecessors, successor, goal, precondition, goal inhibition, conflicting behaviours) except for the RL activation, which is set to 1.0 to be able to overrule decisions with RL. The particular configuration was determined with empirical tests. Here, especially the selection of the Discount-Factor and τ had a major influence on the result.

The results of the experiment are visualised in Figure 22.8. The diagram shows that our approach enables the system to learn the required decisions to complete the mission continuously after 116 steps of exploration and learning from experience. Furthermore, we can also see that the system dynamically adapts to longer episodes with different action sequences, due to the triggered exploration, and still successfully completes the mission once the model initially converged. Moreover, the system shows that it is able to recover after each interchange of the goal target order after 150 episodes. The time of adaptation in the shown sample is 11, 125, and 9 steps. We started with the original Maes' scenario where the spray-painting makes the robot non-operational and then interchanged to the scenario where sanding the board makes non-operational and vice versa. Hence, it seems like adapting to the second scenario is faster to accomplish. This overall pattern stays similar for repeated experiment executions as well as longer samples, which we omitted to be able to visualise the entire sample.

The results are supporting our hypothesis that the integration of RL is fostering self-adaptation capabilities of RHBP and even more enables to learn correct decision without a priori defined effects. Nevertheless, the preparation of the experiment led us to the

insight that it is still necessary to carefully select the configuration of the learning setup, depending on the application scenario. Furthermore, the results show that even though the setup is slightly tuned, learning requires a reasonable amount of training episodes, which is difficult to realise if the system is already deployed to a robot. In consequence, an application in practice can potentially benefit from learning in a simulation environment before deployment.

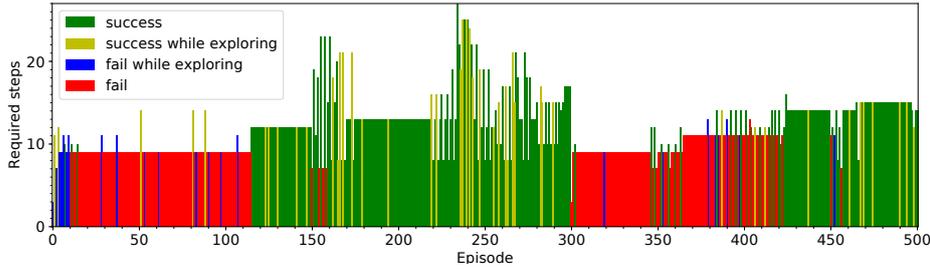


Figure 22.8.: Required decision steps until mission success or failure over learning episodes. Success and failure are differentiated in episodes with and without exploration steps. After 150 episodes the required order of the mission goals is interchanged.

Additionally, we compared the quantitative performance of the RL-based decision-making with the results of RHBP using only the core framework with hybrid decision-making using symbolic planner and behaviour network as well as the behaviour network only with manually tuned weights. These two comparison scenarios reflect the experiments in (Hrabia, Wypler, and Albayrak, 2017), respectively Section 22.1. In detail, the hybrid approach represents the default application of RHBP that benefits from the combination of long-term goal-pursuance provided by the symbolic planner and short-term fast responses, and opportunistic influence of the behaviour network. The approach with the behaviour network only is used for comparison because it was also evaluated in the original experiment.

The results in Table 22.3 show that learned decision sequences are in average 4 decision steps faster than the RHBP solution with a behaviour network only with tuned weights and 7 steps slower than the hybrid approach. Here, the approaches without RL are not deviating in the results due to the static nature of the experiment environment that always leads to the same sequence depending on the particular RHBP configuration. The major reason for the supplementary steps taken by the RL approach is the required

exploration steps that are influencing the overall performance. For this reason, the RL results are classified into successful and failed trials as well as trials with and without exploration. Here, we can observe that failing trials are having fewer steps on average, which indicates that the potentially more optimal RL solutions are more prone to fail. Hence, our RL approach is able to find a solution (local optima) but fails to find the global optimum.

Table 22.3.: Mission steps and success differentiated by exploration.

	mean	std
RHBP Planner and Behaviour Network	7.0	0.0
RHBP tuned Behaviour Network only	18.0	0.0
RHBP RL with Success	13.8	3.3
RHBP RL with Success with Exploration	15.3	4.0
RHBP RL with Success without Exploration	13.5	3.1
RHBP RL with Fail	9.4	1.2
RHBP RL with Fail with Exploration	9.8	2.1
RHBP RL with Fail without Exploration	9.3	1.1

Nevertheless, the additional effort for learning seems reasonable to further increase the overall adaptivity of the system. The conducted experiment underlines the feasibility of the taken approach in terms of being able to learn correct task-decisions with missing information. Moreover, using RL is improving self-adaptation and enabling the system to successfully handle dynamically changed (environmental) target conditions, which have not been known or modelled in advance.

23. Projects

In this chapter, we consider use cases and application from research projects, which applied the RHBP framework in practice. In Section 23.1 details about the first application on a real robot system within a national robot competition are provided. This project especially tests the integration capabilities in a complex ROS environment and proves the general real-time capabilities of the framework. This application verifies again the same requirements ($R1$, $R7$, and $R8$) and objectives ($O1$, $O5$, and $O6$) as the first Maes

experiment but on a real robot in a more challenging environment.

Even though the two related projects presented in Section 23.2 are also using a simulated environment, the considered application scenarios of the discussed contests are exceptionally complex and allow to test especially the performance, scalability, and adaptation capabilities of RHBP. Here, the participation in the first year shows that *Requirement R6* and the corresponding *Objective O4* are feasible in combination with RHBP. The second part, focusing on the participation in 2018 is providing a rich case study on how the proposed combination of decision-making and planning together with self-organisation can look like in a complex application, which especially covers *R3* and *R4*, respectively *Objective O3*. Furthermore, the insights gained in the project also support answering the *Research Question R2*.

Next, Section 23.3 presents details about the application of RHBP in a three-year Unmanned Aerial System (UAS) research project. The discussed applications are using both a realistic simulation environment and real-life experiments. In contrast to the former evaluation use-cases, the research project is also considering the integration with end-users of the robotic system that are providing dynamic goal inputs, while also having a strong focus on the adaptation capabilities of the framework. This project covers many of the general objectives and requirements, such as *R1*, *R3*, *R7*, and *R8*, and their related objectives *O1*, *O2*, *O5*, and *O6*. However, especially interesting are the insights about the tests of *R6* and *R9*. Here, the scope of integration through ROS is even extended to external system components. Additionally, the results of the EffFeu project demonstrators and experiments support addressing *Research Question R1*.

23.1. SpaceBot Cup 2015 Project

The project and experiments described in this section represent the first application of RHBP on a real robot. This application in practice demonstrates the integration capabilities in a complex ROS environment and substantiates the general real-time capabilities of the framework. Particularly, we applied the RHBP solution on a UAS that was developed for the national German SpaceBot Camp Competition 2015, which was initially named SpaceBot Cup. In the contest, autonomous robots are challenged to find objects in an artificial extraterrestrial indoor environment simulating space exploration.

23. Projects

In detail, two of these objects have to be collected, carried to a third object, and assembled together to build a device, which also has to be turned on in order to complete the task ¹.

For the 2015 event, the ground rover of team SEAR (Small Exploration Assistant Rover) from the Institute of Aeronautics and Astronautics of the Technische Universität Berlin (TUB) (Kryza et al., 2015) was supplemented by an autonomous UAV developed by our group of the Distributed Artificial Intelligence Lab (DAI).

The developed ground rover features a manipulator to perform all grasping tasks and is assisted by the UAV in exploration and object localisation. Hence, mapping the unknown environment and locating the objects are the main tasks of the accompanying UAV.

The aerial system can take advantage of its capabilities of being faster than a ground-based system and having fewer issues with rough and dangerous terrain. For this reason, it was the concept of providing an autonomous UAV that rapidly explores the GNSS-denied environment, gathering as much information as possible and providing it to the rover as a foundation for efficient path and mission planning. In particular, it was the goal to support the ground rover with map information as well as locations from the three target objects.

The multi-rotor UAV is based on a commercial assembly kit including a low-level flight controller that is extended with additional sensors and a higher-level computation platform. All intelligence and advanced software modules are used and developed within the ROS environment.

The UAV executes its mission in a completely autonomous fashion and does not rely on remote processing at all.

In particular, all higher-level behaviour is controlled by RHBP. The atomic behaviours for take-off, land, emergency-landing, collision avoidance and exploration are using provided base classes of the framework and are running on one ROS node. RHBP is also supporting a distributed node architecture for the behaviours, but we did not take advantage of it because of the computationally simple nature of most behaviours. Nevertheless, the generalised implementations of the more complex tasks of collision avoidance and exploration are separated in own packages with corresponding separated nodes.

¹Complete task description in German at <http://www.dlr.de/rd/Portaldata/28/Resources/dokumente/rr/AufgabenbeschreibungSpaceBotCup2015.pdf>

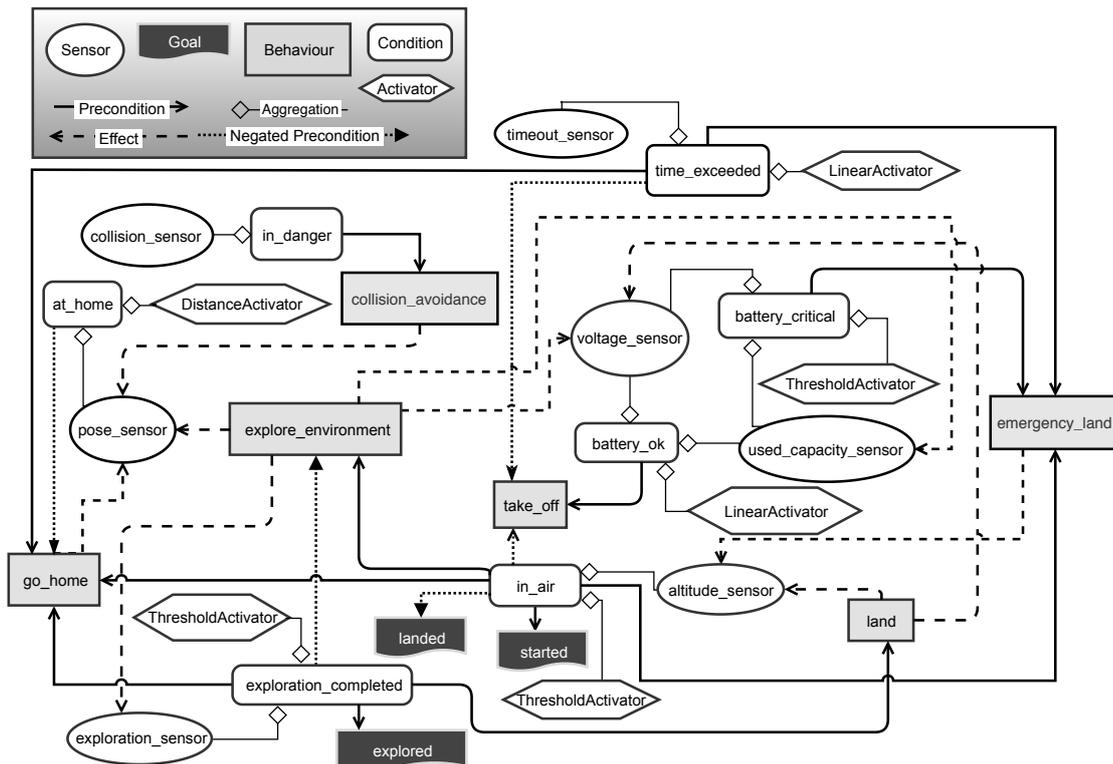


Figure 23.1.: SpaceBot Cup behaviour model.

23. Projects

In order to implement the desired behaviour, we implemented the behaviour model illustrated in Figure 23.1. In order to do so, the provided behaviour base class have been extended for the individual behaviours. Available sensors and abstracted information of the system have been integrated as virtual sensors into the RHBP framework. For that, it was necessary to implement some special sensor wrappers, which extract the needed information from complex ROS message types, like poses (TFs). Furthermore, a special distance activator was implemented to determine the activation in the network based on a *geometry_msgs.msg.Pose* and a desired target pose. The exploration sensor is a wrapper for the exploration module that describes the completeness of the exploration.

The realised UAV capabilities are to take-off and land (regularly at the landing zone after the mission is completed, the time is over, or the battery is depleted or anywhere else in emergency situations), select a position to move to (performed by exploration or return-to-home behaviour and overridden by the obstacle avoidance), and move to the selected location while maintaining constant altitude over ground. While it is operating, the UAV continuously maps the terrain and searches for objects. These activities do not need to be turned on or off explicitly. Given the initial situation that the aircraft is fully charged, on the ground, at the landing zone (also referred to as “home”), and the mission starts, the network will activate the start behaviour first and then cycle between exploration (which retrieves a target location) and, if required, collision avoidance, (thereby mapping the terrain and scanning for objects) until it runs out of battery or completed its exploration mission. Finally, it will select the home location as the target, move there and land.

In general, the developed UAS system is able to autonomously start, land, hover on a position, follow given trajectories, and detect the target objects of the mission. Here, trajectories are generated by an external exploration module and collisions are avoided with the potential field approach, while the whole process is controlled by RHBP as a high-level goal-oriented decision-making and planning component. The capabilities have been empirically tested in the simulation environment MORSE (Echeverria et al., 2011), in the laboratory environment and the contest itself.

The finally integrated system is successfully running onboard of a laptop-like hardware platform (Intel[®] NUC D54250WYB with Intel[®] Core i5-4250U and 8GB RAM), with the CPU having almost 100% load. Nevertheless, the system is responsive and able

to execute all modules in an appropriate refresh rate. Furthermore, most CPU load is generated by the processing of the visual camera data in the object detection and localisation and mapping. The memory usage can be neglected in general, the whole system consumes less than 1GB with an initial map. In the given scenario, the RHBP component is executed with 1Hz and is generating only 1% of the CPU load.

Even though the SpaceBot Cup scenario did not include crucial points in terms of decision-making and planning, rather, it could have been implemented with a simple HSM approach, we have been able to prove that RHBP performs well on real systems running ROS in a dynamic real-time environment. Moreover, we could demonstrate that additional computation demands can be neglected and are not interfering with other more computationally intensive functions such as localisation and mapping. Additionally, this first proof-of-concept on a real robot system showed that the RHBP concept allows for fast and simple integration with other ROS components.

23.2. Multi-Agent Programming Contest

The Multi-Agent Programming Contest (MAPC) is an annual international scientific contest organised by the Clausthal University of Technology². Since 2005, the organisers prepare almost every year a new challenging multi-agent simulation scenario, which has to be solved by the participants (Ahlbrecht, Dix, and Fiekas, 2018; Behrens et al., 2010). The goal of the contest is stimulation of research and education in the area of multi-agent system development and programming through identifying fundamental problems, collecting suitable benchmarks, and gathering test cases for all kind of multi-agent programming languages, platforms, frameworks, and tools.

Participating in the contest has a long tradition at the TUB (e.g. (Heßler, Hirsch, and Keiser, 2007; Heßler, Hirsch, and Küster, 2010; Heßler et al., 2013)), starting in 2007 with different versions of the Java-based Intelligent Agent Componentware (JIAC) framework (Fricke et al., 2001; Hirsch, Konnerth, and Heßler, 2009). The motivation for participation in the contest was always to use it as a platform to apply the generic multi-agent frameworks developed in the institution to several multi-agent problems of the contest and to test the robustness and performance of our solutions. Starting from

²<https://multiagentcontest.org>

23. Projects

2017 this tradition is continued, although now evaluating the RHBP framework, which has its roots in the robotics domain and which is based on the ROS framework.

In particular, RHBP is applied for the implementation of the individual task-level decision-making and planning of the agents, while ROS is used for all inter-agent communication. Due to numerous challenges, we are facing in the application of real robot scenarios (hardware failures, very uncertain environments, and challenges in basic robotic capabilities like object detection and localisation (Hrabia et al., 2017)) it is difficult to evaluate the RHBP framework in applications with a comparable complexity on the task-level like the contest scenario. For this reason, we take advantage of assessing RHBP in such a simulated contest scenario with simplified environmental conditions, which allows us to focus on the general capabilities of our decision-making and planning approach.

In the years of 2016 to 2018, the simulated application scenario was using a discrete and distributed last-mile delivery simulation (Multi-Agent Systems Simulation Platform (MASSim)) on top of geographic map data from different real cities (*OpenStreetMap* data). The simulation allows competition of several teams consisting of independent agents. In all three years, delivery jobs are randomly generated and split into three categories: *Mission jobs* are compulsorily assigned, *auction jobs* are assigned by prior auction, and *regular jobs* are open to everyone. Jobs are monetarily rewarded on fulfilment and can only be accomplished once. Moreover, jobs consist of several items that can be purchased at shops (2016-2017) or gathered in resource nodes(2017-2018), as well as stored in warehouses.

23.2.1. Multi-Agent Programming Contest 2017

The MAPC 2017 team of the TUB results from the project course *Application System Project* that has taken place in the summer term 2017. In this course, two mixed teams of Master's students from business informatics and computer science developed solutions for a simplified version of the official contest scenario, only using 6 agents per team, instead of 28, and reduced task complexity. In particular, the given target was to apply the RHBP framework. The united team for the official contest, named TUBDAI, is formed by volunteer students from the course and their supervisors.

The implementation has been started from scratch in preparation for the summer term, while the final contest implementation was developed based on the most elaborate

student team version of the project course.

The primary strategy of the TUBDAI team is to robustly fulfil as many jobs (mission, priced, auction) as possible. In particular, we favour missions and auctions over priced jobs because fulfilling mission jobs would avoid fines and auction jobs have the advantage of being exclusive for a team once the auction is won. Furthermore, our solution tries to adapt and recover from unexpected situations, for instance, not anymore available items, failed actions, or jobs already fulfilled by another team. Due to the reason that the 2017 implementation was started from scratch, we did not implement posting own jobs or putting special considerations on the opponent observation. Furthermore, we only use a limited set of possible agent actions that are minimally required to fulfil the jobs.

The implemented solution is generally decentralised; each agent processes the perception individually and in parallel. Each agent has its own independent RHBP-based decision-making and planning component. Moreover, the agents exchange messages about their task rating and state, which allows them to determine if they or another agent is responsible for a task. Furthermore, we divided the agents into groups; each group contains an equal amount of agents per role. The groups are automatically assigned at the beginning of the simulation. The number of groups is chosen to have at least one agent per role in a group in order to cope with the role-dependent tool handling capabilities. In consequence, the provided agent role configuration in the contest with 28 agents in the roles of 4 drones, 8 motorcycles, 8 cars, and 8 trucks resulted in 4 static groups, to have at least one drone per group. In each group, one lead agent is responsible for rating and selecting jobs for its group, before it decomposes them into tasks for further coordination. All tasks of a job are rated by all agents of a group. Each agent is communicating its rating to all other agents of its group. This allows agents to determine the task assignment in a decentralised way based on the lowest rate for each task. Each group is operating independently, except for the leaders, who are sharing information about taken jobs and auction bidding with each other to avoid parallel operation on the same job and to not overbid each other in auctions. Additionally, the lead agent of each group is responsible for the bidding process to acquire auction jobs. The groups are used to process different jobs in parallel.

The agents change their behaviour during the simulation runtime with respect to the

23. Projects

assigned task. All agents use the same behaviour implementations. Specific behaviours are configured depending on the agent role, currently assigned task, or events, like detected failures. However, we do not implement special strategies for individual agents, like agents being responsible for buying, delivering, or gathering.

The agents plan their operation (on action-level) until the end of the currently assigned task. An atomic agent task in our implementation is something like deliver, buy, gather, assemble, or assist assemble an item. Thus, the planning considers what to do and where to do it while maintaining and planning all required charging in order to achieve this in the most effective way. Moreover, we applied the routing software GraphHopper³ to directly provide further reasoned information (role specific distances) from the perception as input to the planning. GraphHopper uses the same routing engine as the MASSim server simulation of the contest. Job decomposition implicitly provides the number of steps to be planned depending on the size and complexity of the job.

At the beginning of each simulation step, the task distribution is executed by the group leader to select the next job for the agent group. The task distribution and coordination is a custom implementation that does not apply any other framework for decision-making and planning. Here, we could also have applied the RHBP extension for automated delegation of tasks including decomposition and allocation, but the extension was not yet available at that time. Nevertheless, the experience gained in the contest led to the insight that such extension is necessary for a comprehensive multi-robot decision-making and planning framework.

The individual operational execution of an agent's task is implemented with the RHBP core. On the task execution level, the agent autonomously fulfils a given task, like delivery, assembly, or gather, while it takes care of its battery level and tries to optimise its operation for fast task completion.

The architecture for one individual agent, hence this is instantiated 28 times for all agents, is shown in Figure 23.2. Here, each agent uses a communication bridge component, which we named *mac_ros_bridge*, to communicate with the MASSim simulation server. It handles the authentication process and XML message parsing and generation while converting it into our ROS message domain model. Our domain model basically splits all perception that is provided in one large message into several independent

³<https://www.graphhopper.com>

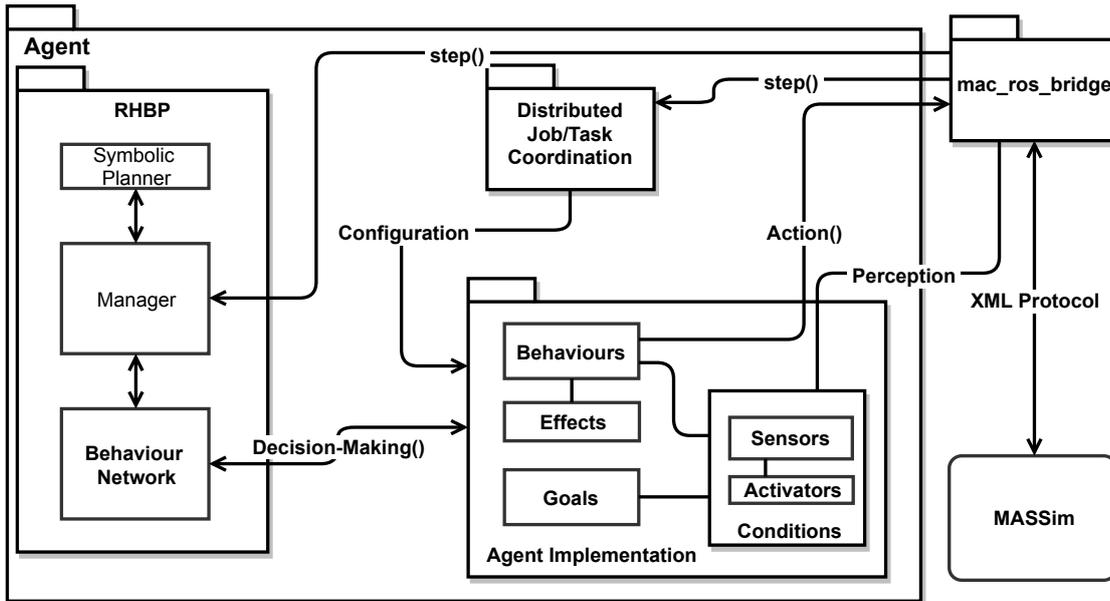


Figure 23.2.: Agent architecture of a single agent in MAPC 2017.

ROS messages, e.g. specific messages for shops, workshops, agents, jobs, auctions, and resources. Here, we also apply some preprocessing, for instance scaling values or converting units. The conversion to ROS messages has the advantage that such messages can easily be consumed by RHPB’s virtual sensor interfaces, as well as enabling the use of ROS debugging capabilities for inspecting the current system state. At the same time, the *mac_ros_bridge* listens for the action feedback on a dedicated communication topic and sends replies to the MASSim server.

Agents exchange messages using the ROS messaging system, which applies a publish-subscribe pattern, called topics. Moreover, the agents use the knowledge base from the RHPB framework that allows for persisting information tuples in a centralised or decentralised fashion. The required core communication in our approach consists of one message per agent and task per simulation round, which has been decomposed from a job. These task messages, which are exchanged by the *Distributed Job/Task Coordination*, are also used to automatically reconfigure the agent’s behaviour implementation to address the current task.

The agent class implementation with a perceive-think-act cycle is making use of one *Manager* and *Symbolic Planner* instance of the RHPB. A callback in this class executes

23. Projects

first the *Distributed Job/Task Coordination* and then the decision-making and planning cycle on every simulation step, triggered by the *request action*, which represents the perception, from the MASSim server. The *request action* is converted to a ROS message by the *mac_ros_bridge*. The decision-making and planning cycle is considered as the *think*. However, in our implementation, it is possible that one decision-making and planning cycle does not directly lead to one simulation action (the *act*) that can be communicated to the server. For this reason, we repeat the decision-making and planning cycle until we get a decision that results in an action or as long as we are in the time limit of a simulation step. In case of failures or timeouts, the agents reply with a recharge action.

