UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Wissensbasierte Systeme

Hierarchische hybride Planung für mobile Roboter

Dissertation

zur Erlangung des Doktorgrades (Dr. rer. nat.)
des Fachbereichs Mathematik/Informatik
der Universität Osnabrück

vorgelegt von

Sebastian Stock

Oktober 2016

Erstgutachter: Prof. Dr. Joachim Hertzberg
Zweitgutachter: Prof. Dr. Susanne Biundo-Stephan

Zusammenfassung

Damit mobile Roboter vielfältige komplexe Aufgaben autonom erfüllen können, benötigen sie Planung, um so entsprechend der Gegebenheiten ihrer Umgebung zu handeln. Durch die stetig zunehmenden Fähigkeiten der Roboterhardware gewinnt die Handlungsplanung und deren Integration in das Gesamtsystem zunehmend an Bedeutung. Die vorliegende Arbeit versucht, einen weiteren Schritt Richtung planbasierter Robotersteuerung zu gehen. Dabei wird zunächst die Verwendung des HTN-Planers SHOP2 [117] in einem Robotersystem, das sich das Lernen aus Erfahrungen zum Ziel gesetzt hat, beschrieben und Wege aufgezeigt, wie die Robustheit des Systems durch die Integration mit anderen Komponenten erhöht werden kann.

Mobilen Robotern stehen unterschiedliche Formen von Wissen, wie temporales oder räumliches Wissen oder Informationen über Ressourcen zur Verfügung. Diese können von SHOP2 jedoch nicht genutzt werden. Um diese Anforderung zu erfüllen, wird in dieser Arbeit der hybride hierarchische Planer CHIMP präsentiert, der die Vorteile hierarchischer Planung und der hybriden Planung als Meta-CSP [28, 104], das die Integration verschiedener Wissensformen erlaubt, kombiniert. Des Weiteren können seine Pläne parallel ausführbare Aktionen enthalten, und zusätzliche Aufgaben können während der Ausführung in den bestehenden Plan integriert werden.

Abstract

To act autonomously and robustly in complex environments, mobile robots require task planning to adapt to the given situation. With the growing capabilities of robot hardware and software this is becoming increasingly important. Yet, only few integrated robotic systems exist. This thesis attempts to make a next step towards robust plan-based robot control. For this, it first describes how the HTN-Planner SHOP2 [117] can be used in a robot control architecture that aims to improve its performance based on experiences. Based on that, means to increase the systems robustness by integrating the planner with other components are discussed.

The environments and tasks of mobile robots contain various forms of knowledge like information about temporal requirements or resources that should be reasoned about in order to achieve a safe and robust behaviour. However, this cannot be fully used by planners like SHOP2. Therefore, this thesis presents the hierarchical hybrid planner CHIMP which combines the advantages of hierarchical planning and hybrid planning with different forms of knowledge as a Meta-CSP [28, 104]. Furthermore, CHIMP's plans can contain actions that can be executed in parallel, and additional goal tasks can be inserted into an existing plan during its execution.

Danksagung

Zuallererst danke ich Herrn Prof. Dr. Joachim Hertzberg dafür, dass er mich seit meinem Bachelorstudium für das Themengebiet der planbasierten Robotersteuerung begeistert hat und mir die Möglichkeit gegeben hat, dies in RACE und anschließend am DFKI zu vertiefen. Zudem danke ich ihm für die außerordentlich angenehme und vorbildliche Zusammenarbeit und dafür, dass er sich bei Fragen und Problemen stets Zeit genommen hat.

Meinen Kollegen an der Universität Osnabrück und am DFKI danke ich für die gute Arbeitsatmosphäre.

Dem RACE-Team danke ich für die nette Zusammenarbeit und die anstrengenden, aber immer interessanten und schönen Code-Sprints und Projekttreffen. Ich blicke immer wieder gerne darauf zurück. Ganz besonders danke ich Martin Günther, Štefan Konečný, Dr. Masoumeh Mansouri und Dr. Federico Pecora für die gute und enge Zusammenarbeit, die mir sehr viel Freude bereitet hat.

Ganz besonderer Dank gilt auch Dr. Kai Lingemann für alle Ratschläge, das stets offene Ohr und natürlich dafür, dass es im Büro jeden Tag aufs Neue interessant ist.

Und schließlich danke ich meiner Frau Ann-Christin für ihre unendliche Unterstützung, ihre Geduld und das Verständnis dafür, dass ich in den letzten Jahren nicht so viel Zeit für sie aufbringen konnte, wie sie es verdient gehabt hätte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Seit den Anfängen der Robotik und der Künstlichen Intelligenz gibt es die Vision von Robotern, die den Menschen in ihrem privaten und beruflichen Leben unterstützen und die Arbeit erleichtern. Der Roboter sollte sein Verhalten autonom an die Bedürfnisse des Menschen und die Anforderungen seiner Umgebung anpassen. Bereits vor fast fünfzig Jahren wurde unter anderem mit dem Roboter SHAKEY an autonomer Handlungsplanung für Roboter und der Planausführung geforscht [122]. Seitdem haben sich die Fähigkeiten mobiler Roboter durch neue Algorithmen, Sensoren, Manipulatoren und andere Hardware stark verbessert. Roboter werden zunehmend in vielfältigen Bereichen eingesetzt. Sie finden Anwendung in Warenhäusern [35, 68] und der Produktion, erste Prototypen selbstfahrender Autos fahren auf Straßen, Staubsaugroboter und Rasenmäher können bereits für den Hausgebrauch gekauft werden [82] und Robotik- und KI-Methoden werden zunehmend im Agrarbereich angewendet [150].

Doch obwohl autonome Handlungsplanung [63] einer der ersten Forschungsschwerpunkte im Bereich der KI und der Robotik gewesen ist, werden auf den Robotern dieser Anwendungsbereiche relativ wenige High-Level-Planungs- und Inferenzverfahren eingesetzt. Stattdessen kommen vielmehr domänenspezifische Handlungsstrategien zum Tragen. Multifunktionale Serviceroboter, die das Haus reinigen, aufräumen, kochen und servieren scheinen trotz Wettbewerben wie RoboCup@Home [177] noch in ferner Zukunft zu liegen. Für diese Art von Robotern sind neben zuverlässigen und sicheren Manipulatoren und Sensoren vor allem solche abstrakteren Verfahren sowie eine geeignete Integration der Einzelkomponenten in eine Kontrollarchitektur notwendig. Der Roboter muss die Folgen seiner eigenen Aktionen und der Aktionen anderer Akteure sowie die Intentionen des Benutzers und mögliche Unsicherheiten seiner Umgebungsinformationen und Sensordaten berücksichtigen und in sein Handeln einbeziehen. Handlungsplanung ist daher ein wichtiger Baustein, um Roboter autonom in komplexen, realen Anwendungsbereichen wie Restaurants einzusetzen: Bereits Aufgaben wie das Servieren eines Kaffees können aus einer Vielzahl von Einzelaktionen und Anforderungen bestehen. Daher sollte dieses Verhalten nicht handkodiert, sondern aufgaben- und kontextadäquat geplant werden.

Ein großer Anteil dieser Arbeit ist im Rahmen des EU-geförderten FP7-Projekts *RACE* [76] entstanden. RACE hat es sich zum übergeordneten Ziel gesetzt, einen weiteren Schritt in Rich-

tung autonomer mobiler Roboter zu gehen, indem der Roboter aus seinen Erfahrungen lernen und sein Verhalten verbessern soll. Ein großer Teil des Projekts hat sich zudem mit hybriden Repräsentationsverfahren und einer Weiterentwicklung der Planung und Planausführung sowie deren Integration für mobile Roboter befasst. Das Ziel dieser Arbeit ist eng mit den Zielen von RACE und den Anforderungen und Aufgaben, die darin zu erfüllen waren, verbunden. Auch das Demonstrationsszenario von RACE in Form eines Serviceroboters, der als Kellner in einem Restaurant agiert, wurde aufgegriffen. Kapitel 2 gibt einen kurzen Überblick über das Projekt.

Unter den verschiedenen Arten von Handlungsplanung aus dem Bereich der Künstlichen Intelligenz bietet Hierarchical Task Network (HTN) Planung [51] einige Vorteile für mobile Roboter und war aus diesem Grund in RACE als zu verwendendes Planungsverfahren gesetzt. Durch die Verwendung von *Methoden*, welche zusätzlich zu den *Operatoren* in der Domänenbeschreibung definiert werden, ist die Planerzeugung auch für größere Problemstellungen vergleichsweise schnell und blockiert die Robotersteuerung somit nicht lange. Zudem wird dem Domänenmodellierer so eine weitere Möglichkeit gegeben, Hintergrundwissen einfließen zu lassen und ein sicheres Roboterverhalten zu gewährleisten, da diese Methoden die möglichen Pläne einschränken. Die Dekompositionshierarchie ermöglicht zudem eine leichte Nachvollziehbarkeit der Pläne. Im Rahmen von RACE war ein weiterer Vorteil, dass Methoden gelernt werden können, um somit das Roboterverhalten basierend auf Erfahrungen zu verbessern. Das erste Teilziel der vorliegende Arbeit war daher zunächst die Integration des frei verfügbaren und vielfach eingesetzten HTN-Planers SHOP2 [117] in ein Robotersystem, welches das Lernen aus Erfahrungen ermöglicht.

Die anfängliche Verwendung von SHOP2 hatte den Vorteil, bereits zu einem frühen Zeitpunkt im Projekt ein lauffähiges System zur Aufnahme erster Robotererfahrungen umsetzen und testen zu können. Allerdings stellen mobile Roboter besondere Anforderungen, die in der Regel nicht von Handlungsplanern berücksichtigt werden. Im Gegensatz zu vielen klassischen Anwendungsgebieten von Planungsalgorithmen kann die Plangenerierung hier häufig nicht als einstufiges Verfahren betrachtet werden, bei dem der Ausgangszustand gänzlich und exakt bekannt ist. Durch eine Diskrepanz zwischen dem vom Roboter angenommenen und dem tatsächlichen Ausgangszustand, durch unerwartete Ereignisse, Sensordatenfehler oder Fehler der Roboteraktionen verläuft die Planausführung oftmals anders als erwartet. Zudem können dem Roboter während der Planausführung zusätzliche Aufgaben gegeben werden. Die Vor- und Nachteile dieses ersten Systems sollen daher analysiert und geeignete Methoden untersucht werden, um dieses System zu verbessern. Dabei steht vor allem die Verbindung zwischen der Planung und der Planausführung im Vordergrund.

Eine weitere wichtige Anforderung ist die Verwendung unterschiedlicher Arten von Wissen für die Robotersteuerung. In komplexen Umgebungen wie beispielsweise einem Restaurant stehen viele Arten von Informationen zur Verfügung. Neben kausalem Wissen sind auch temporales und räumliches Wissen, Wissen über Ressourcen oder Wissen, das von anderen Roboterkomponenten wie zum Beispiel einem Pfadplaner bereitgestellt werden kann, von Bedeutung. Bekommt der Roboter beispielsweise die Aufgabe, einen Kaffee mit Milch und Zucker zu servieren, sollte auch temporales Wissen genutzt werden; so können Pläne verhindert werden, deren Ausführungen zu lange dauern und beispielsweise in einem kalten Kaffee und einem unzufriedenen Gast resultieren

würden. Hybrides Planen unter Einbeziehung unterschiedlicher Wissensrepräsentationsformen für mobile Roboter ist daher ein aktives Forschungsthema. Die meisten Ansätze beschränken sich darauf, zwei oder drei Planungs- bzw. Schlussfolgerungsverfahren zu integrieren. Häufig handelt es sich dabei um problemspezifische Planer für bestimmte Anwendungen. Um jedoch den Planer beziehungsweise Roboter in verschiedenen Umgebungen und für verschiedene Problemstellungen einzusetzen, ist ein allgemeinerer Ansatz notwendig.

Der Meta-CSP-Ansatz [28] ist ein solches generelles Verfahren für hybrides Schlussfolgern. Er basiert auf der Kombination spezieller Inferenzverfahren für verschiedene Constraint-Netze. Dabei werden mehrere Ebenen von Constraint-Netzen definiert. Die höheren Ebenen setzen sich aus Kombinationen tiefer liegender Constraint-Netze zusammen. Zusätzlich zur Konsistenzprüfung der untersten Constraint-Netze können auch Constraints auf den höheren Ebenen definiert werden; diese heißen *Meta-Constraints*. Im Rahmen von RACE wurde an der Universität Örebro das *Meta-CSP-Framework*[1] entwickelt und für Beispiele aus dem Bereich der Robotik eingesetzt [102]. Dieses Framework bietet eine Basis für die Entwicklung hybrider Planer. Planung unter Berücksichtigung vieler verschiedener Arten von Wissen hat jedoch den Nachteil des daraus resultierenden sehr großen hybriden Suchraums, der sich als Kreuzprodukt der einzelnen Suchräume ergibt.

Ein weiteres Ziel dieser Arbeit ist daher die Entwicklung eines hybriden Planungssystems für mobile Roboter, der die Vorteile des ersten prototypischen Gesamtsystems und des Meta-CSP-Ansatzes hinsichtlich der Anforderungen von Roboterumgebungen kombiniert. Umgesetzt und demonstriert wird dies in dem neuen, auf dem Meta-CSP-Framework aufbauenden hierarchischen hybriden Planer CHIMP (Conflict-driven Hierarchical Meta-CSP Planner).

1.1 Wissenschaftlicher Beitrag

Der wissenschaftliche Beitrag dieser Arbeit kann in den folgenden Punkten zusammengefasst werden:

Integration eines gegebenen HTN-Planers in eine Roboterkontrollarchitektur, deren Ziel das Lernen aus Erfahrungen ist. Dies beinhaltet die Modellierung der Planugsdomänenbeschreibung und die Entwicklung eines Domänenformats, das geeignet ist, das Lernen aus Erfahrungen sowie die Planausführung zu unterstützen.

Verbesserung der Robustheit der Planausführung durch Verknüpfung des Planers mit einer konsistenzbasierten Planausführungsüberwachung sowie einem System, das Teile des Plans vor der Ausführung simuliert.

Integration von HTN-Planung und Meta-CSP-Inferenzverfahren durch die Implementierung der HTN-Dekompositionsstrategie als Meta-Constraint im Meta-CSP-Framework. Auf diese Weise wird der Meta-CSP-Planung eine mächtige zusätzliche Heuristik gegeben, um deren großen hybriden Suchraum einzuschränken. Dies erlaubt die Anwendung der

[1] `http://metacsp.org`

Meta-CSP-Planung für größere und komplexere Problemstellungen. Gleichzeitig bleiben die Vorzüge von HTN-Planung für mobile Roboter erhalten.

Der hierarchische hybride Planer CHIMP, der kausales, temporales und ressourcenbasiertes Wissen sowie Wissen, das von einem externen Pfadplaner stammt, für die Planung mobiler Roboter nutzt. Durch seine modulare Struktur kann er zudem um zusätzliche Wissensformen und Schlussfolgerungsmechanismen erweitert werden. CHIMP kann Pläne erzeugen, deren Aktionen teilweise parallel ausgeführt werden können. Zudem können während der Planausführung zusätzliche Aufgaben in den bestehenden Plan integriert werden, ohne die Ausführung pausieren zu müssen.

1.2 Abgrenzung zu anderen Projektarbeiten in RACE

Wie bereits angesprochen, ist ein großer Teil dieser Arbeit als Teil des Projektes RACE entstanden. In RACE haben die verschiedenen Partner sehr eng miteinander kooperiert, und ein Fokus wurde auf die Integration der Teilkomponenten gelegt. Dies macht eine strikte Eins-zu-eins-Zuordnung einzelner Komponenten zu Projektmitgliedern schwierig. Die vorliegende Arbeit profitierte stark von dem wertvollen Feedback und den Ideen der Projektpartner. Daher ist eine Abgrenzung des wissenschaftlichen Beitrags zu der Arbeit anderer Projektmitglieder notwendig:

- Martin Günther von der Universität Osnabrück hatte einen großen Anteil an der entstandenen Softwarearchitektur in RACE. Hierbei war er auch an der Entwicklung des auf einem Zustandsautomaten basierenden Ansatzes zur Planausführung beteiligt. Zudem war er unter anderem federführend bei dem Anchoring von Objekten und bei der Implementierung einzelner Roboteraktionen, wie beispielsweise zur Objektmanipulation. Die genannten Verbindungen zu dieser Arbeit finden sich in den Abschnitten 3.3 und 3.4.

- Hybride Planung war einer der Hauptaufgabenbereiche der Projektpartner der Universität Örebro und resultierte in der Implementierung des Meta-CSP-Frameworks, auf das in Kapitel 4 eingegangen wird. Diese Vorarbeiten motivierten die Entwicklung des in der vorliegenden Arbeit präsentierten Planers CHIMP. CHIMP nutzt einige der vom Meta-CSP-Framework bereitgestellten Funktionen als Bausteine. Zu diesen zählen die Implementierung des temporalen Constraint-Netzes und Reasoners für Allens Intervallalgebra, das Ressourcenscheduling, der grundlegende Mechanismus, mehrere Constraint-Netze in übergeordneten Constraint-Netzen zu kombinieren und Meta-Constraints zu definieren, sowie die Backtracking-Suche zur Auflösung von Konflikten der Meta-Constraints. Für die Planausführung wurden auf den bereitgestellten Funktionen für zeitleistenbasierte Planausführung aufgebaut.

- Das RACE-Blackboard, welches als Wissensspeicher dient, sowie die zugrunde liegende OWL-Ontologie, wurden vorrangig an der Universität Hamburg von Lothar Hotz, Bernd Neumann, Stephanie von Riegen und Pascal Rost implementiert (siehe Abschnitt 3.3). Der Teil der Ontologie, der im Rahmen der vorliegenden Arbeit relevant ist, basiert jedoch wiederum auf der Planungsdomänenmodellierung des Autors.

- Štefan Konečný von der Universität Örebro entwickelte in RACE eine konsistenzbasierte Ausführungsüberwachung (Abschnitt 3.4.5), die in Zusammenarbeit mit ihm mit SHOP2 verknüpft wurde. Im Rahmen dieser Kooperation wurde auch das in Abschnitt 3.4.6 beschriebene Lifting entwickelt.
- Das HIRES-Framework, in dem Roboteraktionen vor ihrer Ausführung in einer Simulation getestet und dadurch parametrisiert werden können, stammt von Sebastian Rockel von der Universität Hamburg. Die in Abschnitt 3.4.6 beschriebene Verknüpfung zwischen einem Handlungsplaner, HIRES und einer semantischen Ausführungsüberwachung wurde in Zusammenarbeit mit Sebastian Rockel und Stefan Konecny konzipiert und umgesetzt.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

Kapitel 1 liefert eine Einführung in das Thema der Arbeit. Der wissenschaftliche Beitrag und der Aufbau der Arbeit werden dargelegt. Zudem wird eine Abgrenzung zu eng verknüpften anderen Arbeiten vorgenommen.

Kapitel 2 gibt einen Überblick über das Projekt RACE, in dessen Rahmen der Hauptbeitrag der vorliegenden Arbeit entstanden ist.

Kapitel 3 gibt zunächst eine Einführung in die HTN-Planung. Anschließend wird auf den Einsatz eines HTN-Planers in der in RACE verwendeten Roboterkontrollarchitektur eingegangen. Es folgt eine Beschreibung der Planausführung, die auf einem Zustandsautomaten basiert, und es werden Erweiterungsmöglichkeiten zur Erhöhung der Robustheit aufgezeigt und demonstriert.

Kapitel 4 führt in die hybride Planung mit dem Meta-CSP-Verfahren ein.

Kapitel 5 präsentiert den hierarchischen hybriden Planer CHIMP, der den wesentlichen wissenschaftlich-technischen Beitrag dieser Arbeit darstellt.

Kapitel 6 diskutiert die Eigenschaften von CHIMP anhand dreier Demonstrationsszenarien.

Kapitel 7 fasst die Arbeit in einem Fazit zusammen und gibt einen Ausblick auf mögliche weiterführende Forschung.

1.4 Vorab publizierte Beiträge und Ergebnisse

Verschiedene Beiträge und Ergebnisse sind während der Arbeit an der vorliegenden Dissertation bereits als Vorabversionen publiziert worden. Die folgende Liste gibt hierzu eine Übersicht:

[76, 144] In [144] wurde auf dem *AAAI Spring Symposium 2013* die RACE-Architektur zum Lernen aus Erfahrungen vorgestellt und die Ziele von RACE beschrieben (vgl. Kapitel 2 und Kapitel 3). [76] fasste in der Zeitschrift *KI-Künstliche Intelligenz 28 (2014)* zum Ende

von RACE dessen Ergebnisse zusammen und ging neben der Architektur auch auf hybride Planung (Kapitel 4) und die Verknüpfung von hybrider und HTN-Planung ein (Kapitel 5).

[95, 119, 159, 172] Auf dem *International Symposium on Robotics 2014* wurde die Integration eines HTN-Planers in die RACE-Architektur und die Planausführung mittels eines Zustandsautomaten vorgestellt [159] (vgl. Kapitel 3.4). Hierauf wird bereits kurz in dem RACE-Deliverable D 4.1 eingegangen [119] und in den Deliverables D 1.1, D 1.3 sowie D 1.4 finden sich Informationen zu der Umgebungsmodellierung und der eingesetzten Wissensrepräsentationen [95, 132, 172].

[89, 161] Die Integration von SHOP2 mit einer konsistenzbasierten Ausführungsüberwachung auf Basis eines speziellen Ausführungsmodells (vgl. Kapitel 3.4.5) und das Lifting von Operatoren (vgl. Kapitel 3.4.6) wurden auf dem *AAAI Spring Symposium 2014* präsentiert [89]. Eine Vorabversion findet sich auch in Deliverable D 4.2 [161].

[143, 160] Die Verknüpfung von Planung, konsistenzbasierter Ausführungsüberwachung und der Imagination von Teilen des Plans mit dem HIRES-Framework (vgl. Kapitel 3.4.6) wurde auf der *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)* publiziert [143]. Eine frühere Beschreibung dieses Ansatzes gibt auch das Deliverable D 4.4 [160].

[158, 160, 162, 163] Der Planer CHIMP (vgl. Kapitel 5) wurde auf der *38th German Conference on Artificial Intelligence (KI 2015)* [162] und der *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)* vorgestellt [163]. Neben der Ausführung auf dem realen PR2-Roboter (vgl. Kapitel 6.2.1) wird in [163] auch auf das Erweitern des Plans während der Ausführung eingegangen (vgl. Kapitel 6.2.2). Vorabversionen dieses Ansatzes wurden bereits in den RACE-Deliverables D 4.3 und D 4.4 beschrieben [158, 160].

Kapitel 2

Das Projekt RACE

Ein großer Anteil der vorliegenden Arbeit ist im Rahmen des Projektes RACE (Robustness by Autonomous Competence Enhancement) entstanden. RACE wurde innerhalb des Siebten Forschungsrahmenprogramms (FP7) der Europäischen Kommission gefördert und lief von Dezember 2011 bis November 2014. Die Abschlussbegutachtung fand im Januar 2015 statt.