The actual operational agent's behaviour is modelled and implemented on the foundation of the RHBP base classes for goals, behaviours, sensors, activators, and conditions. The scenario-specific behaviour model is illustrated in Figure 23.3. The two goals, *money_goal* and *charging_goal*, already frame the targeted autonomous behaviour of an agent. An agent should maintain its battery level and should try to make as much money as possible, with the precondition that the *charging_goal* requires no active task. In order to describe the money goal, we make use of RHBP's GreedyActivator that can be used to formulate conditions that are never satisfied with the current sensor state and always strive for a change. The agents can earn money by executing *item_actions*, even though this is only directly the case for delivery tasks. The task-related behaviours *item_actions* and *goto_task_location* are abstract behaviours that are dynamically configured by the currently assigned task. This task assignment is persisted in the knowledge base and determined by a higher-level coordination logic, which is a custom implementation for the MAPC scenario independent of RHBP. Hence, the combination of the two behaviours *item_actions* and *goto_task_location* allows an agent to travel to the correct location to execute its item-specific task, for instance going to a resource node for gathering, going to a shop for buying, going to a workshop to assist assembly, or going to a storage for delivery. In order to be able to execute the *item_action* behaviour, the agent has to be at the correct and task-dependent location. Travelling to the task-specific location is achieved by executing the *goto_task_location* behaviour. To incorporate the distances to destinations and estimating the required amount of battery, we have implemented the special sensors *RoleChargeDistanceToLocationSensor*, *ClosestFacilityDis-*

tanceSensor, and *RoleDistanceToLocationSensor*. These sensors allow determining the required battery level to travel to a location, the closest facility of a certain type (e.g. charging station or shop) and the distance in meters to a location. Here, the role is considered for the different required routing options (air-routes and street routes) and velocities. The street route distances are determined with the help of the GraphHopper map server. For this purpose, the implemented special sensors automatically calculate distances depending on the current agent position, role, and the task-dependent destination. The effects of behaviours use the static battery costs per simulation step, as well as role and location dependent charging rates and velocities.

An example of a generated PDDL domain and problem description from one simulation step of the discussed model is given in Listing 23.1 and Listing 23.2. The generation is initiated in the behaviours and goals that request information from their bound effects and conditions in every decision step. Behaviours are directly mapped to PDDL actions. Then each condition collects the raw PDDL data that is generated by the connected activators. The generated information depends on the activator type and is threefold, it includes the current sensor state for the PDDL problem, the condition as action precondition or goal condition, and the predicate, respectively function name. The effects return the necessary information about the influenced sensor directly. Next, all the information is aggregated in the manager component of RHBP before it is passed to the connected PDDL planner. At this point, redundant information is also filtered, and the planning progress monitored. Finally, processing the shown planning problem results in a plan sequence of *goto_location* and *item_action*. The resulting sequence is then applied to calculate a plan activation value based on the index position of the behaviour in the plan. This activation value allows to guide the decision-making in the behaviour network because it is combined with the other activation sources from, e.g. predecessors, successors, and preconditions.

The reactive nature of the modelled behaviour network enables fast reactions to unexpected events, like blackouts and failed actions. At the same time, the deliberative part of RHBP allows to optimise for shortest routes and suitable resource management as charging.

In order to determine concrete performance measures for the implemented decision-making and planning approach, an experimental match was conducted with one team

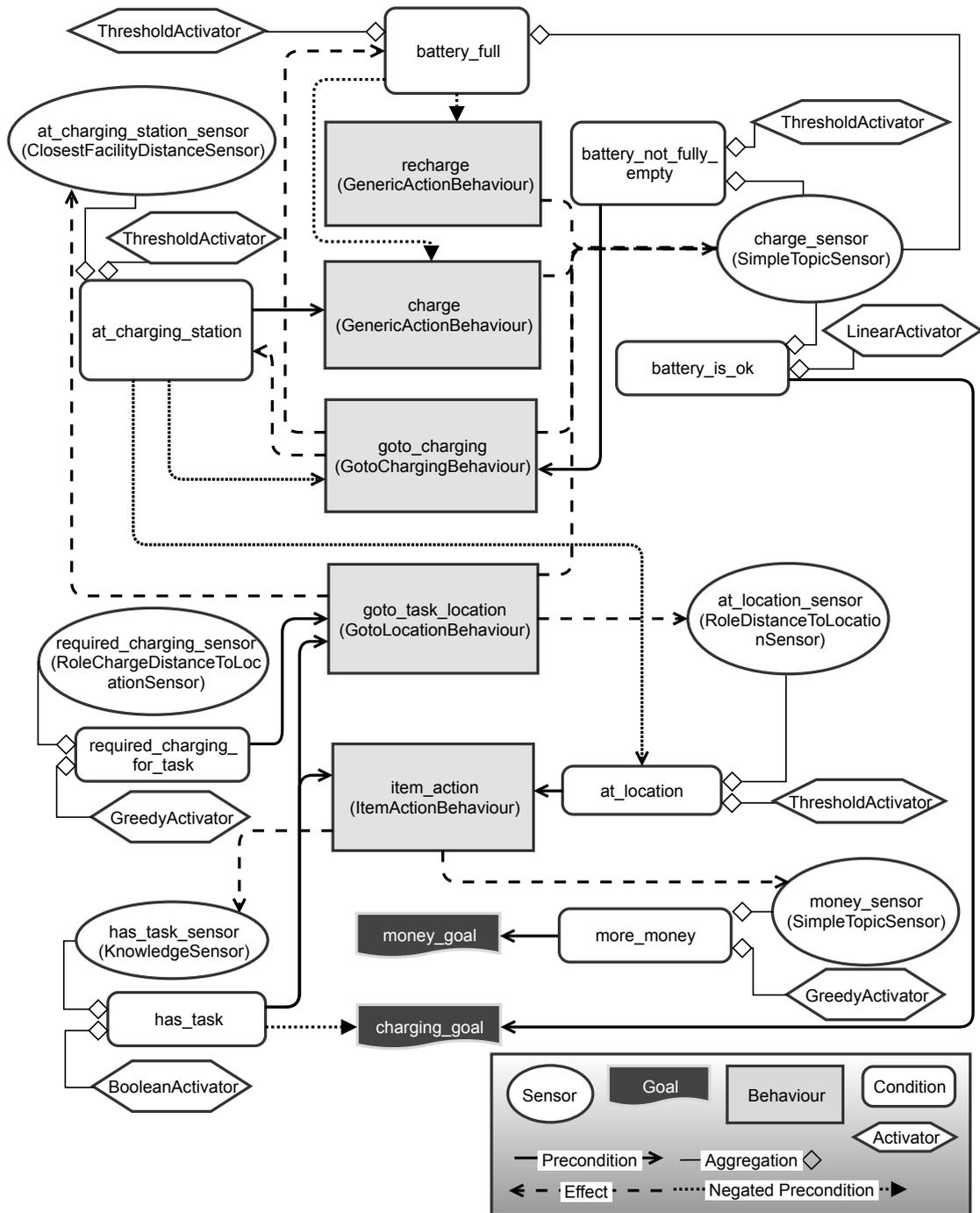


Figure 23.3.: RHPB Behaviour model for agent task execution of TUBDAI 2017.

Listing 23.1: Example PDDL domain generated by RHPB for one agent.

```

(define (domain agentA1)
  (:requirements :strips :adl :equality :negation :conditional-effects :fluents)
  (:predicates
    (has_task))
  (:functions
    (money_sensor)
    (cs_distance)
    (charge_sensor)
    (loc_distance)
    (required_charging)
    (costs))
  (:action goto_location
    :parameters ()
    :precondition (and (has_task) ( >= (required_charging) 10.0)
      (not ( <= (loc_distance) 0.000010 )))
    :effect (and (increase (costs) 1.0) (decrease (loc_distance) 1.0)
      (increase (cs_distance) 1.0) (decrease (charge_sensor) 10)))
  (:action item_action
    :parameters ()
    :precondition (and (has_task) ( <= (loc_distance) 0.00001 ))
    :effect (and (increase (costs) 1.0) (increase (money_sensor) 50)
      (not (has_task))))
  (:action recharge
    :parameters ()
    :precondition (not ( >= (charge_sensor) 250.0 ))
    :effect (and (increase (costs) 1.0) (increase (charge_sensor) 5)))
  (:action goto_charge
    :parameters ()
    :precondition (and (not ( <= (cs_distance) 0.00001 ))
      ( >= (charge_sensor) 10.0 ))
    :effect (and (increase (costs) 1.0) (decrease (cs_distance) 1.0)
      (increase (loc_distance) 1.0) (decrease (charge_sensor) 10)))
  (:action charge
    :parameters ()
    :precondition (and ( <= (cs_distance) 0.00001 )
      (not ( >= (charge_sensor) 250.0 )))
    :effect (and (increase (costs) 1.0) (increase (charge_sensor) 150))))

```

23. Projects

Listing 23.2: Example PDDL problem generated by RHBP for one agent.

```
(define (problem problem-agentA1)
  (:domain agentA1)
  (:init
    (has_task)
    (= (required_charging) 10.0 )
    (= (loc_distance) 0.147355 )
    (= (charge_sensor) 200.0 )
    (= (cs_distance) 1.82968 )
    (= (money_sensor) 49983.0 )
    (= (costs) 0)
  )
  (:goal (and ( >= (money_sensor) 50033.0) ( >= (charge_sensor) 34.0 )
    (not (has_task))))
  (:metric minimize (costs))
)
```

of 28 agents and 3 simulations with 1000 steps each per simulation. This results in 84000 simulation steps that correspond to the same number of decision-making cycles. The resulting mean duration for an action decision has been 0.538s with a standard deviation of 0.131s. In this case, one decision-making cycle is considered as the time from the beginning of the simulation step until the agent has returned its selected action, as explained above this can already include several RHBP decision-making iterations within one step. The experiment was processed on a machine with an Intel® Core™i7-4930K @ 3.40GHz CPU (6 cores with hyper-threading), 32GB RAM and a Samsung SSD 840. Additionally, the computer was running the MASSim simulation at the same time. The system load of the machine is plotted in Figure 23.4. Particularly, the plot shows that the system still has reserves for additional agents or more complex behaviour models. During the contest preparation, we have also been able to test with two instances of our implementation with 28 agents each on the same machine. However, the plot also indicates increasing memory consumption over time, which is probably rooted in the agent behaviour implementation and not in the decision-making itself. We are already aware of some implementation parts that are not properly freeing their resources, as this is not crucial in the limited contest time of max. 3000 steps for one match over 3 simulations.

All in all, the measures from our experiment reveal that the decision-making and planning framework RHBP is scaling up well and it can sufficiently be applied with a large number of agents, respectively robots.

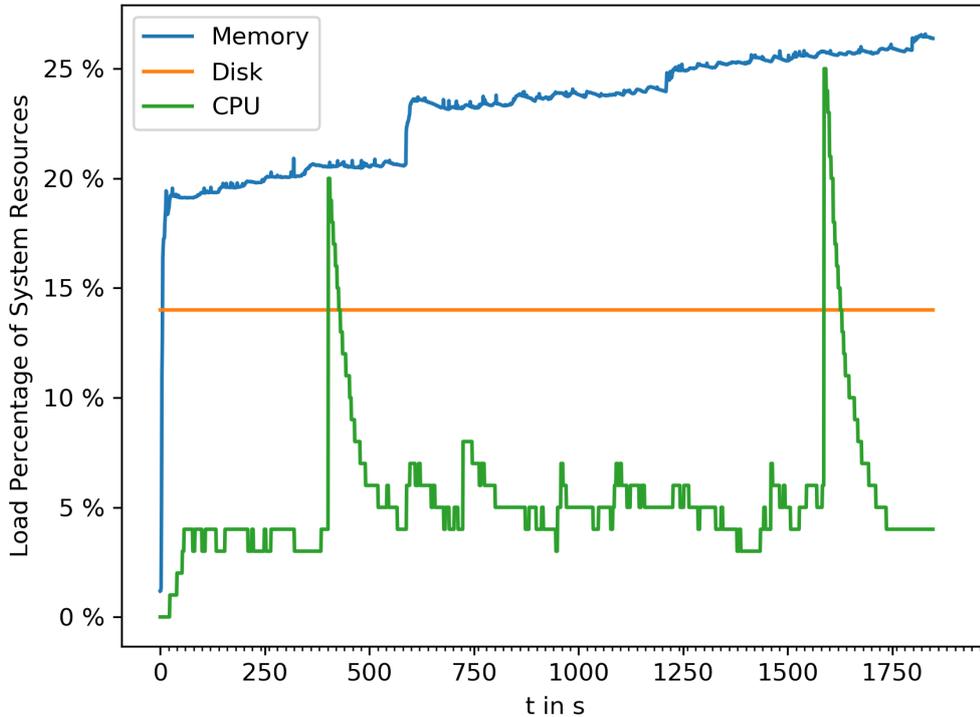


Figure 23.4.: System load during experimental match with 3 simulations á 1000 steps.

The final contest ranking of MAPC 2017 is listed in Table 23.1, while the individual simulation match results are summarised in Table 23.2. To guarantee replicability the source code of all teams, the simulation server, and replays from all matches are available on the official contest homepage⁴. The overall result makes visible that our solution in 2017 was not yet competitive with respect to the quantitative and scenario-specific measures of the contest. During the contest, we were constantly improving our implementation. The implementation was never free of bugs, and for this reason, we kept working and debugging on the code until the last match. Moreover, on the first day of the competition week, our auction bidding implementation was not properly tested

⁴<https://multiagentcontest.org/2017/>

23. Projects

and functioning as expected and therefore we deactivated it after the first simulation for the first day of the competition. The most problematic and critical part of our implementation was the assignment of tasks to individual agents. Here, we had to deal with several bugs and concurrency issues that eventually result in unsolvable jobs or deadlocks. Moreover, during the contest, we discovered several implementation bugs in the team coordination logic. Several constraints, originating in varying role capabilities, lead in major problems, too.

Table 23.1.: MAPC 2017 final ranking.

Pos.	Team	University	Framework / Language	Total Score
1	BusyBeaver	Technische Universität Clausthal (Germany)	Pyson	54
2	Flisvos 2017 (“Hot Stuff” edition)	Rutgers University (USA)	Python	36
2	Jason-DTU	Technical University of Denmark (DTU)	Jason + CArtAgO	36
4	SMART-JaCaMo	Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)	JaCaMo	33
5	lampe	Technische Universität Clausthal (Germany)	LAMPE	16
6	TUBDAI	Technische Universität Berlin (Germany)	RHBP	6
7	Chameleon	Shahid Beheshti University (Iran)	Java	4

Nevertheless, the strongest part of our team is the adaptivity and robustness of the solution, which are very important characteristics in robotics but less important in the 2017 edition of the MAPC. Our implementation can cope well with unexpected situations and misleading perception. We have proven this in the (unfortunately) not counted first simulation of the match against team lampe. In this simulation, a truck of each team was deadlocked in the middle of the map, due to an unexpected simulation server error, this was already mentioned above. Our team was able to deal with the situation of one unusable agent and still made a profit in the simulation. In contrast, the lampe team suffered a lot from this event and experienced substantial loss.

Table 23.2.: MAPC 2017 simulation match results. 3 simulations per match. 3 score points per won simulation if initial seed capital of 50000 is surpassed, 2 score points if not. 1 score point for a draw. Simulation match results in plain text. Score points are boldfaced.

vs	Chameleon	Flisvos 2017	Jason-DTU	lampe	SMART-JaCaMo	TUBDAI
Busy Beaver	186516 : 42899	236028 : -6888	88646 : 77716	80086 : 51626	132160 : 91778	50558 : -53862
	226438 : 41842	179138 : 51995	138592 : 53370	213065 : 49875	165324 : 120778	233538 : -4484
	216723 : 41076	196837 : 53124	128250 : 73308	235248 : 35074	156650 : 109815	257098 : 21993
Chameleon	9 : 0	9 : 0	9 : 0	9 : 0	9 : 0	9 : 0
	46755 : 119691	26018 : 119821	38716 : 210129	27319 : 218952	-1674 : 53602	
	35001 : 156423	29233 : 182542	-5060 : 160596	20026 : 215600	37742 : 51830	
Flisvos 2017	42242 : 167068	11819 : 136206	23787 : 1709	-20120 : 229675	38464 : 13208	
	0 : 9	0 : 9	2 : 6	0 : 9	2 : 6	
	121141 : 35837	106909 : 44669	120429 : 101627	154675 : 2564		
Jason-DTU	125639 : 135220	129985 : 145621	125252 : 70705	108436 : 30650		
	157291 : 48352	177058 : 21699	55394 : 57073	132340 : 16183		
	6 : 3	6 : 3	6 : 3	9 : 0		
lampe	157063 : 53628	149751 : 160820	132481 : -3675			
	139869 : 85659	213863 : 149896	176406 : -10630			
	225755 : 100880	181097 : 105952	115192 : 37275			
SMART-JaCaMo	9 : 0	6 : 3	6 : 3	9 : 0		
	139879 : 150826	107965 : 81801 ⁵				
	130303 : 168878	21695 : 6843				
SMART-JaCaMo	104604 : 129704	44956 : 32526				
	0 : 9	7 : 0				
	119881 : 22001	88611 : 3512				
SMART-JaCaMo	105950 : 30867					
	9 : 0					

⁵ Sim 1 had to be repeated because of a bug in the simulator where multiple agents got stuck in the middle of the map. First result: 17 387 : 51 300.

23.2.2. Multi-Agent Programming Contest 2018

The 2018 edition of the contest featured a modified scenario of 2017. The differences with respect to the delivery jobs have already been mentioned above. Additionally, the scenario was extended with the obligation of building wells for generating the actual points required for winning the matches. Building wells required money that is earned by completing delivery jobs or selling resource items in a shop. The well-building extension in 2018 fosters more interaction and direct competition between the teams aside from increasing the overall search space for finding the most optimal solution. Moreover, the number of used agents per team was increased from 28 to 34 agents.

In the 2018 participation, it was the target to evaluate more recent features of RHBP that have not been tested before. In particular, we examined the knowledge base component more intensively, behaviour hierarchies realised with the *NetworkBehaviours* and the application of the self-organisation extension *so_data*.

The 2018 edition of the contest had 5 international participants, with two independent participations from the TUB. Both teams from TUB are applying a RHBP-based implementation as a succession of the introduction of RHBP in the 2017 contest. However, both teams did only share the starting point with general components developed in the year before, like the protocol proxy `mac_ros_bridge` and the integration of the routing library `GraphHopper`. Despite these general components, both teams developed their solution completely from scratch, which resulted in two very different general strategies. Nevertheless, both implementations are highly decentralised with all agents taking operational decisions autonomously using RHBP behaviour models. Only the evaluation of published delivery jobs is done in centralised components in both cases. Moreover, both teams apply an own contract-net based implementation for the coordination of the assembly and delivery for fulfilling the jobs.

In the following Section 23.2.2, we analyse the implementation and results of the Team *Dumping to gather*, which is formed by two students of the Applied Artificial Intelligence Project. Subsequently, in Section 23.2.2 we discuss details from the second team *TUBDAI* that was realised as a Master's thesis project. Both teams have been supported by the technical supervision of the author of this dissertation. Nevertheless, strategic decision or implementation was never communicated or shared between both teams. The application-specific implementation was only done by the students. The final

ranking of the 2018 contest is listed in Table 23.3, with detailed simulation match results in Table 23.4. To guarantee replicability the source code of all teams, the simulation server, and replays from all matches are available on the official contest homepage⁶.

Table 23.3.: MAPC 2018 final ranking.

Pos.	Team	University	Framework / Language	Total Score
1/2	SMART_JaCaMo	Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)	JaCaMo	33
1/2	TUBDAI	Technische Universität Berlin (Germany)	RHBP	27
3	Jason-DTU	Technical University of Denmark (DTU)	Jason + CArtAgO	21
4	Dumping to Gather	Technische Universität Berlin (Germany)	RHBP	9
5	Akuanduba-UDESC	Santa Catarina State University (UDESC)	n/a	0

Team Dumping to gather

The team *Dumping to gather* opted for a team-oriented strategy where all agents are able to do all types of tasks and independently choose amongst them while cooperating in order to prevent redundant work. In this respect, all agents are responsible for exploration and resource gathering.

The implemented main strategy followed the RHBP approach of 2017 with on demand job processing and item gathering. The implementation is addressing all job types except for auction jobs. Additionally, the newly required well building is also executed on-demand if sufficient money is available.

In general, all agents share information about items, facilities, wells, agents, and visited grid points with each other. The sharing of all permanent information is comprehensively making use of the RHBP knowledge base component that we have introduced in Section 18.4. All non-persistent information is shared through dedicated ROS topics in a broadcast-like manner, which is the case for all information that is automatically updated every simulation step by the MASSim simulation server.

⁶<https://multiagentcontest.org/2018/>

23. Projects

Table 23.4.: MAPC 2018 simulation match results. 3 simulations per match. 3 score points per won simulation. 1 score point for a draw. Simulation match results in plain text. Score points are boldfaced.

vs	Dumping to Gather	Jason-DTU	SMART_JaCaMo	TUBDAI
Akuanduba -UDESC	130 : 6402	401 : 11586	74 : 22271	0 : 13692
	0 : 18677	1014 : 25366	18 : 101995	0 : 46008
	0 : 16936	0 : 1693	65 : 37062	0 : 12228
	0:9	0:9	0:9	0:9
Dumping to Gather		3030 : 3472	2303 : 4581	216 : 11404
		9973 : 14332	9163 : 62413	1192 : 25954
		5847 : 8545	4367 : 29543	798 : 5577
		0:9	0:9	0:9
Jason- DTU			3644 : 6800	146 : 13368
			9142 : 7079	699 : 35190
			770 : 41324	1391 : 9061
			3:6	0:9
SMART- JaCaMo				2337 : 515
				5098 : 1808
				2598 : 600
				9 : 0

For the implementation of the exploration, the simulation map area is partitioned into grid cells reflecting the cell size of the simulation. The coordination of the exploration is implicit: Already visited grid cells are being shared amongst the agents through the knowledge base so that each agent can choose a not yet visited point. Here, the agents are choosing the closest not yet visited cell until all resource nodes have been discovered. However, this implicit self-organised exploration is an own scenario specific implementation that does not apply the self-organisation extension of RHBP.

Furthermore, the new MAPC 2018 scenario features upgrades and selling of item is not used because upgrades seemed too expensive in comparison to the money required for building wells and selling items was also considered as less beneficial in comparison to fulfilling entire jobs.

To coordinate the job fulfilment *Dumping to gather* uses a contract-net-based protocol (Smith, 1980) that enables decentralised coordination. For this reason, each agent has an auctioneer node, which is responsible for managing the auctioning process amongst the agents, and a bidder node, which selects and bids for a sub-task. Every incoming job is assigned to an auctioneer node in a round-robin fashion. The auction is done in stages. First, the job is decomposed into different tasks according to the items and required capacity. It is possible that the tasks are linked since they can depend on a preceding task. For example, if one task contains the gathering of an item, and another task contains the delivery of that item to a store, both tasks should be operated by the same agent. The auctioneer node decomposes the job and publishes the sub-tasks. Secondly, the bidder nodes of the agents decide if they are able to do a sub-task by considering their load, the task's time restrictions and their already assigned plans and bid for the task with the corresponding computed plan. Selecting a bid is done by choosing the plan that will be finished first in simulation step time. In case a job cannot be finished in time, all unused items are stored in storages and later reused.

The agents are largely autonomous without any central decision-making. Although the agents rely on the auctioneer nodes of the other agents to publish sub-tasks and coordinate the internal auctions, the system is resilient against failures of individuals due to the round-robin assignment of the coordinating agent as described before.

If an agent is not executing a sub-task of a delivery job, it is having some idle time. If that is the case, the agents have the following priorities: First, fulfilling jobs, secondly,

23. Projects

building wells if sufficient money is available, thirdly dismantling wells of opponents and lastly exploring the environment. In addition, exploration is done by all agents at the beginning of all simulations to discover all resource nodes that are required to assemble items. Well construction is done in a group of agents to speed up the construction process. The build-up wells are positioned around the border of the simulation map to minimise the probability of accidental discovery by the opponent because the borders are not visited without explicit exploration order.

The implemented general architecture is very similar to the agent architecture of 2017, but it is extended with separated ROS nodes for initiating and participating in auctions used for the contract-net based coordination.

Furthermore, the team developed its own implementation pattern that they called BehaviourGraph. A BehaviourGraph is an abstract component that is used to group certain behaviours, sensors, conditions, and goals of a higher-level task. In contrast to the NetworkBehaviour, which are applied by the *TUBDAI* implementation, the BehaviourGraphs are not adding additional decision-making layers, all behaviours and goals are still registered to the same RHBP manager. BehaviourGraphs are used only for providing convenience functions for common initialisations, updating knowledge, adding preconditions to all contained behaviours, resetting, as well as task-specific functionalities. BehaviourGraphs particularly simplify modelling by avoiding repeated condition assignments.