Das Projektkonsortium bestand aus der Universität Hamburg, der Universität Leeds, der Universität Örebro, der Universität Aveiro, dem Hamburger Informatik Technologie-Center e.V. (HITeC) und der Universität Osnabrück.

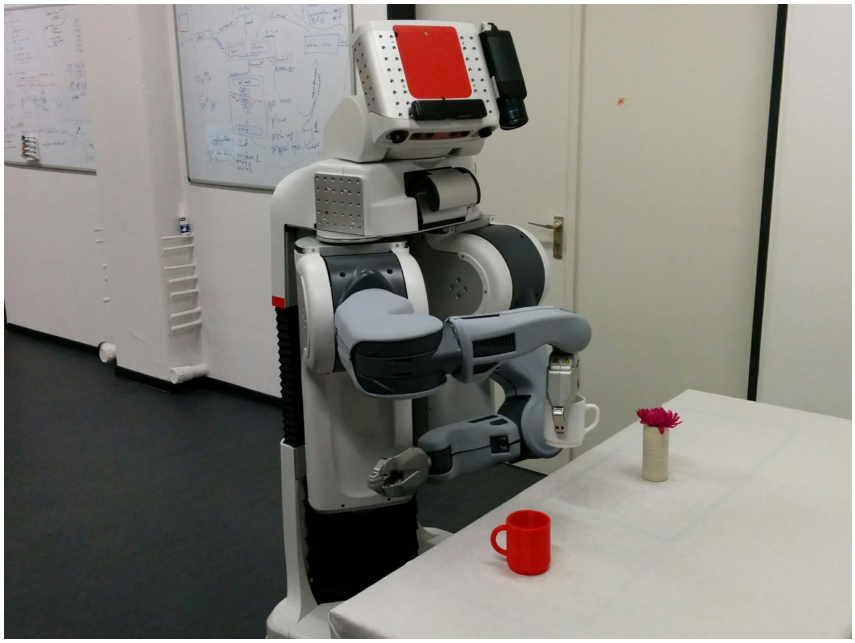


Abbildung 2.1: Der Roboter PR2.

2.1 Projektbeschreibung

Das übergeordnete Ziel von RACE war die Entwicklung eines künstlichen kognitiven Systems in Form eines Serviceroboters, das in der Lage ist, ein Verständnis seiner Umgebung zu erlangen, indem es seine Erfahrungen geeignet abspeichert und nutzt [76]. Basierend auf diesen Erfahrungen sollte der Roboter sein zukünftiges Verhalten verbessern. Um dieses Ziel im Rahmen des Projektes zu erreichen, war Forschung in verschiedenen Themenbereichen sowie deren Integration in ein gemeinsames Robotersystem notwendig. Zu den involvierten Themenfeldern gehörten Wissensrepräsentation und logisches Schließen, Handlungsplanung, planbasierte Robotersteuerung und maschinelles Lernen. Die wissenschaftliche Arbeit innerhalb des Projektes erfolgte in fünf Arbeitspaketen, die im folgenden Überblick kurz beschrieben werden.

WP1: Infrastructure for grounded knowledge representation and reasoning Das erste Arbeitspaket legte den Grundstein für die Wissensrepräsentation und die Inferenzverfahren, die im Rahmen von RACE verwendet wurden. Es beinhaltete die Wahl und Integration unterschiedlicher Wissensrepräsentationsformen und Schlussfolgerungsmethoden, die Modellierung des initialen Roboterwissens sowie die Entwicklung eines allgemeinen Frameworks, das Schlussfolgern unter Verwendung verschiedener Wissensrepräsentationsformen erlaubt.

WP2: Obtaining experiences Das zweite Arbeitspaket beschäftigte sich mit dem Erlangen und der Aufnahme von Erfahrungen über die Umgebung des Roboters und seiner Aktionen. Zu diesen gehören sowohl symbolische als auch subsymbolische Daten. Letztere werden zur Verbesserung der Objekterkennung gespeichert.

WP3: Autonomous creation and adaption of competences Die im vorherigen Arbeitspaket aufgenommenen Erfahrungen wurden hier zum Lernen unterschiedlicher Arten von Wissen verwendet. Neben der angesprochenen Verbesserung der Objekterkennung, zählen hierzu vor allem das Lernen neuer HTN-Methoden, um so direkten Einfluss auf das Roboterverhalten zu erzielen, sowie das Erkennen von Aktivitäten durch das Lernen räumlicher und zeitlicher Relationen zwischen Objekten.

WP4: Planning and executing robot activities by exploiting acquired competences Im vierten Arbeitspaket ist der Hauptbeitrag der vorliegenden Arbeit angesiedelt. Es beschäftigt sich mit der Planung und der Planausführung zur Bereitstellung und unter Verwendung von Erfahrungen. Bereits im ersten Jahr galt es, erste Prototypen entsprechender Module in die Gesamtarchitektur zu integrieren. Ziel war es, bereits zu einem frühen Zeitpunkt erste Erfahrungen zu generieren und für das Lernen bereitzustellen. Im zweiten Jahr waren die verwendeten Repräsentationsformen zu verfeinern und Prototypen durch echte Implementierungen zu ersetzen. Die Vor- und Nachteile des Systems sollten zu Beginn des dritten Jahres analysiert werden. Danach galt es schließlich, einzelne oder mehrere Komponenten zu verbessern oder neu zu implementieren. Letzteres war dabei eine weitestgehend offene Aufgabenstellung.

WP5: Comparative evaluation of competence-enhanced robots Das fünfte Arbeitspaket befasste sich schließlich mit der Bereitstellung einer Simulationsumgebung und der

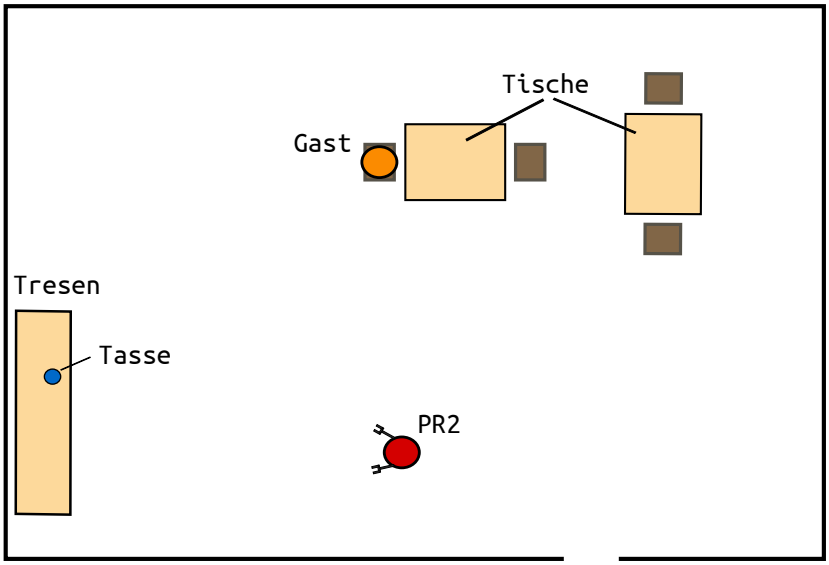


Abbildung 2.2: Schematische Darstellung der Demonstrationsumgebung.

Roboterplattform sowie der Implementierung zusätzlicher Fähigkeiten des Roboters. Zudem galt es, geeignete Kriterien zur Evaluation des Gesamtsystems zu entwickeln.

2.2 Demonstrationszenarien

Als Demonstrationsszenario wurde ein „Kellnerroboter“ in einem Restaurant gewählt. Ein Serviceroboter sollte darin unterschiedliche Aufgaben eines Kellners übernehmen. Zu diesen Beispielaufgaben gehörte das Servieren von Kaffee oder das Abräumen eines Tisches.

An der Universität Hamburg wurde zu Demonstrationszwecken eine beispielhafte kleine Restaurantumgebung aufgebaut. Sie bestand aus zwei Tischen und einem Tresen. Der schematische Aufbau dieser Umgebung wird in Abbildung 2.2 dargestellt, und das Foto in Abbildung 2.3 zeigt den PR2-Roboter darin. Ein typisches Lernszenario in RACE begann zunächst mit einem oder mehreren Durchläufen, in denen dem Roboter relativ detaillierte Anweisungen zum Lernen einer Aufgabe geben wurden. Beispielsweise wurde der Roboter zuerst angewiesen, zum Tresen zu fahren, nachdem er dort angekommen ist wurde ihm gesagt eine bestimmte Tasse zu greifen, anschließend sollte er zu einer bestimmten Seite des Tisches, an dem sich der Gast befindet, fahren und schließlich sollte der die Tasse auf einem vordefinierten Bereich auf dem Tisch platzieren. Nachdem diese Schritte erfolgreich ausgeführt wurden, wurde dem Roboter mitgeteilt, dass diese Abfolge von Aktionen eine `ServeGuest`-Aktivität sei [144]. In einem zweiten Durchlauf sitzt der Gast an einem anderen Platz, dem Roboter werden erneut detaillierte Instruktionen ähnlicher Granularität gegeben, und wiederum wird dem Roboter abschließend mitgeteilt, dass es sich dabei um eine `ServeGuest`-Aktivität gehandelt habe. Aus diesen konkreten Beispielen soll der Roboter anschließend das generelle Konzept `ServeGuest` lernen, sodass ihm im nächsten Schritt

Abbildung 2.3: Die Demonstrationsumgebung in der Universität Hamburg.

in einer wiederum leicht veränderten Startsituation nur noch diese eine Zielaufgabe gegeben zu werden braucht und er dafür einen vollständigen Plan generiert und ausführt. Das Lernen selbst ist allerdings nicht Gegenstand dieser Arbeit, weshalb für Details auf die in RACE entwickelten Ansätze von Neumann et al. [118] sowie Mokhtari et al. [112] verwiesen sei.

2.3 Rahmenbedingungen für diese Arbeit

Durch den Projektplan und das Demonstrationsszenario von RACE wurden bereits einige Rahmenbedingungen vorgegeben, die für die vorliegende Arbeit übernommen wurden. Diese werden im Folgenden vorgestellt.

Als Roboterframework wurde das *Robot Operating System* (ROS) [138] gewählt. ROS ist im aktuellen Forschungsumfeld und insbesondere in Verbindung mit dem PR2 sehr weit verbreitet und steht unter der BSD-Lizenz. Die in dieser Arbeit verwendete Version ist ROS-Fuerte[1].

Von der Universität Hamburg wurde ein PR2-Roboter zur Verfügung gestellt. Der PR2 ist eine Roboterplattform, die von dem Unternehmen Willow Garage hergestellt wurde und vor allem in der Forschung eingesetzt wird. Abbildung 2.1 zeigt einen PR2. Er verfügt über einen omnidirektionalen Antrieb und zwei Arme mit sieben Freiheitsgraden. Als Sensoren verfügt er unter anderem über einen nickenden Laserscanner unterhalb seines Kopfes, einen Laserscanner oberhalb seiner Basis, der zur Navigation eingesetzt wird, eine inertiale Messeinheit sowie eine RGBD-Kamera vom Typ ASUS Xtion Pro Live. Letztere wurde auf dem Kopf des Roboters

[1] `http://wiki.ros.org/fuerte`

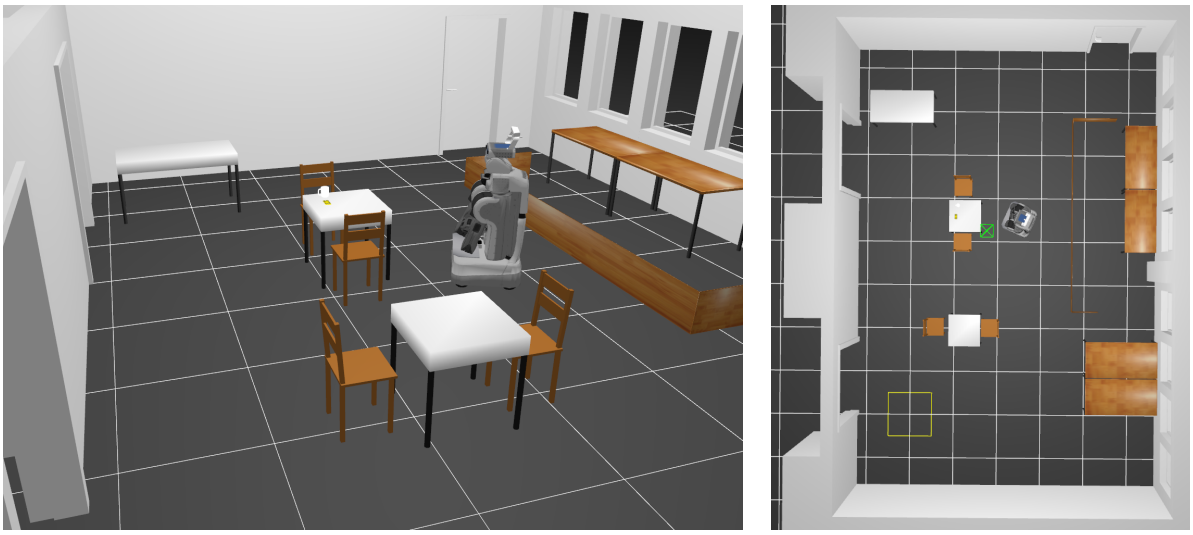


Abbildung 2.4: Nachbildung der Demonstrationsumgebung in Gazebo.

montiert. Zudem war der PR2 lange Zeit die Standard-Entwicklungsplattform für ROS und wird somit sehr gut in ROS Fuerte unterstützt.

Die Demonstrationsumgebung sowie der Roboter befanden sich somit in der Universität Hamburg. Dennoch sollte allen Projektpartnern die Möglichkeit gegeben werden, das Gesamtsystem jederzeit zu testen. Zu diesem Zweck wurde der Robotersimulator Gazebo[2] [88] eingesetzt. Gazebo bietet eine 3D-Simulation für viele Roboterplattformen wie beispielsweise den PR2. Die Demonstrationsumgebung wurde von der Universität Hamburg originalgetreu nachgebildet. Zwei Bildschirmaufnahmen der Demonstrationsumgebung in Gazebo finden sich in Abbildung 2.4.

Im Projektplan von RACE wurde in den Arbeitspaketen, welche die planbasierte Robotersteuerung betreffen, bereits festgelegt, dass in den ersten beiden Projektjahren frei verfügbare Softwaremodule verwendet und integriert werden sollten. Hier wurde bereits konkret die Vorgabe getroffen, zunächst den HTN-Planer SHOP2 [117] beziehungsweise JSHOP2 einzusetzen. Es galt, SHOP2 dabei nicht nur zur reinen Planung zu verwenden, sondern auch zusätzliche Informationen für das Execution Monitoring und die Lernverfahren zu generieren und die Ergebnisse des Lernens wiederum in die Planung einfließen zu lassen. Die Integration von SHOP2 in ein Robotersystem wird daher auch in dieser Arbeit thematisiert und im nachfolgenden Kapitel beschrieben.

Für das dritte Projektjahr war im Projektantrag versprochen, einen völlig neuen Planer zu konzipieren, der auf die Behebung der in den ersten beiden Projektjahren ausgemachten Schwächen von SHOP2 abzielen sollte. Dieser Planer war im Projektplan weder funktional noch konzeptuell genauer beschrieben. CHIMP ist dieser Planer.

[2] `http://gazebosim.org`

Kapitel 3

Verwendung eines HTN-Planers in einem Robotersystem

3.1 Planende Roboter

Einfach ausgedrückt ist das Ziel der Handlungsplanung die Generierung einer Abfolge von Aktionen, mit der ein gewünschtes Ziel erreicht werden kann. Damit war die Planung von Beginn an eng mit einem der generellen Ziele der Künstlichen Intelligenz, dem Verstehen und Erschaffen intelligenter Systeme [145], verbunden. Insbesondere war sie damit auch schon früh mit dem Forschungsgebiet der Robotik verknüpft und führte mit dem Stanford Research Institute Problem Solver (STRIPS) [55] zu einem der ersten Planungssysteme. STRIPS wurde zusammen mit PLANEX [54] als Ausführungskomponente auf dem Roboter SHAKEY [122] eingesetzt. STRIPS und seine gleichnamige Domänenbeschreibungssprache legten den Grundstein für das, was heute unter dem Begriff *klassische Planung* bekannt ist. Ein Rückblick über die Anfänge von SHAKEY, STRIPS und die daraus entstandenen frühen Anfänge hierarchischer Planung am Stanford Research Institute findet sich in [123, Kapitel 12].

Ausgangspunkt der Planung ist ein *Planungsproblem*, welches den aktuellen Zustand, das zu erreichende Ziel sowie die zur Verfügung stehenden Aktionen spezifiziert. Das Ziel wird in der klassischen Planung als Zustand angegeben, den es ausgehend vom initialen Zustand zu erreichen gilt. In anderen Planungsverfahren wie der HTN-Planung können als Ziel jedoch auch Aufgaben angegeben werden, die zu erfüllen sind. Die Zustände werden häufig in einer Sprache der Prädikatenlogik erster Stufe mit endlich vielen Prädikatensymbolen und Konstantensymbolen, aber ohne Funktionensymbole repräsentiert. Ein Zustand ist dann eine Menge variablenfreier Atome. Für die Aktionen, die in der Umgebung durchgeführt werden können, wird dem Planer eine formale Beschreibung in Form von *Operatoren* gegeben. Operatoren bestehen aus Vorbedingungen und Effekten. Vorbedingungen spezifizieren das, was in dem jeweiligen Zustand erfüllt sein muss, um den Operator anwenden zu können. Die Effekte sind hingegen die Änderungen der Fakten, die sich durch Anwendung des Operators beziehungsweise durch das Ausführen der zugehörigen Aktion in dem Zustand ändern.

Häufig wird für Operatoren ein an die STRIPS-Domänenbeschreibungssprache angelehntes Format verwendet, das die Effekte eines Operators in zwei Listen für die negativen und positiven Effekte aufteilt. Von STRIPS sind die Action Description Language (ADL) [134] sowie die heute als Standard von vielen klassischen Planern eingesetzte Planning Domain Definition Language (PDDL) [107] abgeleitet.

Klassische Planung trifft allerdings einige vereinfachende Annahmen, die nicht unbedingt zu den realen Anforderungen von Roboterumgebungen passen. So geht sie davon aus, dass der Zustandsraum der Umgebung endlich, vollständig beobachtbar sowie statisch ist, dass die Aktionen deterministisch sind und keine explizite Dauer besitzen, sondern unmittelbar fertig gestellt werden, und dass es sich bei den resultierenden Pläne um linear geordnete Listen von Aktionen handelt [63].

Planung und Robotik haben sich seit ihren Anfängen etwas von einander entfernt und sich als eigenständige Disziplinen entwickelt. Die Handlungsplanung beschäftigte sich mehr mit der Entwicklung besserer und schnellerer Planungsalgorithmen und setzte auf eine stärkere Formalisierung. Dabei ließ sie jedoch häufig die speziellen Anforderung der Robotik außer Acht. In dieser Hinsicht konnten mit Planungsgraphen [17] und heuristischer Zustandsraumsuche [22] große Geschwindigkeitsfortschritte erzielt werden. Bekannte heuristische Planer sind Fast-Forward (FF) [77] und Fast Downward [73]. Für einen ausführlichen Überblick über das Themengebiet der Handlungsplanung sei auf [63, 64] verwiesen.

Auf der anderen Seite konzentrierte sich die Robotik eher auf die Hardware und tiefer angesiedelte Fähigkeiten von Robotern wie die Bewegungsplanung und die Sensordatenverarbeitung. So ist es auch heute noch häufig der Fall, dass Robotersysteme keinen Handlungsplaner einsetzen, sondern dass das Verhalten des Roboters in einem von Hand kodierten Zustandsautomaten vordefiniert ist. In allen Bereichen der Robotik und vor allem in der Entwicklung von Roboterhardware wurden in den letzten Jahren große Fortschritte erzielt. Eine herausfordernde Aufgabe ist jedoch weiterhin, alle diese Komponenten in ein Gesamtsystem zu integrieren, damit komplexe Aufgaben ausführen zu können und sie so für reale Anwendungen interessant zu machen. Eine Einführung in die Robotik vermitteln Herzberg et al. [75] und ein breiter Überblick über das Themengebiet wird im Springer Handbook of Robotics gegeben [151]. LaValle [94] geht vor allem ausführlicher auf die Bewegungsplanung ein, behandelt aber auch in einem gewissen Maße Planung unter Unsicherheit.

Als extremes Gegenstück zur planbasierten Robotersteuerung entstand in den 1980er Jahren die verhaltensbasierte Robotik [25, 26]. Sie stellte eine Gegenbewegung zu einer starken Fokussierung auf symbolische Repräsentationen und Schlussfolgerungsmechanismen für die Robotersteuerung dar. Stattdessen wurden Methoden entwickelt, die ein komplexes Roboterverhalten auf Basis der reinen Sensordaten erzielen konnten. Dieser Ansatz beschränkt sich jedoch zumeist auf ein rein reaktives Verhalten und ist daher eher für die Navigation als für komplexe, aus einer Vielzahl unterschiedlicher Aktionen bestehende Aufgaben geeignet.

Neben der Planung sind auch die Ausführungsüberwachung und die Kontrollarchitekturen wichtige Themen für die planbasierte Steuerung von Robotern; auf diese wird im Laufe dieses Kapitels noch genauer eingegangen. Darüber hinaus müssen Sensordaten von Objekten in der Umgebung

mit ihren symbolischen Repräsentationen verknüpft und diese Beziehungen aufrecht erhalten werden, um so einen konsistenten Zustand der Umgebung verwalten und für die Planung verwenden zu können. Dieses Problem ist als Objektverankerung bekannt [33].

3.2 HTN-Planung

Parallel zur klassischen Planung hat sich Hierarchial Task Network (HTN)-Planung als ein weiteres wichtiges Planungsverfahren entwickelt. Die Ursprünge der HTN-Planung gehen auf Sacerdotis Planer NOAH [148] und Tates NONLIN [164] zurück. Sacerdoti erkannte die Vorteile partiell geordneter Pläne und stellte fest, dass das Zulassen einer zur Planungszeit nicht linearen Ordnung auch für Pläne, die im fertigen Zustand vollständig geordnet sind, vorteilhaft sein kann. Er führte prozedurale Netze als Graphen ein, deren Knoten Aktionen unterschiedlicher Abstraktionsebenen repräsentieren und die mit anderen Aktionen verbunden sind. NOAH verwendete eine hierarchische Dekompositionsstrategie. Die Knoten des prozeduralen Netzes besaßen Prozeduren, die neue Knoten erzeugen konnten. Auf diese Weise erhielt man während der Planung eine Hierarchie unterschiedlicher Abstraktionsebenen. Dieser Ansatz hierarchischer Planung unter Verwendung einer partiellen Ordnung wurde in dem Planer NONLIN, der von der Generierung prozeduraler Netze im Bereich Operations Research motiviert wurde, erweitert. Eine seiner wichtigen Erweiterungen war die Einführung zusätzlicher Entscheidungspunkte für das Backtracking. Dadurch war NONLIN beispielsweise in der Lage, frühere Entscheidungen zur Auflösung konfligierender Aktionen zu revidieren oder alternative Dekompositionsmethoden zu verwenden, falls eine andere in eine Sackgasse führte. Noch vor NOAH und NONLIN entwickelte Sacerdoti mit ABSTRIPS [147] eine weitere Form hierarchischer Planung, bei der eine Abstraktion auf Merkmalsebene umgesetzt wurde. Vorbedingungen von Operatoren wurden auf mehreren Abstraktionsebenen definiert, und STRIPS wurde mehrmals angewendet, wobei zunächst die Vorbedingungen der niedrigen Abstraktionsebenen ignoriert wurden und der Plan somit schrittweise verfeinert wurde. Im Gegensatz zu NOAH und NONLIN hatten ABSTRIPS-Operatoren jedoch noch alle dieselbe Abstraktionsebene. [74, Kapitel 4 und 5] unterscheidet hierfür zwischen Situationsabstraktion, wie sie ABSTRIPS durchführt, und Operatorabstraktion, wie sie bei der HTN-Planung üblich ist.