The implemented behaviour model used for decision-making is visualised in Figure 23.5. Notably, the conditions modelled for the BehaviourGraphs are reflecting the aforementioned main priorities of the agents. Each main task is modelled in one BehaviourGraph. The implementation of the job fulfilment and related coordination, see lower left BehaviourGraph, is integrated similarly to the implementation of *TUBDAI* 2017. Here, the actual job tasks are determined by the RHBP-independent coordinator and then operated by the multi-purpose `execute_plan` and `finish_plan` behaviour, which is similar to the approach of *TUBDAI* 2017. The comparison between the model of *Dumping to gather* in Figure 23.5 and the previous version of *TUBDAI* 2017 in Figure 23.3 is also visualising the increased complexity of the model with more behaviours, conditions, and goals. In detail, the old *TUBDAI* model contained only 5 behaviours and 2 goals, whereas we have 5 BehaviourGraphs with all in all 5 goals and 13 behaviours. Never-

theless, a detailed comparison shows that the increased complexity is deduced from the new scenario that requires additionally well-building, well dismantling, and exploration. In the 2017 scenario, exploration was not mandatory because resource nodes have not been required to assemble items.

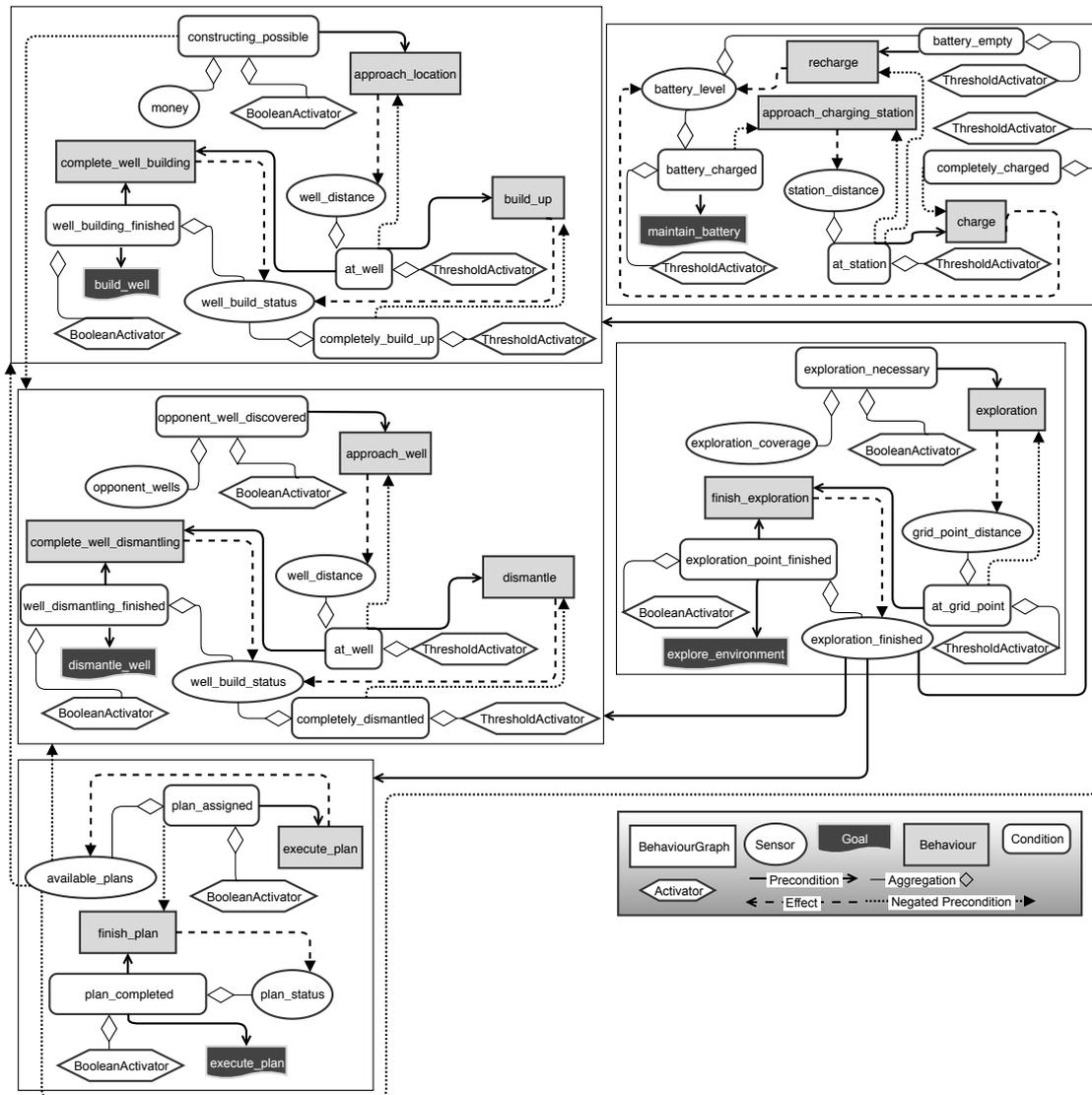


Figure 23.5.: Behaviour model for agent task execution of *Dumping to gather*. BehaviourGraphs are containers to apply conditions to multiple behaviours.

The team *Dumping to gather* achieved a consistent performance with their implementation, resulting in a 4th place. They have been able to address all mandatory elements of

23. Projects

the scenario and made points and gained money in all simulation matches. They gained a clear win against team UDESC, which was having general problems with timeouts etc. Furthermore, the match against Jason-DTU has been very close and counterbalanced but was barely lost. The other two matches against the two best teams of *TUBDAI* (2018) and Smart_JaCaMo have been lost more clearly.

Team TUBDAI

The main strategy of the *TUBDAI* implementation is a stockpile with feedback strategy. Here, it is the goal to gather resources, assemble items in advance, and fulfil jobs that can be performed with the available item stock. First, the agents explore the environment until a resource node for each base item is discovered. Then all agents start to gather resources to achieve a stock of base items depending on the assigned priority for further assembly. Agent groups are formed dynamically to assemble items once the agent capacity is exceeded. The jobs are then prioritised according to a calculated reward. The reward is given by a ratio of required work and revenue. If a job reaches the defined reward threshold and all items of the job are currently in stock, the job is executed. Furthermore, urgent item demands such as induced by mission jobs, which would result in fines if not fulfilled, are also considered on demand based on feedback from other higher-level components like the job planner by dynamically adjusting the priorities of the respective items. The feedback allows reacting and changing the priorities of finished products and base items, which results in better adaptability to changes on demand, while still maintaining the efficiency of a general stockpile strategy. The advantage of this stockpile with feedback strategy is a fast job performance, enabling decentralised decision-making by individual agents to avoid a single point of failure, while also enabling on-demand execution based on the priority feedback. A disadvantage is that assembled finished products may not match any job and cannot be used, hence potentially wasting time resources.

Even though *TUBDAI* aimed for a decentralised solution, the rating of jobs is centralised in one agent for simplification because there is no difference in between the job perception amongst the agents and the cost-benefit analysis is as well agent independent. The distribution of task from the decomposed jobs is then realised using a contract net protocol (Smith, 1980) involving all agents in a decentralised and distributed fash-

ion. All other components are completely decentralised, too. Each agent has its own RHBP-based decision-making and planning component.

For the execution of jobs, an algorithm involving a chain of decisions has been developed. At the end of the chain, the scenario-specific job planner inspects which items currently provide high money returns. This information is converted into a finished product prioritisation. This prioritisation is then used by the next link in the decision chain, the assembly decision. The agents always decide autonomously which items should be assembled next and share their decision with the others. This decision is mainly based on what is needed for job execution as well as on the available items, and the finished products that are already assembled. In turn, this decision creates a prioritisation of base items that are most needed for assembly. These prioritisations are then used by the gathering algorithm to decide which base item to gather. An important point is that the taken decisions are exchanged amongst the agents to avoid conflicts and unwanted parallel work.

One advantage of the stockpiling strategy is that it creates job idle time, respectively agents for dismantling, building, and exploration because not all agents are always striving for fulfilling the currently available jobs. Particularly, this additional freedom for individual context-specific decisions of the agents is fostering the adaptation and reaction to varying opponent strategies.

The self-organised exploration of the environment is implemented with the `so_data` library of RHBP. Each agent emits its own location with `SoMessages`, so others do not explore these points as well while selecting a target location close-by that has not been visited for the longest time. The exchanged `SoMessages` are filtered in a decentralised manner in each agent by the `SoBuffer` module instance of the `so_data` library, which provides the base for the calculation of a discrete heat map. Likewise, the heat map is used to select the appropriate exploration target locations, while the initial exploration phase is stopped after resource nodes for all available base items have been discovered. Later during the match, one drone agent is also exclusively patrolling the map border to discover opponent wells, which would be difficult to discover accidentally during normal job operation.

The general idea behind the well-building is to build wells at locations that are difficult to discover or difficult to reach by the agents of the other team to neglect the requirement

23. Projects

of an explicit well defence strategy. More details about the particular implementation are provided below in this section.

The dismantling of opponent wells is executed if no own wells are constructed by the agent and if opponent well locations are known. In particular, the agents are prioritising the closest near-by wells for dismantling.

The agent architecture is as well based on the 2017 approach but it replaces significant parts of the ROS-topic-based communication in the perception with custom information provider modules, which are feeding the information more directly into RHBP sensors. The information providers avoid some communication overhead coming from ROS communication to increase the efficiency of the implementation. Furthermore, the *TUBDAI* implementation is not making use of the knowledge base and instead it shares information more distributively with the self-organisation library `so_data` of RHBP .

In contrast to the last-years participation and to the implementation of *Dumping to gather* NetworkBehaviours are frequently used for structuring and controlling the major responsibilities of the agents on the highest decision-making level. In particular, NetworkBehaviours are modelled for controlling resource exploration, discovering opponent wells, dismantling opponent wells, building wells, gathering base items, assembling finished item products, and delivering jobs. All NetworkBehaviour implementations are inheriting from an abstract scenario-specific NetworkBehaviour implementation `GoAndDoNetworkBehaviour` that incorporates battery management and travelling on the simulated map, which is a basic capability of all higher-level tasks in the MAPC scenario. The high-level decision-making behaviour model is visualised in Figure 23.6. Considering the fact that this model covers only the highest-level of decision-making with additional nested models within each of the shown NetworkBehaviours a comparison with the model of the previous participation in 2017, illustrated in Figure 23.3, indicates that the complexity of the 2018 *TUBDAI* implementation is considerably larger. Similarly, a comparison against the other TUB team of 2018 shows that the model has more conditions and relationships, as we have to compare the NetworkBehaviours against the BehaviourGraphs of *Dumping to gather*. Moreover, we see that the structure of the high-level tasks is considerable different, while *TUBDAI* is inherently including the charging responsibility in each NetworkBehaviour, *Dumping to gather* is using a distinct BehaviourGraph for it. Overall, we see that the structure of *TUBDAI* is more fine-

grained, e.g., we have distinct NetworkBehaviours for assembly, delivery, and gathering. Likewise, the entire number of behaviour and goal instances is considerably larger, but this is partly because of redundant charging goals and behaviours in each NetworkBehaviour. In detail, the old *TUBDAI* model contained only 5 behaviours and 2 goals, whereas *TUBDAI* 2018 has 3 goals and 7 NetworkBehaviours each again containing 2 goals and 4-5 behaviours.

During the realisation of the *TUBDAI* implementation, we discovered a new general implementation pattern for lower-level decisions in sensor, condition, behaviour, and goal implementations. The new implementation pattern is taken from the self-organisation extension of RHBP, which offers a component called DecisionPattern that can be used by certain behaviours and sensors to share low-level decisions between the decision-making layer of RHBP and the actual implementation of the behaviour. This implementation pattern is further generalised and not anymore applied only for self-organisation related low-level decisions. Instead, the DecisionPattern works as an aggregate that takes a low-level decision like selecting the closest charging station, which is then shared amongst the sensors and conditions of behaviours and goals as well as the actual behaviour implementation. This pattern was found to be useful for sharing information between behaviours and sensors of one agent. At the end, this pattern is used frequently for sharing information between behaviours and sensors of one agent for various lower-level decisions, including those that did not involve self-organisation.

All in all, the last-minute changes of the well-building strategy paid off because this has been a unique strategy, which was not expected by the opponents. Here, this particular strategy becomes especially attractive, as the original well-building strategy has been changed three days before the contest and the RHBP-based architecture supported a quick integration of the new strategy, which did not require comprehensive code changes. Originally, it was the plan to build wells with trucks at the edges of the map area. Unfortunately, this turned out to be less efficient with the final contest maps published three days before the contest. Instead, we shifted to the strategy that making use of so-called off-road locations on the map in order to neglect an explicit well defence strategy. Off-road locations are locations that are not connected to the street network and thus only reachable by drones.

The conducted last-minute changes comprise shifted role responsibilities like only

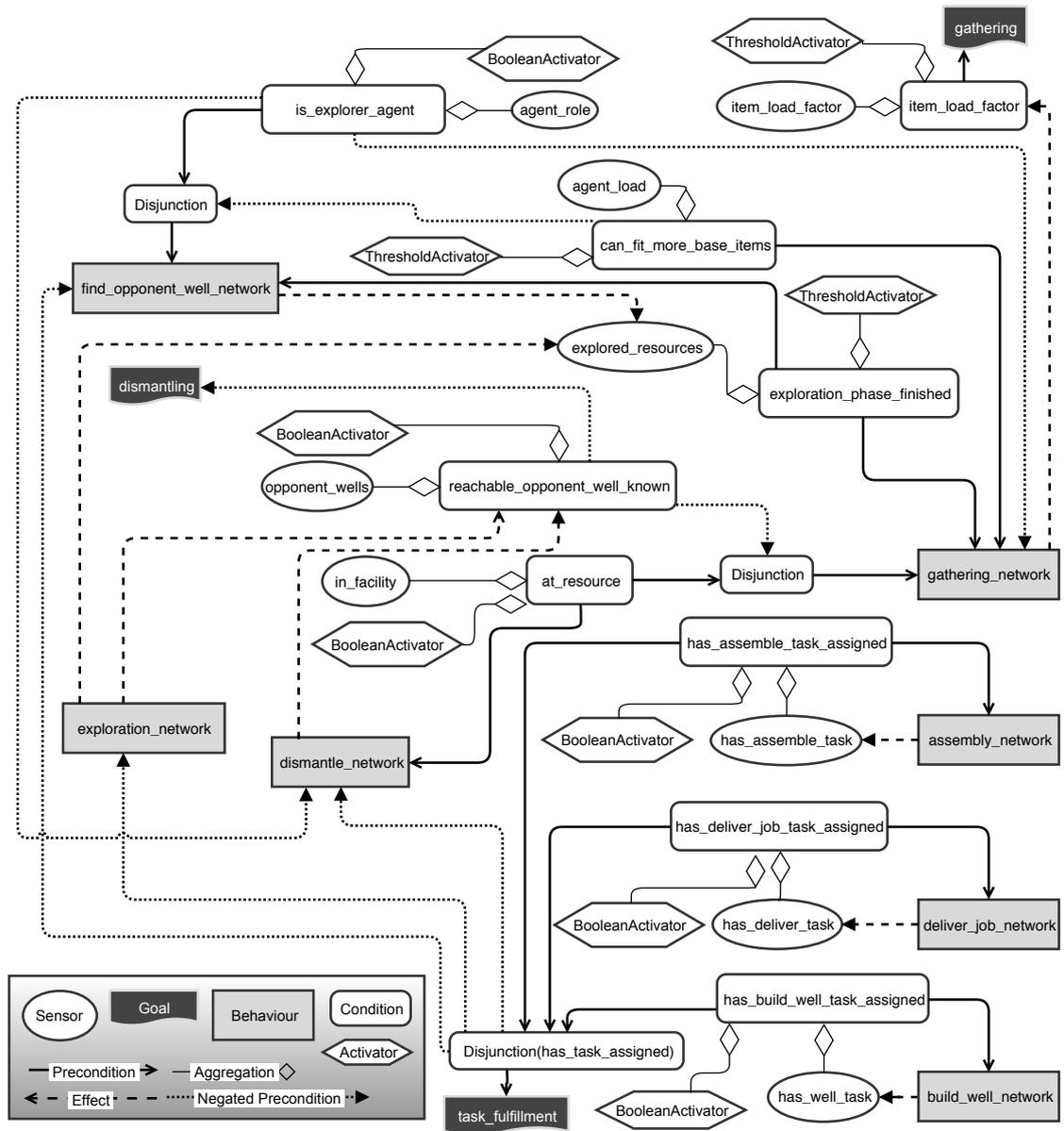


Figure 23.6.: High-level decision-making behaviour model for agent task execution of TUBDAI 2018. All listed behaviours are NetworkBehaviours containing nested behaviour models. Each NetworkBehaviour contains 2 goals and 4-5 behaviours.

drones building wells; a changed exploration that is not focusing anymore on the map borders; less priority on the job fulfilment because the mandatory and rare drones are often busy with well-building; and a higher priority on dismantling to efficiently use the increased job idle time for non-drone agents.

The encapsulation of code within behaviours and aggregation in `NetworkBehaviours` made these changes very intuitive and robust. By duplicating certain `NetworkBehaviours` and switching around preconditions and effects, most of the strategy was adapted, requiring only small code changes within the behaviours. It was stated by the student that similar changes would have been potentially much more difficult to achieve with a traditional sequential programming approach.

The runtime adaptiveness of RHP could be observed at the match against `Akuanduba-UDESC`. While many resources were usually used for dismantling opponent wells, there was no dismantling required against `Akuanduba-UDESC` because of their timeout issues, which resulted in almost 100% inactivity of their agents. This freed up resources for other tasks for our agents. The agents were able to adapt to this unexpected situation and increased their job performance from an average of 63 jobs to 127 jobs. This shows that `TUBDAI` agents adapted well to a situation without opponent wells by shifting priorities accordingly.

Moreover, the simulation configuration used in the contest was very different from the sample configurations that have been published together with the server source code for the contest preparation. The biggest difference was that it was very easy to gain money for building wells within the contest. While in the sample configurations (which we assumed to be similar to the contest configuration) most jobs offered rewards of less than 500, the jobs in the contest had much higher rewards, i.e. jobs exceeding 10,000 in reward, whereas building wells stayed on the same price level. In consequence, building wells became easier, and the strategy of building and defending more critical. Nevertheless, the `TUBDAI` implementation has shown that it was able to adapt and handle this unexpected setup successfully.

In the end, `TUBDAI` only lost the final match against `SMART_JaCaMo`. The reason was that they efficiently dismantled our non-defended off-road constructed wells exclusively with two of their drones, which have been only responsible for discovering and dismantling of our wells. Moreover, their skill was upgraded directly at simulation start,

23. Projects

so they dismantled more efficiently. Furthermore, due to the road-agents not dismantling at all, team SMART_JaCaMo did not suffer from failed goto actions like other teams. A question that might come up at this point is why our RHBP-based approach was not able to adapt automatically to this situation. The reason is that RHBP is only having the opportunity of adaptation if alternative behaviour implementations are available, which was not the case for the *TUBDAI* implementation.

Nevertheless, a detailed analysis of the replays and the published code of SMART_JaCaMo showed that their unique strategy was also not a result of their adaptive strategy or implementation, but rather a result of their last-minute changes in implementation the human team has made after analysing the matches before the very last match of the competition between *TUBDAI* and SMART_JaCaMo. We could prove this by playing about 30 simulations with the a priori published simulation sample configurations and the not modified SMART_JaCaMo code in which SMART_JaCaMo was not able to win any simulation against our team. In detail, the SMART_JaCaMo team added a second drone for exploration, implemented immediate skill upgrade after simulation start, disabled dismantling in trucks, enabled dismantling for exploration drones, and created a second drone exploration algorithm, that targets locations that are typically used by our agents to build wells. All these changes were made in short time-frame while we were competing against the other three teams. The changes seemed to be very robust and side-effect free, which is impressive for such a substantial last-minute change. Nevertheless, it has to be stated that the SMART_JaCaMo approach did not follow the rules of the competition because teams are encouraged to refrain from code changes during the contest that are not pure bug fixes of their own strategy. This fact also leads to an official correction of the final placement by the steering committee of the competition resulting in a shared top spot between *TUBDAI* and team SMART_JaCaMo⁷.

23.2.3. Multi-Agent Programming Contest Conclusion

The evaluation in the MAPC shows that RHBP has reached a maturity level that allows for successful competition with other multi-agent frameworks in a quantitative evaluation in general, which is especially demonstrated by winning the shared top spot of *TUBDAI* in 2018. However, the results are still dependent on the developed scenario-

⁷<https://multiagentcontest.org/2019/01/23/results.html>

23.3. EffFeu Project: Efficient Operation of Unmanned Aerial Vehicles for Industrial Firefighters

specific strategy, which is proven by the different success of the two teams using RHBP in 2018. Nevertheless, the participation in both years provided several examples that have verified the robustness and adaptation capabilities of the approach in totally unexpected situations, such as the server deadlock in 2017, strongly biased simulation configurations and the unexpected passive competitor Akuanduba-UDESC in 2018. Furthermore, the 2018 edition again illustrated the scalability of our solution because even with the increased number of agents, it was still possible to run both solutions in two ROS instances on the same machine as in 2017.

Additionally, the participating students reported that the necessary separation of concerns and the creation of small components due to the given sensor, behaviour, and goal structure of RHBP is fostering a modular system architecture. Similarly, this modularisation allows to adapt a solution quickly to changed requirements from a developer point of view. Especially these advantages have been demonstrated with the last-minute changes of the well-building strategy of TUBDAI in 2018, which have been implemented only three days before the contest.

Future participation could also explore the application of RHBP for the task/job coordination as well as testing not yet in the MAPC context used features of the RHBP framework. For instance, the explicit task coordination with the *Decomposition Manager* for the coordination and cooperation of a multi-agent system as well as the reinforcement learning extension could be examined in more detail in future contest editions.

23.3. EffFeu Project: Efficient Operation of Unmanned Aerial Vehicles for Industrial Firefighters

The project described in the following evaluates the RHBP in a complex robotics scenario testing various capabilities of the framework such as the self-adaptation of the decision-making and planning. Different from previous evaluation experiments and research projects, this project is also considering the integration with end-users of the robotic system and how actual user commands are mapped to the RHBP approach.

The number of unmanned aerial system (UAS) applications for supporting common firefighters and industrial firefighters is growing in recent years (Skorput, Mandzuka, and Vojvodic, 2016; Twidwell et al., 2016). In particular, exploration, surveillance,

23. Projects

monitoring, and documentation of accidents, dangerous places, and critical infrastructure are evaluated. Up to this point, most gathered information, for instance, video footage, is analysed manually by trained professionals without any additional support. Moreover, aerial systems do always require a trained pilot, an extra person who is using either semi-automated control based on GNSS or full manual control in GNSS-denied environments (e.g. indoors or close to environmental structures such as buildings or trees). Such analysis and control create an enormous psychological and emotional load, especially in critical and hectic situations, apart from the fact that needing two extra persons to operate the drone is unacceptable.

In order to enable a mission-guided application of drones and reduce the load of manual control and analysis, the project *Efficient Operation of Unmanned Aerial Vehicles for Industrial Fire Fighters* (EffFeu) aims for a holistic integration of UAS in the daily work of industrial firefighters.

The project is organised into three subprojects, one for each of the three consortial partners: Gemtec GmbH is dealing with the integration of UAS into the safety and security management system and the development of the drone hardware, ART+COM AG investigates interactive user interfaces for UAS, and DAI-Labor of TU Berlin extends the intelligence of the UAS.

In detail, the Gemtec subproject is focused on integrating the sensory output of drones, its sensor payloads and UAS controls into their safety and security management system so that data and information acquired can be seamlessly used by control room personnel and task force for human decision-making.

The ART+COM subproject works on the augmentation of the camera stream with additional relevant information in real-time. They also develop user interfaces that guarantee the intuitive and goal-oriented interaction with the drone, by either by the control room personnel or by the task force on-site.

The subproject of the DAI-Labor focuses especially on objects and situation recognition, autonomous navigation, and the intelligent mission-oriented control of aerial systems. In the context of this dissertation, the focus lies on the mission-oriented control of the aerial systems, which is realised on the foundation of the RHBP framework.

In the following, we introduce the overall architecture of the project and its components first in Section 23.3.1. Subsequently, we describe how the mission-guided control

based on RHBP has been realised in Section 23.3.2, which provides details about three particularly implemented scenarios. Finally, Section 23.3.3 assesses the results obtained with RHBP as well as the perspective of the overall project.

23.3.1. EffFeu System Architecture

In order to provide some technical background about the project infrastructure and the involved components, the EffFeu architecture is briefly described in the following.

The nature of the EffFeu architecture is a holistic structure of a distributed system of distributed systems. Figure 23.7 gives an overview of the overall EffFeu architecture.

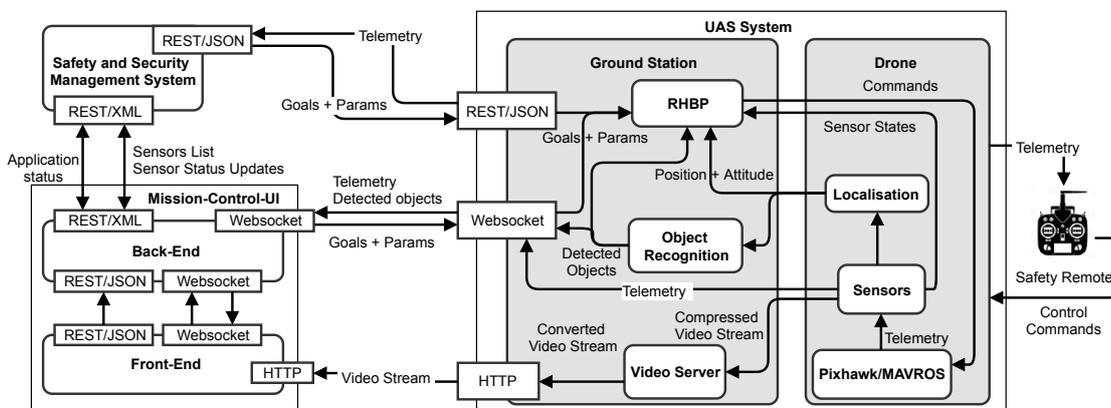


Figure 23.7.: EffFeu system architecture. Arrows indicate the direction of the data flow.

The main components of the system are reflecting the responsibilities of the project partners, namely the *Safety and Security Management System*, the *Mission-Control-UI*, and the *UAS System*.

The *Safety and Security Management System* software allows to integrate, monitor, and control various sensors and actors of industrial plants, for instance, fire warning devices, video systems, and access control systems. Responsible for the operation of the software are experts that manage occurring alarms and events in a central control room, e.g. on an industrial plant with own fire brigade, like a chemical plant.

The *Mission-Control-UI* is an intuitive and goal-oriented user interface that is used by the rescue forces in operation on a ruggedised mobile tablet computer. It allows to select and configure high-level mission goals for the drones and gives access to the automatically gathered information, like recognised objects, as well as the current system

23. Projects

state. To manage several parallel clients, the component is split into a *Front-End* and *Back-End*.

DAI-Labor is concentrating on the drone and ground station components and the provisioning of interfaces, which can be used by the partners both for monitoring the drone and its sensor outputs as well as for controlling it using high-level mission goals. The *UAS system* consists of the major components: *RHBP*, *Object Recognition*, *Localisation*, and *Web Video Server*. Each component is embedded in several ROS processes and can reside and run either on the drone or on the ground station computer.

The *RHBP* component is responsible for the task-level decision-making and planning of the system. It is instructed by high-level mission goals that are selected and parametrised in the user interface of the *Safety and Security Management System* component or the *Mission-Control-UI*. The RHBP core interprets the goals and creates the flow of activities that are then translated to low-level drone behaviours and executed. More details are given below in Section 23.3.2.

The *Localisation* component enables seamless operation of the drones indoor and outdoor based on GNSS information, Red Green Blue Depth (RGBD)-based Simultaneous Localization and Mapping (SLAM), visual odometry and prior known environment maps.

The *Object Recognition* component is a one-shot multi-object detector for multiple categories and varying resolutions. The pre-trained part may reside on the drone itself, the online training unit is running on the ground station. According to the firefighter's preferences and priorities, it is capable of detecting persons, vehicles and hazardous goods in real-time, also reconstructing the 3D world coordinates of objects in the image.

The main task of the *ground station* is to provide gateway-like functions such as converting telemetry data and sensor information into needed representations and providing them over different interfaces. Here, the function of the ground station is bilateral because also incoming control messages will be translated and delivered to according components.

The *Localisation*, *Object Recognition*, and the input from external components provided by the *ground station* interfaces contribute information that can be used by RHBP for the task-level decision-making and planning.

The *Safety Remote* is only a backup control instance that allows taking over control in case the system behaves maliciously aside from being currently required for regulatory

reasons.

Furthermore, the drone itself is controlled on low-level by a Pixhawk autopilot system with a PX4 software stack that is instructed by the *RHBP*. All higher-level components of the UAS, including RHBP, Localisation, and Object Recognition, are implemented with the ROS framework. The ROS component instances, so-called nodes, can be freely distributed between the ground station computer and the drone itself. In particular, the ROS nodes on the drone are running on a comparable powerful small-size x86 companion computer (Intel[®] NUC7i7BNH) that has a serial connection with MAVLink protocol to the Pixhawk.

The novelty of the project architecture is the realisation of a holistic integration of UAS based on ROS with additional components that are required to achieve a seamless and mission-oriented workflow for rescue forces. Here, the additional components from the perspective of the UAS are the Safety and Security Management System and the Mission-Control-UI. To our best knowledge, this is the first time that an industrial Safety and Security Management System is equipped with means for controlling and monitoring drones in operation. Furthermore, the integration with the Mission-Control-UI is different from existing control approaches because it entirely focuses on a mission-oriented and task-based perspective instead of a traditional motion-oriented control.

23.3.2. Mission-guided Control with RHBP

Today's control of drones is very much focused on the spatial control of the systems; the users either control full manually, or drones follow preprogrammed GNSS trajectories in an automated fashion. However, most professional users are not interested in the actual aerial system and which trajectories it has to fly in order to collect the required information. An alternative is a mission-guided control that defines the scope of operation for the drone with goal specifications. In order to realise a mission-guided control of drones in the EffFeu project, RHBP is used to implement the execution and decision-making. Before details about three different application scenarios are presented in Section 23.3.2, Section 23.3.2, and Section 23.3.2, we first outline how RHBP is interfaced from an end-user perspective in the project context in the following subsection.

End-user and UI integration

In EffFeu, the user interfaces are connected to RHBP through an external goal API that lists all available goals and possible parametrisations. Subsequently, the user interface allows activating and configuring goals on demand through the external goal API. The high-level goals that are enabled by the user are mapped to corresponding RHBP goals that are automatically instantiated or configured by the interface component. Here, it is also possible that a single goal from the user perspective, like follow a person, is internally decomposed to several RHBP components. We differentiate such user goals from the internal RHBP goal components. A user goal is, for instance, a combination of internal safety goals considering the battery consumption and the actual goal of holding a certain distance to the tracked object. Moreover, in some cases, user goals are a composition of RHBP goals, behaviours, and sensors instances to be able to parametrise and configure the behaviours. This is necessary because RHBP does not support the passing of parameters on the logical planning level, which are required for a behaviour to be executable. In consequence, running behaviour instances that require additional information, like the flight destination, need to be dynamically instantiated or adjusted during runtime, independent of RHBP's plan execution. In the EffFeu scenario, we dynamically create instances of the behaviours with the goal specific parameters because altering existing instances is problematic as the goals are allowed to be instantiated multiple times. For other more abstract behaviours such as take-off, landing, and collision avoidance, the corresponding user goal does only contain goals that consist of conditions formed with RHBP sensors and activators. The detailed relationships and communications of the EffFeu RHBP component introduced in Section 23.3.1 are visualised in Figure 23.8. Furthermore, the diagram highlights that the management of the user goals is handled in a scenario, respectively mission-specific manager component, which is responsible for creating or deleting the RHBP components that are used to model the achievement of a certain goal.

Subsequently, the architecture presented in Figure 23.9 is a further extended version of Figure 23.8 that incorporates the mission-guided control of multiple drones. In detail, this is achieved by replacing the direct goal instantiation from the *Scenario Manager* with the creation of *DelegationGoals* and replacing the default RHBP Manager class with the further extended *DelegationManager*. Both classes are from the RHBP *Delegation*

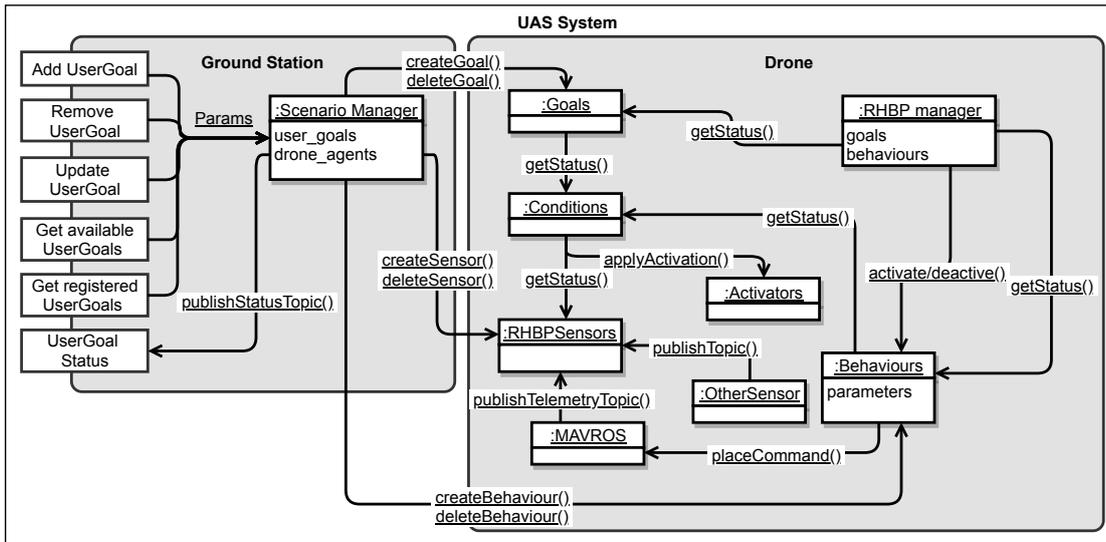


Figure 23.8.: RHPB component relationships and external user goal API in UML communication diagram style without particular communication sequence. Arrows show messages and directions. Plural names of object instances indicate that multiple objects of this type are possible.

Component and enable a distributed decomposition, allocation, and delegation of tasks through an auction-based algorithm, as explained in Chapter 21. Due to the fact that some user goals require special behaviour and sensor instances instantiated dynamically during runtime, we have extended the approach for the mission-guided control of a single drone from above. Likewise, such special instances are created by the *Scenario Manager*, but in contrast to the single drone approach, these sensors and behaviour are created before the decomposition is started and only the drone that gets the task finally delegated retains them while they are removed from all other drones. Moreover, we use the configuration of the *DelegationManager* that always responds to CFPs and run each auction until all robots have replied.

Milestone scenario

A first simple example of the application of RHPB in EffFeu is illustrated in Figure 23.10. The shown behaviour model was used as the mid-project milestone demonstrator as the first proof-of-concept. In the scenario, the mission goal is to explore a selected environment, and once a person is found to circumfly (inspect) the found person to gather

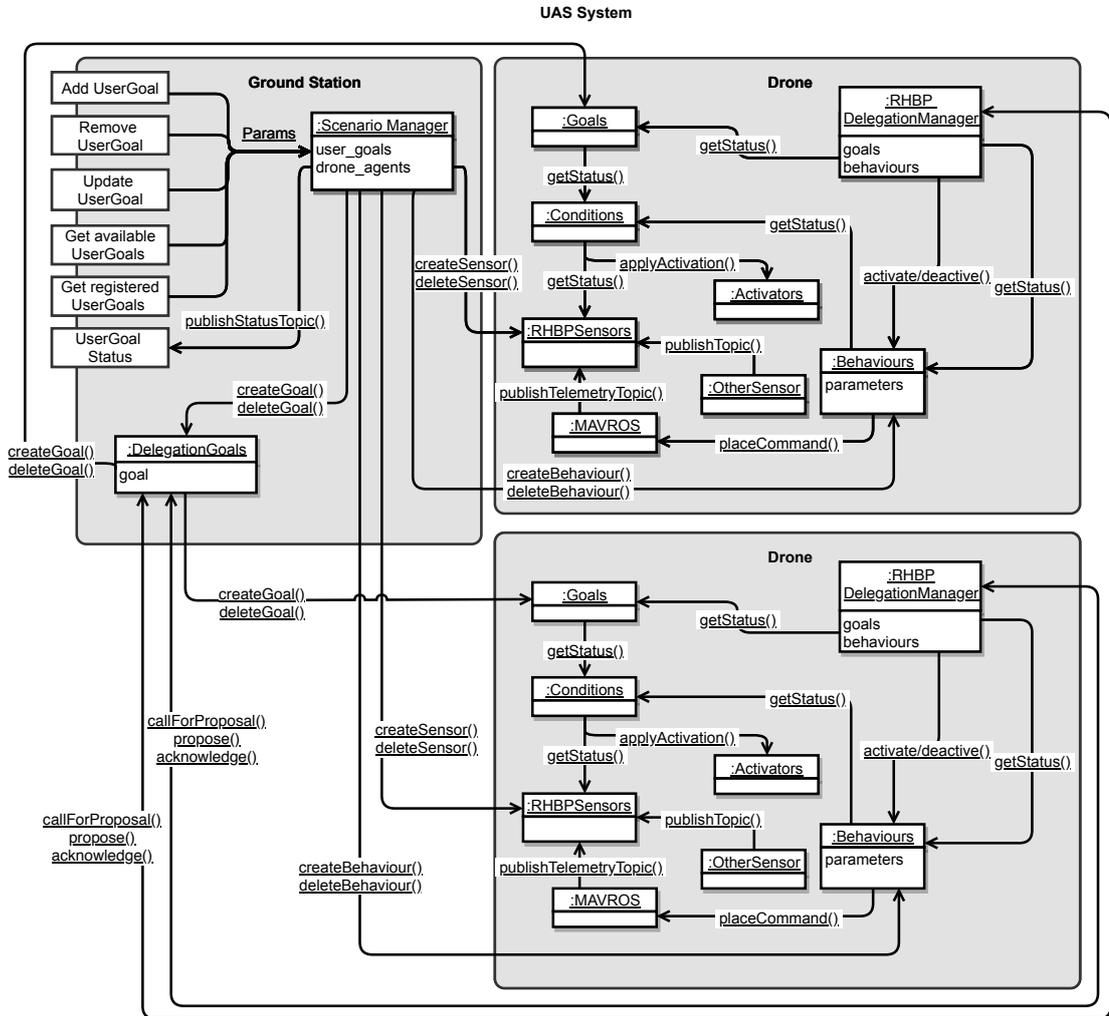


Figure 23.9.: RHBP component relationships and external user goal API in a multi-robot setup. Only two drones are depicted for simplicity. UML communication diagram style without particular communication sequence. Arrows show messages and directions. Plural names of object instances indicate that multiple objects of this type are possible.

more information about the situation. To model this scenario, we use two goals, one for initiating the exploration behaviour, and one for detecting a person. The *recognised_objects_sensor* connects to the results of our object detection, whereas the *coverage_sensor* describes the exploration completion of the selected area percentagewise. Additionally, the implemented behaviour model considers the current battery level of the drone to safely land in case of an empty battery or prevent mission launches without sufficient battery.

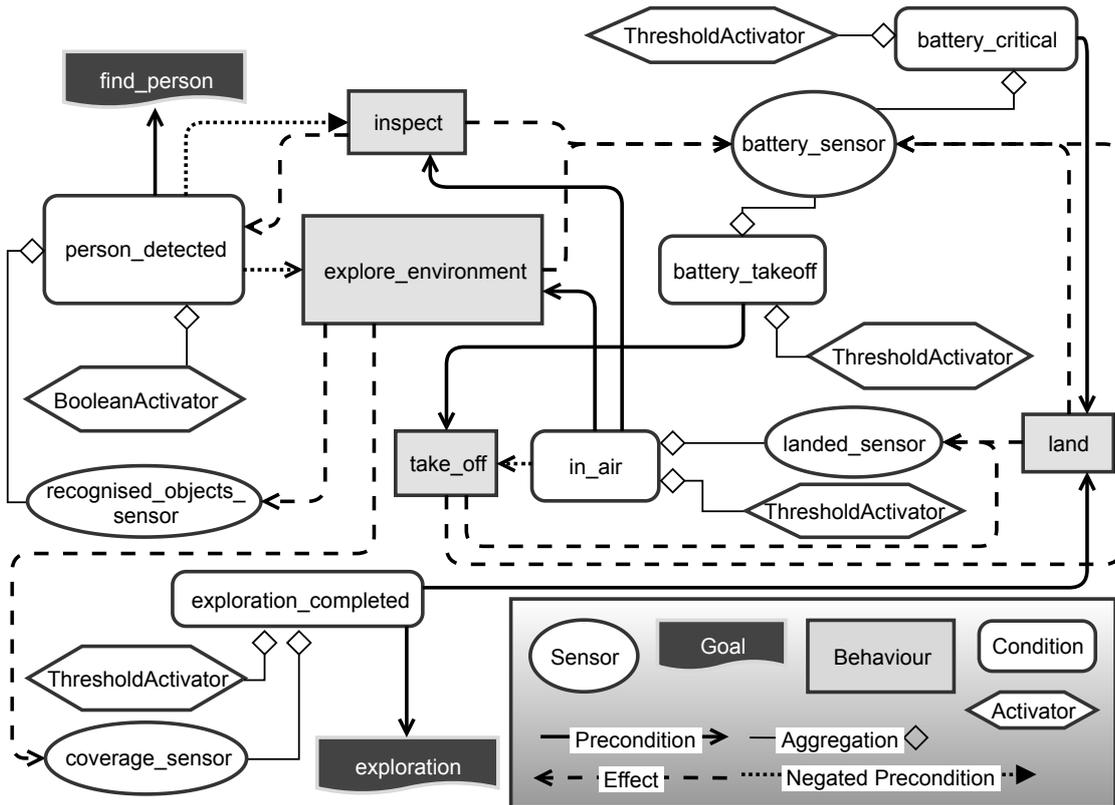


Figure 23.10.: EffFeu project mid-project milestone scenario behaviour network model.

Undoubtedly, the given scenario is comparable simple and it would not require a complex planning and decision-making system. Nevertheless, the implemented model performed well in a qualitative evaluation during the milestone demonstration, where the drone showed the intended behaviour after the corresponding user goal was created. Creating the goal was both demonstrated through the *Mission-Control-UI* and the *Safety and Security Management System* of our partners. The intention of this example is to

23. Projects

illustrate the application and integration options with a simple scenario. A further developed system being equipped with more different behaviours and other goal sets is described and evaluated in the following subsection.

Dynamic scenario

The milestone scenario explained in the last section is statically initialised and executed without showing the benefits of the RHBP approach in a dynamic environment. Notably, different behaviours that can be used to achieve the same result but being different in their particular characteristics, such as specific preconditions and effects, as well as changing the mission goal during runtime make the application of a decision-making and planning component meaningful because it allows to efficiently chose the most suitable behaviour depending on the current situation. In the following, we will explore such more complex and dynamic application scenario that has been realised after the project milestone.

The complex scenario comprises some changes with respect to the milestone scenario. First, the main task of the scenario is modified in the way that now finding a person and inspecting the environment as well as exploring the given area for the purpose of creating a map have the same priority. This is in contrast to the milestone scenario wherein the exploration is only necessary to find the person.

Secondly, additional possible behaviours have been integrated, namely *arm* for arming the drone automatically, *idle* to model an option of doing nothing on the ground, *hover* to wait in the air, and two alternative exploration behaviours. The exploration behaviours differ in the exploration speed and the quality of exploration. The slow exploration is traversing the area of interest in parallel lines with a smaller distance and higher overlap in comparison to the fast exploration. This difference results in a longer flight trajectory for the slow exploration but a higher probability of finding objects with the object recognition. The *remember_position* is an example of a behaviour that is only affecting internal system states by storing the position of the person after it has been found.

The new behaviour network model is illustrated in Figure 23.11. The diagram also shows that we added additional goals to improve the overall mission execution. Particularly, the *exploration goal* is fulfilled by completing the exploration of the area of

interest. This can be potentially achieved by running both exploration behaviours; however, they have been modelled with different effect intensities to represent the above-described differences, in detail, *exploration_slow* has a greater influence on the *recognised_objects_sensor* and a smaller influence on the progress of the *mission_coverage_sensor*. The *inspect_object* goal is achievable by inspecting the area around the found object.

The additional *battery* goal is a maintenance goal that tries to minimise the usage of the battery. Here, the so-called *GreedyActivator* of RHBP is used to model a condition that is never satisfied and always aiming for a maximisation of the particular sensor.

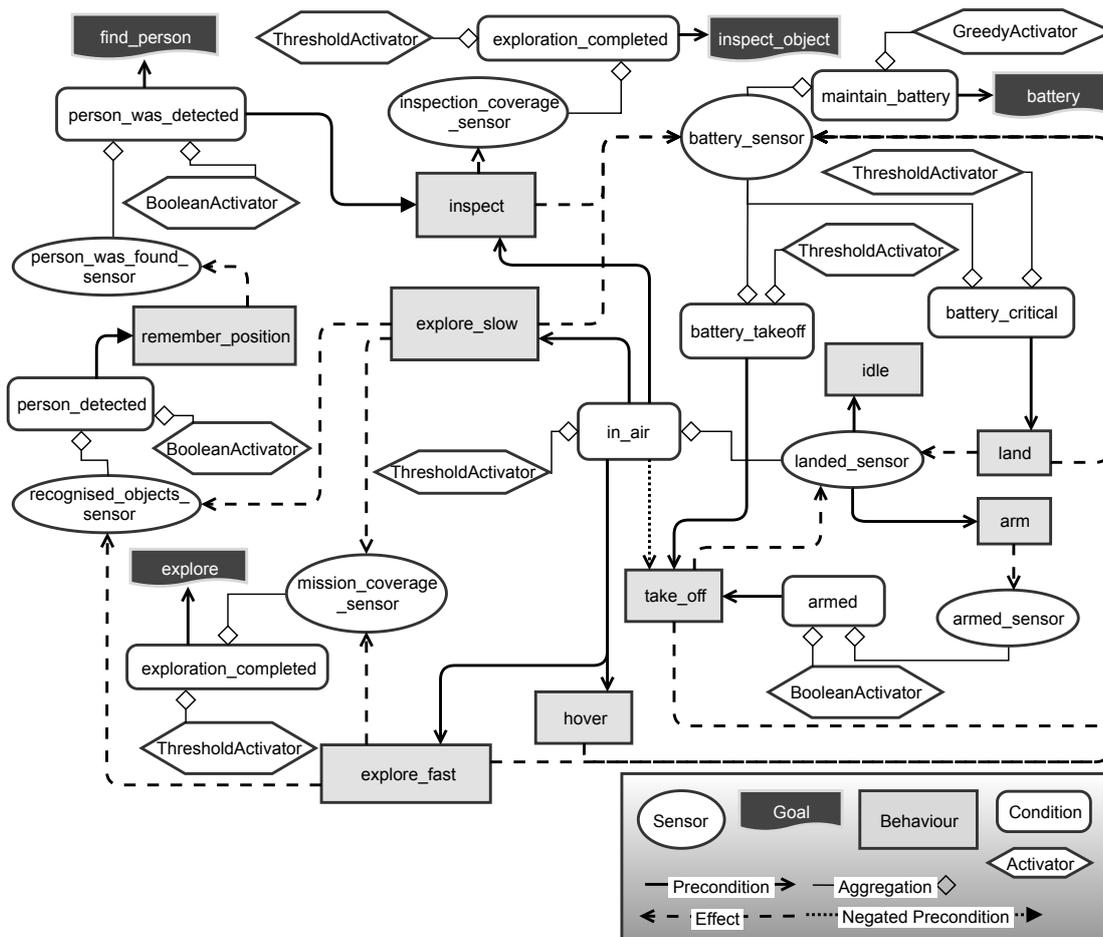


Figure 23.11.: More complex behaviour network model including more and alternative behaviours for similar effects and additional goals.

In order to evaluate the complex behaviour model, we are using a simulation environment that we implemented with the generic Morse simulator (Echeverria et al., 2011).

23. Projects

The simulation environment has been extended to mimic the same ROS API as we are using on the real drone for external control. The original API in use is a combination of the MAVLink protocol of the PX4 firmware and the MAVROS package for the automated creation of ROS bindings. This approach allows us to easily test our implementations before they can be seamlessly transferred to the actual hardware. This works exceptionally well for the mission-guided control because here we are mostly interested in the taken high-level decision and not in the detailed motion of the drone, which is not simulated precisely. Using a simulator for evaluating the higher-level behaviour has the advantage of always having the same conditions, which simplifies the reproduction of experimental results.