Etwas aktuellere hierarchische Planer, die in einer Vielzahl realer Anwendungen eingesetzt worden sind, sind SIPE [175] beziehungsweise SIPE-2 [176] und die NONLIN-Nachfolger O-Plan [34] und O-Plan2 [165]. Da sie sich auf reale Anwendungen konzentriert haben, besitzen sie viele für die Praxis vorteilhafte Eigenschaften. Sie können nichtlineare Pläne erstellen und Ressourcen sowie Constraints in der Planung berücksichtigen. Zudem besitzen sie Ausführungskomponenten, mit denen die Planung eng verzahnt ist, wodurch Pläne während der Ausführung angepasst und repariert werden können. Weiterhin verfügen sie über grafische Benutzerschnittstellen zur Anzeige der Pläne. Der häufig eingesetzte Planer SHOP2 zeichnet sich hingegen vor allem durch seine Geschwindigkeit aus; auf ihn wird in Abschnitt 3.2.2 genauer eingegangen.

Obwohl HTN-Planung zu den am häufigsten in realen Anwendungen eingesetzten Planungsverfahren zählt, fehlte ihr lange Zeit eine theoretische Basis. Diese wurde erst 1994 von Erol,

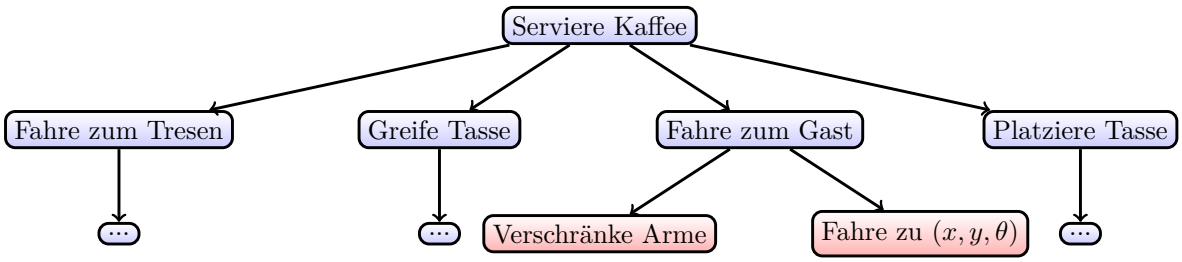


Abbildung 3.1: Beispiel für eine Aufgabenhierarchie für das Servieren von Kaffee. Komplexe Aufgaben sind in blau und primitive Aufgaben sind in rot eingezeichnet.

Hendler und Nau [51] gegeben und gezeigt, dass HTN-Planung ausdrucksstärker als klassische Planung ist. Mit UMCP präsentierten sie zudem den ersten als vollständig und korrekt bewiesenen Planungsalgorithmus für HTN-Planung [52].

Darüber hinaus gibt es Arbeiten, die HTN-Planung mit anderen Planungsverfahren wie beispielsweise mit Partial-Order-Causal-Link (POCL)-Planung und Scheduling integrieren [16, 149], und mit ANML existiert eine ausdrucksstarke Domänenbeschreibungssprache, die im Gegensatz zu PDDL auch die Definition von HTN-Methoden unterstützt [155]. Kapitel 4 befasst sich ausführlich mit der Integration unterschiedlicher Planungs- und Schlussfolgerungsmechanismen.

3.2.1 Formale Beschreibung der HTN-Planung

HTN-Planung basiert allgemein auf der Idee, dass Pläne für zu erfüllende Aufgaben auf unterschiedlichen Abstraktionsebenen repräsentiert werden können. So kann beispielsweise das Servieren eines Kaffees wie in Abbildung 3.1 in die Teilaufgaben „Fahre zum Tresen“, „Greife die gefüllte Tasse“, „Fahre zum Gast“ und „Platziere die Tasse vor dem Gast“ aufgeteilt werden. Diese Teilaufgaben lassen sich wiederum dekomponieren, bis eine unterste Abstraktionsebene erreicht ist, in der die Aufgaben direkt ausgeführt werden können. In der Abbildung ist dies für „Verschränke Arme“ und „Fahre zu (x, y, θ)“ der Fall. Diese Aufgaben unterster Abstraktionsebene werden auch als *primitive Aufgaben* bezeichnet, wohingegen die höher liegenden Aufgaben *komplexe Aufgaben* genannt werden und primär der Planung selbst dienen. Auf diese Weise entsteht eine Aufgabenhierarchie.

Das Ziel der Planung ist in diesem Fall auch nicht wie in der klassischen Planung das Erreichen eines Zielzustandes, sondern das Erfüllen einer oder mehrerer *Zielaufgaben* mittels ausführbarer, primitiver Aufgaben. Für die Planung von primitiven Aufgaben verwendet auch die HTN-Planung Operatoren, die den Weltzustand verändern. Anders als in der klassischen Planung werden zusätzlich zu den Operatoren *Methoden* verwendet, die jeweils eine mögliche Dekomposition einer komplexen Aufgabe in Teilaufgaben beschreiben. Auf diese Weise verwendet die HTN-Planung eine mächtige Heuristik, die den Suchraum der Planung stark einschränken kann, weil der Planer nur solche Dekompositionen durchführen kann, für die entsprechende Methoden vorliegen. Ebenso kann dem Planer auf diese Weise Hintergrundwissen zur Verfügung gestellt werden und es lassen sich formal gültige, jedoch vom Anwender unerwünschte Pläne besser

vermeiden. Beides sind Gründe für die häufige Anwendung dieser Art der Planung für reale Aufgabenstellungen.

Zur Repräsentation dienen dem Planer Aufgabennetze, die aus Aufgaben und Relationen zwischen diesen bestehen. [63] unterscheidet zwischen einfachen Aufgabennetzen (Simple Task Network (STN)) und hierarchischen Aufgabennetzen (Hierarchical Task Network) für die Einführung der HTN-Planung. STN-Planung ist eine Unterkategorie der HTN-Planung.

Ein einfaches Aufgabennetz sei nach [63, Definition 11.2] folgendermaßen definiert:

Definition 3.1. Ein *einfaches Aufgabennetz* ist ein azyklischer gerichteter Graph $w = (U, E)$, der aus einer Menge U von Knoten, die jeweils Aufgaben repräsentieren, und einer Menge von Kanten E besteht. w ist grundinstanziiert, wenn alle Aufgaben in U grundinstanziiert sind, d.h. wenn sie keine Variablen enthalten. w ist *primitiv*, wenn alle Aufgaben in U primitiv sind, und sonst *nicht-primitiv*. Die Kanten in w geben eine partielle Ordnung der Aufgaben in U an: Existiert ein Pfad von u nach v, so gilt $u \prec v$.

In Abschnitt 5.1.6 wird später noch eine Definition der Methoden gegeben, die in dem neu entwickelten Planer CHIMP verwendet werden. Zudem wird die Syntax von Methoden der Planer SHOP2 und CHIMP in Anhang B und Anhang A beschrieben. Zunächst genügt jedoch die folgende allgemeine Definition einer STN-Methode nach [63, Definition 11.4]:

Definition 3.2. Eine *STN-Methode* ist ein 4-Tupel

$$m = (\text{name}(m), \text{aufgabe}(m), \text{vorbed}(m), \text{dekomp}(m)),$$

für das gilt:

- name(m) ist der Name $n(x_1, x_2, \ldots, x_k)$ der Methode. Dabei ist n ein eindeutiges Methoden-Symbol und $x_1, x_2, \ldots, x_k$ sind die Symbole aller in der Methode verwendeten Variablen;
- aufgabe(m) ist eine komplexe Aufgabe, auf die die Methode angewendet werden kann;
- vorbed(m) ist eine Menge von Literalen, die die Vorbedingungen der Methode darstellen;
- dekomp(m) ist ein Aufgabennetz, welches die Teilaufgaben sowie deren Ordnungsrelationen beinhaltet.

Eine Methode ist in einem Zustand s anwendbar, wenn die positiven Vorbedingungen aus vorbed(m) in s enthalten sind, die negativen Vorbedingungen aus vorbed(m) hingegen nicht.

Beide in dieser Arbeit eingesetzten Planer verwenden eine Version beziehungsweise die Grundidee des *Partial-Order-Forward-Decomposition (PFD)*-Algorithmus [63, Kapitel 11.4], welcher daher an dieser Stelle kurz besprochen werden soll und in Listing 3.1 dargestellt ist. Er führt eine vorwärts gerichtete Tiefensuche durch. Aufgrund der Vorwärtssuche ist der jeweilige Zustand in jedem Planungsschritt bekannt, da alle früher im Plan vorkommenden Operatoren bereits angewendet worden sind. Der PFD-Algorithmus ist eine Verallgemeinerung des Total-Order-Forward-Decomposition (TFD)-Algorithmus, der eine vollständige Ordnung der Teilaufgaben einer Methode verlangt.

Algorithmus 3.1 PFD-Algorithmus nach [63, Abbildung 11.9].

1: **Prozedur** PFD(s, w, O, M)
2: **falls** $w = \emptyset$ **dann**
3: **gib zurück** {}
4: **Ende falls**
5: wähle ein $u \in w$, das keine Vorgänger in w hat
6: **falls** t_u ist eine primitive Aufgabe **dann**
7: $aktiv \leftarrow \{(a, \sigma) | a$ ist eine Grundinstanz eines Operators aus O, σ ist eine Ersetzung, für die gilt name$(a) = \sigma(t_u)$ und a ist in s anwendbar$\}$
8: **falls** $aktiv = \emptyset$ **dann**
9: **gib zurück** Fehler
10: **Ende falls**
11: wähle $(a, \sigma) \in aktiv$
12: $\pi \leftarrow \text{PFD}(\gamma(s, a), \sigma(w - \{u\}), O, M)$
13: **falls** $\pi =$ Fehler **dann**
14: **gib zurück** Fehler
15: **sonst**
16: **gib zurück** $a.\pi$
17: **Ende falls**
18: **sonst**
19: $aktiv \leftarrow \{(m, \sigma) | m$ ist eine Grundinstanz einer Methode aus M, σ ist eine Ersetzung, für die gilt name$(m) = \sigma(t_u)$ und m ist in s anwendbar$\}$
20: **falls** $aktiv = \emptyset$ **dann**
21: **gib zurück** Fehler
22: **Ende falls**
23: wähle $(m, \sigma) \in aktiv$
24: wähle ein Aufgabennetz $w' \in \delta(w, u, m, \sigma)$
25: **gib zurück** PFD(s, w', O, M)
26: **Ende falls**
27: **Ende Prozedur**

Die Eingabedaten für PFD sind der initiale Zustand s und ein Aufgabennetz w sowie eine aus Operatoren O und Methoden M bestehende STN-Planungsdomäne. PFD wählt zunächst nichtdeterministisch einen Knoten aus dem Aufgabennetz aus, zu dem keine zu ihm gerichtete Kante existiert und der somit keine Vorgänger besitzt (Zeile 5). t_u sei dabei die von dem Knoten u repräsentierte Aufgabe. Wenn dies eine primitive Aufgabe ist (Zeile 6), wird in der Planungsdomäne nach allen Operatoren gesucht, die mit einer geeigneten Namensersetzung auf die Aufgabe angewendet werden können und deren Vorbedingungen in dem aktuellen Zustand s erfüllt sind. Wenn kein passender Operator existiert, so existiert kein Plan, der die Aufgaben erfüllen würde. Andernfalls wird nichtdeterministisch ein solcher Operator und zugehörige Namensersetzung ausgewählt. Damit kann PFD rekursiv erneut aufgerufen werden (Zeile 12). Hierfür wird ein aktualisierter Zustand übergeben, der sich aus der Anwendung des Operators auf s mittels der Anwendungsfunktion γ ergibt. Zudem wird aus dem zu übergebenden Aufgabennetz die aktu-

elle Aufgabe u entfernt. Wenn dieser rekursive Aufruf für das verbleibende Aufgabennetz einen Plan in Form einer Liste von Operatoren findet, wird der aktuelle Operator a am Anfang dieses Plans eingefügt und der neue Plan zurück gegeben (Zeile 16). Andernfalls wird der Fehler an die aufrufende Instanz weiter gereicht.

Mit komplexen Aufgaben und Methoden wird ab Zeile 18 analog verfahren. Zunächst werden alle Methoden und passende Namensersetzungen ausgewählt, die im aktuellen Zustand s anwendbar sind. Anschließend erfolgt daraus eine nichtdeterministische Auswahl einer solchen Methode m, die mittels Ersetzungsfunktion δ auf das Aufgabennetz angewendet wird. δ entfernt die ursprüngliche Aufgabe u aus dem Aufgabennetz und ersetzt sie durch eine Kopie des Aufgabennetzes aus m. Hierbei ist darauf zu achten, dass die früheren Ordnungsrelationen, die u betreffen, korrekt auf die Aufgaben des einzufügenden Aufgabennetzes übertragen werden [63]. Da hierfür mehrere Alternativen existieren können, gibt δ eine Menge von Aufgabennetzen an, aus der eines nichtdeterministisch ausgewählt wird (Zeile 24). Mit diesem wird der PFD-Algorithmus wiederum rekursiv aufgerufen und das Ergebnis direkt zurück gegeben. Der Zustand s ändert sich durch die Anwendung von Methoden nicht.

Auch wenn der PFD-Algorithmus die Verwendung partiell geordneter Methoden erlaubt, sind die resultierenden Pläne dennoch immer vollständig geordnet. Die Möglichkeit, partiell geordnete Methoden verwenden zu können, bietet jedoch eine größere Freiheit bei der Modellierung der Methoden und damit verbunden häufig auch eine Reduzierung der Methodenanzahl, da die partielle Ordnung mehrere Alternativen erlaubt.

3.2.2 Der Planer SHOP2

Ein weit verbreiteter HTN-Planer ist SHOP2 (Simple Hierarchical Ordered Planner 2) [117]. Er wird als freie Software unter den Lizenzen GPLv2, LGPLv2 sowie MPL1.1 verbreitet, und mit JSHOP2 existiert auch eine Java-Version. Bekanntheit erlangte der Planer durch seine erfolgreiche Teilnahme bei der International Planning Competition im Jahre 2002, bei der er eine Auszeichnung erhielt. Seither wurden SHOP2 und sein Vorgänger SHOP [116] in vielen unterschiedlichen Anwendungsbereichen eingesetzt [115]. Zu diesen gehören die Planung von Evakuierungen, die Einschätzung terroristischer Bedrohungen, die Integration von Softwaresystemen, die Materialauswahl in der Produktionstechnik sowie die automatische Zusammenstellung von Webdiensten.

Einer der Vorzüge von SHOP2 ist der Geschwindigkeitsvorteil im Vergleich zu anderen HTN-Planern. Dies resultiert aus der Verwendung des PFD-Algorithmus und seiner Eigenschaft, die Aufgaben in derselben Reihenfolge zu planen, in der sie später in dem Plan vorkommen und ausgeführt werden [117]. Aufgrund dieser Strategie kann SHOP2 in jedem Planungsschritt den in der Situation erwarteten Weltzustand $\gamma(s, a)$ (Zeile 12) intern verwalten und durch Anwendung der Nachbedingungen der Operatoren fortführen. Dies erleichtert auch das Prüfen der Vorbedingungen von Methoden beziehungsweise Operatoren stark und erlaubt die Verwendung komplexer Ausdrücke in den Vorbedingungen, für welche beispielsweise auch beliebige Ausdrücke in der Programmiersprache Lisp verwendet werden dürfen. Andererseits hat dies allerdings auch zur Folge, dass die generierten Pläne vollständig geordnet sind.

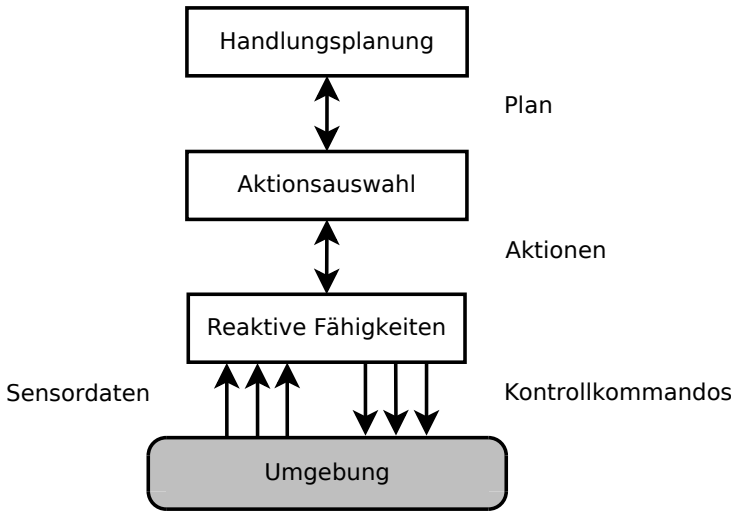


Abbildung 3.2: Die 3-Schichten-Architektur nach [135]. Darstellung nach [75].

Da SHOP2 ausgiebig im Projekt RACE eingesetzt wurde und diese Arbeit einige Beispiele aus der für RACE erstellten Domänenbeschreibung enthält, wird in Anhang B kurz auf die Möglichkeiten der Definition von Operatoren und Methoden für SHOP2 eingegangen.

3.3 Die RACE-Architektur

In RACE wurde eine Vielzahl verschiedener Softwarekomponenten entwickelt. Eine entscheidende Frage war dabei, wie diese Komponenten in ein gemeinsames System integriert werden konnten, sodass sie auf die gewünschte Weise zusammen spielen und ein robustes Robotersystem bilden konnten. Es musste somit eine geeignete Roboterkontrollarchitektur konzipiert und entwickelt werden. Nach [151, Kapitel 8] sei das Konstruieren einer Roboterkontrollarchitektur sogar eher eine Kunst als eine Wissenschaft. Dort wird auch ein Überblick über verschiedene Formen von Architekturen gegeben. Hertzberg et al. [75, Kapitel 9] gehen im Rückblick auf die Debatte zwischen Verfechtern einer sequenziellen Organisation von Modulen wie in der SMPA-Architektur (Sense-Model-Plan-Act) und verhaltensbasierten Architekturen wie der Subsumptions-Architektur ein. Bei der SMPA-Architektur besteht der Kontrollfluss aus einer sequenziellen Abfolge aus Sensordatenerfassung, Umgebungsmodellierung, Planung und Aktionsausführung. Mit letzterer nimmt der Roboter Einfluss auf die Umgebung, welche daraufhin wiederum mit den Sensoren erfasst wird. Durch die potentiell lange Laufzeit der Umgebungsmodellierung und der Planung besitzt das System allerdings einen zeitlich langen Kontrollzyklus. Dies hindert den Roboter daran, schnell reagieren zu können. Diesen Nachteil umgeht die Subsumptionsarchitektur [25] ins andere Extrem. Die Kontrollmodule werden nebenläufig ausgeführt und Ergebnisse bestimmter Module können die Ergebnisse anderer Module überschreiben. Dadurch ist es jedoch schwierig, über mehrere Kontrollzyklen hinweg ein langfristiges komplexes Roboterverhalten zu erzielen.

Als Kompromiss entstand die 3-Schichten-Architektur [135], die die Vorteile beider Ansätze aufgreift. Abbildung 3.2 zeigt eine schematische Darstellung. Sie ist eine hybride Architektur, die sowohl kurze als auch lange Kontrollzyklen beinhaltet. In der obersten Schicht wird von einem Handlungsplaner ein Plan erstellt, der das übergeordnete Verhalten des Roboters zum Erreichen seiner langfristigen Ziele vorgibt. Die mittlere Schicht wählt aus diesem Plan die nächste Aktion aus und instanziiert sie gegebenenfalls. Zudem können hier auch Aktionen des Plans ähnlich wie bei der hierarchischen Planung in primitivere Aktionen dekomponiert werden, um so eine weitere Abstraktionsschicht zwischen der Planung und den primitiven Aktionen des Roboters zu erhalten. Je nach Implementierung können auf der mittleren Schicht auch Ressourcen zugewiesen und ein auf der obersten Schicht noch partiell geordneter Plan linearisiert werden. Darüber hinaus kann diese Schicht auch den Handlungsplaner zu einer Neuplanung veranlassen. Auf diese Weise fungiert sie als Vermittler zwischen der Handlungsplanung und den primitiven reaktiven Kontrollmechanismen des Roboters, die auf der untersten Ebene angesiedelt sind.

Die LAAS-Architektur [1] ist eine weitere 3-Schichten-Architektur. In ihr besteht die oberste Schicht neben dem Planer auch aus einer Ausführungsüberwachung. Diese oberste Schicht kann wiederum aus mehreren internen Schichten bestehen und beispielsweise zwischen Missions- und Aufgabenplanung trennen. Auf diese Weise ist die Planung enger mit der Ausführungsüberwachung verknüpft. Die mittlere Ebene ist hingegen relativ schlank gehalten. Sie ist rein reaktiv und kontrolliert lediglich die Ausführung der auf der untersten Ebene implementierten Funktionen. Die engere Verbindung zwischen der Planung und der Ausführungsüberwachung führt zu mehr Flexibilität in der Adaption des Planes beziehungsweise der Durchführung einer Neuplanung.

Die genannten Architekturschemata betrachten mit dem Kontrollfluss jedoch nur einen Aspekt der Organisation der verschiedenen in einem Robotersystem eingesetzten Module. In RACE mussten auch Möglichkeiten zur Wissensrepräsentation, zur Datenhaltung und zur Integration der lernenden Komponenten berücksichtigt werden. Das Resultat ist eine modulare Architektur, deren übergeordnetes Ziel es ist, eine Verbesserung des Roboterverhaltens auf Basis früherer Erfahrungen zu ermöglichen [144]. Speziell waren einige Nebenbedingungen zu berücksichtigen: Mehrere verschiedene Schlussfolgerungsmodule müssen integriert werden und auf denselben symbolischen Daten arbeiten; alle Wissensformen, die für das Lernen relevant sein können, müssen als *Erfahrungen* gespeichert werden, welche die Eingabedaten für die lernenden Module sind; das gelernte Wissen muss wiederum zurück in die Planungs- und Schlussfolgerungsprozesse gespeist werden. Aus diesen Gründen wurde eine Art Blackboard-Architektur [50, 121] verwendet. Ihre zentrale Komponente ist das *Blackboard*, welches als gemeinsamer Wissensspeicher der anderen Komponenten fungiert. Das Blackboard basiert in RACE auf einer OWL-Ontologie und ist als RDF-Datenbank implementiert. Alle Komponenten schreiben *Ereignisse* dort hinein beziehungsweise bekommen sie als Ergebnis von Anfragen wiederum zurück geliefert. Anstatt direkt miteinander zu kommunizieren, erfolgt der Hauptanteil der Kommunikation zwischen den Komponenten über das Blackboard. Anders als in der klassischen Blackboard-Architektur werden jedoch teilweise auch direkte Kommunikationswege verwendet. Ereignisse sind all jene Dinge, Veränderungen und Aktivitäten, die innerhalb einer bestimmten Zeitspanne in der Umgebung des Roboters auftreten. Dies schließt die Aktivitäten des Roboters, die Beobachtung, dass sich ein Objekt in einem bestimmten Gebiet befindet, die Aktivitäten von Gästen und die Zustände der Körperteile des Roboters wie beispielsweise die Verschränkung des linken Roboterarmes ein.

Alle Ereignisse verfügen über Zeitstempel für ihre Start- und Endzeitpunkte. Da das Blackboard alle diese Ereignisse speichert, beinhaltet es sowohl den aktuellen als auch vergangene Weltzustände aus Sicht des Roboters und stellt diese Informationen den anderen Modulen zur Verfügung. Wie im Folgenden noch näher erläutert wird, handelt es sich aus Sicht des Kontrollflusses allerdings wiederum auch um eine 3-Schichten-Architektur mit einem HTN-Planer in der obersten, einer schlanken Aktionsauswahlkomponenten in Form eines Zustandsautomaten in der mittleren Schicht und schließlich auf unterster Ebene den als ROS-Actions implementierten Roboterfähigkeiten sowie mehrere Komponenten, die Sensordaten verarbeiten.

Eine Mischung aus einer Blackboard-Architektur und einer 3-Schichten-Architektur wird auch in dem CogX Dora-System umgesetzt [69]. Es verwendet das CAST-Framework (Cosy Architecture Schema Toolkit) [71], welches Werkzeuge zur Entwicklung eines verteilten Blackboard-System mit mehreren Wissensrepräsentationsebenen durch Kombination verschiedener Unterarchitekturen bereitstellt. Die mittlere Schicht dient hier jedoch hauptsächlich der Bereitstellung einer konsistenten Umgebungsrepräsentation für die darüber liegende Planungsschicht. Dies geschieht auf Basis von Informationen der untersten Ebene, deren Komponenten ihr Wissen ebenfalls in Form eines weiteren Blackboards teilen. In RACE ist das Blackboard hingegen parallel zu den obersten beiden Ebenen angeordnet und die Komponenten auf unterster Ebene tauschen Informationen über Datenströme in Form von ROS-Topics aus. Der Grund für diese Unterschiede liegt unter anderem in den unterschiedlichen Zielen der Systeme. Während eine der Hauptaufgaben von Dora die Exploration der Umgebung und das Füllen von Lücken in dessen Wissen ist, dient das Blackboard in RACE auch dem Sammeln von Informationen, die für die lernenden Komponenten relevant sind.

In diesem Aspekt ähnelt die RACE-Architektur auch KnowRob [166], bei welchem es sich um eine Wissensrepräsentationssystem für Roboter handelt, das verschiedene Wissensrepräsentations- und Schlussfolgerungsmechanismen verbindet und die gesammelten Daten für das Lernen und Beantworten späterer Anfragen speichert. Auch KnowRob verwendet dabei OWL-Ontologien. Während der Fokus in RACE allerdings auf dem Speichern von Informationen für das Lernen und Verbessern zukünftiger Aktionen durch Konzeptualisierung liegt, bietet KnowRob hingegen weitere Mechanismen, um Informationen für die aktuelle Ausführung einer Aktion bereitstellen zu können. Hierzu gehören virtuelle Wissensbasen, deren Daten nicht explizit repräsentiert, sondern bei jeder Anfrage aus den verschieden Informationsquellen generiert werden. Häufige Anwendungsbereiche sind komplexe Manipulationsaufgaben wie das Zubereiten eines Pfannkuchens. Das Speichern von Episoden von Handlungen wird mit openEASE noch weiter in den Fokus gerückt [11]. OpenEASE verwendet KnowRob und bietet dafür eine webbasierte Schnittstelle für Menschen und Roboter um Informationen aufgezeichneter Episoden abzufragen. Es speichert dabei nicht nur symbolische sondern auch kontinuierliche Sensordaten oder Posen und Trajektorien des Roboters und von Objekten, sodass die Handlungen später möglichst genau nachvollzogen werden können.

Die Verbindung einer OWL-Ontologie mit einem HTN-Planer setzt auch Hartantos Hybrid Deliberative Layer (HDL) [70] um, welcher die oberste Schicht einer 3-Schichten-Architektur bildet. Anders als in RACE dient die Ontologie und ihre A-Box dort jedoch nicht vorrangig dem gemeinsamen Austausch von Daten zwischen den verschiedenen Komponenten und dem Speichern

für späteres Lernen. Stattdessen wird die Ontologie für eine Vorverarbeitung des Planungsproblems genutzt. Sie beruht auf der Erkenntnis, dass in größeren Domänen einige Elemente der Umgebung für das Planungsproblem irrelevant sind und lediglich die Laufzeit der Planung erhöhen. HDL repräsentiert zusätzlich zur Umgebung auch die Planungsdomäne in einer Ontologie und generiert mit Hilfe von OWL-Schlussfolgerungsmechanismen eine gefilterte Domänen- und Problembeschreibung für den Planer.

Eine mögliche alternative Vorgehensweise zum Blackboardsystem, die auch in RACE diskutiert wurde, wäre die ausschließliche Verwendung von Datenströmen anstelle eines Datenspeichers wie dem Blackboard gewesen. Eine auf Datenströmen basierende Kommunikation und Organisation wurde unter anderem von DyKnow [72] umgesetzt. Zudem wird ein solches Publish-Subscribe-Muster häufig auch in ROS verwendet und mittels ROS-Topics realisiert [138]. Anstatt alle Ereignisse zum Blackboard zu senden, würden die Komponenten ihre Informationen auf speziellen ROS-Topics veröffentlichen, welche wiederum von anderen Komponenten abonniert werden können. Dieses Vorgehen ist geeignet, wenn die einzelnen Komponenten nur über wenige spezielle Arten von Informationen auf dem Laufenden gehalten werden müssen und die Kommunikation kurzfristig und ohne großen Overhead vonstatten gehen sollte. Ein Beispiel hierfür ist die Übertragung von Sensordaten von einem Laserscanner an eine Objekterkennungskomponente. Dennoch zeigt DyKnow, dass auch mit Datenströmen die Repräsentation von Daten auf mehreren Abstraktionsebenen möglich ist. Daten aus verschiedenen Quellen werden dabei von bestimmten Komponenten fusioniert und an Komponenten höherer Abstraktionsebene gesendet. Hierzu benötigt DyKnow allerdings mit KPL (Knowledge Processing Language) eine dedizierte Beschreibungssprache für die Spezifikation der Datenströme und Prozesse. Bei der einfachen Verwendung von ROS-Topics können jedoch auch einzelne Nachrichten verloren gehen. Für das Beispiel des Laserscanners hätte der Verlust einer einzelnen Nachricht meistens keine großen Auswirkungen, da die Nachrichten mit einer sehr hohen Frequenz gesendet werden. Bei symbolischen Daten über den Zustand des Roboters und seiner Umgebung könnte ein Datenverlust jedoch schnell zu einem unerwünschten Roboterverhalten oder fehlerhaften Lernergebnissen führen. Die Verwendung eines Blackboards hat hingegen den Vorteil, dass nicht jede Komponente den aktuellen Weltzustand vorhalten und aktualisieren muss, sondern die von ihr benötigten Informationen gezielt abfragen kann. Inkonsistenzen zwischen den Zuständen unterschiedlicher Komponenten werden so vermieden. Zudem können einzelne Komponenten so auch zu einem späteren Zeitpunkt gestartet werden, da keine Gefahr besteht, wichtige Informationen zu verpassen.

Anders als bei einer klassischen Blackboard-Architektur können manche Module jedoch auch direkt untereinander kommunizieren. Abbildung 3.3 zeigt ein Diagramm der RACE-Architektur. Die Sensoren und Controller des Roboters senden kontinuierliche Daten per ROS-Topics zu den Komponenten, die die symbolische Propriozeption und die symbolische Perzeption übernehmen. Basierend auf den kontinuierlichen Daten erzeugen diese Komponenten diskrete Daten in Form von Ereignissen und senden diese über einen ROS-Service an das Blackboard. Ereignisse sind hierbei Instanzen von OWL-Klassen, die in der OWL-Ontologie definiert sind und als Basis des Blackboards dienen. Beispielsweise überwacht eine Komponente den Torso des Roboters. In der OWL-Ontologie sind die Torsopositionen mit den Klassen `TorsoUpPosture`, `TorsoMiddlePosture` und `TorsoDownPosture` modelliert. Intern verfügt die Komponente über

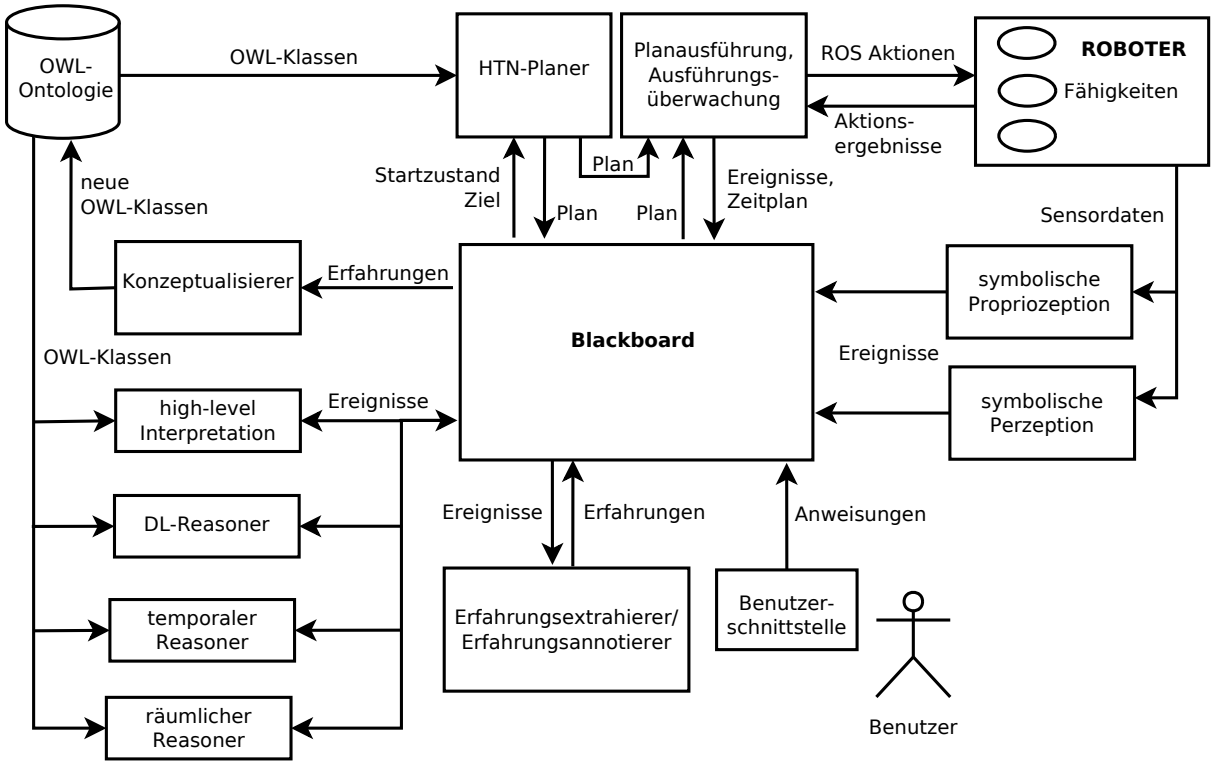


Abbildung 3.3: Überblick über die RACE-Architektur nach [144].

eine Funktion, die den vom Torsocontroller gelieferten kontinuierlichen Positionswert auf eine dieser Klassen abbildet. Wenn auf diese Weise eine Änderung der diskrete Torsoposition festgestellt wird, sendet die Komponente zwei Ereignisse zum Blackboard: Zum einen wird die Instanz der alten Torsoposition beendet, indem bei dem Ereignis der Zeitstempel für die Endzeit gesetzt und es erneut zum Blackboard gesendet wird. Zum anderen wird eine Instanz des neu erkannten Klasse erstellt und mit Startzeitpunkt versehen in das Blackboard eingefügt. Der Endzeitpunkt wird noch offen gelassen. Somit ist der aktuelle Weltzustand des Roboters definiert als alle sich im Blackboard befindenden Ereignisse, deren Endzeitpunkt nicht gesetzt ist.

Für die Kommunikation zwischen dem Blackboard und den anderen Komponenten werden *Fluenten* als gemeinsames Datenaustauschformat für die Übertragung von Ereignissen verwendet. Sie repräsentieren Instanzen von Klassen der OWL-Ontologie zusammen mit Zeitstempeln für die Start- und Endzeiten. Hierfür wurde ein entsprechender ROS-Nachrichtentyp für die generelle Repräsentation eines Fluenten erstellt. Die Nachrichtendefinition besteht aus dem Namen und Typen des Fluenten, das heißt den Namen der Instanz und der OWL-Klasse, einer Liste von Eigenschaften sowie zwei flexiblen Zeitpunkten für die Start- und Endzeiten. Bei den Eigenschaften handelt es sich um Objekteigenschaften oder Datentypeigenschaften. Sie bestehen aus einem Rollentyp, dem Typen des Wertes sowie dem Wert selbst. Durch die ausschließliche Verwendung von Fluenten für die Kommunikation mit dem Blackboard dient die OWL-Ontologie als gemeinsame Sprache aller Komponenten.

Wenn neue Fluenten in das Blackboard eingefügt oder bestehende Fluenten aktualisiert werden, ist das häufig wiederum für andere Komponenten relevant. Aus diesem Grund wurden an der Universität Hamburg *Demand-Topics* als eine Funktionalität des Blackboards implementiert, über die eine Benachrichtigung jener Komponenten erfolgen kann [133]. In einer Konfigurationsdatei kann spezifiziert werden, dass neue oder aktualisierte Fluenten, die bestimmte Kriterien erfüllen, auf einem frei wählbaren ROS-Topic veröffentlicht werden. Als Kriterien können bestimmte Klassen und Eigenschaften der relevanten Fluenten spezifiziert werden oder es können vom Fluenten zu erfüllende SPARQL-Anfrage vorgegeben werden. Interessierte Komponenten können das jeweilige ROS-Topic abonnieren und erhalten auf diese Weise sofort die entsprechenden Fluenten zugesandt.

Der *Erfahrungsextrahierer* protokolliert die Existenz von Ereignissen, die potentiell für das Lernen aus Erfahrungen interessant sind. Die Ereignisse werden in *Episoden* organisiert und gespeichert. Die Generierung von Erfahrungen geschieht auf Basis von Kommandos über die *Benutzerschnittstelle*. Der vorgesehene Ablauf sieht derart aus, dass der Benutzer dem Roboter zunächst mitteilt, dass die folgenden Ereignisse Teil einer neuen Erfahrung sein werden. Anschließend kann der Benutzer dem Roboter eine zu erfüllende Aufgabe geben. Beide Instruktionen werden von der Benutzerschnittstelle an das Blackboard und von diesem weiter an den Erfahrungsextrahierer gesendet. Der Erfahrungsextrahierer erstellt daraufhin eine neue Instanz der Klasse `Intent`, welche die Zielaufgabe als Eigenschaft besitzt, und sendet sie als Fluent zum Blackboard. Mittels Demand-Topic wird der Fluent zum *HTN-Planer* weiter gereicht. Daraufhin fragt der Planer die für ihn relevanten Ereignisse vom Blackboard ab und generiert so den initialen Zustand für die Planung. Details hierzu werden in Abschnitt 3.4 gegeben. Für dieses Planungsproblem generiert er einen Plan und sendet diesen an das Blackboard oder direkt zur *Planausführungskomponente*. Diese übernimmt zwei Aufgaben: Neben dem reinen Ausführen der Aktionen mit den Fähigkeiten des Roboters überwacht sie auch die erfolgreiche Ausführung basierend auf den geplanten Vorbedingungen und Effekten der Aktionen. Die erfolgreiche oder fehlerhafte Ausführung jeder Aktion wird zusammen mit ihren Start- und Endzeitpunkten als Fluenten in das Blackboard eingetragen. Falls ein Fehler auftritt, wird die Ausführung des Gesamtplans abgebrochen und gegebenenfalls ein neuer `Intent` mit derselben Zielaufgabe generiert und auf diese Weise eine erneute Planung angestoßen. Der Planer würde in diesem Fall den aktualisierten Weltzustand abfragen. Alternativ kann die Ausführung vom Benutzer auch zu jeder Zeit über ein Abbruch-Kommando gestoppt werden.

Damit der Roboter sein Verhalten auf Basis seiner Erfahrungen verbessern kann, werden die Methoden und Operatoren der HTN-Domänenbeschreibung ebenfalls in der Ontologie repräsentiert. Der *Konzeptualisierer* bekommt die Episoden als Eingabedaten und aktualisiert die Ontologie mit seinen gelernten Konzepten neuer Methoden. Anschließend können diese Änderungen wiederum aus der Ontologie in die Domänenbeschreibung des Planers eingespeist werden.

Die hier beschriebene RACE-Architektur hat verschiedene Vor- und Nachteile. So ist die Verwendung einer OWL-Ontologie als Sprachspezifikation, auf deren Basis alle Komponenten Daten in das Blackboard schreiben und wieder auslesen können, attraktiv. Auf diese Weise ist allen Komponenten immer bekannt, welche Eigenschaften eines Fluenten zu erwarten sind oder gesetzt werden müssen. OWL wurde an dieser Stelle auch daher verwendet, da die gesammelten

Daten so direkt als Eingabe für die an der Universität Hamburg entwickelten Lernverfahren dienen und deren gelernten OWL-Klassen wiederum als Basis für ein verbessertes Umgebungs- und Aktionsmodell des Roboters genutzt werden können. Allerdings haben diese Vorteile einen erhöhten Modellierungsaufwand und eine erhöhte Inflexibilität zum Preis. Wenn sich aufgrund von geänderten Anforderung in der Implementierung einer Komponente etwas ändert, so muss auch gleichzeitig immer die Ontologie und möglicherweise auch alle anderen Komponenten, die diese Klasse verwenden, angepasst werden. Der Grundsatz, alle Daten in das Blackboard einzutragen und nur über dieses an andere Komponenten zu senden, erzeugt vor allem dann einen unnötig hohen Aufwand, wenn diese Daten nur für die Kommunikation zweier Komponenten benötigt werden und für die Lernverfahren nicht relevant sind. In diesem Fall eine direkte Kommunikation ohne den Umweg über das Blackboard vorzuziehen. Ein letzter wichtiger Punkt ist die Abhängigkeit aller Komponenten vom Blackboard. Aus softwaretechnischer Sicht ergibt sich daraus die Gefahr, dass sich das System nur als Ganzes verwenden lässt und sich die einzelnen Komponenten nur schwer in andere Systeme einbinden lassen. Dies hängt jedoch von der Implementierung der Komponenten ab und ist daher weniger eine Schwäche der Architektur, sondern vielmehr eine mit ihr verbundene Gefahr, die bei der Implementierung berücksichtigt und möglichst umgangen werden sollte. Dies betrifft in ähnlicher Weise auch die Verwendung von ROS als Roboterframework. In beiden Fällen ist darauf zu achten, dass die eigentliche Funktionalität von den spezifischen Anwendungsdaten und den anderen Komponenten getrennt wird. Das lässt sich durch die Kapselung der Basisfunktionalität in einer vom verwendeten Framework unabhängigen Bibliothek erreichen. Diese Bibliothek kann von einem Wrapper an das Blackboard angebunden und mit spezifischem Anwendungswissen ausgestattet werden. Bei dem Planer CHIMP und dem Meta-CSP-Framework ist diese Trennung gelungen, andere Komponente wie die Planausführungsüberwachung lassen sich jedoch nicht direkt unabhängig vom Blackboard oder ROS wiederverwenden.

Dadurch, dass das Blackboard auf einer OWL-Ontologie basiert, beinhaltet es hauptsächlich symbolische Daten. Als Speicher von Trainingsdaten für die Verbesserung der Objekterkennungsverfahren kann es daher nicht verwendet werden. Aus diesem Grund wurde an der Universität Aveiro eine NoSQL-Datenbank für die Speicherung von RGBD-Sensordaten in das System integriert und zum Lernen und späteren Erkennen neuer Objektkategorien eingesetzt [127].