A difference to former applications of the RHBP framework is that in the EffFeu project goals are dynamically created, enabled, and disabled during runtime through the above-described external user interface. Aside from feasibility experiments in (Hrabia, Wypler, and Albayrak, 2017) and (Hrabia, Kaiser, and Albayrak, 2017) RHBP has been so far used only with a single drone in a static mission of the SpaceBot Cup (Hrabia et al., 2017), as well as with static mission goals in simulated multi-robot scenarios of the Multi-Agent Programming Contest (Hrabia et al., 2018b; Krakowczyk et al., 2018).

In the following experiments, we wanted to see if our framework is able to properly handle the adaptation to dynamically enabled/disabled goals. Moreover, we investigated to which extent the influence of the symbolic planner in our hybrid approach is supporting the overall mission accomplishment and adaptation in such scenarios. Particularly, we analysed the influence on the efficiency, how fast is the mission accomplished, as well as the adaptation capabilities, how fast is the system reacting to environmental and mission changes.

In order to examine the influence of dynamically created, enabled, and disabled goals, we have scripted a scenario within the simulation environment for the above-described behaviour model. The scenario starts with the goals *find_person*, *explore*, and *battery* enabled. After 18 decision-making steps the *find_person* and *explore* goal are disabled and enabled again after step 24. An additional *inspect_object* goal is then created after step 30. In this scenario, all goals except for the *battery* goal are achievement goals that are removed after they are completed. Moreover, environmental changes, respectively perception changes are simulated through the suddenly detected target object. Decision-

making and planning is executed with one step per second.

For the investigation of the influence of the symbolic planner in such dynamic scenarios, we repeated the experiment with different weights for the influence of the symbolic planner, which have to be in a range of 0 to 1. Furthermore, we limited RHBP to enable only one behaviour at a time because in the given setup behaviours would heavily conflict with each other by sending different low-level control commands at the same time. Other activation weights: activation by situation, predecessors, successors, and conflictors (see Section 18.1 for details) that are used in the RHBP decision-making are set to a medium influence of 0.5 except for the activation by goals that we empirically configured to 0.7 to foster goal pursuance in all experiments. We have fixed the weights except the planner weight to empirically determined values because we are especially interested in the influence of the planner as a major component of our hybrid approach. All other weights are only relevant for the decision-making of the behaviour network layer itself, and the fine-tuning of its decision-making, which influences the stability but not the overall adaptation means.

The diagrams in Figure 23.12 visualise a selection of the most characteristic experiment runs with logged decisions and corresponding goals. The drawn bars describe which behaviour is selected for execution as well as which goals are enabled in a particular decision-making step. We decided to show the run with very little symbolic planning influence (weight=0.1) instead of the run without influence. The reason is that both result sequences are very similar, but with weight=0.0 the complete execution takes just longer (230 decision steps), which makes it more difficult to depict the diagram in comparison to the other results. Decision steps without a decision taken are possible if the current sensor's respective drone state together with precondition setup does not allow for any behaviour to be activated. This is sometimes occurring due to some not in the behaviour model specified drone states, which can occur when the drone is in the transition between being in-air and landed.

In general, the results show that in all configurations, the given mission is completed after some time independent of the planner influence. Furthermore, RHBP is able to handle dynamically created or enabled and disabled goals during runtime, see (a) and (b). The goal pursuance of the system without (much) influence from the planner (c) is not sufficient, the system is having problems to handle the conflicting goals of maintaining

23. Projects

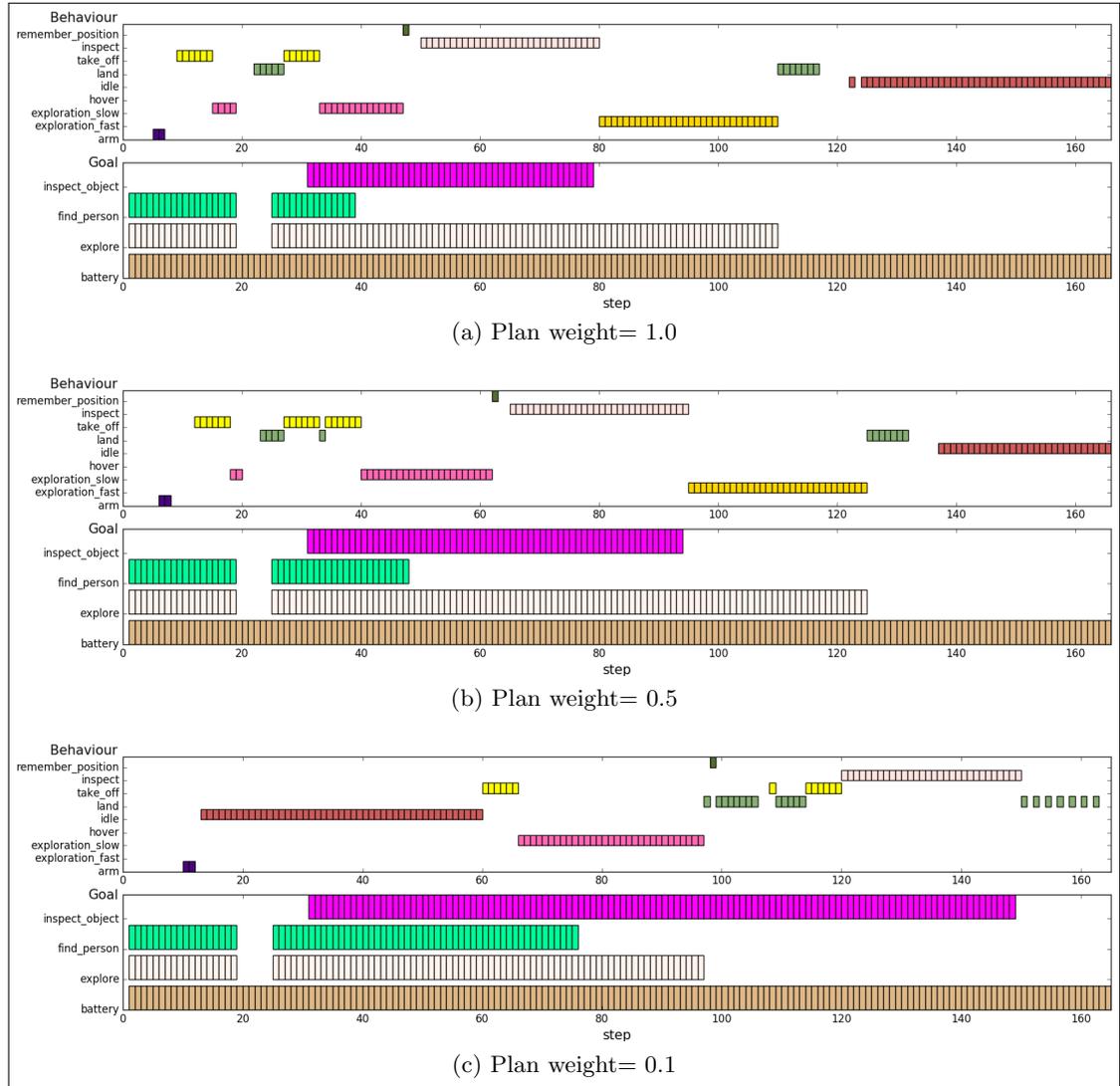


Figure 23.12.: Selected behaviours and enabled goals over time ($\frac{\text{step}}{s}$). Comparison between different characteristic symbolic planning influences specified by weights. A larger weight corresponds to a larger influence.

the battery and completing all other tasks, which require battery capacity. This becomes visible already in (b) where the system suddenly decides for landing one time after step 30. In (c) this is more obvious because without the additional *inspect_goal*, the overall activation of the battery consuming goals is not high enough to start the overall mission. Alternative exploration behaviours are correctly selected in (a) and (b) depending on the currently available boundary conditions. First, the *exploration_slow* is selected until the person is found. After the person is found, the *exploration_fast* is favoured to fasten the exploration of the remaining area. Adaptation capabilities in the sufficiently completed missions of (a) and (b) are indicated by the number of required steps after the goals are activated, deactivated, completed, and time to consider the detected person. Both configurations (a) and (b) show a very similar performance. The adaptation time to the found person is 2 steps, to reactivated goals (*find_person*, *explore*) 2 steps, and to completed goals (*inspect_object*, *explore*) 1 step for both (a) and (b). Only the response to the temporary deactivation of the goals (*find_person*, *explore*) at the beginning of the mission is slightly different. Here, configuration (a) is adapting after 3 steps, while (b) requires 4 steps.

Greater influence from the planner fosters the efficiency, the entire mission is completed in a shorter time, (a) is faster than (b), and (b) is faster than (c). The adaptation capabilities, once the goals are sufficiently pursued, are not much affected by the influence of the symbolic planner. Nevertheless, reducing the influence of the planner makes decision-making less stable, see unnecessary take-offs and landings in (c). Even though more stable decision-making could also be achieved by tuning the model and weights more carefully, the result would depend on the particular scenario implementation and would have to be revised every time when the implementation is changed, which underlines the results obtained with the initial experiments of Section 22.1.

Results underpin that the hybrid RHBP approach, which applies long-term planning to direct the short term decision-making, is reasonable to foster the goal pursuance of the system as well as the efficiency. Furthermore, the influence of the symbolic planner is especially useful in situations of conflicting and dynamically changing goals, whereas influence on adaptation time is negligible.

Multi-robot scenario

This scenario is further extending the dynamic scenario of the previous section towards multiple drones. In particular, the behaviour model of the dynamic scenario is instantiated two times for two independent drones, each having its own RHBP core and decision-making cycle. In detail, we apply the architecture for multiple robots outlined at the end of Section 23.3.2. This results in user goals not being directly mapped to RHBP goals, behaviours, and sensors that are assigned to the drone. Instead, each registered user goal is instantiated as a *DelegationGoal* first before it is finally delegated to a particular robot. To simplify the integration, we have combined multiple goals of one user goal to one *DelegationGoal* with a compound condition that represents all goals.

The RHBP weight configuration is the same as in the previous experiment with plan weight = 1.0. Moreover, the cost function of the delegation module is using its default configuration with *Task-Limit-Utilisation-Factor*= 1.0, *Workload-Proportional-Factor*= -0.025, *Additional-Workload-Factor*= 1.0, *Additional-Delegation-Factor*= 1.0, *Cooperation-Amount-Factor*= 1.0, and *Contractor-Number-Factor*= 0.1. This default configuration is putting a higher priority on the individual costs required to fulfil the additional goal and less on how much the overall plan will be extended.

The simulation and scenario from the previous single drone experiment are reused in a slightly modified version to focus on the relevant aspects for testing the features of the delegation component. In detail, two areas, instead of only one area, have to be explored by the drones to find one person. Additionally, the *inspect_object* goal is created after the person has been found to enable a consideration of the person's position in the delegation. The underlying scenario script creates two user goals for exploration and finding the person directly at the beginning of the execution in step 0. After 10 exploration steps of UAV1, the person is detected, and an additional inspection user goal is registered. Moreover, we do not make use of the battery goal to focus entirely on user goals that are delegated to one or the other drone, depending on the auction results.

Figure 23.13 shows a diagram representation of the sequence of activated behaviours and enabled goals for both UAVs. Exploration behaviours are duplicated because we have one set per area that was available at least during the decision steps of the auction. We can observe that the two auctions for each exploration and finding the person goal are delegated to one of the drones at the end. Each delegation, including the auction

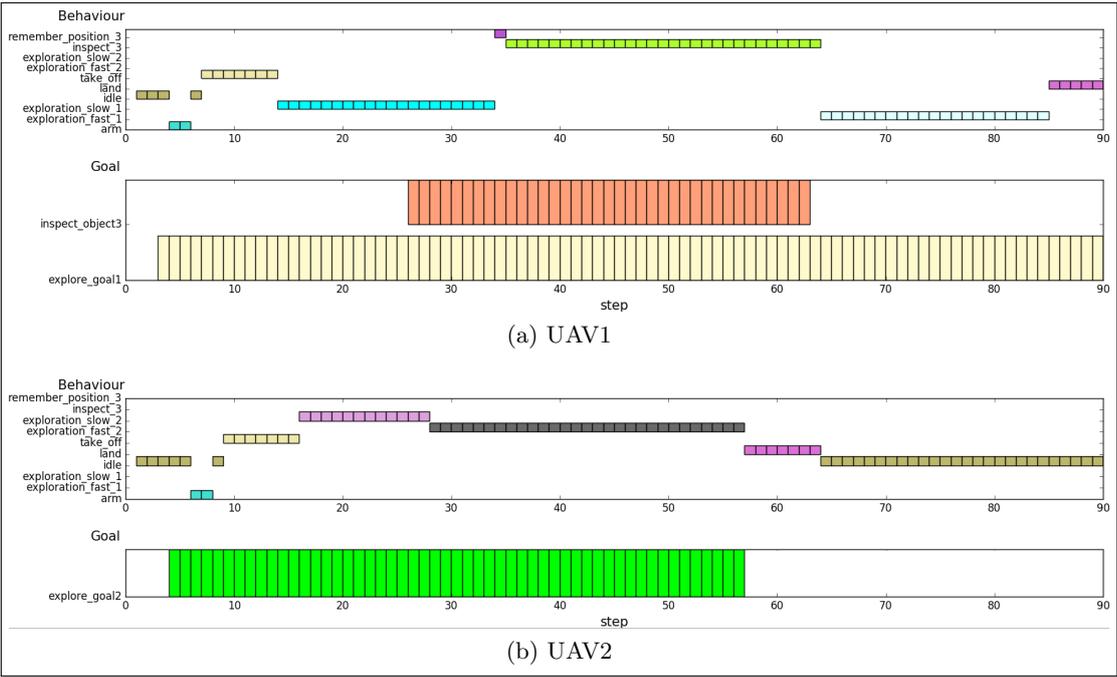


Figure 23.13.: Multi-robot scenario applying task decomposition, allocation, and delegation. Selected behaviours and enabled goals over time ($\frac{\text{step}}{s}$). Diagrams for each UAV individually.

23. Projects

execution takes two decision-making steps, which corresponds to the minimally required additional steps for processing an auction. In consequence, we can also deduce that none of the robustness features of the delegation component has been used such as the selection of an alternative proposal or reiteration of the overall auction in case of failures. In our scenario, the drone with shorter overall trajectory, corresponding to a more concise plan, received one goal each. Moreover, both drones apply, similar to the experiment in the previous section, the slow and more fine-grained exploration first, until the person has been detected. After the person has been detected, it takes again two decision-making cycles to execute the auction and assign the *inspect_object* goal to UAV1. At the same time, UAV2 switches to the faster and less precise exploration behaviour in order to complete the rest of its previously assigned exploration area. The same is done by UAV1 after it has completed the inspection of the area surrounding the found person.

Even though we are not making use of all the decomposition, allocation, and delegation capabilities the RHBP component provides, such as the `DelegationBehaviours` and `DelegableBehaviours`, the experiment illustrates how the component can be applied in practise in a multi-robot RHBP scenario. Particularly, it allows for avoiding static assignments of goals and makes use of explicit coordination within the framework. Furthermore, it was possible to successfully decompose, allocate, and delegate three independent user goals with the default utility function of the delegation component. At the same time, the drones still adapted successfully to the changes in the environment. Nevertheless, the experiment also shows that the decentralised process of decomposition, allocation, and delegation requires additional decision-making steps for the execution of the auctions, which slightly slows down the adaptation process.

23.3.3. EffFeu Project Summary

The scenarios of EffFeu project are interesting for the evaluation of the RHBP framework presented in this work because they are another proof-of-concept on how the framework can be applied in real-life applications and it especially shows how the framework can be integrated into a complex distributed system architecture with end-user interfaces for non-experts. Here, it is peculiarly remarkable that the system has proven its potential to deal with dynamically created, deleted, enabled, and disabled goals. Furthermore, the complex example does also illustrate the capabilities of the general approach in terms of

adaptivity because the system has been able to dynamically select the most appropriate behaviour of a similar type depending on the boundaries defined by the currently valid goals. Moreover, the multi-robot, respectively multi-drone scenario serves as a proof-of-concept for the application of the delegation component within the RHBP environment. Particularly, we could present how the delegation extension can be used to inject external user goals in an explicitly coordinated fashion.

All in all, the EffFeu project developed the foundation to make the integration of UAVs into the safety and security management system easy and easy to use. The efficient use of drones in the daily routine of industrial firefighters and in case of emergency is still challenging. With our system under development, firefighters can already benefit from a mission-guided control and enhanced adjustable autonomy of the system that reduces their cognitive load. They can already assign mission goals with priorities to drones, which are then reassembled and planned as sequences of actions by the drone autonomously. Altogether, our approach enables a transition from motion and trajectory-based drone control towards a goal and mission-oriented control and application in the operation of autonomous drones.

24. Comparison with other Frameworks

RHBP is the first task-level decision-making and planning framework that comprises general coordination means, self-organisation features, and general adaptation capabilities. The analysis of the related work did not lead to an alternative framework that has a comparable feature set. Even more, the hybrid approach of RHBP that combines traditional symbolic planning with reactive behaviour networks is inherently different from all related work that is following one of the combined directions. All in all, this makes a direct comparison very difficult because missing parts would have to be manually developed for alternative approaches. In consequence, the comparison would strongly depend on such custom-made extensions.

Furthermore, the idea behind the development of this thesis is to develop a framework that is striving for increased adaptation capabilities with respect to the given goals. In

24. Comparison with other Frameworks

this account, we do not target the most optimal or most efficient solution. We are rather looking for the most flexible approach that potentially has the highest degree of freedom for adaptation. Nevertheless, measuring such potential for adaptation is difficult because we are missing a quantitative scale, which would be intuitively available for an approach that is more concerned about the efficiency of task fulfilment. In consequence, the so far discussed evaluation scenarios are either presenting a proof-of-concept, which shows that the approach works in general, see Chapter 22, or give concrete examples on how the system has been successfully applied and integrated into research projects as presented in Chapter 23. Nevertheless, the very successful participation in the MAPC shows that RHBP is also able to compete with other multi-agent frameworks in a quantitative evaluation. However, the remainder of this chapter is giving an additional perspective by looking on the possibilities and capabilities of related approaches in comparison to the RHBP in order to provide a classification with respect to the state-of-the-art. Whereby the following Section 24.1 presents a comparison against related symbolic planning frameworks, behaviour network concepts, and hybrid approaches. Subsequently, Section 24.2 focuses solely on a structured comparison against related work from decomposition and delegation due to the fact that none of the approaches, we compared against in the first section, is considering decomposition and delegation explicitly.

24.1. Comparison against symbolic planning frameworks, behaviour networks and hybrid planning approaches

Table 24.1 collects various features of different task-level decision-making and planning approaches that have been discussed throughout this dissertation and compares several of them regarding their feature fulfilment. For details about the particular frameworks, please also see Chapter 11. The table makes visible that only the new RHBP approach is able to combine all the capabilities in one comprehensive framework. Especially the support of multiple robots and their coordination is neglected by most other approaches. Here, RHBP is the only one that enables implicit coordination through its comprehensive integration of self-organisation patterns. Moreover, aside from RHBP, only SkiROS (Rovida et al., 2017) and partially ROSPlan (Cashmore et al., 2015) consider explicitly the coordination of multiple robots at all, but this is limited to a centralised approach

24.2. Comparison against other decomposition and delegation approaches

with one task manager/planner and centralised knowledge sharing. A decentralised consideration of explicit coordination with decomposition, delegation, and allocation as supported by RHBP is missing in general. For this reason, we are also evaluating this part separately in the next Section 24.2 with corresponding related work independent of particular frameworks. Likewise, besides RHBP, only the symbolic planning approaches SkiROS and ROSPlan have features for knowledge sharing. Hence, RHBP is the first behaviour network, respectively hybrid behaviour network, based approach that takes into account knowledge sharing.

Furthermore, only a small amount of solutions support a direct ROS integration, which makes their application and comparison in practice more challenging. Similarly, the integration of learning is rather limited in the two approaches that have considered it. Particularly, the realised approach of Jung, 1998 is able to learn direct correlations of binary effects after the execution of behaviours, but delayed effects or fluent values are not studied. Likewise, reinforcement learning was as well considered on a conceptual level for determining effects by Decugis and Ferber, 1998, but this has never been realised or evaluated.

24.2. Comparison against other decomposition and delegation approaches

Due to the fact that RHBP is the only framework that comprises a decentralised consideration of explicit coordination with decomposition, delegation, and allocation, we separately compare this feature against general related work independent of a specific task-level decision-making and planning framework integration.

In Table 24.2 and Table 24.3, we compare the decomposition, allocation, and delegation features of the solution presented in Chapter 21 against several related approaches. Please refer to Chapter 12 for additional background about the mentioned solutions. Approaches that are not listed in both tables are either missing the decomposition or delegation part.

In Table 24.2 *No a priori definition* means that the solution does not require a (complete) definition of the decompositions in advance. *Generatable during runtime* is realised if decompositions can be generated dynamically during runtime. *Dependencies between*

24. Comparison with other Frameworks

Table 24.1.: Comparison of the presented RHBP (Hrabia 2019) against existing symbolic planning frameworks, behaviour networks and hybrid planning approaches. [Y=Yes, N=No, P=Partially, M=Manually]

		Maes (1989) [MASM]	Decugis and Ferber (1998)	Jung (1998) [ABBA]	Hertzberg et al. (1998)	Allgeuer and Behnke (2013)	Lee and Cho (2014)	Cashmore et al. (2015) [ROSPlan]	Rovida et al. (2017) [SkiROS]	Hrabia 2019 [RHBP]
Behaviour Network	Included	Y	Y	Y	Y	Y	Y	N	N	Y
	Learning	N	P	Y	N	N	N	N	N	Y
	Hierarchies	N	Y	N	N	N	N	N	N	Y
Symbolic Planning	Included	N	N	N	Y	N	Y	Y	Y	Y
	Standard Planner	N	N	N	ADL	N	N	PDDL	PDDL	PDDL
	Generating plan descriptions	N	N	N	N	N	N	P	P	Y
General	ROS Integration	N	N	N	N	Y	N	Y	Y	Y
	Parallel Behaviour Execution	N	N	N	N	Y	N	N	N	Y
	Non-binary Preconditions	N	N	N	Y	Y	N	Y	Y	Y
Coordination	Multi-Robot Support	N	N	M	N	N	N	P	Y	Y
	Explicit	N	N	N	N	N	N	P	Y	Y
	Implicit	N	N	N	N	N	N	N	N	Y
	Knowledge Sharing	N	N	N	N	N	N	Y	Y	Y

24.2. Comparison against other decomposition and delegation approaches

sub-goals refers to the support of condition dependencies between multiple delegated goals. *Multi-agent decomposition* describes if only a single agent or centralised instance is decomposing or if different agents are doing this in a distributed fashion. *Comparable agent decompositions* refers to the feature of an automated selection of the most suitable decomposition according to a certain utility function. *Decentralised decomposition selection* is possible if multiple agents in the system are involved in the decomposition selection in contrast to a single central agent. *Utilising decomposition for allocation* is supported if the determined decomposition is forming the foundation for the task allocation in the next step.

The listed features in Table 24.3 are comparing our delegation capabilities against related work. *Decentralised* is available if the delegation is not coordinated by a central instance. *Agent as auctioneer* means that each agent can become an auctioneer and no external instances are required. *Goal combinations possible* is available if the allocation supports the allocation and delegation of goal sets and does not require delegating each sub-goal individually. *Different decompositions possible* describes a feature that enables to compare and select from different possible delegations. *Considering efficiency for allocation* is available if the approach is trying to find the most efficient allocation of tasks and not just a possible solution. *Decentralised efficiency determination* is given if the agents are calculating the efficiency autonomously in a decentralised way. *Agent solution paths not required* is valid if the approach works without sharing the complete decomposition and allocation with each other. *Direct delegation* refers to the option of directly delegating a task between agents without selecting them explicitly based on a certain algorithm.

All in all, both tables indicate that the realised RHBP solution has the most comprehensive feature set of the compared decompositions approaches. Only RHBP allows for an automatic decomposition without a priori definition of a complete decomposition model. Even though this is only partially possible due to the necessary usage of *DecompositionBehaviours* that work as hooks for potential decompositions and delegations. Moreover, only RHBP and (Doherty et al., 2016) support a decentralised selection of the applied decomposition in case of several alternatives. Furthermore, only RHBP and (Zlot and Stentz, 2005) are able to select the most suitable decomposition according to a utility function.

24. Comparison with other Frameworks

Table 24.2.: Comparing RHBP decomposition features against existing solutions.
[Y=Yes; N=No; P=Partially]

	Simmons and Apfel- baum, 1998 TDL	Lemaire, Alami, and Lacroix, 2004	Doherty et al., 2016	Zlot and Stentz, 2005	RHBP
No a priori definition	N	N	N	N	P
Generatable during runtime	Y	N	Y	Y	Y
Dependencies between sub-goals	Y	Y	Y	Y	Y
Multi-agent decomposi- tion	N	N	Y	Y	Y
Comparable agent de- compositions	N	N	N	Y	Y
Decentralised decomposi- tion selection	N	N	Y	N	Y
Utilising de- composition for allocation	N	Y	Y	Y	Y

24.2. Comparison against other decomposition and delegation approaches

Table 24.3.: Comparison of the RHBP delegation features against existing solutions.
[Y=Yes; N=No; P=Partially]

	Mills-Tettey, Stentz, and Dias, 2007 Hungarian algorithm	Rassenti, Smith, and Bulfin, 1982 Combinatorial Auction	Botelho and Alami, 1999 M+	Doherty et al., 2016	Zlot and Stentz, 2005	RHBP
Decentralised	N	N	Y	Y	Y	Y
Agent as auctioneer	N	Y	N	N	Y	Y
Goal combinations possible	N	Y	N	N	Y	N
Different decompositions possible	N	N	N	Y	Y	Y
Considering efficiency for allocation	Y	Y	Y	N	Y	Y
Decentralised efficiency determination	N	N	Y	N	N	Y
Agent solution paths not required	Y	Y	Y	Y	N	Y
Direct delegation	N	N	Y	Y	N	P

24. Comparison with other Frameworks

The realised RHBP extension supports most delegation features, except for the allocation and delegation of goal combinations. Here, RHBP requires to first allocate all goals to an agent, which then could further delegate sub-goals again. However, solutions that support goal combinations are less decentralised, which was a core requirement for our approach to improve the robustness. Moreover, *Direct delegation* is only possible in RHBP by making use of manually modelled NetworkBehaviours, which result in explicit direct delegations. This minor disadvantage is not considered problematic as long as it is possible to decompose and delegate sub-goals in general.

Part VI.

Summary and Discussion

The remaining chapters of this dissertation are a retrospection on the results and achievements as well as an outlook into the next possible research directions. In Chapter 25, we highlight and summarise the important aspects of the realised approach. Next, in Chapter 26, we review and answer the proposed research questions, objectives, and requirements. This is continued with a discussion of experienced limitations of the realised approach in Chapter 27. Finally, Chapter 28 outlines some possible research directions that would further broaden the applicability of the results of this dissertation.

25. Conclusion

In this work, we presented and developed the RHBP framework¹ that addresses the needs of the robotics community and especially ROS community for a goal-driven adaptive decision-making and planning package for task-level behaviour control and coordination of multi-robot systems.

The concept for the development of the framework is based on findings of the initial analysis of existing approaches and related work from the different affected research areas. Notably, we examined individual decision-making and planning for single robot systems, centralised explicit coordination, decision-making and planning of general-purpose multi-robot systems and self-organised coordination of many simple robots. In detail, the analysis indicates that the existing work in the research directions is missing a comprehensive concept that enables to combine the different advantages in one consistent manner. In particular, the specific existing solutions have shown various limitations: They are either not goal-oriented, too abstract and are missing relevant support for an implementation in practice, or are too application-specific, which complicates reuse and portability, or limited in their expressiveness, due to strict formalisations.

The developed abstract component-oriented architecture of the framework is built around a core component for task-level decision-making and planning. This RHBP core combines the advantages of a reactive and adaptive behaviour-based decision-making with a symbolic planner that enables appropriate goal pursuance. In particular, we ex-

¹Source code available online: <https://github.com/ros-hybrid-behaviour-planner>

tended and combined existing concepts of reactive and hybrid behaviour-based decision-making in a ROS-integrated solution. The developed solution automatically generates the required plan descriptions (domain and problem) for the PDDL-based planner and supports multiple robots alongside all features that have only been available in single behaviour-network, symbolic planning, and hybrid planning approaches but have never been combined.

Furthermore, the general underlying concept of hybrid behaviour networks is extended with capabilities that allow for the creation of hierarchical abstraction layers for the decision-making through the encapsulation of additional decision-making and planning instances within one behaviour. Additionally, accompanying knowledge base components simplify the exchange and persistence of knowledge amongst the agents in the system, which also enables the realisation of MAPE-K feedback architectures.

Aside from the available encapsulation of complete behaviour models within a behaviour, which enables structuring and decomposition based on an a priori defined behaviour model, the task decomposition and delegation extension fosters a more decoupled and decentralised solution. This extension supports the dynamic and decentralised creation of sub-goals for individual robots in a goal-oriented top-down fashion. The combination of this extension for explicit coordination together with implicit coordination though in a single robotics framework is inherently unique and provides maximum flexibility and adaptation capabilities to the particular application scenario.

Moreover, the RHBP framework contains a general-purpose self-organisation library for the multi-robot domain. The self-organisation library (`so_data`) provides a wide range of mechanisms and allows for additional extensions due to its modular and generic architecture. The further introduced integration of the self-organisation library into the RHBP core closes a gap between the research fields of decision-making, planning, and self-organisation. It enables the development of autonomous robots resolving tasks with complex dependencies while allowing for self-organised coordination and problem resolving within one framework. Both worlds can directly interact with each other by making use of the same domain model, information sources and abstract system capabilities. The integration into the popular ROS framework does also support a fast adoption and integration into existing solutions and software ecosystems.

Furthermore, we have extended the self-organisation library with the concept and

implementation of SO Coordinator components that help to abstract the actual self-organisation mechanisms from the intentions of a system designer. Moreover, this approach provides the necessary infrastructure to collect abstract self-organisation mechanisms and their configuration in a common database. While the current determination of the mechanism scores is relying on expert experience, it already includes infrastructure to enable more sophisticated approaches in the near future, like online and offline machine learning or applying evolutionary strategies.

In addition, the incorporation of reinforcement learning further enhances the self-adaptation capabilities of the RHBP core component by automatically learning beneficial behaviours for different goal-sets independent of the behaviour model and its a priori defined relationships.

The realised RHBP framework is also tested in various experiments, which underlines that the approach works in general as intended. Aside from such proof-of-concepts, the evaluation in several research projects highlights that RHBP is applicable in complex robot and multi-agent use-cases in practice.

Especially, achieving the shared top spot in the international MAPC 2018 shows that RHBP is not only fostering the adaptation capabilities, it is also competitive for the implementation of application scenarios that are more concerned about clear quantitative measures.

Last but not least, the realised framework has been extensively applied in teaching for the realisation of different implementations related to the MAPC in several project courses of the TUB. Here, RHBP has been used by all in all 54 students in the courses Application System Project and Applied Artificial Intelligence Project from summer term 2017 until the summer term 2019, which again underlines the maturity of the approach.

26. Revisiting Research Questions, Objectives, and Requirements

The goal of this thesis is to close the discovered research gap between the different research directions of individual decision-making and planning for single systems, centralised explicit coordination, decision-making and planning of general-purpose multi-robot systems as well as self-organised coordination of many simple robots.

As a guideline of the research conducted in this dissertation, following research questions have been proposed in Chapter 3.

Research Questions:

Q1 How does the integration of self-adaptive decision-making and task-level planning allow robotic systems to cope with dynamic environments?

Q2 In what way can the combination of task-level decision-making and planning with self-organisation improve the robustness of coordination for multi-robot systems?

Next, we get back to each of the research questions and discuss how far they can be answered.

Q1: The successful application of the RHBP framework in the EffFeu project demonstrates that self-adaptive decision-making and planning is facilitating the operation of robots in dynamic environments. Here, especially the declarative creation of a behaviour model that avoids static state transitions and the continuously repeated decision-making and planning cycle enables the system to automatically respond to uncertainty and dynamic changes in the environment. Moreover, the shown facilitation of adaptation capabilities is twofold. First, the system is able to dynamically adjust to changed environmental conditions by selecting the most appropriate behaviours from several alternatives depending on the currently enabled goals, which even works if no direct path to the goals is known to the planner due to the opportunistic property of the behaviour network. Secondly, the demonstrations substantiate how the system is coping appropriately with dynamically added, removed, and updated goals throughout a mission. This

26. Revisiting Research Questions, Objectives, and Requirements

also underlines the reusability because an existing behaviour model for a robot system can just be applied to a new application scenario by formulating new goals. Undoubtedly, this has limitations because the behaviour model needs to contain behaviours that allow for approaching the given goals in general, but the particular way on how the behaviours are combined is decided dynamically and does not have to be modelled in advance. Moreover, the RL extension is also fostering long-term adaptation to unforeseen changes in the model as well as in the environment but it requires a decent amount of learning episodes, as shown in the experiments in Section 22.4.

Q2: The successful participation in the MAPC, especially in the year 2018 with the shared top spot in the competition has proven that a combination of self-organisation, explicit coordination, and individual task-level decision-making and planning is beneficial, especially if a complex problem has to be addressed with multiple robots in a coordinated fashion. In this use-case, the application of self-organisation mechanisms enables a robust, distributed, and implicitly coordinated exploration of the environment, while the team of simulated autonomous robots uses explicit coordination in parallel to handle mutually dependent tasks for assembly and delivery of items. Furthermore, this is combined with individual decision-making and planning to dynamically handle failures, deal with uncertainty, manage resources and plan the trajectories. All these aspects are coherently expressed in one behaviour model containing several abstraction levels of decisions. In this particular use-case, the taken approach has also proven with the last-minute changed well-building strategy that the strict separation of concerns induced by splitting robot capabilities into several behaviour networks, behaviours, conditions, and goals fosters a fast adaptation from the development point of view. The experience of the robust and successful combination of implicit and decentralised, explicit coordination within the MAPC led to the implementation of the specific delegation component of the RHBP framework afterwards. This component covers decomposition, allocation, and delegation of tasks. Whereby the delegation component is a further developed generalisation of the scenario-specific approach used in the MAPC. Here, we consequently considered decentralisation and robustness in the concept. This component could also show its functionality for realising the coordination of externally provided user goals in the EffFeu project.

Furthermore, the formulated research questions led to a number of objectives and requirements that provided the foundation for the further developed approach. The objectives and requirements repeated below are discussed in detail in Chapter 15.

Objectives:

- O1** Developing a new reusable task-level decision-making and planning framework for mobile robots in dynamic and uncertain environments.
- O2** Supporting self-adaptation through algorithms and conceptual integration.
- O3** Supporting self-organisation with an automatic selection of suitable mechanisms.
- O4** Enabling the realisation of multi-robot applications including means for task partitioning and distribution.
- O5** Providing capabilities for describing the system goals and boundaries wherein the system is supposed to operate autonomously and adaptively.
- O6** Aiming for coherent, integrated design and implementation of adaptation capabilities within general decision-making and planning.

Requirements:

- R1** Integrate various behaviour network concepts into one extended solution.
- R2** Extending the hybrid behaviour network concept with more advanced learning capabilities.
- R3** Allowing for multiple hierarchy, respectively abstraction levels in the decision-making.
- R4** Integrating implicit coordination through self-organisation in a goal-oriented manner.
- R5** Enabling an automated selection of self-organisation mechanism based on expert knowledge.
- R6** Explicitly addressing automated task decomposition, allocation and delegation.
- R7** Using the standardised planning interface PDDL.

R8 Implementing one coherent object-oriented domain model.

R9 Applying the ROS middleware and Python programming language for the realisation.

These objectives and requirements are achieved with the realisation of the modular framework RHBP that is seamlessly integrated into the popular robotics ecosystem ROS (*R9*). The implemented framework is supporting reuse through its component-driven architecture and it can independently be used for decentralised single as well as multi-agent or multi-robot systems (*O1* and *R3*). In particular, the approach enables in practice a coherent and integrated design and implementation of the decision-making and planning as well as coordination application logic within one software ecosystem that features a common domain model (*O6* and *R8*). This becomes visible through the reuse of the RHBP core component for various aspects of the framework, such as supporting the goal-oriented usage of self-organisation mechanisms as well as using the same components for the particular implementation.

The presented core implementation of RHBP, consisting of a hybrid reactive decision-making and a PDDL-based symbolic planning layer (*R1* and *R7*), enables a declarative, flexible, and capability-oriented implementation of robot systems, while the coherently integrated extensions for implicit coordination based on self-organisation (*O3*, *R4*, and *R5*) fosters the adaptation capabilities for multi-robot systems. Likewise, the non-mandatory integration of reinforcement learning (*O2* and *R2*) allows to further increase the adaptation capabilities of single robot systems in many situations. Furthermore, the extension for a simplified and automated explicit task decomposition and delegation (*O4* and *R6*) gives more control for the multi-robot coordination if this is required, e.g. for complex tasks with mutual dependencies and the necessity of decomposition. Even more, as all features can be seamlessly combined in one behaviour model, which describes the multi-robot system with its capabilities and goals, it is possible to tailor the autonomy in an application-specific manner. In this spirit, the declaratively modelled preconditions guarantee the outer boundary of the autonomy (*O5*).

The answered questions and achieved objectives underline that the problem addressed in this thesis has been successfully approached. Nevertheless, there is always room for improvement; hence, the next chapter discusses some limitations as well as possible solutions.

27. Discussion

Despite the fact that the developed RHBP framework can be applied for general-purpose task-level decision-making and planning in a modular and reusable fashion, the evaluation in practice has also revealed some limitations. The encountered limitations are examined in the remainder of this chapter.

First, the implemented RHBP framework does not have an explicit notion of time. The frequency of the main decision-making feedback loop can be used as a rough time reference but behaviour executions are not limited to the duration of a step, and the system developer would have to manually ensure that the effects of behaviours correspond with the average execution time per step. In consequence, if the consideration of time is crucial for the decision-making, the system developer is responsible for creating a consistent behaviour model that reflects a time reference on the level of decision steps.

Secondly, while the application of RHBP is certainly improving the adaptation capabilities of a multi-robot system, developing a behaviour model requires careful empirical tests to make sure that the system works as intended. Here, it is especially useful to evaluate an implementation first in a simulated environment. Although an extensive usage of preconditions for all behaviours allows to create certain guarantees about the expectable runtime behaviour, this is also automatically limiting the degree of freedom for future adaptations. Alternative approaches that are able to rely on model checking have a clear advantage in this respect but are lacking the necessary degree of freedom for adaptation on the other hand. A further in-depth study on how aspects of model checking could be incorporated into the presented behaviour-network-based approach is an exciting direction for future research.

Thirdly, it is imaginable that several robots in a multi-robot system adapt at the same time to changed environmental conditions or that the individual adaptations influence each other. In the worst case, the entire system might reach an oscillating state. Avoiding such situations requires a coordination of the individual adaptations amongst the agents. A possible solution is presented in (Weißbach et al., 2017), here the authors

27. Discussion

developed a transaction protocol to ensure a consistent adaptation even in the presence of communication problems or failing agents. The approach of Weißbach et al. (2017) could also be adapted and integrated into the presented RHBP framework in the future.

Next, RHBP is making use of a standardised planning interface, although the taken approach of reusing existing PDDL-based planners through a PDDL-interface enables exchangeability of the symbolic planning component, it is also limiting possible extensions and avoids a more enclosing integration. This is especially restricting the monitoring capabilities. Here, it could be interesting to explore opportunities of a planner that is specifically developed for and integrated into RHBP. Such kind of specialised planner could allow for additional features like plan-reuse, online plan altering, and the identification of missing behaviours for a further extended automated task decomposition. Plan-reuse and online plan altering means that the planner avoids to replan and search through the entire state space every time again from scratch to improve the performance over continuous decision-making steps. Instead, it tries to identify intelligently which sub-branches of the created search graph have been already analysed in the past and can be reused or modified in the current situation. The usefulness of an automated identification of missing behaviours respectively edges in the planner search space has been already discussed in Chapter 21. Conducting research in this challenging direction would allow to further improve the automated decomposition capabilities of the system in the future.

As far as known, the RHBP framework has been the first approach that combines decision-making and planning as well as coordination through self-organisation in a coherent way. Nevertheless, it marks a starting point that can be further improved. A possible improvement could be a more sophisticated selection of self-organisation patterns in the coordination mechanism selector component. Here, it seems promising to have an in-depth investigation on possible applications of machine learning as an extension of the current expert knowledge database approach.

Furthermore, the application in practice within projects involving an end-user such as EffFeu yields to the insight that a backtracking and recoding of why certain decision have been taken are sometimes desired by users to understand why the system is behaving as it is. Realising such insights within the RHBP seems feasible in general but it would require additional considerations on how the recorded information can be post-processed

in a human understandable fashion. These considerations are in line with the current trend towards research in the direction of human-understandable and explainable AI in general (Hagras, 2018; Wachter, Mittelstadt, and Floridi, 2017).

Finally, the currently realised reinforcement learning capabilities of RHBP show a lot of potentials but it could also be further improved in the future. In this case, it would be especially useful to reduce the number of necessary learning episodes until the solution converges, which is also a general research challenge in the robotics RL domain (Kober, Bagnell, and Peters, 2013). Moreover, the current RL implementation is not explicitly addressing learning in a cooperative multi-agent setup. Thus, a consideration of Multi-Agent Reinforcement Learning (MARL) would allow to take advantage of distributed learning in the future.

28. Future Work

The presented framework RHBP for adaptive decision-making and planning in the multi-robot domain was developed with a focus on modular expandability. For this reason, there are many connecting factors that simplify future extensions. Some possible directions for future research have already been identified in the previous chapter, which discussed the current limitations of the presented approach. The ideas presented in the remainder of this chapter are not driven by current shortcomings. Instead, they describe possibilities that would allow to further evaluate and broaden the applicability of the RHBP approach.

First, intuitive extensions are the development of additional application-specific sensors, behaviours, and activators. Such reusable extensions could potentially foster the dissemination because they would simplify the application in other use cases by other researchers. Additionally, the implementation could be optimised more comprehensively in terms of runtime performance, e.g. increasing concurrent executions.

In the previous chapter, we already discussed that developing a behaviour model with the framework requires careful empirical testing but also the design and implementation phase could be assisted by additional tool support to improve the usability. Ideas that

28. *Future Work*

could foster such software engineering aspects could be, for instance, an automated generation of (dynamic) behaviour diagrams from code or runtime environment. Likewise, the automated generation of code stubs from a visual editing tool could facilitate the usage, especially for beginners.

The so far targeted multi-robot applications have considered the multi-robot system as a tool that is autonomously operating based on the goals provided by a human. However, more investigation could be done towards scenarios that are relying on a closer interaction between humans and robots. Possible applications areas could be robots as human co-workers in automation or large heterogeneous swarm system consisting of humans as well as robots, e.g. in search and rescue operations.

Finally, all the different tested application and evaluation scenarios led to additional insights, useful improvements, and new ideas for supplementary features. For this reason, it is certainly encouraged to further apply RHBP in new application scenarios to broaden the applicability of the solution. Moreover, it would be especially interesting to analyse the results of a permanently deployed RHBP-based industrial multi-robot solution, which operates outside of the extent of research projects with an only limited scope of demonstrators.

Bibliography

- Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. (2016). ‘Tensorflow: a system for large-scale machine learning.’ In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Vol. 16, pp. 265–283.
- Ahlbrecht, Tobias, Jürgen Dix, and Niklas Fiekas (Oct. 2018). ‘Multi-agent programming contest 2017’. In: *Annals of Mathematics and Artificial Intelligence* 84.1, pp. 1–16.
- Alami, R., R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand (Apr. 1998). ‘An Architecture for Autonomy , An Architecture for Autonomy’. In: *The International Journal of Robotics Research* 17.4, pp. 315–337.
- Albayrak, S. and H. Krallmann (1992). ‘Verteilte kooperierende wissensbasierte Systeme in der Fertigungssteuerung’. In: *Information als Produktionsfaktor*. Ed. by Winfried Görke, Hermann Rininsland, and Max Syrbe. Springer Berlin Heidelberg, pp. 564–576.
- Ali, Syed Mustafa, Robert M Zimmer, and CM Elstob (July 1998). ‘The question concerning emergence: Implications for artificiality’. In: *AIP Conference Proceedings* 437.1, pp. 138–156.
- Allgeuer, Philipp and Sven Behnke (2013). ‘Hierarchical and state-based architectures for robot behavior planning and control’. In: *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conf. on Humanoid Robots, Atlanta, USA*.
- Arkin, Ronald C. (1998). *Behavior-based robotics*. MIT press.
- Ashley-Rollman, Michael P., Seth Copen Goldstein, Peter Lee, Todd C. Mowry, and Padmanabhan Pillai (Oct. 2007). ‘Meld: A declarative approach to programming ensembles’. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 2794–2800.
- Ay, Nihat, Ralf Der, and Mikhail Prokopenko (Sept. 2012). ‘Guided self-organization: perception–action loops of embodied systems’. In: *Theory in Biosciences* 131.3, pp. 125–127.
- Bachrach, Jonathan, James McLurkin, and Anthony Grue (2008). ‘Protoswarm: A Language for Programming Multi-robot Systems Using the Amorphous Medium Abstraction’. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3. AAMAS '08*. Estoril, Portugal: International Foundation for Autonomous Agents and Multiagent Systems, pp. 1175–1178.
- Balch, Tucker and Ronald C. Arkin (1998). ‘Behavior-based formation control for multi-robot teams’. In: *IEEE Transactions on Robotics and Automation* 14.6, pp. 926–939.
- Balch, Tucker and Maria Hybinette (2000). ‘Social potentials for scalable multi-robot formations’. In: *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*. Vol. 1, pp. 73–80.

BIBLIOGRAPHY

- Barnes, Laura E., Mary Anne Fields, and Kimon P. Valavanis (Dec. 2009). ‘Swarm Formation Control Utilizing Elliptical Surfaces and Limiting Functions’. In: *Trans. Sys. Man Cyber. Part B* 39.6, pp. 1434–1445.
- Bashyal, Shishir and Ganesh Kumar Venayagamoorthy (2008). ‘Human swarm interaction for radiation source search and localization’. In: *IEEE Swarm Intelligence Symposium, 2008. SIS 2008*, pp. 1–8.
- Beal, Jacob and Jonathan Bachrach (2006). ‘Infrastructure for engineered emergence on sensor/actuator networks’. In: *IEEE Intelligent Systems* 21.2, pp. 10–19.
- Bechon, Patrick and Jean-Jacques Slotine (2012). ‘Synchronization and quorum sensing in a swarm of humanoid robots’. In: *arXiv preprint arXiv:1205.2952*.
- Behera, Lalit Kumar and Aparna Sasidharan (2011). ‘Ant Colony Optimization for Cooperation in Robotic Swarms’. In: *Advances in Applied Science Research* 2.3, pp. 476–482.
- Behrens, Tristan, Mehdi Dastani, Jürgen Dix, Michael Köster, and Peter Novák (Aug. 2010). ‘The multi-agent programming contest from 2005–2010’. In: *Annals of Mathematics and Artificial Intelligence* 59.3, pp. 277–311.
- Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). ‘The arcade learning environment: An evaluation platform for general agents’. In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.
- Blum, Avrim L. and Merrick L. Furst (1997). ‘Fast planning through planning graph analysis’. In: *Artificial Intelligence* 90.1–2, pp. 281–300.
- Bohren, Jonathan and Steve Cousins (Dec. 2010). ‘The SMACH High-Level Executive [ROS News]’. In: *Robotics Automation Magazine, IEEE* 17.4, pp. 18–20.
- Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York, NY, USA: Oxford University Press, Inc.
- Bonet, Blai and Hector Geffner (2001). ‘Heuristic search planner 2.0’. In: *AI Magazine* 22.3, p. 77.
- Bonjean, Noëlie, Wafa Mefteh, Marie Pierre Gleizes, Christine Maurel, and Frédéric Migeon (2014). ‘ADELFE 2.0’. In: *Handbook on Agent-Oriented Design Processes*. Ed. by Massimo Cossentino, Vincent Hilaire, Ambra Molesini, and Valeria Seidita. Springer Berlin Heidelberg, pp. 19–63.
- Botelho, Sylvia da Costa and Rachid Alami (1999). ‘M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement’. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE, pp. 1234–1239.
- Brambilla, Manuele, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo (Mar. 2013). ‘Swarm robotics: a review from the swarm engineering perspective’. In: *Swarm Intelligence* 7.1, pp. 1–41.
- Brambilla, Manuele, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo (2012). ‘Property-driven design for swarm robotics’. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 139–146.
- Branke, Jürgen, Moez Mnif, Christian Muller-Schloer, and Holger Prothmann (2006). ‘Organic Computing - Addressing complexity by controlled self-organization’. In: *2th*