3.4 SHOP2 in der RACE-Architektur

Nachdem ein Überblick über die RACE-Architektur gegeben wurde, soll jetzt auf die Verwendung von SHOP2 und auf die Planausführungskomponenten im Detail eingegangen werden. Zu Beginn von RACE herrschte bei den Projektpartnern die übereinstimmende Meinung, dass die frühzeitige Integration von Prototypen in das Zielsystem für einen erfolgreichen Projektverlauf essentiell sein werde. Das Ziel war es daher, bereits zum Ende des ersten Projektjahres ein lauffähiges Robotersystem demonstrieren zu können, mit dem Erfahrungen aufgenommen werden und als Eingabedaten für die Lernverfahren dienen können. Das Arbeitspaket, das sich mit planbasierter Robotersteuerung auseinandersetzte, sah die Integration und Modifikation bestehender Komponenten vor, die den Stand der Forschung zielgerichteter planbasierter Robotersteuerung

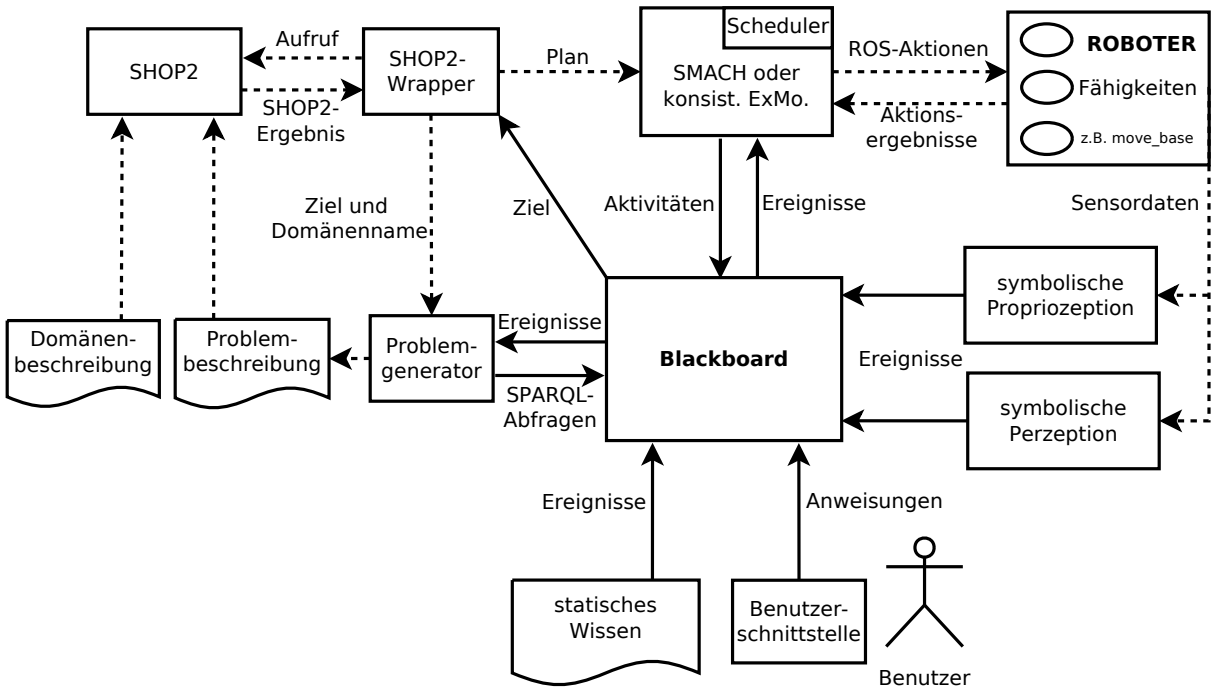


Abbildung 3.4: Detaillierte Sicht auf die für die planbasierte Robotersteuerung beteiligten Komponenten nach Integration des Planers SHOP2. Fluenten sind mit durchgezogenen Linien eingezeichnet und sonstige Aufrufe und Datenübertragungen mit gestrichelten Linien.

widerspiegelten. Dies war für die ersten beiden Projektjahre geplant. Für das dritte Jahr sah die Arbeitsbeschreibung ein weitestgehend offen gehaltenes Teilarbeitspaket vor:

> „In this task, the planning and temporal reasoning methods, formalisms and implementations are evaluated, and possibly modified, changed, or re-implemented in the light of the knowledge gained from the Year-2 Demonstrator. This is an open-ended task in the sense that we cannot plan ahead at project definition time what exactly such changes will be like.“ [139]

Dieser Abschnitt setzt sich zunächst mit der Umsetzung der ersten Aufgabenstellung auseinander, das heißt der Integration von SHOP2 und einer Planausführung, die auf einem Zustandsautomaten basiert. Zunächst wird dafür ein detaillierter Überblick über die an der Planung- und Planausführung beteiligten Komponenten gegeben. Kapitel 5 behandelt schließlich den im Rahmen des finalen Teilarbeitspakets entwickelten Planer CHIMP.

3.4.1 Ein detaillierter Blick auf die Architektur zur planbasierten Robotersteuerung

Während Abbildung 3.3 einen allgemeineren Überblick über die in RACE verwendete Gesamtarchitektur gegeben hat, zeigt Abbildung 3.4 eine detaillierte Ansicht der an der für die Planung

```
!ProblemClass
Class: 'race:Kitchenware'
IncludeSubclasses: True
KeepSubclassNames: True
---
!ProblemClass
Class: 'upper:On'
Properties: ['hasArea', 'hasPhysicalEntity']
IncludeSubclasses: True
KeepSubclassNames: False
---
!ProblemClass
Class: 'race:Arm'
Properties: ['hasArmPosture', 'hasGripper']
IncludeSubclasses: True
KeepSubclassNames: False
```

Listing 3.1: Auszug einer Konfigurationsdatei zur Abfrage der für die Planung relevanten Fluenten aus dem Blackboard.

und Ausführung relevanten Komponenten und insbesondere die Integration von SHOP2 beziehungsweise JSHOP2. (J)SHOP2 wurde ohne Modifikationen als Bibliothek eingesetzt. Für die Planausführung und Ausführungsüberwachung wurden, wie im weiteren Verlauf noch beschrieben wird, SMACH oder eine konsistenzbasierte Ausführungsüberwachung, die in der Abbildung als ExMo abgekürzt wird, eingesetzt. Fluenten werden hier nur für die Kommunikation mit dem Blackboard verwendet, während manche Komponenten auch über eigene ROS-Nachrichtentypen oder direkte Programmaufrufe kommunizieren, was in jener Abbildung mit gestrichelten Pfeilen gekennzeichnet ist.

Zur Anbindung von SHOP2 an die anderen Komponenten wurde ein *Wrapper* als ROS-Node implementiert. Der Wrapper abonniert Demand-Topics, auf welchen das Blackboard neue Ziele für die Planung sendet, und ruft daraufhin den *Problemgenerator* für die Erzeugung einer Problembeschreibungsdatei für SHOP2 auf. Anschließend wird SHOP2 gestartet, dessen Plan geparst und an die Planausführung gereicht.

Der Problemgenerator fragt alle Ereignisse aus dem Blackboard ab, die Teil des aktuellen Weltzustand des Roboters und relevant für die Planung sind. Er transformiert diese Ereignisse in das vom Planer erwartete und zur Domänenbeschreibung passende Repräsentationsformat. Damit der Problemgenerator weiß, welche Fluenten und welche deren Eigenschaften relevant sind, kann ihm dies in einer Konfigurationsdatei im YAML-Format [13] mitgeteilt werden. Listing 3.1 gibt ein Beispiel dreier Einträge einer solchen Konfigurationsdatei. Neue Einträge werden mit dem Schlüsselwort `!ProblemClass` eingeleitet und einer aus drei Bindestrichen bestehenden Zeile voneinander getrennt. Der einzige notwendige Schlüssel ist `Class`, mit dem die OWL-Klasse der relevanten Fluenten angegeben werden muss. Mit `IncludeSubclasses` kann mitgeteilt werden,

```
1 (Table table1)
2 (Arm leftArm_1)
3 (ArmPosture leftArm_1 armUnnamedPostureState)
4 (ManipulationArea mas1 spatialExtent_1 table1)
5 (HasPlacingArea mas1 pawr1)
6 (PlacingArea pawer1 spatialExtent_5 table1)
7 (Coffee mug1)
8 (On mug1 pawer1)
```

Listing 3.2: Beispiel eines ersten Prototyps eines Repräsentationsformats, bei dem kein direkter Zusammenhang zwischen der OWL-Ontologie und dem Format des Weltzustands für SHOP2 bestand.

ob nur die direkten Instanzen jener Klasse oder auch die der Unterklassen zu betrachten sind. Mit den anderen beiden Schlüsseln kann die Art definiert werden, wie die Fluenten in die Problembeschreibung einzutragen sind. Wenn `KeepSubclassNames` auf den Wert `True` gesetzt ist, wird immer der spezifischste Klassenname der OWL-Instanz und andernfalls der bereits unter `Class` angegebene verwendet. Unter `Properties` kann schließlich noch angegeben werden, welche Eigenschaften der Instanz relevant sind und in die Problembeschreibung eingetragen werden müssen. In dem gegebenen Beispiel werden alle Instanzen der Klasse `race:Kitchenware` inklusive ihrer Unterklassen abgefragt. Dies beinhaltet auch die Klassen `race:Mug` und `race:Knive`. Da für den Planer beispielsweise die Information, ob es sich bei einem Objekt um einen Becher oder ein Messer handelt, relevant ist, wird der spezifischste Klassenname in die Problembeschreibung eingetragen. Im nächsten Abschnitt wird das Format beschrieben, in welchem die auf diese Weise ermittelten Informationen in die Problembeschreibung für SHOP2 eingetragen werden.

3.4.2 Format der Zustandsbeschreibung für SHOP2

In den ersten beiden Projektjahre wurde sich ausgiebig mit der Entwicklung einer Domänenbeschreibung für SHOP2 befasst, die den Anforderungen von RACE und den Demonstrationsszenarien genügt. Das Format, in welchem der Weltzustand des Roboters repräsentiert wird, ist ein wichtiger Aspekt davon. Dies betrifft die Repräsentation der Fakten in der Problembeschreibung und in den Vorbedingungen und Effekten der Operatoren beziehungsweise Methoden in der Domänenbeschreibung. Dieses Format ist nicht allein für die Planung, sondern auch für die Planausführungskomponente von Bedeutung, da letztere die geplanten Vorbedingungen und Effekte verarbeiten, vom Blackboard abfragen und überprüfen muss. Die Planausführungskomponente sollte dafür kein zusätzliches handkodiertes, domänenspezifisches Wissen benötigen.

Im Gegensatz zu diesen Anforderungen wurden beispielsweise in einem ersten Prototypen noch die in Listing 3.2 exemplarisch aufgeführten Atome verwendet. Die Modellierung erfolgte hier noch vorrangig aus der Sicht des Planers und wurde von den Informationen, die in den Vorbedingungen der Operatoren und Methoden benötigten wurden, getrieben. Häufig wird dabei wie für `(Table table1)` lediglich die Klasse gefolgt von dem Instanznamen angegeben, und teilweise folgen darauf noch Werte bestimmter Eigenschaften, wie der aktuelle Zustand des Arms.

Andere Atome gaben zwischen zwei Instanzen bestehende Relationen an (Zeilen 5 und 7). Diese Relationen können allerdings in der Ontologie auf zwei verschiedene Arten repräsentiert sein. So ist einerseits `HasPlacingArea` eine Eigenschaft der `ManipulationArea` und andererseits ist `On` eine eigenständige Klasse. Der entscheidende Nachteil dieser Repräsentation ist jedoch, dass es keine direkte Transformationsvorschrift von einer Instanz aus der Ontologie in das Format der Problembeschreibung und umgekehrt gibt. Für jeden einzelnen Eintrag in der Problembeschreibung musste daher im Programmcode beschrieben werden, wie die benötigten Informationen aus dem Blackboard abzufragen und in welcher Reihenfolge sie in die Problembeschreibung zu schreiben waren. Wenn der Roboter aus seinen Erfahrungen neue Planungsmethoden mit zusätzlichen Arten von Vorbedingungen lernt, müsste somit daraufhin auch der Programmcode des Problemgenerators angepasst werden, damit diese neuen benötigten Informationen in die Problembeschreibung geschrieben und vom Planer verwertet werden können. Auf der anderen Seite besteht weiterhin das angesprochene Problem, dass auch die Planausführungsüberwachung die Semantik der Vor- und Nachbedingungen der Operatoren verstehen muss, um diese überprüfen zu können.

Aus diesem Grund wurde ein Format für die Repräsentation von Fakten in der Beschreibung des Weltzustands entworfen, das die oben genannten Kriterien erfüllt, indem es eng an das Format der OWL-Ontologie angelehnt ist. Es werden nur zwei Arten von Atomen verwendet:

1. Instanzen werden in der Form `(Instance` *Konzeptname Instanzname*`)` repräsentiert.
2. Die Eigenschaften jener Instanzen werden mit `(`*Eigenschaftsname Instanzname Wert*`)` aufgeführt.

Listing 3.3 zeigt Beispiele verschiedener Einträge, die sich auf Basis der Konfigurationsbeschreibungen in Listing 3.1 ergeben können. Die Zeilen 1, 2 und 5 listen drei Instanzen der Klassen `Mug`, `On` und `Arm` auf. Zusätzlich werden in der Konfigurationsdatei deren speziell nachgefragten Eigenschaften aufgeführt. Hier werden auch zwei weitere Vorteile der expliziten Auflistung der Eigenschaften ersichtlich. Zum einen macht die Nennung des Namens der Eigenschaft die Semantik des Fluenten deutlicher. Zum anderen kann auf diese Weise feingranular in den Vorbedingungen der Methoden und Operatoren angegeben werden, welche Eigenschaften für deren Anwendung wirklich relevant sind. Beispielsweise ist bei einem Arm häufig dessen Pose, jedoch nicht der Instanzname des zugehörigen Greifers relevant.

```
1 (Instance Mug mug1)
2 (Instance On on4)
3 (hasArea on4 paer1)
4 (hasPhysicalEntity on4 mug1)
5 (Instance Arm rightArm1)
6 (hasArmPosture rightArm1 armTuckedPosture3)
7 (hasGripper rightArm1 rightGripper1)
```

Listing 3.3: Beispiel für die Repräsentation von aus dem Blackboard extrahierten Instanzen in der Problembeschreibung von SHOP.

Wenn eine gelernte Methode neue zusätzliche Vorbedingungen benötigt, muss lediglich die Konfigurationsdatei um einen geeigneten Eintrag ergänzt werden, sodass der Problemgenerator die zusätzlichen Informationen vom Blackboard abfragt und in die Problembeschreibung hinzufügt.

3.4.3 Modellierung der Planungsdomäne für die RACE-Szenarien

Die Modellierung einer HTN-Planungsdomäne für einen mobilen Serviceroboter ist noch immer eine anspruchsvolle Aufgabe. Es gibt eine große Anzahl verschiedener Möglichkeiten der Modellierung und es müssen viele verschiedene Anforderungen berücksichtigt werden. So stellt Ronny Hartanto [70] fest, dass kein allgemein anerkanntes Vorgehen dafür existiere und weist darauf hin, dass mehrere Iterationen der Verfeinerung notwendig seien. Letzteres traf auch besonders im Falle der RACE-Domäne zu, da sie von mehreren anderen Komponenten anderer Projektpartner beeinflusst wurde beziehungsweise wiederum Einfluss auf diese hatte. Darüber hinaus schlägt Hartanto ein vierstufiges Vorgehen vor, bei dem zunächst die Aktionen und Ziele bestimmt, anschließend die Task-Netze beziehungsweise die HTN-Methoden definiert werden, an dritter Stelle die Domänenbeschreibung programmiert und im vierten Schritt schließlich getestet wird. Diese Schritte sind notwendig; jedoch sind manche dieser Punkte noch sehr generell formuliert und bedürfen einer detaillierteren Betrachtung. So ist die Modellierung und Repräsentation der Umgebung und des Roboters ein weiterer Schritt nach der Bestimmung der zur Verfügung stehenden beziehungsweise benötigten Aktionen des Roboters. Darüber hinaus gilt es, die Nebenbedingungen zu identifizieren, die von der Umgebung, den Aufgaben sowie der Roboterplattform auferlegt werden.

Als Randbemerkung sei erwähnt, dass es meist nicht eine alleinige richtige Art der Domänenmodellierung gibt, sondern zumeist mehrere Alternativen existieren, die verschiedene Vor- und Nachteile besitzen. Auch sollte die Domäne in den meisten Fällen, über die gesamte Lebenszeit des Roboters betrachtet, nicht als statisch und endgültig angesehen werden, sondern stattdessen auch immer Ziel möglicher Verbesserungen sein; sei es durch autonomes Lernen oder durch manuelle Anpassungen.

Festlegung der Aufgaben und Roboteraktionen

Der erste Schritt bei der Modellierung einer Planungsdomäne ist die Festlegung geeigneter Aufgaben und Aktionen. Dafür gilt es zunächst, eine geeignete unterste Abstraktionsebene zu wählen, bis zu welcher die Roboteraktionen und die Informationen über die Umgebung zu repräsentieren sind. Dies hängt von den Sensoren und physischen Fähigkeiten des Roboters, den bereits für das Robotersystem zur Verfügung stehenden Modulen zur Wahrnehmung und Manipulation, den Wissensrepräsentations- und Schlussfolgerungsmöglichkeiten des Planers und natürlich von der Aufgabenstellung, die der Roboter in dem speziellen Anwendungsszenario zu erfüllen hat, ab.

Auch wenn SHOP2 auch numerische Variablen repräsentieren und mit ihnen Berechnungen durchführen kann, liegt seine Stärke in der symbolischen beziehungsweise kausalen Planung. Es existieren zwar Verfahren zur Repräsentation von temporalen Informationen wie der Dauer eines

Operators, auf welche in Abschnitt 4.2 kurz eingegangen wird. Anspruchsvolle räumliche oder temporale Inferenz- oder Planungsverfahren, das Scheduling von Ressourcen oder gar Bewegungsplanung bietet SHOP2 jedoch nicht. Aus diesem Grund muss die Anwendungsumgebung diskretisiert und symbolisch modelliert werden. Hierbei sollten folgende Fragen berücksichtigt werden:

1. Welche Art von Aufgaben muss der Roboter durchführen?
2. Welche primitiven Fähigkeiten und Aktionen sind bereits auf dem Roboter implementiert?
3. Sind zusätzliche Aktionen notwendig, um das erwünschte Roboterverhalten zu ermöglichen?
4. Was muss für jede dieser Aktionen erfüllt sein, um sie ausführen zu können?
5. Welche Auswirkungen haben die Ausführungen der Aktionen?

Im ROS-Framework existieren bereits einige Software-Komponenten, die grundlegende Fähigkeiten für den PR2 in Form von ROS-Actions implementieren. Beispielsweise gibt es fertige Komponenten zum Verschränken der Arme, zum Bewegen des Torsos oder für die Navigation zu einer gegebenen Pose in der Umgebung. Diese ROS-Actions wurden als unterste Abstraktionsebene für die primitiven Operatoren des Planers gewählt. Da SHOP2 keine räumlichen Schlussfolgerungsfähigkeiten besitzt, wurde der Operator für die Navigation auf das Fahren in ein gegebenes Gebiet beschränkt. Die folgenden 13 Operatoren wurden für SHOP2 modelliert:

`!tuck_arms(?left_arm_goal ?right_arm_goal)` Mit diesem Operator verschränkt der Roboter seine Arm oder bewegt sie zurück in eine Position seitlich vom Körper. Die Zielpositionen können für jeden Arm einzeln unabhängig voneinander angeben werden.

`!move_arm_to_side(?arm)` Dieser Operator bewegt einen der beiden Roboterarme zur Seite.

`!move_arms_to_carryposture()` Wenn der Roboter Objekte in der Hand hält, kann er seine Arme nicht mit `!tuck_arms` verschränken, da die Objekte mit der Basis des Roboters kollidieren würden. Mit diesem Operator bewegt der Roboter beide Arme in Positionen, mit denen er die Objekte sicher transportieren kann.

`!move_base(?preManArea)` Der Roboter fährt mit den gewöhnlichen, von ROS bereitgestellten Navigationsverfahren in ein bestimmtes Gebiet, das eine Instanz von `PreManipulationArea` sein muss.

`!move_base_blind(?area)` Um Objekte von einem Tisch oder Tresen zu greifen oder darauf zu platzieren, muss der Roboter näher an sie heran fahren. Dies ist mit den gewöhnlichen Navigationsverfahren auf Grund der eingebauten Hindernisvermeidung nicht möglich. Mit diesem Operator kann sich der Roboter auf geradem Wege dichter an das Objekt heran bewegen, ohne dass ihn die Hindernisvermeidung zu früh davon abhält.

`!move_torso(?torsoposture)` Hiermit wird der Torso in eine niedrige, mittlere oder hohe Position bewegt.

!pick_up_object(?object ?arm) Durch Anwendung dieses Operators greift der Roboter ein bestimmtes Objekt mit einem bestimmten Arm.

!place_object(?object ?arm ?placingArea) Mit diesem Operator platziert der Roboter ein Objekt, das er in einem seiner Greifer hält, in ein bestimmtes Gebiet auf einem Tisch oder Tresen.

!place_place_on_tray(?object ?arm ?trayArea) Ähnlich zum vorherigen Operator kann der Roboter hiermit ein Objekt, das er in dem Greifer des Arms `?arm` hält, auf seinem Tablet abstellen.

!pick_from_tray(?object ?arm ?trayArea) Hiermit kann ein Objekt vom Tablett gegriffen werden.

!observe_objects_on_area(?placingArea) Wenn der Roboter vor dem Tisch oder Tresen steht, kann er mit diesem Operator gezielt auf einen bestimmten Bereich des Tisches beziehungsweise Tresens schauen und versuchen, Objekte zu erkennen.

!wait_until_unblocked(?area) Es kann vorkommen, dass bei der Navigation der gewählte Pfad durch einen Gast oder andere Hindernisse blockiert ist. Mit diesem Operator wartet der Roboter bis das Hindernis verschwunden und der Pfad wieder frei befahrbar ist.

!point_head(?point) Mit diesem Operator bewegt der Roboter seinen Kopf und richtet seinen Blick auf einen bestimmten Punkt in der Umgebung.

Zusätzlich zu diesen Operatoren, deren korrespondierende Aktionen physisch auf dem Roboter ausgeführt werden, wurden Operatoren benötigt, die nur für die Planausführungsüberwachung oder den Planer selbst relevant sind:

!replan(?task ?argument1 ?argument2) Mit diesem Operator kann die Planausführung angewiesen werden, eine Neuplanung zu veranlassen. Die Argumente des Operators sind wiederum eine Zielaufgabe, die geplant werden soll, sowie deren Argumente. Dies wird vor allem dann genutzt, wenn zum initialen Planungszeitpunkt nicht genügend Informationen zur Verfügung stehen und die Umgebung erst weiter erkundet werden muss, bevor der restliche Teil des Plans erstellt werden kann.

!imagine(?task ?arg1 ?arg2) Mit diesem Operator wird der Planausführung signalisiert, dass eine bestimmte Aufgabe zuerst in der Simulation getestet werden sollte. Hierauf wird in Abschnitt 3.4.6 genauer eingegangen.

!create_lifted_mug(?mug ?lifted_name) Dieser Operator erzeugt eine neue Platzhalter-Instanz eines Bechers im internen Zustand des Planers. Auf diesen sowie den folgenden Operator wird in Abschnitt 3.4.6 eingegangen.

!lift_mug(?mug ?lifted_name) Mit diesem Operator wird die Instanz eines bestimmten Bechers aus dem internen Planungszustand entfernt und von einer Platzhalter-Instanz eines Bechers ersetzt.

Umgebungsmodellierung

Der PR2-Roboter gibt bereits einige Nebenbedingungen vor, die bei der Modellierung der Umgebung und der Roboteraktionen zu berücksichtigen sind. Die für gewöhnlich mit ROS und dem PR2 verwendeten Navigationsalgorithmen [105, 106] nutzen die Sensordaten der Laserscanner zwar, um intern ein 3D-Voxelgitter als Belegtheitskarte freier, belegter und unbekannter Zellen zu verwalten, die globale und lokale Pfadplanung wird jedoch aus Effizienzgründen nur zweidimensional durchgeführt. Dabei wird die Belegtheitskarte in eine 2D-Kostenkarte projiziert. Sowohl die Berechnung der Kostenkarte als auch die lokale Pfadplanung mittels Dynamic Window Approach [57] verwenden lediglich einen vordefinierten 2D-Grundriss des Roboters. Dieser 2D-Grundriss ist statisch und berücksichtigt nicht die aktuelle Konfiguration der Roboterarme, sondern nimmt an, dass diese verschränkt sind. Wenn sich die Arme beispielsweise an der Seite befinden und über den angenommenen Grundriss hinaus gehen, können diese mit Hindernissen kollidieren. Aus diesem Grund wurde die Vorgabe an den Handlungsplaner gegeben, dass die Arme beim Fahren immer verschränkt sein sollten. Zum Verschränken der Arme existierte ebenfalls bereits eine ROS-Action [110]. Diese sollte jedoch nicht verwendet werden, wenn der Roboter Objekte in seinen Greifern hält. Die Objekte würden bei der Ausführung der Aktion nicht berücksichtigt und mit der Roboterbasis oder dem anderen Arm kollidieren. An der Universität Hamburg wurde daher wurde für das Tragen von Objekten eine alternative Aktion implementiert, die die Arme in eine etwas höhere Position bringt und so für kleine Objekte wie Becher eine Kollision vermeidet. Eine weitere Nebenbedingung ist für das Greifen von Objekten zu beachten: Hierfür sollte sich der Roboter bereits in der Nähe des Objekts befinden, und die Arme müssen beide in einer seitlichen Position sein, damit die Sicht auf das Objekt nicht durch die Arme verdeckt ist und die Arme eine kollisionsfreie Trajektorie ohne die Tischkante als Hindernis haben.