- International Symposium on Leveraging Applications of Formal Methods, Verification and Validation ISoLA*. IEEE, pp. 185–191.
- Bresciani, Paolo, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos (May 2004). ‘Tropos: An Agent-Oriented Software Development Methodology’. In: *Autonomous Agents and Multi-Agent Systems* 8.3, pp. 203–236.
- Breuer, Thomas, Geovanny R Giorgana Macedo, Ronny Hartanto, Nico Hochgeschwender, Dirk Holz, Frederik Hegger, Zha Jin, Christian Müller, Jan Paulus, Michael Reckhaus, et al. (2012). ‘Johnny: An Autonomous Service Robot for Domestic Environments’. In: *Journal of Intelligent and Robotic Systems* 66.1-2, pp. 245–272.
- Brooks, Rodney A. (1986). ‘A robust layered control system for a mobile robot’. In: *Robotics and Automation, IEEE Journal of* 2.1, pp. 14–23.
- Brun, Yuriy, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw (2009). ‘Engineering self-adaptive systems through feedback loops’. In: *Software engineering for self-adaptive systems*. Springer, pp. 48–70.
- Camazine, Scott, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau (Aug. 2003). *Self-Organization in Biological Systems*. Princeton, N.J.; Oxford: Princeton Univers. Press.
- Campo, Alexandre, Shervin Nouyan, Mauro Birattari, Roderich Groß, and Marco Dorigo (Oct. 2006). ‘Enhancing cooperative transport using negotiation of goal direction’. In: *BNAIC 2006 - Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence*. University of Namur. Namur, Belgium, pp. 365–366.
- Cashmore, Michael, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras (2015). ‘ROSPlan: Planning in the Robot Operating System.’ In: *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, pp. 333–341.
- Cheng, Jiming, Winston Cheng, and Radhika Nagpal (2005). ‘Robust and self-repairing formation control for swarms of mobile agents’. In: *AAAI*. Vol. 5, pp. 59–64.
- Chu, Hoang Nam, Arnaud Glad, Olivier Simonin, Francois Sempe, Alexis Drogoul, and Francois Charpillet (2007). ‘Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation’. In: *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Vol. 1. IEEE, pp. 442–449.
- CogniTeam, Ltd. (2015). *CogniTao (Think As One)*. URL: <http://www.cogniteam.com/cognitao.html> (visited on 02/09/2016).
- Cui, Yanzhe, Richard M. Voyles, Joshua T. Lane, and Mohammad H. Mahoor (Sept. 2014). ‘ReFrESH: A self-adaptation framework to support fault tolerance in field mobile robots’. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1576–1582.
- Dantu, Karthik, Bryan Kate, Jason Waterman, Peter Bailis, and Matt Welsh (2011). ‘Programming Micro-aerial Vehicle Swarms with Karma’. In: *9th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’11. New York, NY, USA: ACM, pp. 121–134.
- Das, Rajarshi, James P. Crutchfield, Melanie Mitchell, and James E. Hanson (1995). ‘Evolving Globally Synchronized Cellular Automata’. In: *Proceedings of the 6th Inter-*

BIBLIOGRAPHY

- national Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 336–343.
- De Nicola, Rocco, Gianluigi Ferrari, Michele Loreti, and Rosario Pugliese (2013). ‘A language-based approach to autonomic computing’. In: *Formal Methods for Components and Objects*. Ed. by Bernhard Beckert, Ferruccio Damiani, FrankS. de Boer, and MarcelloM. Bonsangue. Vol. 7542. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 25–48.
- De Rosa, Michael, Seth Goldstein, Peter Lee, Padmanabhan Pillai, and Jason Campbell (May 2008). ‘Programming modular robots with locally distributed predicates’. In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3156–3162.
- De Wolf, Tom and Tom Holvoet (2005). ‘Emergence Versus Self-Organisation: Different Concepts but Promising When Combined’. In: *Engineering Self-Organising Systems*. Ed. by Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal. Springer Berlin Heidelberg, pp. 1–15.
- De Wolf, Tom and Tom Holvoet (2007). ‘Design patterns for decentralised coordination in self-organising emergent systems’. In: *Engineering Self-Organising Systems*. Springer, pp. 28–49.
- De Wolf, Tom, Giovanni Samaey, and Tom Holvoet (2005). ‘Engineering self-organising emergent systems with simulation-based scientific analysis’. In: *4th International Workshop on Engineering Self-Organising Applications*, pp. 146–160.
- Decugis, Vincent and Jacques Ferber (1998). ‘An Extension of Maes’ Action Selection Mechanism for Animats’. In: *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior on From Animals to Animats 5*. Cambridge, MA, USA: MIT Press, pp. 153–158.
- Deneubourg, Jean-Louis, Serge Aron, Simon Goss, and Jacques M Pasteels (1990a). ‘The self-organizing exploratory pattern of the argentine ant’. In: *Journal of insect behavior* 3.2, pp. 159–168.
- Deneubourg, Jean-Louis, Simon Goss, Nigel Franks, Ana Sendova-Franks, Claire De-train, and Laetitia Chrétien (1990b). ‘The Dynamics of Collective Sorting Robot-like Ants and Ant-like Robots’. In: *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*. Cambridge, MA, USA: MIT Press, pp. 356–363.
- Desai, Jaydev P., James P. Ostrowski, and Vijay Kumar (2001). ‘Modeling and control of formations of nonholonomic mobile robots’. In: *IEEE Transactions on Robotics and Automation*, 17.6, pp. 905–908.
- Dias, Mary Bernardine, Robert Zlot, Nidhi Kalra, and Anthony Stentz (July 2006). ‘Market-Based Multirobot Coordination: A Survey and Analysis’. In: *Proceedings of the IEEE* 94.7, pp. 1257–1270.
- Digani, Valerio, Lorenzo Sabattini, Cristian Secchi, and Cesare Fantuzzi (July 2015). ‘Ensemble Coordination Approach in Multi-AGV Systems Applied to Industrial Warehouses’. In: *IEEE Transactions on Automation Science and Engineering* 12.3, pp. 922–934.
- Doherty, Patrick, Jonas Kvarnström, Piotr Rudol, Marius Wzorek, Gianpaolo Conte, Cyrille Berger, Timo Hinzmann, and Thomas Stastny (2016). ‘A collaborative frame-

- work for 3d mapping using unmanned aerial vehicles’. In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, pp. 110–130.
- Dorigo, Marco, Mauro Birattari, and Thomas Stutzle (2006). ‘Ant colony optimization’. In: *IEEE computational intelligence magazine* 1.4, pp. 28–39.
- Dorigo, Marco, Dario Floreano, Luca Maria Gambardella, Francesco Mondada, Stefano Nolfi, Tarek Baaboura, Mauro Birattari, Michael Bonani, Manuele Brambilla, Arne Brutschy, Daniel Burnier, Alexandre Campo, Anders Lyhne Christensen, Antal Decugniere, Gianni Di Caro, Frederick Ducatelle, Eliseo Ferrante, Alexander Forster, Alexander Martinez Gonzales, Jerome Guzzi, Valentin Longchamp, Stephane Magnenat, Nithin Mathews, Marco Montes de Oca, Rehan O’Grady, Carlo Pinciroli, Giovanni Pini, Philippe Retornaz, James Roberts, Valerio Sperati, Timothy Stirling, Alessandro Stranieri, Thomas Stutzle, Vito Trianni, Elio Tuci, Ali Emre Turgut, and Florian Vaussard (2013). ‘Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms’. In: *IEEE Robotics Automation Magazine* 20.4, pp. 60–71.
- Dorigo, Marco, Vito Trianni, Erol Şahin, Roderich Groß, Thomas H. Labella, Gianluca Baldassarre, Stefano Nolfi, Jean-Louis Deneubourg, Francesco Mondada, Dario Floreano, and Luca M. Gambardella (2004). ‘Evolving self-organizing behaviors for a swarm-bot’. In: *Autonomous Robots* 17.2-3, pp. 223–245.
- Ducatelle, Frederick, Gianni A. Di Caro, and Luca M. Gambardella (2010). ‘Cooperative Self-organization in a Heterogeneous Swarm Robotic System’. In: *12th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’10. New York, NY, USA: ACM, pp. 87–94.
- Ducatelle, Frederick, Alexander Förster, Gianni A. Di Caro, and Luca M. Gambardella (2009a). ‘Supporting navigation in multi-robot systems through delay tolerant network communication’. In: *IFAC Proceedings Volumes* 42.22, pp. 25–30.
- Ducatelle, Frederick, Alexander Förster, Gianni A. Di Caro, and Luca Maria Gambardella (2009b). ‘New task allocation methods for robotic swarms’. In: *9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions*.
- Echeverria, Gilberto, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan (May 2011). ‘Modular open robots simulation engine: MORSE’. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 46–51.
- Edmonds, Bruce (2005). ‘Using the Experimental Method to Produce Reliable Self-Organised Systems’. In: *Engineering Self-Organising Systems: Methodologies and Applications*. Ed. by Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal. Springer Berlin Heidelberg, pp. 84–99.
- Edmonds, Bruce and Joanna J. Bryson (2004). ‘The Insufficiency of Formal Design Methods ” The Necessity of an Experimental Approach - for the Understanding and Control of Complex MAS’. In: *3th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS ’04. Washington, DC, USA: IEEE Computer Society, pp. 938–945.
- Eguchi, Amy and Hiroyuki Okada (2018). ‘If You Give Students a Social Robot? - World Robot Summit Pilot Study’. In: *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. HRI ’18. Chicago, IL, USA: ACM, pp. 103–104.

BIBLIOGRAPHY

- Fernandez-Marquez, Jose Luis, Giovanna Di Marzo Serugendo, and Sara Montagna (2011). ‘Bio-core: Bio-inspired self-organising mechanisms core’. In: *International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*. Springer, pp. 59–72.
- Fernandez-Marquez, Jose Luis, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, and Josep Lluís Arcos (May 2012). ‘Description and composition of bio-inspired design patterns: a complete overview’. In: *Natural Computing* 12.1, pp. 43–67.
- Fikes, Richard E. and Nils J. Nilsson (1972). ‘STRIPS: A new approach to the application of theorem proving to problem solving’. In: *Artificial intelligence* 2.3, pp. 189–208.
- Foote, Tully (Apr. 2013). ‘tf: The transform library’. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop, pp. 1–6.
- Foukarakis, Michalis, Asterios Leonidis, Margherita Antona, and Constantine Stephanidis (2014). ‘Combining Finite State Machine and Decision-Making Tools for Adaptable Robot Behavior’. In: *Universal Access in Human-Computer Interaction. Aging and Assistive Environments*. Ed. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Alfred Kobsa, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Demetri Terzopoulos, Doug Tygar, Gerhard Weikum, Constantine Stephanidis, and Margherita Antona. Vol. 8515. Cham: Springer International Publishing, pp. 625–635.
- Francesca, Gianpiero, Manuele Brambilla, Arne Brutschy, Lorenzo Garattoni, Roman Miletitch, Gaëtan Podevijn, Andreagiovanni Reina, Touraj Soleymani, Mattia Salvato, Carlo Pinciroli, Franco Mascia, Vito Trianni, and Mauro Birattari (Sept. 2015). ‘AutoMoDe-Chocolate: automatic design of control software for robot swarms’. In: *Swarm Intelligence* 9.2-3, pp. 125–152.
- Franceschelli, Mauro, Daniele Rosa, Carla Seatzu, and Francesco Bullo (2013). ‘Gossip algorithms for heterogeneous multi-vehicle routing problems’. In: *Nonlinear Analysis: Hybrid Systems* 10, pp. 156–174.
- Franklin, Stan and Art Graesser (Jan. 1997). ‘Is It an agent, or just a program?: A taxonomy for autonomous agents’. In: *Intelligent Agents III Agent Theories, Architectures, and Languages*. Ed. by Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings. Lecture Notes in Computer Science 1193. Springer Berlin Heidelberg, pp. 21–35.
- Fricke, Stefan, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sessler, and Sahin Albayrak (2001). ‘Agent-based telematic services and telecom applications’. In: *Communications of the ACM* 44.4, pp. 43–48.
- Gancet, Jeremi and Simon Lacroix (2007). ‘Embedding heterogeneous levels of decisional autonomy in multi-robot systems’. In: *Distributed Autonomous Robotic Systems 6*, pp. 263–272.
- Garlan, David, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste (Oct. 2004). ‘Rainbow: architecture-based self-adaptation with reusable infrastructure’. In: *Computer* 37.10, pp. 46–54.
- Gat, Erann (1998). ‘On three-layer architectures’. In: *Artificial intelligence and mobile robots*, pp. 195–210.

- Ge, Shuzhi Sam and Yan Juan Cui (Oct. 2000). ‘New potential functions for mobile robot path planning’. In: *IEEE Transactions on Robotics and Automation* 16.5, pp. 615–620.
- Gelernter, David, Nicholas Carriero, and Sarat Chandran (1985). *Parallel programming in Linda*. Yale University. Department of Computer Science.
- Gerkey, Brian P. and Maja J. Mataric (2002). ‘Sold!: Auction methods for multirobot coordination’. In: *IEEE transactions on robotics and automation* 18.5, pp. 758–768.
- Gershenson, Carlos (2007). ‘Design and control of self-organizing systems’. PhD thesis. Vrije Universiteit Brussel.
- Goebel, Patrick (Apr. 2013). *ROS By Example*. Lulu, pp. 61–88.
- Goldstein, Jeffrey (1999). ‘Emergence as a construct: History and issues’. In: *Emergence* 1.1, pp. 49–72.
- Goodrich, Michael A., Brian Pendleton, P. B. Sujit, and José Pinto (2011). ‘Toward human interaction with bio-inspired robot teams’. In: *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. IEEE, pp. 2859–2864.
- Graff, Daniel, Jan Richling, and Matthias Werner (2013). ‘Programming and Managing the Swarm - An Operating System for an Emerging System of Mobile Devices’. In: *IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*. Vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, pp. 9–16.
- Graff, Daniel, Jan Richling, and Matthias Werner (Apr. 14, 2014). ‘jSwarm: Distributed Coordination in Robot Swarms’. In: *Proceedings of the International Workshop on Robotic Sensor Networks 2014*. Berlin.
- Groß, Roderich, Michael Bonani, Francesco Mondada, and Marco Dorigo (2006). ‘Autonomous self-assembly in swarm-bots’. In: *Robotics, IEEE Transactions on* 22.6, pp. 1115–1130.
- Habibi, Golnaz, Zachary Kingston, Zijian Wang, Mac Schwager, and James McLurkin (2015). ‘Pipelined consensus for global state estimation in multi-agent systems’. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1315–1323.
- Hagras, Hani (Sept. 2018). ‘Toward Human-Understandable, Explainable AI’. In: *Computer* 51.9, pp. 28–36.
- Haken, Herman (1984). *The science of structure: synergetics*. Van Nostrand Reinhold.
- Hasselt, Hado V. (2010). ‘Double Q-learning’. In: *Advances in Neural Information Processing Systems*, pp. 2613–2621.
- Hayes-Roth, Barbara (Aug. 1985). ‘A Blackboard Architecture for Control’. In: *Artificial Intelligence* 26.3, pp. 251–321.
- Hernandez-Sosa, Daniel, Antonio Carlos Dominguez-Brito, Cayetano Guerra-Artal, and Jorge Cabrera-Gómez (2005). ‘Runtime self-adaptation in a component-based robotic framework’. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2700–2705.
- Hertzberg, Joachim, Herbert Jaeger, Uwe R. Zimmer, and Philippe Morignot (1998). ‘A framework for plan execution in behavior-based robots’. In: *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational*

BIBLIOGRAPHY

- Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*. IEEE, pp. 8–13.
- Heßler, Axel, Benjamin Hirsch, and Jan Keiser (2007). ‘Collecting Gold. JIAC IV Agents in MULTI-AGENT PROGRAMMING CONTEST.’ In: *Proceedings of the Fifth International Workshop on Programming Multi-Agent Systems. At AAMAS 2007. Honolulu, HI, USA*.
- Heßler, Axel, Benjamin Hirsch, and Tobias Küster (Aug. 2010). ‘Herding cows with JIAC V’. In: *Annals of Mathematics and Artificial Intelligence* 59.3, pp. 335–349.
- Heßler, Axel, Thomas Konnerth, Pawel Napierala, and Benjamin Wiemann (2013). ‘Multi-Agent Programming Contest 2012: TUB Team Description’. In: *The Multi-Agent Programming Contest 2012 Edition: Evaluation and Team Descriptions*. Ed. by Michael Köster, Federico Schlesinger, and Jürgen Dix. IfI Technical Report Series IfI-13-01. Institut für Informatik, Technische Universität Clausthal, pp. 86–97.
- Hester, Todd, Michael Quinlan, and Peter Stone (May 2012). ‘RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for robot control’. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 85–90.
- Hirsch, Benjamin, Thomas Konnerth, and Axel Heßler (2009). ‘Merging Agents and Services — the JIAC Agent Platform’. In: *Multi-Agent Programming: Languages, Tools and Applications*. Ed. by Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni. Springer, pp. 159–185.
- Hoffmann, Jörg (2001). ‘FF: The fast-forward planning system’. In: *AI magazine* 22.3, p. 57.
- Hoffmann, Jörg (2002). ‘Extending FF to Numerical State Variables’. In: *In Proceedings of the 15th European Conference on Artificial Intelligence*. Wiley, pp. 571–575.
- Hrabia, Christopher-Eyk (Sept. 2016). ‘A Framework for Adaptive and Goal-Driven Behaviour Control of Multi-robot Systems’. In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pp. 275–280.
- Hrabia, Christopher-Eyk, Martin Berger, Axel Hessler, Stephan Wypler, Jan Brehmer, Simon Matern, and Sahin Albayrak (2017). ‘An Autonomous Companion UAV for the SpaceBot Cup Competition 2015’. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, pp. 345–385.
- Hrabia, Christopher-Eyk, Michael Franz Ettlinger, and Axel Hessler (2020). ‘ROS Hybrid Behaviour Planner: Behaviour Hierarchies and Self-Organisation in the Multi-Agent Programming Contest’. In: *The Multi-Agent Programming Contest (MAPC 2018) (Lecture Notes in Computer Science)*. to appear.
- Hrabia, Christopher-Eyk, Axel Hessler, Yuan Xu, Jan Brehmer, and Sahin Albayrak (2018a). ‘EffFeu Project: Efficient Operation of Unmanned Aerial Vehicles for Industrial Fire Fighters’. In: *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications. DroNet’18. Munich, Germany: ACM*, pp. 33–38.
- Hrabia, Christopher-Eyk, Axel Hessler, Yuan Xu, Jacob Seibert, Jan Brehmer, and Sahin Albayrak (2019). ‘EffFeu Project: Towards Mission-Guided Application of Drones in Safety and Security Environments’. In: *Sensors* 19.4.

- Hrabia, Christopher-Eyk, Tanja Katharina Kaiser, and Sahin Albayrak (2017). ‘Combining self-organisation with decision-making and planning’. In: *Multi-Agent Systems and Agreement Technologies*. Springer, pp. 385–399.
- Hrabia, Christopher-Eyk, Patrick Marvin Lehmann, and Sahin Albayrak (July 2019). ‘Increasing Self-Adaptation in a Hybrid Decision-Making and Planning System with Reinforcement Learning’. In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 01, pp. 469–478.
- Hrabia, Christopher-Eyk, Patrick Marvin Lehmann, Nabil Battjbuier, Axel Hessler, and Sahin Albayrak (2018b). ‘Applying robotic frameworks in a simulated multi-agent contest’. In: *Annals of Mathematics and Artificial Intelligence* 84.1-2, pp. 117–138.
- Hrabia, Christopher-Eyk, Marco Lützenberger, and Sahin Albayrak (2018). ‘Towards adaptive multi-robot systems: self-organization and self-adaptation’. In: *The Knowledge Engineering Review* 33, e16.
- Hrabia, Christopher-Eyk, Nils Masuch, and Sahin Albayrak (2015). ‘A Metrics Framework for Quantifying Autonomy in Complex Systems’. In: *Multiagent System Technologies : 13th German Conference, MATES 2015, Cottbus, Germany, September 28 - 30, 2015, Revised Selected Papers*. Ed. by P. Jörg Müller, Wolf Ketter, Gal Kaminka, Gerd Wagner, and Nils Bulling. Cham: Springer International Publishing, pp. 22–41.
- Hrabia, Christopher-Eyk, Marc Schmidt, Andrea Marie Weintraud, and Axel Hessler (2020). ‘Distributed Decision-Making based on Shared Knowledge in the Multi-Agent Programming Contest’. In: *The Multi-Agent Programming Contest (MAPC 2018) (Lecture Notes in Computer Science)*. to appear.
- Hrabia, Christopher-Eyk, Stephan Wypler, and Sahin Albayrak (Apr. 2017). ‘Towards Goal-driven Behaviour Control of Multi-Robot Systems’. In: *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 166–173.
- Iocchi, Luca, Daniele Nardi, Maurizio Piaggio, and Antonio Sgorbissa (2003). ‘Distributed coordination in heterogeneous multi-robot systems’. In: *Autonomous Robots* 15.2, pp. 155–168.
- Jennings, Nicholas R., Katia Sycara, and Michael Wooldridge (Mar. 1998). ‘A Roadmap of Agent Research and Development’. In: *Autonomous Agents and Multi-Agent Systems* 1.1, pp. 7–38.
- Joshi, R.K., Paola Carbone, Feng-Chao Wang, Vasyl G. Kravets, Ying Su, Irina V. Grigorieva, HA Wu, Andre K. Geim, and Rahul Raveendran Nair (Feb. 2014). ‘Precise and Ultrafast Molecular Sieving Through Graphene Oxide Membranes’. In: *Science* 343.6172, pp. 752–754.
- Jung, David (1998). ‘An architecture for cooperation among autonomous agents’. PhD thesis. University of South Australia.
- Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). ‘Planning and acting in partially observable stochastic domains’. In: *Artificial intelligence* 101.1-2, pp. 99–134.
- Kam, Moshe, Xiaoxun Zhu, and Paul Kalata (1997). ‘Sensor fusion for mobile robot navigation’. In: *Proceedings of the IEEE* 85.1, pp. 108–119.
- Kauffman, Stuart A. (1995). *At Home in the Universe: The Search for Laws of Self-organization and Complexity*. Oxford University Press.

BIBLIOGRAPHY

- Kazadi, Sanza T. (2000). ‘Swarm engineering’. phd. California Institute of Technology.
- Keller, Thomas, Patrick Eyerich, and Bernhard Nebel (2012). ‘Task Planning for an Autonomous Service Robot’. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Ed. by Erwin Prassler, Marius Zöllner, Rainer Bischoff, Wolfram Burgard, Robert Haschke, Martin Hägele, Gisbert Lawitzky, Bernhard Nebel, Paul Plöger, and Ulrich Reiser. Springer Berlin Heidelberg, pp. 117–124.
- Kephart, Jeffrey O. and David M. Chess (2003). ‘The vision of autonomic computing’. In: *Computer* 36.1, pp. 41–50.
- Kernbach, Serge, Eugen Meister, Florian Schlachter, Kristof Jebens, Marc Szymanski, Jens Liedke, Davide Laneri, Lutz Winkler, Thomas Schmickl, Ronald Thenius, Paolo Corradi, and Leonardo Ricotti (2008). ‘Symbiotic Robot Organisms: REPLICATOR and SYMBRION Projects’. In: *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. PerMIS ’08. New York, NY, USA: ACM, pp. 62–69.
- Kimura, Hajime, Masayuki Yamamura, and Shigenobu Kobayashi (1995). ‘Reinforcement learning by stochastic hill climbing on discounted reward’. In: *Machine Learning Proceedings 1995*. Elsevier, pp. 295–303.
- Klavins, Eric (2004). ‘A language for modeling and programming cooperative control systems’. In: *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*. Vol. 4. IEEE, pp. 3403–3410.
- Kleiner, Alexander, Rodrigo Baravalle, Andreas Kolling, Pablo Pilotti, and Mario Munich (Sept. 2017). ‘A solution to room-by-room coverage for autonomous cleaning robots’. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5346–5352.
- Kloetzer, Marius and Calin Belta (May 2006). ‘Hierarchical abstractions for robotic swarms’. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*, pp. 952–957.
- Kober, Jens, J. Andrew Bagnell, and Jan Peters (2013). ‘Reinforcement learning in robotics: A survey’. In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.
- Koepke, Hoyt (2011). ‘Why Python rocks for research’. In: *Hacker Monthly* 8.
- Kohl, Nate and Peter Stone (2004). ‘Policy gradient reinforcement learning for fast quadrupedal locomotion’. In: *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*. Vol. 3. IEEE, pp. 2619–2624.
- Korsah, G. Ayorkor, Anthony Stentz, and M. Bernardine Dias (Oct. 2013). ‘A comprehensive taxonomy for multi-robot task allocation’. In: *The International Journal of Robotics Research* 32.12, pp. 1495–1512.
- Krakowczyk, Daniel, Jannik Wolff, Alexandru Ciobanu, Dennis Julian Meyer, and Christopher-Eyk Hrabia (2018). ‘Developing a Distributed Drone Delivery System with a Hybrid Behavior Planning System’. In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, pp. 107–114.
- Kramer, Jeff and Jeff Magee (2007). ‘Self-managed systems: an architectural challenge’. In: *Future of Software Engineering, 2007. FOSE’07*. IEEE, pp. 259–268.