Auf Grund dieser Anforderungen werden die Positionen der Arme durchgängig überwacht, zu einer Unterklasse der Klasse `ArmPosture` diskretisiert und als Fluenten in das Blackboard geschrieben. Auf ähnliche Weise wird mit den Positionen des Torsos als Unterklassen von `TorsoPosture` verfahren. `ArmPosture` und `TorsoPosture` sind zusammen mit `GripperPosture` und `Holding` Unterklassen von `RobotPosture`, dessen Taxonomie in Abbildung 3.5 dargestellt ist. Die Zustände der Greifer sind jedoch für die Planungsdomäne nicht relevant, weil das Öffnen und Schließen der Greifer direkt in den Aktionen zur Objektmanipulation implementiert ist und nicht geplant werden braucht. Stattdessen ist für den Planer nur die Information relevant, ob der Roboter aktuell etwas in seinen Greifern hält. Dies wird mit Fluenten der Klasse `Holding` repräsentiert. Auch für den Torso gibt es Einschränkungen für verschiedene Aktionen. So sollte der Torso beim Fahren unten sein, um den Körperschwerpunkt des Roboters zu senken und so ein Schwanken oder gar Umfallen beim abrupten Bremsen oder Beschleunigen zu vermeiden. Beim Tragen von Objekten darf der Torso jedoch wiederum nicht nach ganz unten bewegt werden, da dies selbst bei einer `CarryPosture` der Arme zu einer Kollision mit der Roboterbasis führen würde. Deshalb ist in diesem Fall eine mittlere Torsoposition zu verwenden. Zur Manipulation von Objekten auf einem Tisch sollte der Torso hingegen nach oben gefahren werden, um dem Roboter einen besseren Blick auf den Tisch zu geben und die Anzahl der möglichen Armtrajektorien, die sich in Kollision mit dem Tisch befinden würden, zu reduzieren.

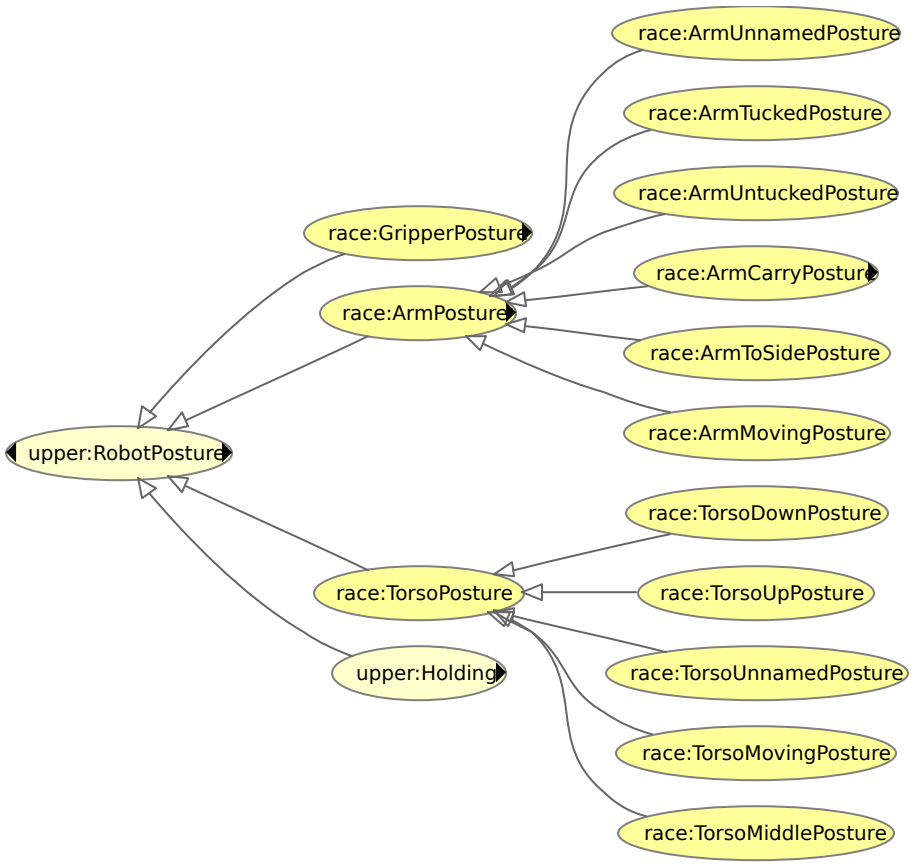


Abbildung 3.5: Taxonomie der Positionen des Torsos und der Arme.

Die bereits vorhandenen Navigationsmechanismen des PR2-Roboters haben noch eine weitere Konsequenzen für die Domänenmodellierung: Damit der Roboter Objekte auf dem Tisch greifen oder platzieren kann, muss er sehr dicht an den Tisch heran fahren. Im Idealfall sollte sich die Basis des Roboters sogar leicht unter dem Tisch befinden. Jedoch würde der Pfadplaner dieses dichte Heranfahren wegen der angesprochenen Projektion seiner dreidimensionalen Umgebungsdaten in eine zweidimensionale Kostenkarte nicht zulassen, um so eine erwartete Kollision zu vermeiden. Daher wurde von Projektpartnern zusätzlich zu den bestehenden Navigationsverfahren eine zweite ROS-Action implementiert, mit der sehr nahe an ein Objekt geradeaus heran gefahren werden kann. Um diese Aktion zu nutzen wurde die Demonstrationsumgebung, wie in Abbildung 3.6 gezeigt, in Bereiche aufgeteilt. Für die Navigation werden in der Ontologie als Bereiche die `PreManipulationArea` und die `ManipulationArea` unterschieden. Instanzen von `PreManipulationArea` können direkt mit dem Standard-Pfadplaner angefahren werden. Diese Bereiche sind weit genug von den Tischen beziehungsweise dem Tresen entfernt, um die Arme kollisionsfrei verschränken zu können. Andererseits sind sie auch so gewählt, dass die jeweils zugehörige `ManipulationArea` mit jener anderen Aktion geradeaus erreicht werden kann. Die `ManipulationAreas` sind wiederum so nahe am Tisch, dass Objekte, die sich auf einer entsprechenden `PlacingArea` auf dem Tisch befinden, mit den Armen erreicht werden können. Eine

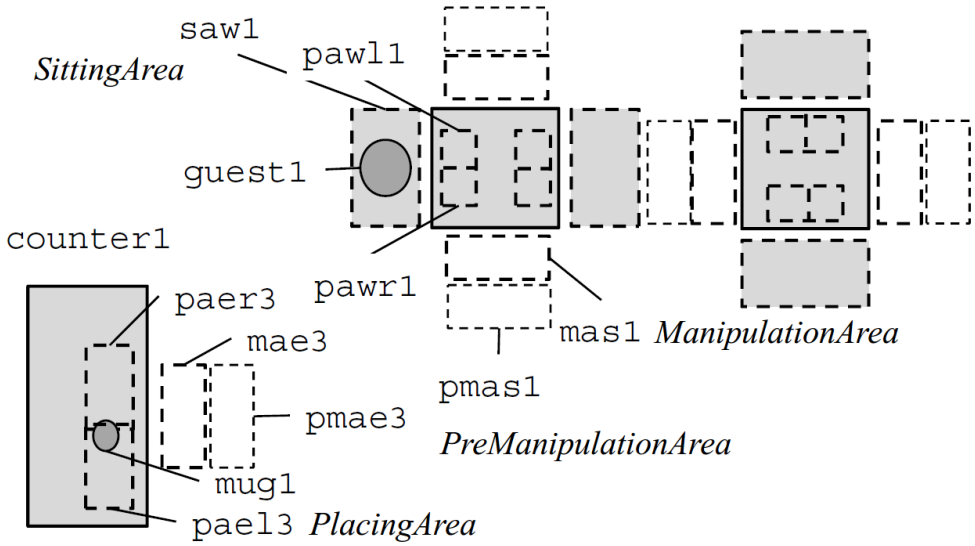


Abbildung 3.6: Repräsentation der Demonstrationsumgebung, die aus zwei Tischen und einem Tresen besteht [159].

`PlacingArea` kann wiederum mit einer `SittingArea` verbunden sein, auf der ein Gast sitzen kann. Wenn sich der Roboter in einer `ManipulationArea` befindet, sollte er nur Objekte greifen oder platzieren. Für andere Aktionen, die eine Armbewegung involvieren, sollte er zunächst zurück in die `PreManipulationArea` fahren.

Verwendung von Fluenten in der Planungsdomäne

Wie bereits erwähnt wurde bei der Domänenmodellierung für SHOP2 darauf geachtet, dass sie mit der Repräsentation in der Ontologie kompatibel ist. Dies spiegelt sich auch in der Art und Weise wider, mit der Vorbedingungen und Effekte der Operatoren repräsentiert sind. Wenn der Roboter eine Aktion ausführt, werden existierende Fluenten beendet und neue erstellt. Führt der Roboter beispielsweise die Aktion zum Verschränken beider Arme aus, werden die Fluenten, die die alten Armpositionen repräsentieren, beendet, neue Instanzen von `ArmTuckedPosture` erstellt und als Fluenten an das Blackboard gesendet. Bei letzteren ist nur die Startzeit gesetzt und die Endzeit noch offen gelassen. Das Beenden eines Fluenten bedeutet, dass dessen Endzeitpunkt gesetzt wird. Neue Fluenten unterscheiden sich zudem durch ihren eindeutigen Namen, der auch in den Operatoren und Methoden verwendet wird, von älteren Fluenten der gleichen Klasse . Dies wurde durch folgende Modellierung der Operatoren erreicht:

1. Fluenten, die durch Anwendung eines Operators beziehungsweise der entsprechenden Aktion beendet werden, werden in den negativen Effekten des Operators aufgelistet. Dies ist eine normale Vorgehensweise, die auch in alternativen Repräsentationsformen gebräuchlich ist.

2. Der Index des aktuellen Planungsschritts wird mit einem *Zähler* zwischengespeichert. Die-

```
1  (:- (inc_cnt ?newValue ?oldValue)
2   (assign ?newValue (1 + '?oldValue))
3  )
4
5  (:- (new_constant ?name ?prefix ?cnt)
6   ((assign ?name (intern (format nil "~A~A" '?prefix '?cnt))))
7  )
8
9  (:- (new_constant ?name ?prefix ?cnt ?postfix)
10  ((assign
11     ?name
12     (intern (format nil "~A~A~A" '?prefix '?cnt '?postfix))
13  ))
14 )
```

Listing 3.4: Axiome für das Inkrementieren des Zählers und die Erzeugung eines neuen Instanznamens.

ser Zähler wird als ganzzahliger Wert i von einem atomaren Ausdruck der Form `(CNT i)` repräsentiert und bei jeder Anwendung eines Operators inkrementiert. Er ist nicht für andere Module sondern nur für den Planer selbst relevant. Das Inkrementieren ist direkt als Teil der Operatordefinitionen implementiert, sodass keine Modifikation von SHOP2 selbst notwendig gewesen ist. Dafür muss lediglich der alte Zählerwert als Teil der Vorbedingungen aus dem Weltzustand abgefragt, mit Hilfe eines Axioms erhöht, anschließend in den Effekten das `CNT`-Atom entfernt und mit neuem Wert wieder hinzugefügt werden. Das Axiom zum Inkrementieren ist in den ersten beiden Zeilen von Listing 3.4 dargestellt.

3. Als positiver Effekt eines Operators werden neue Fluenten erzeugt. Um diese Fluenten von früheren unterscheiden und in das Blackboard eintragen zu können benötigen sie einen eindeutigen neuen Namen. Hier kommt der Zähler in Spiel, indem er als Teil des Namens verwendet wird. Mit dem Axiom `new_constant`, das in Listing 3.4 abgedruckt ist, wird der Name als neue Konstante erzeugt. Er setzt sich aus einem Präfix, welches meist ein der Klassenname des Fluenten ist, dem Zähler sowie einem optionalen Postfix zusammen. Ein Postfix wird verwendet, wenn mehrere Fluenten derselben Klasse benötigt werden.

Als ein Beispiel zeigt Listing 3.5 die vollständige Definition eines Operators für das Bewegen eines Arms in die seitliche Position `ArmToSidePosture`. Die Vorbedingungen stellen zunächst sicher, dass nicht beide Arme verschränkt sind, da die Aktion sonst nicht ausgeführt werden kann. In den Zeilen 11, 13, 18 und 24 wird der alte Zählerstand abgefragt, erhöht und im Zustand des Planers aktualisiert. Zeile 12 erstellt eine Konstante für den Instanznamen der neuen Position. Dies könnte zum Beispiel `armToSidePosture3leftArm1` sein. Diese Instanz wird in den Zeilen 22 und 23 schließlich zum Zustand hinzugefügt und zudem als Wert der `HasArmPosture`-Eigenschaft des jeweiligen Arms gesetzt.

```
1  (:operator (!move_arm_to_side ?arm)
2   ( ; Vorbedingungen
3    (Instance Arm leftArm1)
4    (Instance Arm rightArm1)
5    (not ((HasArmPosture leftArm1 ?oldLeft)
6          (Instance ArmTuckedPosture ?oldLeft)
7          (HasArmPosture rightArm1 ?oldRight)
8          (Instance ArmTuckedPosture ?oldRight)))
9    (HasArmPosture ?whicharm ?oldPosture)
10   (Instance ?oldPostureType ?oldPosture)
11   (CNT ?cnt)
12   (new_constant ?newPosture armToSidePosture ?cnt ?arm)
13   (inc_cnt ?cntNew ?cnt)
14  )
15  ( ; Negative Effekte
16   (HasArmPosture ?arm ?oldPosture)
17   (Instance ?oldPostureType ?oldPosture)
18   (CNT ?cnt)
19  )
20  ( ; Positive Effekte
21   (Instance ArmToSidePosture ?newPosture)
22   (HasArmPosture ?arm ?newPosture)
23   (CNT ?cntNew)
24  )
25 )
```

Listing 3.5: Operator für das Bewegen eines Armes zur Seite.

Modellierung der Aufgaben-Hierarchie

Fast ebenso wichtig wie die Definition der Operatoren ist die Modellierung der Hierarchie von Aufgaben und ihre Repräsentation in Form von Methoden. Der Planer kann nur diejenigen Pläne erstellen, die sich aus der Aufgabenhierarchie ergeben. Wenn eine Umgebungssituation und Zielaufgabe bei der Modellierung nicht bedacht und in einer Methode mit geeignete Vorbedingungen berücksichtigt wurde, kann SHOP2 keinen Plan finden. Daher ist das Testen der Planungsdomäne für viele verschiedene Startzustände essentiell.

Für das Erstellen der Methoden hat sich ein Top-down-Verfahren als vorteilhaft erwiesen, in dem mit den Aufgaben höchster Ebene begonnen wird und für die unterschiedlichen Anfangszustände geeignete Teilaufgaben gewählt werden. Dabei sollte auf eine geeignete Granularität der unterschiedlichen Ebenen der Teilaufgaben geachtet werden. Wenn die Teilaufgaben so gewählt werden, dass Methoden sehr viele Teilaufgaben haben, kann dies bedeuten, dass Methoden stark an eine bestimmte Situation angepasst sind. Somit wären wiederum eine große Anzahl

verschiedener Methoden für dieselbe Aufgabe notwendig, um alle Situationen abzudecken. Die Teilaufgaben sollten stattdessen auch für andere Aufgaben und Situationen relevant sein.

In den Demonstrationsszenarien von RACE waren `serve_coffee_to_guest` für das Servieren eines Kaffees und `clear_table` für das Abräumen eines Tisches die am häufigsten genutzten Zielaufgaben. Beide Aufgaben beinhalten hauptsächlich den Transport eines Objektes. Daher wurde `move_object` als erste Teilaufgabe gewählt, da sie so in Methoden beider übergeordneten Aufgaben verwendet werden konnte und gleichzeitig auch eine weitere sinnvolle Zielaufgabe darstellt, die dem Roboter vom Benutzer gegeben werden kann. Ähnlich zum Planer muss der Domänendesigner diesen Prozess rekursiv durchführen und geeignete Dekompositionen für die neuen Aufgaben identifizieren, ohne dabei das Gesamtbild aus dem Auge zu verlieren. Beispielsweise besteht das Transportieren eines Objekts wiederum aus dem Holen sowie dem anschließenden Platzieren des Objekts in dem gewünschten Bereich der Umgebung. Wenn der Roboter das Objekt hingegen bereits in der Hand hält, kann für das Transportieren eine weitere Methode geschrieben werden, in der das Holen ausgelassen wird. Für das Holen oder Platzieren des Objekts muss der Roboter in den meisten Fällen zu dem Objekt fahren, was wiederum eine geeignete Aufgabe ist, die in vielen verschiedenen Aufgaben verwendet werden kann. Das Ergebnis dieses Prozessen sind Aufgabenhierarchien, die ähnlich zu dem sind, was der HTN-Planer später als Plan generiert. Abbildung 3.7 zeigt ein Beispiel für eine solche Aufgabenhierarchie für das Servieren eines Kaffees. Die Pfeile zeigen darin die Reihenfolge der Teilaufgaben an.

Nachdem die verschiedenen Teilaufgaben gewählt wurden, können die Methoden formal in der SHOP2-Syntax inklusive Vorbedingungen aufgeschrieben werden. Hierbei müssen auch die von der Roboterplattform auferlegten Nebenbedingungen beachtet werden. Im Falle der RACE-Domäne waren von diesen Nebenbedingungen hauptsächlich die Methoden für die Aufgabe `assume_driving_pose` sowie `assume_manipulation_pose` betroffen. Ziel dieser Aufgaben ist es, dass der Roboter eine Pose einnimmt, die zum Fahren oder für die Objektmanipulation geeignet ist. Um alle Eventualitäten abzudecken, war dafür eine vergleichsweise große Anzahl an Methoden notwendig. Eine möglichst modulare Implementierung der Teilaufgaben und zugehörigen Methoden erwies sich als vorteilhaft. So gibt es einzelne Teilaufgaben, die jeweils geeignete Positionen des Torsos und der Arme sicherstellen.

Die vollständige Domänenbeschreibung für SHOP2 für das Restaurantszenario findet sich unter `https://github.com/sebastianstock/race-shop2`.

3.4.4 Planausführung basierend auf einem Zustandsautomaten

Nach der Generierung des Planes erfolgt dessen Ausführung. Zu Beginn von RACE wurde hierfür ein auf einem Zustandsautomaten basierender Ansatz verwendet. Die Grundidee ist es, aus dem Plan, der als Liste von Operatoren vorliegt, einen Zustandsautomaten zu generieren und auf diese Weise auszuführen [159]. Für die Erstellung und Verwendung des Zustandsautomaten wird die Bibliothek SMACH [18, 19] verwendet. Neben der Möglichkeit, hierarchische und nebenläufige Zustände zu modellieren, ist die gute Integration in das ROS-Framework ein großer Vorteil, weshalb diese Bibliothek in verschiedenen Robotikanwendungen eingesetzt wird.

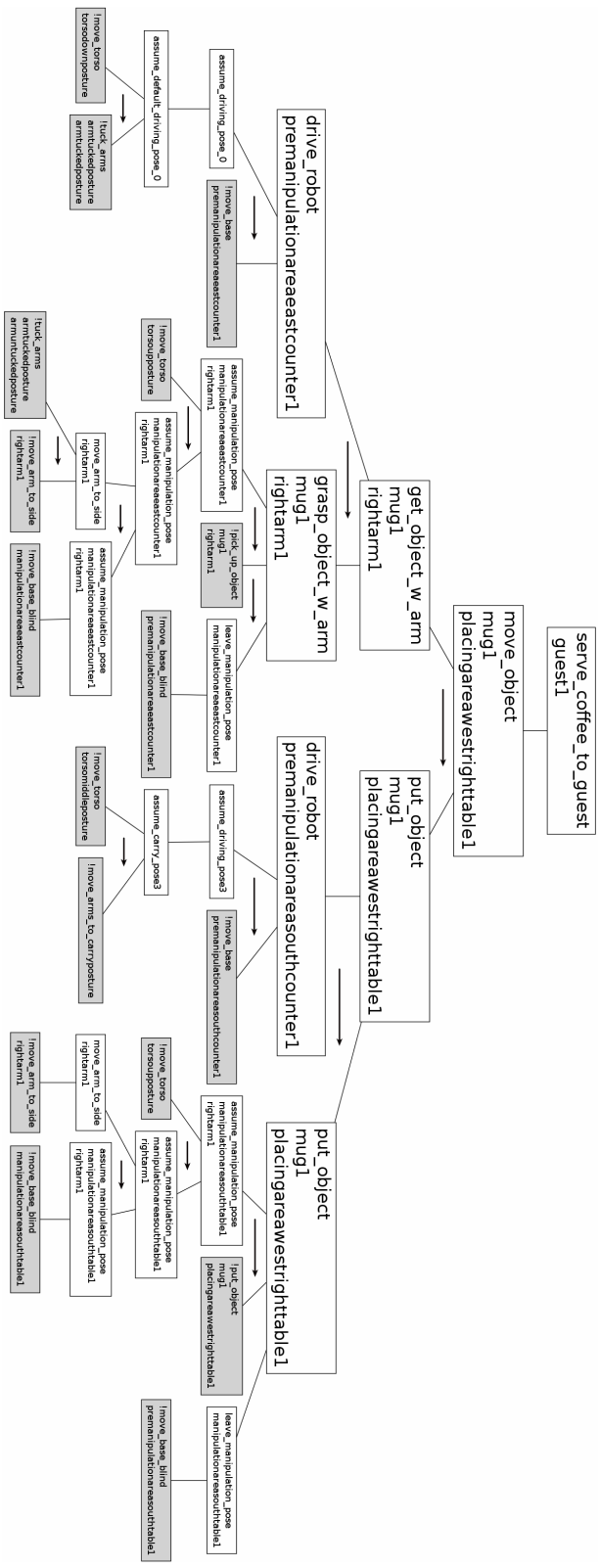


Abbildung 3.7: Beispiel einer Aufgabenhierarchie für das Servieren eines Kaffees.

Für gewöhnlich wird der Zustandsautomat allerdings handkodiert. So wird in [20] eine handkodierter Zustandsautomat für ein Beispielszenario verwendet, in dem ein PR2 Getränke aus einem Kühlschrank holt. BakeBot [21] verwendet SMACH für das Backen von Plätzchen mit einem PR2, wobei der Fokus auf der Integration der tiefer liegenden Bewegungsplanung und der 3D-Sensordatenverarbeitung liegt. ROS Commander [120] verwendet ebenfalls SMACH und gibt dem Benutzer eine grafische Benutzerschnittstelle zur Hand, um hierarchische Zustandsautomaten für unterschiedliche Roboterverhalten aus vorgegebenen Bausteinen zusammen zu bauen. Mit Hilfe von zweidimensionalen Markern können die auf diese Weise implementierten Verhaltensweisen mit Objekten oder Orten verknüpft werden.

Jedoch kann die reine Robotersteuerung mit SMACH für komplexere Anwendungen an seine Grenzen stoßen. Die zusätzliche Verwendung eines Planungssystems bietet mehr Möglichkeiten und Flexibilität, sich an verschiedene Situationen und Anforderungen anzupassen. Die Kombination des SHOP2-Planers und SMACH bietet so die Vorteile beider Verfahren. Diese Idee wird auch in dem Framework Task Compiler [126] umgesetzt, das zeitgleich zu dem hier beschriebenen Verfahren entwickelt wurde. Es verwendet eine PDDL [107] Domänenrepräsentation und FF [77] als Planer und erstellt damit ebenfalls einen Zustandsautomaten. Zudem werden auch lokale und globale Fehlerbehebungszustände eingefügt.

Nach der Planung wird der Zustandsautomat zur Laufzeit generiert. Für jeden Operator des Planes wird in SMACH ein Zustand erstellt und zu dem entsprechenden vorherigen Zustand verbunden. Für jede Art von Operator wurde dafür eine Methode implementiert, die den SMACH-Zustand derart generiert, dass die benötigten Parameter entsprechend gesetzt und bei der Ausführung die zur Aktion gehörigen ROS-Actions aufgerufen werden. Wenn die Aktion erfolgreich ausgeführt wird, geht der Zustandsautomat in den nächsten Zustand über und andernfalls in einen Fehlerzustand, in welchem die Zielaufgabe erneut in das Blackboard eingefügt und dadurch eine Neuplanung veranlasst wird. Eine mögliche Erweiterung für diese Art der Planausführung wäre die Implementierung spezieller Zustände zum Umgang mit Fehlern einzelner Aktionen. Beispielsweise könnte ein Fehlerzustand für das Greifen eines Bechers ein erneutes Greifen versuchen oder andernfalls den Roboter zurück in eine sichere Position mit etwas Abstand vom Tisch fahren lassen.

Eine Limitierung von SHOP2s PFD-Algorithmus ist die Eigenschaft, dass obwohl der Planungsprozess selbst eine partielle Ordnung der Aufgaben beinhalten kann, die resultierenden Pläne immer vollständig geordnet sind. Der PR2-Roboter selbst wäre allerdings in der Lage, bestimmte Aktionen auch parallel auszuführen. Beispielsweise kann der PR2 gleichzeitig seine Arme verschränken und den Torso bewegen. Dies würde in einer kürzeren Gesamtausführungszeit resultieren.

Um diese Aktionen dennoch parallel ausführen zu können wurde an der Universität Hamburg ein zusätzliches Modul implementiert und zwischen die Planung und Generierung des Zustandsautomaten hinzugefügt [48]. Dieses Modul versucht, Teile des Planes zu parallelisieren und anschließend als parallele Zustände in den Zustandsautomaten einzufügen. Die Parallelisierung geschieht auf Basis einer per Hand kodierten Abbildung jeden Operators auf benötigte Ressourcen. Alle Ressourcen besitzen die Kapazität 1. Sie werden verwendet, um einen gerichteten Graphen zu erstellen, indem die Knoten die Operatoren repräsentieren und Kanten zwischen Operatoren

existieren, die die gleiche Ressource verwenden. Die Kanten sind von dem Operator, der früher in dem Plan vorkommt, zu dem späteren gerichtet. Auf diese Weise wird eine partielle Ordnung der Operatoren ermittelt und für die Generierung des hierarchischen Zustandsautomaten benutzt, indem Container-Zustände Aktionen beinhalten, die parallel ausgeführt werden können. In den Demonstrationsszenarien, in denen dieses Verfahren eingesetzt wurden, wurde damit das gewünschte Ergebnis erzielt und die Ausführungszeit reduziert. Ein genereller Nachteil ist jedoch, dass es auf der Annahme beruht, dass jedes Paar von Operatoren, für die in der zusätzlichen Konfigurationsdatei nicht angegeben wurde, dass sie dieselbe Ressource benutzen, prinzipiell parallel ausgeführt werden können. Der Planer könnte jedoch andere Gründe haben, dass die Operatoren in der gegebenen Reihenfolge angeordnet sind. In der SHOP2-Domänenbeschreibung kann für die Teilaufgaben von Methoden beispielsweise explizit angegeben werden, ob die Reihenfolge der Teilaufgaben fest ist oder nicht. Daher wäre ein integrierter Planer wünschenswert, der Handlungsplanung und Scheduling von Ressourcen kombiniert sowie Pläne mit parallel ausführbaren Aktionen generiert. Eine Möglichkeit hierzu wird in den Kapiteln 4 und 5 vorgestellt.

3.4.5 Integration von SHOP2 und konsistenzbasierter Ausführungsüberwachung

Die auf einem Zustandsautomaten basierende Planausführung eignete sich für eine frühzeitige Integration des Planers und der zu Grunde liegenden Roboteraktionen. So konnten bereits im ersten Jahr von RACE einfache Versionen der Demonstrationsszenarien zum Servieren von Kaffee oder Abräumen des Tisches gezeigt und Erfahrungen aufgenommen werden. Es fehlte jedoch die für ein Robotersystem wichtige Ausführungsüberwachung. In SMACH wird die nächste Aktion direkt ausgeführt, sobald die vorhergehende erfolgreich abgeschlossen ist. Um jedoch ein sicheres Roboterverhalten zu gewährleisten, sollten vor dem Ausführen einer Aktion deren Vorbedingungen überprüft werden. Dies ist vor allem in dynamischen Umgebungen wichtig. Ebenso sind auch die erwarteten Effekte einer Operation von Bedeutung, um zu überprüfen, ob sie erfolgreich war und es Sinn ergibt, mit der Ausführung des Planes fortzufahren. So überprüfen manche Aktionen nicht korrekt, ob etwas fehlgeschlagen ist und geben in manchen Fällen selbst dann eine Erfolgsmeldung zurück, wenn der erwartete Effekt nicht eingetreten ist. Beispielsweise kann es beim Greifen eines Objekts passieren, dass dieses bedingt durch einen instabilen Griff durch die Greifer des Roboters rutscht. Die Aktion selbst bemerkt dieses nicht und gibt eine Erfolgsmeldung zurück. Daher ist eine Planausführungsüberwachung notwendig, die überprüft, ob der erwartete Effekt tatsächlich eingetreten ist, sodass andernfalls geeignete Gegenmaßnahmen ergriffen werden können. Teilweise wäre dies auch im Zustandsautomaten möglich, indem die direkten Vorbedingungen und Effekte beim Übergang vom einen zum nächsten Zustand überprüft werden.

PLANEX war eines der ersten Systeme zur Planausführung und Ausführungsüberwachung, das dies erreichte [54]. Es verwendete Dreieckstabellen zur Repräsentation der Abhängigkeiten zwischen Effekten einer Aktion auf andere Aktionen. Pettersson [136] gibt einen ausführlichen Überblick über die Ausführungsüberwachung in der Robotik. Dabei werden zumeist jedoch ausschließlich explizite Effekte von Aktionen überwacht. Dies verlangt, dass die Effekte einer Aktion

direkt beobachtbar sind. Bouguerra et al. [23] verwenden semantisches, in Beschreibungslogiken repräsentiertes Domänenwissen, um implizite Effekte zu inferieren und überwachen zu können. Beispielsweise kann damit die Ausführung der Aktion, in die Küche zu fahren, mit der Frage, ob sich ein Kühlschrank und ein Herd in dem Raum befinden, überprüft werden. Durch die Überprüfung mehrerer impliziter Effekte können Fehler in der Wahrnehmung des Roboters ausgeglichen werden.

Andere Ansätze setzen auf spezielle Sprachen als Austauschformat zwischen Planung und Ausführung beziehungsweise Ausführungsüberwachung. So verwenden Doherty et al. [43] die temporale logikbasierte Sprache TAL (Temporal Action Logic) für die Planung, Ausführung und Ausführungsüberwachung. Darin lassen sich Formeln spezifizieren, deren Gültigkeit bei der Ausführung zu überprüfen sind. Die zu überwachenden Formeln ließen sich in ihrem System zwar auch automatisch generieren, dennoch würden sie hauptsächlich vom Benutzer modelliert, um sicherzustellen, dass diese auch für eine korrekte Ausführung relevant seien. Beispiele für spezielle Ausführungssprachen sind RAP [56], das in der LAAS-Architektur [1] verwendete PRS [60], die C++-Erweiterung TDL [153] und PLEXIL [170], welche in [151, Kapitel 8.3] verglichen werden. Das Ausführungssystem T-REX [108, 109] dient der Ausführung zeitleistenbasierter temporaler Pläne, die mit dem Planer Europa [58, 83] generiert werden. Dazu verwendet T-REX mehrere gekoppelte Kontrollschleifen, die die Teilziele, die Pläne und die Sensorinformationen auf verschiedenen Ebenen repräsentieren, deren Ausführung mit einem Constraint-basierten Ansatz überwachen und gegebenenfalls reparieren. Eine weitere aktuelle Sprache zur Repräsentation von Plänen ist die CRAM Plan Language [10]. Sie wird vor allem zusammen mit dem bereits in Abschnitt 3.3 erwähnten Wissensrepräsentationssystem KnowRob eingesetzt und erlaubt die abstrakte Spezifikation von Handlungs- und Manipulationsplänen, die durch Anfragen an KnowRob instantiiert werden und dabei auch die Adaptionen während der Ausführungszeit und die Einbeziehung von Sensordaten erlaubt.

Des Weiteren kann es auch vorkommen, dass bei der Ausführung einer Aktion indirekte Nebeneffekte auftreten, die die Aktion selbst nicht betreffen und erst zu einem späteren Zeitpunkt von Bedeutung werden. Wenn der Roboter beispielsweise ein Objekt in einem seiner Greifer hält und damit zu einem Tisch fährt, könnte es passieren, dass ihm das Objekt während der Fahrt aus dem Greifer rutscht. Da dies jedoch nicht mit den erwarteten und explizit aufgelisteten positiven oder negativen Effekten des Fahrens in Konflikt steht, würde der Roboter mit der Planausführung fortfahren und erst beim Abstellen auf den Tisch feststellen, dass dies nicht möglich ist.

Von Stefan Konecny wurde in RACE eine konsistenzbasierte Ausführungsüberwachung (Consistency Based Execution Monitoring, CBEM) [89] entwickelt, die die Planausführung mit SMACH ersetzt. Sie basiert auf einer fortlaufenden Konsistenzprüfung des Plans und der realen Daten des Roboters mit Hilfe eines temporalen Constraint-Netzes, in welches der Plan inklusive der Vorbedingungen und Effekte transformiert wird. Dieses kann zudem mit einem speziellen Ausführungsmodell (Execution Model) angereichert werden. Das Ausführungsmodell kann zum einen zusätzliche Informationen über die spezielle Roboterplattform oder die Umgebung enthalten, um so die Planungsdomäne allgemeingültiger modellieren zu können. Zum anderen kann es sich dabei aber auch um Wissensrepräsentationsformen handeln, die vom Planer nicht berücksichtigt

werden. Im Fall von SHOP2 werden durch den Plan nur rein kausale Informationen gegeben. Es fehlen detaillierte temporale Informationen über die Beziehungen zwischen den Vorbedingungen, Aktionen und Effekten. Diese werden deshalb in Form von temporalen Relationen in Allens Intervallalgebra [2] als Ausführungswissen hinzugefügt. Im Vergleich zu Bouguerra et al. wird hier zusätzlich zum kausalen und semantischen somit auch temporales Ausführungswissen eingesetzt.

Während der Planausführung fügt die konsistenzbasierte Ausführungsüberwachung die realen, auf den Sensordaten des Roboters basierenden Informationen in ihr Constraint-Netz ein und verknüpft sie mit den geplanten Elementen. Nach jedem Hinzufügen neuer Informationen wird eine Konsistenzprüfung des temporalen Constraint-Netzes durchgeführt.

In dem temporalen Constraint-Netz werden zwei zusätzliche Fluenten verwendet, die explizit die Vergangenheit und die Zukunft repräsentieren. Die aktuelle Ausführungszeit entspricht dabei dem Endzeitpunkt der Vergangenheit und dem Startzeitpunkt der Zukunft. Alle noch im Plan vorkommenden Fluenten, die noch nicht eingetreten sind, werden zunächst mit der Zukunft verbunden. Bei deren Eintreten wird die Relation entsprechend aktualisiert. Details hierzu werden in der bereits erwähnten Publikation [89] gegeben. Auf diese Weise wird eine im Plan vorkommende Aktion genau dann ausführbar, wenn ihre Vorbedingungen erfüllt sind und ihre Startzeit somit der aktuellen Zeit entspricht. In diesem Fall veranlasst die Ausführungsüberwachung den Start der entsprechenden Aktionen auf dem Roboter, indem sie einen ROS-Service der Ausführungszwischenschicht aufruft. Jene Komponente beinhaltet die Funktionalitäten, die zuvor in den SMACH-Zuständen vorlag und sorgt so für den Aufruf der primitiven Roboterfähigkeiten. Im Gegensatz zur klassischen Ausführungsüberwachung, die primär der Überwachung diente, sorgt die hier verwendete konsistenzbasierte Ausführungsüberwachung also auch für die Ausführung der Aktionen.

3.4.6 Der Umgang mit Unsicherheit

Wie im vorherigen Abschnitt über die Ausführungsüberwachung bereits angeklungen, ist der Umgang mit Unsicherheit ein zentrales Thema in der planbasierten Robotersteuerung, auch wenn dies häufig noch ausgeblendet wird. In realen und dynamischen Umgebungen existieren verschiedene Quellen für Unsicherheiten. In den meisten Fällen liegen dem Roboter keine gesicherten vollständigen Umgebungsinformationen vor. So können Objekte von anderen Akteuren bewegt worden sein, ohne dass der Roboter dies sofort bemerken kann, da er sich an einem anderen Ort der Umgebung aufhält. Zudem kann sich die Umgebung auch ändern, während der Roboter bereits einen Plan ausführt, und diesen so ungültig machen. Und selbst wenn die Umgebung bekannt ist, können auch in der Ausführung einzelner Aktionen Fehler oder unerwartete Effekte auftreten.

In der Künstlichen Intelligenz gibt es eine Vielzahl verschiedener Methoden für Wissensrepräsentations- und Schlussfolgerungsverfahren unter Unsicherheit. An dieser Stelle kann daher nicht im Detail darauf eingegangen werden. Ein Überblick wird in [145, Kapitel 11.3 und Teil IV] gegeben. Unsicherheit ist auch speziell auf allen Ebenen der Robotik ein Thema und spielt beispielsweise auch in der Umgebungswahrnehmung, Lokalisierung und Kartierung eine wichtige

Rolle. Thrun et al. [167] geben eine Einführung in den Einsatz probabilistischer Methoden für diese Anwendungsbereiche. Neben der Verwendung in grundlegenden Roboterfunktionalitäten wie der Lokalisierung wird aber auch auf einem höheren Abstraktionsniveau mit Unsicherheit umgegangen.

In einem Markow-Entscheidungsprozess (MDP) versucht ein Agent, den Nutzen seiner Entscheidungen beziehungsweise Aktionen zu optimieren, wobei die Ausführung der Aktionen nicht deterministisch ist. Dazu wird die Markow-Annahme verwendet, dass die Wahrscheinlichkeit für das Erreichen eines Folgezustands nur von der letzten Aktion und dem letzten Zustand abhängt, nicht jedoch von noch früheren Zuständen und Aktionen. In MDPs ist der aktuelle Zustand stets vollständig bekannt. Eine Erweiterung von MDPs sind partiell beobachtbare Markow-Entscheidungsprobleme, die als POMDPs [84] bezeichnet werden, und in denen der aktuelle Zustand nicht vollständig beobachtbar oder bekannt ist. Mit diesen Eigenschaften erfüllen sie zwei wichtige Anforderungen vieler Anwendungen in der Robotik. Problematisch ist jedoch der sehr große Zustandsraum, wodurch die vollständige Steuerung eines Roboters in größeren realen Anwendungen derzeit nicht möglich ist. Auch wenn Beispiele für die High-Level-Robotersteuerung mit POMDPs in stark strukturierten Anwendungsumgebungen unter Verwendung einer Hierarchisierung mehrerer POMDPs existieren [137], werden sie derzeit eher für Teilaufgaben wie die Navigation [128] oder Objektmanipulation [81, 91] eingesetzt. Ein aktuelles Thema ist auch die Kombination hierarchische Planung, Bewegungsplanung und POMPDs [86].

Manche Planungsansätze versuchen bereits initial einen vollständigen Plan zu erstellen, der mit allen Unsicherheiten des Planungsproblems umgehen kann. Konformante Planung [145, Kapitel 11.3] geht davon aus, dass keine Sensoren zur Verfügung stehen, mit denen die initialen Unsicherheiten über die Umgebung aufgeklärt werden könnten. Der Plan sollte für alle möglichen Ausprägungen des teilweise unbekannten Weltzustands gültig sein. Conformant-FF[79] ist beispielsweise eine Erweiterung des Planers FF [77] für konformante Planung. Kontingenzplanen [145, Kapitel 11.3] berücksichtigt hingegen auch Wahrnehmungen. Für die unterschiedlichen möglichen Ergebnisse von Wahrnehmungen werden dabei alternative Aktionen geplant. Die Pläne enthalten dadurch eine Vielzahl möglicher bedingter Verzweigungen für die jeweiligen Sensoreindrücke. Gerade in realen und komplexen Umgebungen führt dies zu einem sehr großen Suchraum. Beispiele für Kontingenzplaner sind Contingent-FF [78], welches wiederum eine Erweiterung von Conformant-FF beziehungsweise FF ist. Die Vielzahl von Objekten und Quellen von Unsicherheiten machen die oben genannten Ansätze jedoch für reale Robotikanwendungen ungeeignet.

Anstatt einen Plan zu erstellen, der alle Eventualitäten berücksichtigt, bietet sich in der Robotik eine engere Verknüpfung von Planung, Ausführung und Neuplanung beziehungsweise Modifizierung des Planes an. Weser et al. [174] verwenden JSHOP2 für die Steuerung eines Serviceroboters in partiell beobachtbaren Umgebungen. Im Falle fehlender Informationen für die Planung einer Aufgabe wird zunächst ein aus Wahrnehmungsaktionen bestehender Plan erstellt. Nach der Ausführung dieses Plans oder im Falle eines Ausführungsfehlers wird eine Neuplanung durchgeführt, in der die eigentliche Aufgabe mit den neu gewonnenen Informationen geplant wird. Zudem wird ein weiterer Ansatz vorgestellt, in dem der Planer nach der Wahrnehmungsaktion nicht stoppt, sondern mit einem erwarteten Ergebnis dieser Wahrnehmungsaktion fortfährt

und einen vollständigen Plan erstellt. Nur wenn das reale vom erwarteten Ergebnis abweicht, wird erneut geplant. Dieses Vorgehen ist damit sehr ähnlich zu dem im nächsten Unterabschnitt präsentierten Ansatz.

ACogPlan [125] ist hingegen eine Erweiterung von SHOP, die in der Lage ist, in nicht abgeschlossenen Domänen für einen Roboter zu planen. Der Planer berücksichtigt dabei auch solche HTN-Methoden und Operatoren, deren Vorbedingungen mit den bekannten Fakten nicht erfüllbar sind, die aber im Falle weiterer verfügbarer Informationen erfüllt werden könnten. Der Planer würde die Methode somit als möglicherweise anwendbar deklarieren und die betreffende Aufgabe zunächst nicht weiter dekomponieren. Zudem werden wiederum Wahrnehmungsaktionen in den Plan eingefügt, um die notwendigen Informationen zu akquirieren. Andere Teile des Plans werden dann zunächst ausgeführt, und nach der Wahrnehmungsaktion wird Basis der neuen Informationen mit der Planung der zunächst noch ausgelassen komplexen Aufgabe fortgefahren. Dies ist damit ein Beispiel für kontinuierliche Planung [39]. Kontinuierliche Planungssysteme, wie sie in den letzten Jahren vermehrt in der Robotik eingesetzt werden, verschachteln die Planung und Ausführung ineinander. Teile der Planung werden dabei auf einen später Zeitpunkt verschoben, um für sie so die Ergebnisse von Wahrnehmungsaktionen nutzen und den Plan verfeinern zu können [24]. Weitere Beispiele kontinuierlicher Planung in der Robotik sind die Arbeiten von Dornhege und Hertle [46], von Kaelbling und Lozano-Perez [85] sowie von Hofmann et al. [80]. Erstere präsentieren ein integriertes Planungs- und Ausführungssystem für ein Demonstrationsszenario, in dem der Roboter die Aufgabe bekommt, eine Haushaltsumgebung aufzuräumen. Es wird dabei eine Architektur umgesetzt, die eine Schleife aus Planung, einer Überprüfung der Vorbedingungen der nächsten Aktion und anschließender Ausführung der Aktion oder vollständiger einer Neuplanung umsetzt. Kaelbling und Lozano-Perez setzen hingegen nicht auf eine Neuplanung, sondern auf einen hierarchischen Ansatz, in dem abstrakte Teilaufgaben teilweise erst zu einem späteren Zeitpunkt der Ausführung konkreter geplant werden. Dies setzt voraus, dass Aktionen reversibel sind, da sich der Robotern andernfalls in eine Sackgasse manövrieren könnte. Der kürzlich vorgestellte Ansatz von Hofmann et al. ermöglicht kontinuierliche Planung mit der auf dem Situationskalkül basierenden Programmiersprache GOLOG [96] durch die Integration von PDDL-Planern. Auch hier werden für Teile des Plans zunächst wie in [24] Platzhalter verwendet und erst zu einem späteren Zeitpunkt verfeinert, wenn genügend Informationen zur Verfügung stehen.

Viele der genannten Verfahren setzen spezielle Planungsverfahren voraus, die explizit auf den Umgang mit Unsicherheiten ausgelegt sind. Im restlichen Teil dieses Abschnitts sollen stattdessen zwei Verfahren diskutiert werden, die einen herkömmlichen Planer wie SHOP2 verwenden können und dennoch in einem gewissen Maße mit Unsicherheiten in der Umgebung und Varianzen in der Planausführung umgehen zu können. Diese Verfahren wurden als Teil von RACE entwickelt, um die Robustheit des Gesamtsystems zu erhöhen.