- Krupitzer, Christian (2018). ‘A framework for engineering reusable self-adaptive systems’. PhD thesis.
- Krupke, Dominik, Maximilian Ernestus, Michael Hemmer, and Sándor P Fekete (Sept. 2015). ‘Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces’. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 413–420.
- Kryza, Lennart, Sascha Kapitola, Cem Avsar, and Klaus Briess (Dec. 2015). ‘Developing Technologies for Space on a Terrestrial System: A cost effective Approach for Planetary Robotics Research’. In: *1st Symposium on Space Educational Activities*. Padova, Italy.
- Kudelski, Michal, Marco Cinus, Luca Gambardella, and Gianni A. Di Caro (2012). ‘A framework for realistic simulation of networked multi-robot systems’. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5018–5025.
- Kumar, Ashish, Sanjeev Sharma, Ritu Tiwari, and Samriddhi Majumdar (2012). ‘Area Exploration by Flocking of Multi Robot’. In: *Procedia Engineering* 41, pp. 377–382.
- Kumar, Vignesh and Ferat Sahin (Oct. 2003). ‘Cognitive maps in swarm robots for the mine detection application’. In: *IEEE International Conference on Systems, Man and Cybernetics, 2003*. Vol. 4, pp. 3364–3369.
- Lee, Young-Seol and Sung-Bae Cho (2014). ‘A Hybrid System of Hierarchical Planning of Behaviour Selection Networks for Mobile Robot Control’. In: *International Journal of Advanced Robotic Systems* 11.4, pp. 1–13.
- Lei, Bin and Wenfeng Li (2008). ‘Formation control for multi-robots based on flocking algorithm’. In: *International Conference on Intelligent Robotics and Applications*. Springer, pp. 1238–1247.
- Lemaire, Thomas, Rachid Alami, and Simon Lacroix (2004). ‘A distributed tasks allocation scheme in multi-UAV context’. In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 4. IEEE, pp. 3622–3627.
- Lemos, Rogério de, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaella Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke (2013). ‘Software Engineering for Self-Adaptive Systems: A Second Research Roadmap’. In: *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Springer Berlin Heidelberg, pp. 1–32.
- Li, Jincheng, Jie Chen, Pengbo Wang, and Chunsheng Li (2018). ‘Sensor-oriented path planning for multiregion surveillance with a single lightweight UAV SAR’. In: *Sensors* 18.2, p. 548.

BIBLIOGRAPHY

- Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). ‘Continuous control with deep reinforcement learning’. In: *arXiv preprint arXiv:1509.02971*.
- Maes, Pattie (1989). ‘How to do the right thing’. In: *Connection Science* 1.3, pp. 291–323.
- Mamei, Marco, Matteo Vasirani, and Franco Zambonelli (2004). ‘Experiments of morphogenesis in swarms of simple mobile robots’. In: *Applied Artificial Intelligence* 18.
- Mamei, Marco, Matteo Vasirani, and Franco Zambonelli (2005). ‘Self-organizing spatial shapes in mobile particles: The TOTA approach’. In: *Engineering Self-Organising Systems*. Springer, pp. 138–153.
- Masar, Marek (June 2013). ‘A biologically inspired swarm robot coordination algorithm for exploration and surveillance’. In: *IEEE 17th International Conference on Intelligent Engineering Systems (INES)*, pp. 271–275.
- Matarić, Maja J. (Dec. 1995). ‘Issues and approaches in the design of collective autonomous agents’. In: *Robotics and Autonomous Systems. Moving the Frontiers between Robotics and Biology* 16.2–4, pp. 321–331.
- Medermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
- Mettler, Tobias, Michaela Sprenger, and Robert Winter (2017). ‘Service robots in hospitals: new perspectives on niche evolution and technology affordances’. In: *European Journal of Information Systems* 26.5, pp. 451–468.
- Mills-Tettey, G. Ayorkor, Anthony Stentz, and Mary Bernardine Dias (July 2007). *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs*. Tech. rep. CMU-RI-TR-07-27. Pittsburgh, PA: Carnegie Mellon University.
- Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). ‘Playing atari with deep reinforcement learning’. In: *arXiv preprint arXiv:1312.5602*.
- Mokarizadeh, S., A. Grosso, M. Matskin, P. Kungas, and A. Haseeb (May 2009). ‘Applying Semantic Web Service Composition for Action Planning in Multi-robot Systems’. In: *2009 Fourth International Conference on Internet and Web Applications and Services*, pp. 370–376.
- Montagna, Sara, Mirko Viroli, Jose Luis Fernandez-Marquez, Giovanna Di Marzo Seruendo, and Franco Zambonelli (June 2013). ‘Injecting Self-Organisation into Pervasive Service Ecosystems’. In: *Mobile Networks and Applications* 18.3, pp. 398–412.
- Morandini, Mirko, Frédéric Migeon, Marie-Pierre Gleizes, Christine Maurel, Loris Penserini, and Anna Perini (Nov. 2009). ‘A Goal-Oriented Approach for Modelling Self-organising MAS’. In: *Engineering Societies in the Agents World X. Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, pp. 33–48.
- Moriarty, David E, Alan C Schultz, and John J Grefenstette (1999). ‘Evolutionary algorithms for reinforcement learning’. In: *Journal of Artificial Intelligence Research* 11, pp. 241–276.

- Nagasaka, Yasunori, Kazuhito Murakami, Tadashi Naruse, Tomoichi Takahashi, and Yasuo Mori (Jan. 2001). ‘Potential Field Approach to Short Term Action Planning in RoboCup F180 League’. In: *RoboCup 2000: Robot Soccer World Cup IV*. Ed. by Peter Stone, Tucker Balch, and Gerhard Kraetzschmar. Lecture Notes in Computer Science 2019. Springer Berlin Heidelberg, pp. 345–350.
- Naghsh, Amir M., Jeremi Gancet, Andry Tanoto, and Chris Roast (2008). ‘Analysis and design of human-robot swarm interaction in firefighting’. In: *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*. IEEE, pp. 255–260.
- Nagpal, Radhika (2002). ‘Programmable self-assembly using biologically-inspired multi-agent control’. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. ACM, pp. 418–425.
- Nagpal, Radhika (2004). ‘A Catalog of Biologically-Inspired Primitives for Engineering Self-Organization’. In: *Engineering Self-Organising Systems*. Ed. by Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli. Springer Berlin Heidelberg, pp. 53–62.
- Nebel, Bernhard and Yuliya Babovich-Lierler (2004). ‘When are behaviour networks well-behaved?’ In: *Proceedings of the 16th European Conference on Artificial Intelligence*. IOS Press, pp. 672–676.
- Newman, David V. (1996). ‘Emergence and Strange Attractors’. In: *Philosophy of Science* 63.2, pp. 245–261.
- Nguyen, Hai, Matei Ciocarlie, Kaijen Hsiao, and Charles C. Kemp (May 2013). ‘ROS commander (ROSCo): Behavior creation for home robots’. In: *2013 IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany: IEEE, pp. 467–474.
- Nicola, Rocco De, Diego Latella, Alberto Lluch Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi, and Andrea Vandin (2015). ‘The SCEL Language: Design, Implementation, Verification’. In: *Software Engineering for Collective Autonomic Systems*. Ed. by Martin Wirsing, Matthias Hözl, Nora Koch, and Philip Mayer. Lecture Notes in Computer Science 8998. Springer International Publishing, pp. 3–71.
- Nicolis, Gregoire (1989). ‘Physics of far-from-equilibrium systems and self-organisation’. In: *The New Physics* 11, pp. 316–347.
- Noël, Victor and Franco Zambonelli (2015). ‘Methodological Guidelines for Engineering Self-organization and Emergence’. In: *Software Engineering for Collective Autonomic Systems*. Ed. by Martin Wirsing, Matthias Hözl, Nora Koch, and Philip Mayer. Lecture Notes in Computer Science 8998. Springer International Publishing, pp. 355–378.
- Nolfi, Stefano and Dario Floreano (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press Cambridge.
- Olfati-Saber, Reza (2006). ‘Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory’. In: *IEEE Transactions on Automatic Control* 51.3, pp. 401–420.
- Oliehoek, Frans A., Matthijs T. J. Spaan, Bas Terwijn, Philipp Robbel, and João V. Messias (2017). ‘The MADP Toolbox: An Open Source Library for Planning and

BIBLIOGRAPHY

- Learning in (Multi-)Agent Systems’. In: *Journal of Machine Learning Research* 18.89, pp. 1–5.
- Oreizy, Peyman, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf (May 1999). ‘An Architecture-Based Approach to Self-Adaptive Software’. In: *IEEE Intelligent Systems* 14.3, pp. 54–62.
- Panait, Liviu and Sean Luke (2005). ‘Cooperative multi-agent learning: The state of the art’. In: *Autonomous Agents and Multi-Agent Systems* 11.3, pp. 387–434.
- Parker, Lynne E. (Apr. 1998). ‘ALLIANCE: An architecture for fault tolerant multirobot cooperation.’ In: *IEEE Transactions on Robotics and Automaton* 14.2, pp. 220–240.
- Parunak, H. Van, Michael Purcell, and Robert O’Connell (2002). ‘Digital pheromones for autonomous coordination of swarming UAV’s’. In: *1st UAV Conference*, p. 3446.
- Pednault, Edwin P. D. (1989). ‘ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus’. In: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 324–332.
- Penders, Jacques, Lyuba Alboul, Ulf Witkowski, Amir Naghsh, Joan Saez-Pons, Stefan Herbrechtsmeier, and Mohamed El-Habbal (Jan. 2011). ‘A Robot Swarm Assisting a Human Fire-Fighter’. In: *Advanced Robotics* 25.1-2, pp. 93–117.
- Perico, Danilo H., Thiago P. D. Homem, Aislan C. Almeida, Isaac J. Silva, Claudio O. Vilão, Vinicius N. Ferreira, and Reinaldo A. C. Bianchi (Aug. 2018). ‘Humanoid Robot Framework for Research on Cognitive Robotics’. In: *Journal of Control, Automation and Electrical Systems* 29.4, pp. 470–479.
- Peters, Jan and Stefan Schaal (2006). ‘Policy gradient methods for robotics’. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, pp. 2219–2225.
- Peters, Jan, Sethu Vijayakumar, and Stefan Schaal (2003). ‘Reinforcement learning for humanoid robotics’. In: *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20.
- Picard, Gauthier and Marie-Pierre Gleizes (2004). ‘The ADELFE methodology’. In: *Methodologies and Software Engineering for Agent Systems*. Springer, pp. 157–175.
- Pincioli, Carlo and Giovanni Beltrame (2016). ‘Buzz: An extensible programming language for heterogeneous swarm robotics’. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, pp. 3794–3800.
- Portugal, David and Rui Rocha (2011). ‘A survey on multi-robot patrolling algorithms’. In: *Doctoral Conference on Computing, Electrical and Industrial Systems*. Springer, pp. 139–146.
- Prehofer, Christian and Christian Bettstetter (2005). ‘Self-organization in communication networks: principles and design paradigms’. In: *IEEE Communications magazine* 43.7, pp. 78–85.
- Preisler, Thomas, Ante Vilenica, and Wolfgang Renz (July 2013). ‘Decentralized Coordination in Self-Organizing Systems based on Peer-to-Peer Coordination Spaces’. In: *Electronic Communications of the EASST* 56.0, pp. 1–13.

- Prokopenko, Mikhail (2009). ‘Guided self-organization’. In: *Human Frontiers Science Program (HFSP) Journal* 3.5, pp. 287–289.
- Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng (2009). ‘ROS: an open-source Robot Operating System’. In: *ICRA workshop on open source software 3.3.2*, p. 5.
- Rabe, Markus and Felix Dross (2015). ‘A reinforcement learning approach for a decision support system for logistics networks’. In: *Proceedings of the 2015 Winter Simulation Conference*. IEEE Press, pp. 2020–2032.
- Ramirez, Andres J., Adam C. Jensen, and Betty H.C. Cheng (June 2012). ‘A taxonomy of uncertainty for dynamically adaptive systems’. In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, pp. 99–108.
- Ranjbar-Sahraei, Bijan, Gerhard Weiss, and Ali Nakisae (2012). ‘A multi-robot coverage approach based on stigmergic communication’. In: *German Conference on Multiagent System Technologies*. Springer, pp. 126–138.
- Rassenti, Stephen J., Vernon L. Smith, and Robert L. Bulfin (1982). ‘A combinatorial auction mechanism for airport time slot allocation’. In: *The Bell Journal of Economics*, pp. 402–417.
- Reina, Andreagiovanni, Marco Dorigo, and Vito Trianni (Sept. 2014). ‘Towards a Cognitive Design Pattern for Collective Decision-Making’. In: *Swarm Intelligence*. Springer International Publishing, pp. 194–205.
- Ren, Wei and Randal W. Beard (2004). ‘A decentralized scheme for spacecraft formation flying via the virtual structure approach’. In: *AIAA Journal of Guidance, Control, and Dynamics* 27, pp. 73–82.
- Reshamwala, Alpa and Deepika P. Vinchurkar (2013). ‘Robot path planning using an ant colony optimization approach: A survey’. In: *International Journal of Advanced Research in Artificial Intelligence* 2, pp. 65–71.
- Reynolds, Craig W. (1999). ‘Steering behaviors for autonomous characters’. In: *Game developers conference*. Vol. 1999, pp. 763–782.
- Rovida, Francesco, Matthew Crosby, Dirk Holz, Athanasios S Polydoros, Bjarne Grossmann, Ronald PA Petrick, and Volker Krüger (2017). ‘SkiROS—A Skill-Based Robot Control Platform on Top of ROS’. In: *Robot Operating System (ROS)*. Springer, pp. 121–160.
- Rovida, Francesco, Bjarne Grossmann, and Volker Krüger (Sept. 2017). ‘Extended behavior trees for quick definition of flexible robotic tasks’. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6793–6800.
- Rovida, Francesco and Volker Krüger (2015). ‘Design and development of a software architecture for autonomous mobile manipulators in industrial environments’. In: *Industrial Technology (ICIT), 2015 IEEE International Conference on*. IEEE, pp. 3288–3295.
- Şahin, Erol (Jan. 2005). ‘Swarm Robotics: From Sources of Inspiration to Domains of Application’. In: *Swarm Robotics*. Ed. by Erol Şahin and William M. Spears. Lecture Notes in Computer Science 3342. Springer Berlin Heidelberg, pp. 10–20.

BIBLIOGRAPHY

- Sanner, Scott (2010). ‘Relational dynamic influence diagram language (rddl): Language description’. In: *Unpublished ms. Australian National University*, p. 32.
- Savidis, Anthony, Margherita Antona, and Constantine Stephanidis (2005). ‘A decision-making specification language for verifiable user-interface adaptation logic’. In: *International Journal of Software Engineering and Knowledge Engineering* 15.06, pp. 1063–1094.
- Schmeck, Hartmut (May 2005). ‘Organic computing - a new vision for distributed embedded systems’. In: *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pp. 201–203.
- Semertzidou, Christina, Nikolaos I. Dourvas, Michail-Antisthenis Tsompanas, Andrew Adamatzky, and Georgios Ch. Sirakoulis (2016). ‘Introducing Chemotaxis to a Mobile Robot’. In: *Artificial Intelligence Applications and Innovations: 12th IFIP WG 12.5 International Conference and Workshops, AIAI 2016, Thessaloniki, Greece, September 16-18, 2016, Proceedings*. Ed. by Lazaros Iliadis and Ilias Maglogiannis. Cham: Springer International Publishing, pp. 396–404.
- Serugendo, Giovanna Di Marzo, Marie Pierre Gleizes, and Anthony Karageorgos (2006). ‘Self-Organisation and Emergence in MAS: An Overview.’ In: *Informatica (Slovenia)* 30.1, pp. 45–54.
- Serugendo, Giovanna Di Marzo, Marie-Pierre Gleizes, and Anthony Karageorgos (2005). ‘Self-organization in multi-agent systems’. In: *The Knowledge Engineering Review* 20.2, pp. 165–189.
- Sheh, Raymond, Sören Schwertfeger, and Arnoud Visser (Oct. 2016). ‘16 Years of RoboCup Rescue’. In: *KI - Künstliche Intelligenz* 30.3, pp. 267–277.
- Sierhuis, Maarten, Jeffrey M. Bradshaw, Alessandro Acquisti, Ron Van Hoof, Renia Jeffers, and Andrzej Uszok (2003). ‘Human-agent teamwork and adjustable autonomy in practice’. In: *Proceedings of the seventh international symposium on artificial intelligence, robotics and automation in space (I-SAIRAS)*.
- Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). ‘Mastering the game of Go with deep neural networks and tree search’. In: *nature* 529.7587, p. 484.
- Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). ‘Mastering the game of go without human knowledge’. In: *Nature* 550.7676, pp. 354–359.
- Simmons, Reid G. (Feb. 1994). ‘Structured control for autonomous robots’. In: *IEEE Transactions on Robotics and Automation* 10.1, pp. 34–43.
- Simmons, Reid and David Apfelbaum (Oct. 1998). ‘A task description language for robot control’. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications*. Vol. 3, 1931–1937 vol.3.
- Skorput, P., S. Mandzuka, and H. Vojvodic (Sept. 2016). ‘The use of Unmanned Aerial Vehicles for forest fire monitoring’. In: *2016 International Symposium ELMAR*, pp. 93–96.

- Smith, Reid G. (1980). ‘The contract net protocol: High-level communication and control in a distributed problem solver’. In: *IEEE Transactions on Computers* 12, pp. 1104–1113.
- Song, Kai-Tai, Chi-Yi Tsai, and Cheng-Hsien Chiu Huang (Sept. 2008). ‘Multi-robot cooperative sensing and localization’. In: *2008 IEEE International Conference on Automation and Logistics*, pp. 431–436.
- Song, Peng and Vijay Kumar (2002). ‘A Potential Field Based Approach to Multi-Robot Manipulation’. In: *In: IEEE Int’l. Conf. on Robotics and Automation*, 1217–1222.
- Steghöfer, Jan-Philipp, Hella Seebach, Benedikt Eberhardinger, and Wolfgang Reif (2014). ‘PosoMAS: An Extensible, Modular SE Process for Open Self-organising Systems’. In: *PRIMA 2014: Principles and Practice of Multi-Agent Systems*. Ed. by Hoa Khanh Dam, Jeremy Pitt, Yang Xu, Guido Governatori, and Takayuki Ito. Vol. 8861. Lecture Notes in Computer Science. Springer International Publishing, pp. 1–17.
- Stroupe, Ashley W., Martin C. Martin, and Tucker Balch (2001). ‘Distributed sensor fusion for object position estimation by multi-robot systems’. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 2. IEEE, pp. 1092–1098.
- Sudeikat, Jan, Lars Braubach, Alexander Pokahr, Wolfgang Renz, and Winfried Lamersdorf (2009). ‘Systematically engineering self-organizing systems: The SodekoVS approach’. In: *Electronic Communications of the EASST* 17, pp. 1–12.
- Sudeikat, Jan and Wolfgang Renz (2009). ‘MASDynamics: Toward Systemic Modeling of Decentralized Agent Coordination’. In: *Kommunikation in Verteilten Systemen (KiVS)*. Ed. by Klaus David and Kurt Geihs. Informatik aktuell. Springer Berlin Heidelberg, pp. 79–90.
- Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge.
- Thrun, Sebastian (Mar. 2002). ‘Probabilistic Robotics’. In: *Communication of the ACM* 45.3, pp. 52–57.
- Tomic, Teodor, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grix, Felix Ruess, Michael Suppa, and Darius Burschka (Sept. 2012). ‘Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue’. In: *IEEE Robotics Automation Magazine* 19.3, pp. 46–56.
- Twidwell, Dirac, Craig R. Allen, Carrick Detweiler, James Higgins, Christian Laney, and Sebastian Elbaum (2016). ‘Smokey comes of age: unmanned aerial systems for fire management’. In: *Frontiers in Ecology and the Environment* 14.6, pp. 333–339.
- Tzafestas, Spyros G. (July 2018). ‘Mobile Robot Control and Navigation: A Global Overview’. In: *Journal of Intelligent & Robotic Systems* 91.1, pp. 35–58.
- Vallati, Mauro, Lukas Chrupa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, et al. (2015). ‘The 2014 international planning competition: Progress and trends’. In: *AI Magazine* 36.3, pp. 90–98.
- Varga, Maja, Meysam Basiri, Gregoire Heitz, and Dario Floreano (2015). ‘Distributed formation control of fixed wing micro aerial vehicles for area coverage’. In: *Intelligent*

BIBLIOGRAPHY

- Robots and Systems (IROS)*, 2015 IEEE/RSJ International Conference on. IEEE, pp. 669–674.
- Villegas, Norha M., Gabriel Tamura, Hausi A. Müller, Laurence Duchien, and Rubby Casallas (2013). ‘DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems’. In: *Software Engineering for Self-Adaptive Systems II*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Vol. 7475. Springer Berlin Heidelberg, pp. 265–293.
- Viroli, Mirko, Matteo Casadei, and Andrea Omicini (2009). ‘A framework for modelling and implementing self-organising coordination’. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, pp. 1353–1360.
- Viscido, Steven V., Julia K. Parrish, and Daniel Grünbaum (2004). ‘Individual behavior and emergent properties of fish schools: a comparison of observation and theory’. In: *Marine Ecology Progress Series* 273, pp. 239–250.
- Wachter, Sandra, Brent Mittelstadt, and Luciano Floridi (2017). ‘Transparent, explainable, and accountable AI for robotics’. In: *Science Robotics* 2.6, eaan6080.
- Watkins, Christopher J.C.H. and Peter Dayan (1992). ‘Q-learning’. In: *Machine learning* 8.3-4, pp. 279–292.
- Wegner, Peter (May 1997). ‘Why Interaction is More Powerful Than Algorithms’. In: *Communications of the ACM* 40.5, pp. 80–91.
- Weiss, Gerhard, Lars Braubach, and Paolo Giorgini (1999). ‘Intelligent agents’. In: *Handbook of Technology Management*.
- Weißbach, Martin, Philipp Chrszon, Thomas Springer, and Alexander Schill (Sept. 2017). ‘Decentrally Coordinated Execution of Adaptations in Distributed Self-Adaptive Software Systems’. In: *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 111–120.
- Winfield, Alan F. T., Christopher J. Harper, and Julien Nembrini (July 2004). ‘Towards Dependable Swarms and a New Discipline of Swarm Engineering’. In: *Swarm Robotics*. Springer Berlin Heidelberg, pp. 126–142.
- Wolf, Tom De and Tom Holvoet (2004). ‘Emergence and Self-Organisation: a statement of similarities and differences’. In: *Lecture Notes in Artificial Intelligence*. Springer Verlag, pp. 96–110.
- Wooldridge, Michael J. (2000). *Reasoning about rational agents*. MIT press.
- Yang, Bin, Yongsheng Ding, Yaochu Jin, and Kuangrong Hao (Oct. 2015a). ‘Self-organized Swarm Robot for Target Search and Trapping Inspired by Bacterial Chemotaxis’. In: *Robotics and Autonomous Systems* 72.C, pp. 83–92.
- Yang, Cheng-Hsuan, Ming-Chang Wen, Yi-Chu Chen, and Shih-Chung Kang (2015b). ‘An optimized unmanned aerial system for bridge inspection’. In: *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*. Vol. 32. IAARC Publications, p. 1.
- Yliniemi, Logan, Adrian K. Agogino, and Kagan Tumer (Dec. 2014). ‘Multirobot Coordination for Space Exploration’. In: *AI Magazine* 35.4, pp. 61–74.
- Yogeswaran, Mohan and SG Ponnambalam (2010). ‘Swarm robotics: An extensive research review’. In: *Advanced Knowledge Application in Practice*. Ed. by Igor Fürstner. InTech.

- Yoon, Sung Wook, Alan Fern, and Robert Givan (2007). ‘FF-Replan: A Baseline for Probabilistic Planning.’ In: *ICAPS*. Vol. 7, pp. 352–359.
- Younes, Håkan L.S., Michael L. Littman, David Weissman, and John Asmuth (2005). ‘The first probabilistic track of the international planning competition’. In: *Journal of Artificial Intelligence Research* 24, pp. 851–887.
- Yu, Chih-Han and Radhika Nagpal (2009). ‘Self-adapting modular robotics: A generalized distributed consensus framework’. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1881–1888.
- Yu, Chih-Han and Radhika Nagpal (July 2011). ‘A Self-adaptive Framework for Modular Robots in a Dynamic Environment: Theory and Applications’. In: *The International Journal of Robotics Research* 30.8, pp. 1015–1036.
- Zhao, Qingying, Min Li, Zhongya Wang, Jie Li, and Jun Luo (Dec. 2015). ‘A Quorum Sensing algorithm to control nanorobot population and drug concentration in cancer area’. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 42–47.
- Zlot, Robert and Anthony Stentz (2005). ‘Complex task allocation for multiple robots’. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, pp. 1515–1522.