Lifting

Planer wie SHOP2 verwenden die Annahme, dass die Umgebung statisch und deren Zustand zum Zeitpunkt der Planung vollständig bekannt ist. Mit diesen Annahmen erstellt SHOP2 einen

Plan, der aus grundinstanziierten Operatoren besteht. Dadurch wird dem Roboter für die Planausführung genau vorgegeben, welche Tasse er mit welchem Arm zu greifen hat. In vielen Fällen ist diese genaue Spezifikation erwünscht und erforderlich, manchmal ist sie jedoch zu restriktiv und überspezifiziert. Wenn es beispielsweise das Ziel ist, dem Gast einen Kaffee zu servieren, ist es irrelevant, welche Instanz eines mit Kaffee gefüllten Bechers verwendet wird. In dem in [89] beschriebenen Demonstrationsszenario startet der Kellnerroboter in der dem Tresen gegenüber liegenden Seite des Raumes und kann daher nicht sehen, welche Objekte sich auf dem Tresen befinden. Somit kann er nur potentiell veraltetes Wissen über die Objekte verwenden, die er bei beim letzten Vorbeifahren auf dem Tresen gesehen hat. Damit generiert SHOP2 einen Plan, der zum Beispiel das Greifen des Bechers `mug1` mit dem rechten Arm beinhaltet. Wenn der Plan anschließend ausgeführt wird, kann `mug1` jedoch bereits zuvor oder während der Ausführung von einem Kellner oder anderen Roboter vom Tresen entfernt worden sein. Wenn der Roboter den Tresen erreicht, würde die Aktion zum Greifen von `mug1` daher selbst dann fehlschlagen, wenn sich ein anderer mit Kaffee gefüllter Becher dort befinden würde, der ebenso verwendet werden könnte.

Die Verbindung des bestehenden Planers mit der Planausführung führt zu einer Strategie der frühen Bindung an Entscheidungen. Zum Umgang mit diesem Nachteil wurde ein als *Lifting* [89] bezeichnetes Verfahren implementiert. Dabei wird während der Planung von einigen spezifischen Instanzen von Objekten abstrahiert, indem sie durch Platzhalter ersetzt werden. Diese Platzhalter werden erst später direkt während der Ausführung der betreffenden Aktion durch reale Instanzen ersetzt. Ein Vorteil diesen Verfahrens ist, dass auf Seiten des Planers nur Anpassungen an der Planungsdomäne notwendig sind, ohne den Programmcode des Planers selbst modifizieren zu müssen. Im Folgenden wird kurz auf die dafür notwendigen Anpassungen der Planungsdomäne eingegangen.

```
1  (:operator (!create_lifted_mug ?lifted_name ?area)
2   ((not (Instance Mug ?lifted_name)) ; kein Platzhalter vorhanden
3    (CNT ?cnt)
4    (new_constant ?new_on On ?cnt)
5    (inc_cnt ?cntn ?cnt))
6   ((CNT ?cnt))
7   ((Instance Mug ?lifted_name)
8    (Lifted ?lifted_name)
9    (Instance On ?new_on)
10   (hasPhysicalEntity ?new_on ?lifted_name)
11   (hasArea ?new_on ?area)
12   (CNTN ?cntn)))
13 (:operator (!lift_mug ?mug ?lifted_name)
14  ((Instance Mug ?mug))
15  ((Instance Mug ?mug))
16  ((IsLiftedAs ?mug ?lifted_name)))
```

Listing 3.6: Zwei Operatoren für das Lifting von Objekten vom Typ `Mug`.

```
1  (:method (lift_mugs_on_area ?lifted_name ?area)
2   ((Instance Mug ?mug)
3    (not (Lifted ?mug))
4    (Instance On ?on)
5    (HasArea ?on ?area)
6    (HasPhysicalEntity ?on ?mug)
7   )
8   ((create_lifted_mug ?lifted_name ?area)
9    (!lift_mug ?mug ?lifted_name)
10   (lift_mugs_on_area ?lifted_name ?area)
11  )
12  ()
13  ()
14 )
15 (:method (create_lifted_mug ?lifted_name ?area)
16  ((not (Instance Mug ?lifted_name))) ; doesn't already exist
17  ((!create_lifted_mug ?lifted_name ?area))
18  ()
19  ()
20 )
```

Listing 3.7: Zwei Methoden zum Ersetzen aller sich auf einem bestimmten Gebiet befindenden Objekte vom Typ `Mug` durch eine Platzhalterinstanz.

Um Lifting nur mit Mitteln der Domänenbeschreibungssprache zu erreichen, werden zusätzliche Methoden und Operatoren modelliert, die die jeweiligen Instanzen durch Platzhalterinstanzen ersetzen. Für das gegebene Szenario wurde dies für Objekte vom Typ `Mug` implementiert. Das Verfahren selbst ist jedoch unabhängig von der Art des Objekts. Die zwei zusätzlichen Operatoren sind in Listing 3.6 aufgeführt. Der Operator `!create_lifted_mug` erstellt eine neue Platzhalterinstanz für Objekte vom Typ `Mug`, die sich in einem bestimmten Gebiet befinden. Zudem erstellt der Operator auch eine neue Instanz vom Typ `On`, wodurch der Platzhalter von den anderen Operatoren und Methoden wie eine normale Instanz vom Typ `Mug` verwendet werden kann. Mit dem Operator `!lift_mug` kann eine bestehende Instanz im Zustand des Planers durch den Platzhalter ersetzt werden. Die Instanz selbst wird entfernt, was mit dem Prädikat `IsLiftedAs` markiert wird. Die beiden Operatoren werden von den in Listing 3.7 aufgeführten Methoden verwendet. Die Methode `lift_mugs_on_area` ersetzt rekursiv alle `Mug`-Instanzen in einem bestimmten Gebiet durch Platzhalter.

Damit der Platzhalter bei der Planausführung korrekt mit einer realen Instanz ersetzt werden kann, muss der Roboter neue Informationen darüber sammeln, welche Objekte sich tatsächlich auf dem Tresen befinden. Dies wird durch eine Beobachtungsaktion erzielt, die ausgeführt wird, wenn der Roboter direkt vor dem Tresen steht. Bei dieser Aktion bewegt der Roboter seinen Kopf, sodass er gezielt auf einen gewünschten Bereich schauen kann. Dadurch kann er die Ob-

jekte auf dem Tisch erkennen und die Einträge im Blackboard entsprechend aktualisieren. Die Aktion wurde als Teilaufgabe in die Methode zum Greifen des Objekts eingefügt. Somit ist die Beobachtungsaktion auch Teil des Plans.

Für die Planausführung und Ausführungsüberwachung wurde wiederum die konsistenzbasierte Ausführungsüberwachung verwendet. Damit konnte das beschriebene Szenario für das Holen eines Bechers erfolgreich in einer Gazebo-Simulation ausgeführt werden [89]. Lifting bietet auf diese Weise eine Möglichkeit, auch dann einen herkömmlichen Planer wie SHOP2 verwenden zu können, wenn eine Late-Commitment-Strategie wünschenswert ist, um so die Wahrscheinlichkeit von Ausführungsfehlern zu reduzieren.

Betrachtet man das gegebene Szenario allein, so hätte die Domänenbeschreibung alternativ auch direkt derart modelliert werden können, dass das Lifting nicht notwendig ist. Dabei würde man die Operatoren und Methoden zum Greifen so implementieren, dass sie keine Instanz des Objekts, sondern nur dessen Typ als Parameter erhalten und die Entscheidung welches Objekt gegriffen wird, der Planausführung überlassen. Dies hätte jedoch wiederum andere Nachteile: So könnte damit im Umkehrschluss kein Plan für das Transportieren einer bestimmten Objektinstanz erstellt werden, falls der Gast beispielsweise eine Lieblingstasse besitzt. Eine Domänenbeschreibung mit beiden Arten von Operatoren und Methoden würde wiederum einen erhöhten Aufwand in der Modellierung der Planungsdomäne und der Implementierung der Roboteraktionen mit sich bringen. Lifting bietet hier somit den Vorteil, beide Szenarien mit nur einem leicht erhöhten Modellierungsaufwand zu realisieren. Es ist ein allgemeines Verfahren, das auch zusammen mit anderen HTN-Planern verwendet werden kann. Dennoch wäre es vorzuziehen, wenn diese späte Variablenbindung und Anpassung der Pläne zur Laufzeit direkt vom Planer zur Verfügung gestellt würde.

Imagination

Als weiteres Verfahren zum Umgang mit Unsicherheiten wurde in RACE von Sebastian Rockel die *Imagination* mit dem HIRES-Framework [141] entwickelt. Dessen Idee ist es, bestimmte Aktionen des Plans während der Ausführung zunächst mit unterschiedlichen Parametern in einer Simulation zu testen und den Plan zur Ausführungszeit entsprechend zu modifizieren beziehungsweise zu vervollständigen. So können für einzelne Aktionen verschiedene Parameter getestet und ausgewählt werden, und potentielle Ausführungsfehler können frühzeitig vorausgeahnt und Gegenmaßnahmen getroffen werden.

Dies ist ähnlich zu Arbeiten von Kunze et al., in welchen eine Physiksimulation zur Ermittlung geeigneter Parametrierungen für die Objektmanipulation eingesetzt wird [90]. Jene setzen ein Inferenzsystem um, das beispielsweise das Greifen eines Eies oder das Wenden eines Pfannkuchens mit verschiedenen Parametern simuliert, die Ergebnisse protokolliert sowie in eine Repräsentation mittels Zeitlinien transformiert, auf deren Basis schließlich Anfragen in PROLOG gestellt und ausgewertet werden können. In der genannten Arbeit wird die Simulation jedoch nicht während der physischen Planausführung durchgeführt und das System ist nicht vollständig mit der Handlungsplanung und Planausführung integriert.

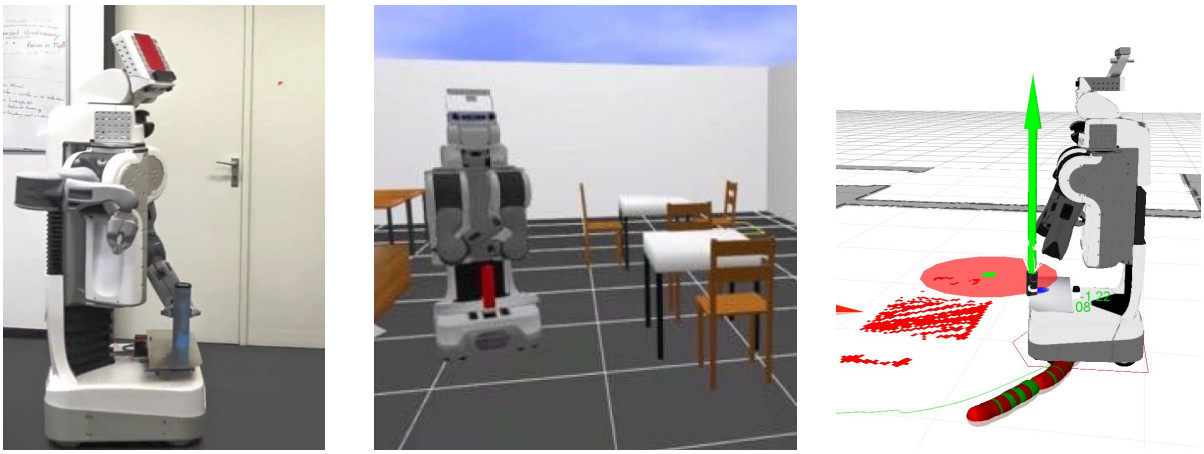


Abbildung 3.8: Realer (links) und in Gazebo simulierter (Mitte) PR2 mit einem auf seiner Basis montiertem Tablett und einem sich darauf befindlichen schlanken Zylinder („Pfeffermühle“). Rechts ist eine Visualisierung in RVIZ mit eingezeichneter Orientierung der Pfeffermühle zu sehen. Die linke und rechte Abbildung stammen aus [143], die mittlere aus [141].

Der Hauptbeitrag an dem hier vorgestellten Verfahren liegt bei Sebastian Rockel und dem von ihm entwickelten HIRES-Framework [142]. Der eigene Beitrag des Autors der vorliegenden Arbeit liegt in der Anbindung an den SHOP2-Planer und die notwendigen Anpassungen der Domänenbeschreibung [143]. Zudem lag eine Beteiligung an der Systemarchitektur vor.

HIRES verwendet Gazebo [88] als Simulator. Die Simulation ist auf modernen Rechnern hinreichend performant, dass einzelne später im Plan vorkommende Aktionen auf einem anderen per Netzwerk verbundenen Rechner in Gazebo getestet werden können, während der Roboter den Beginn des Planes ausführt.

Die Integration der Planung mit SHOP2, der semantischen Ausführungsüberwachung sowie der Imagination wurde in einem Beispielszenario demonstriert, in dem der Roboter eine Pfeffermühle vom Tresen holen und auf einem Tablett zu einem Tisch transportieren sollte. Wie in Abbildung 3.8 zu sehen ist, wurde hierfür ein Tablett auf die Basis des PR2s montiert. Die Gefahr dieser Aufgabe besteht darin, dass die Pfeffermühle in Abhängigkeit von der gewählten Geschwindigkeit beziehungsweise Beschleunigung und der vom Pfadplaner gewählten Trajektorie von dem Tablett fallen kann. Daher wurde die Aktion `!move_base` um einen Geschwindigkeitsparameter erweitert. Dieser kann die symbolischen Werte `fast` oder `slow` annehmen. Anhand der Ergebnisse der Simulation wird nun entschieden, welcher der beiden Werte besser geeignet ist, um eine hinreichende Erfolgswahrscheinlichkeit zu gewährleisten, ohne immer nur langsam fahren zu müssen. Hierbei wird berücksichtigt, wie sehr die Pfeffermühle auf dem Tablett wackelt oder gar umfällt. Dazu wird die Orientierung der Pfeffermühle überwacht, wie rechts in Abbildung 3.8 zu sehen ist. Zudem wird auch die Ausführungszeit der simulierten Aktion ermittelt, da sie der Planausführung einen gewichtigen Anhaltspunkt liefert, ob ausreichend Zeit übrig bleibt, um langsam zu fahren. Dieses Vorgehen zur Wahl einzelner Parameter wird in Abbildung 3.9 schematisch dargestellt.

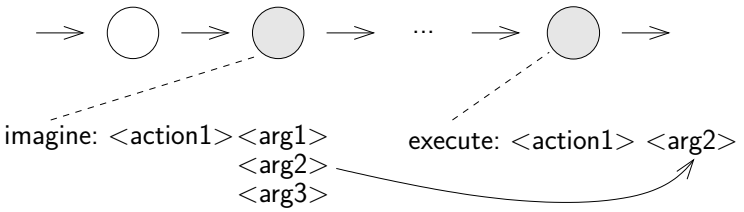


Abbildung 3.9: Schematische Darstellung der Wahl noch offener Parameter durch Imagination. Eine Aktion wird zunächst mit mehreren möglichen Parametern in der Simulation getestet. Auf Basis deren Ergebnisse wird einer dieser Parameter für die reale Ausführung der Aktion ausgewählt. Die Grafik stammt aus [143].

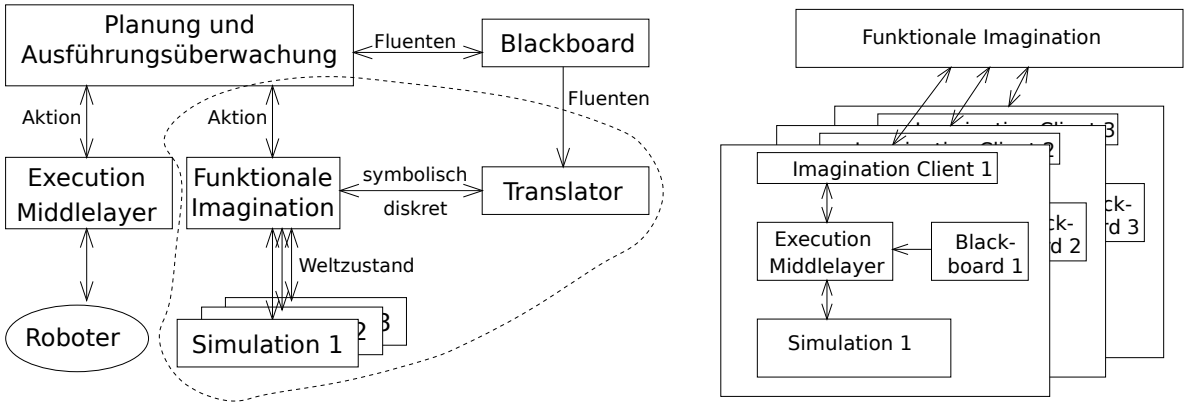


Abbildung 3.10: Zusammenspiel zwischen Planung, Planausführung und Simulation mit HIRES. Darstellungen nach [143].

Abbildung 3.10 zeigt das Zusammenspiel zwischen HIRES, dem Planer und der semantischen Ausführungsüberwachung. Der linke Teil der Abbildung gibt einen Überblick über das Gesamtsystem. Die Planung und die Planausführungsmodule sind hier zur Vereinfachung zusammengefasst. Sie führen die Aktionen auf dem Roboter durch Aufruf über den *Execution Middlelayer*, in dem die primitiven Roboteraktionen implementiert sind, aus. Die Abbildung verdeutlicht, dass die Imagination parallel zu dieser Ausführung auf dem physischen Roboter angegliedert ist. Die *Funktionale Imagination* koordiniert das Testen einer Aktion für verschiedene Parameternistanziierungen. Für jede mögliche Parameterwahl wird eine Simulation auf einem separaten Rechner gestartet. Dies wird detaillierter im rechten Teil der Abbildung dargestellt. Für jede gestartete Simulation wird auch ein eigener ROS-Master sowie eine Untermenge des RACE-Systems inklusive Blackboard und Execution Middlelayer gestartet.

Die Simulation benötigt die Posen aller zu simulierenden Objekte sowie die gewünschte Roboterkonfiguration inklusive der Posen der Arme und des Torsos. Hierbei ist zu beachten, dass in der Simulation nicht der zum Startzeitpunkt der Simulation reale Weltzustand direkt abgebildet werden soll, sondern der beim späteren Start der zu testenden Aktion erwartete Zustand. Dazu wird der erwartete Zustand auf Basis des aktuellen Zustands und der bis zum Start der zu testenden Aktion noch fehlenden Aktionen generiert.

```
1  (:method (move_object_imagine ?object ?toPlacingArea)
2   ((Instance Tray ?tray)
3    (HasManipulationArea ?toPlacingArea ?manArea)
4    (HasPreManipulationArea ?manArea ?preArea)
5   )
6   ((!imagine !move_base_param ?preArea slow/fast)
7    (get_object_w_tray ?object)
8    (put_object ?object ?toPlacingArea)
9   )
10 )
```

Listing 3.8: Methode zum Transportieren eines Objekts auf einem Tablett inklusive Testen unterschiedlicher Fahrgeschwindigkeiten in Simulation.

Zudem benötigt Gazebo metrische Daten, wohingegen der Plan und die Problembeschreibung nur symbolische Informationen enthalten. An dieser Stelle kommt der *Translator* ins Spiel, der die symbolischen Informationen in die für die Initialisierung der Simulation benötigten metrischen Posen übersetzt. Beispielsweise enthält der Plan oder die Problembeschreibung des Planes nur die rein symbolische Information, dass sich die Pfeffermühle auf einem bestimmten Bereich des Tisches befindet. Gazebo benötigt hingegen eine konkrete Pose, die die Pfeffermühle einnehmen soll. Sofern die Pfeffermühle nicht bereits zuvor mit einer anderen Aktion bewegt wird, extrahiert der Translator die Pose, in der die Pfeffermühle zuletzt gesehen wurde, aus dem Blackboard. Wenn zwischenzeitlich eine andere Aktion die Pfeffermühle manipulieren wird und sich deren Pose daher noch ändern wird, verwendet der Translator eine Standard-Pose innerhalb des jeweiligen Gebiets.

Der Plan enthält Informationen, welche Aktionen simuliert werden sollen, welche Parameter zu variieren sind und wann die Simulationen gestartet werden können. Dies geschieht allein durch eine Erweiterung der Planungsdomäne, ohne Änderungen an SHOP2 vornehmen zu müssen. Es wurde der Operator `!imagine ?task ?arg1 ?arg2` erstellt, der abgesehen vom obligatorischen Inkrementieren des Zählers weder Vorbedingungen noch Effekte besitzt. Der Operator signalisiert der Planausführung, dass an dieser Stelle im Plan die Simulation der als Parameter übergebenen Aufgabe `?task` mit den Argumenten `?arg1` und `?arg2` zu starten ist. Da auch die Argumente der zu simulierenden Aufgabe wiederum Argumente des `!imagine`-Operators sind, kann dieser spezielle Operator nur für Aufgaben mit genau zwei Argumenten angewendet werden. Sollen auch Aufgaben mit einer anderen Argumentenanzahl simuliert werden, sind analog dazu weitere solcher `!imagine`-Operatoren zu erstellen.

In dem Beispielszenario wird jener `!imagine`-Operator von der zusätzlich erstellten Methode `move_object_imagine` verwendet. Sie ist eine modifizierte Version der Methode `move_object`. Listing 3.8 zeigt ihre Definition. Die Durchführung der Imagination erfolgt hier direkt zu Beginn als erste Teilaufgabe. Für die Aufgabe `!move_base_param` werden die Parameter `slow` und `fast` getestet.

Das Zusammenspiel der verschiedenen Module wurde erfolgreich für das Beispielszenario mit einem PR2 demonstriert. Details hierzu finden sich in [143]. Das Ausführen der beiden Simulationen für die Parameter `slow` und `fast` dauerte 109 Sekunden beziehungsweise 76 Sekunden, während die reale Ausführung des ganzen Szenarios 384 Sekunden benötigte. Insbesondere waren die Simulationen, welche parallel zu der realen Ausführung durchgeführt wurden, bereits vor dem Greifen der Pfeffermühle vom Tresen und damit rechtzeitig vor deren Transport auf dem Tablett beendet. Die Imagination führte somit zu keiner Verlängerung der Gesamtdauer der Planausführung. Für das schnelle Fahren wurde in der Simulation ein mehrfaches Schwanken der Pfeffermühle festgestellt, wohingegen sie sich bei langsamer Fahrt nicht bewegte und letzteres somit von HIRES einen höheren Konfidenzwert erhielt. Da auch die längere Ausführungszeit einer langsamen Fahrt nicht in Konflikt mit anderen Nebenbedingungen stand, hat die semantische Ausführungsüberwachung schließlich den Parameter `slow` für die Aktion `!move_base_param` gewählt. Der Plan wurde damit erfolgreich ohne Umfallen der Pfeffermühle auf dem realen PR2 ausgeführt. Zum Vergleich wurde das System ohne Imagination getestet. Dabei ist die Pfeffermühle in einem von vier realen Versuchen bei schneller Fahrt umgefallen. Für weitere experimentelle Resultate sei auf [143] sowie [141] verwiesen.

Eine mögliche Erweiterung dieses Verfahrens wäre die Simulation ganzer Pläne. Eine Limitierung ist allerdings noch die Laufzeit, da die Simulation sehr rechenintensiv ist. Dadurch kann die Simulation mehrere Aktionen auf heutigen Rechnern nicht zeitgleich zur realen Ausführung der ersten Aktionen des Plans ausgeführt werden. Der Roboter müsste somit längere Zeit auf das Ergebnis der Simulation warten, bevor er dessen Ergebnis nutzen und fortfahren kann.

3.5 Diskussion

In diesem Kapitel wurde die Verwendung des weit verbreiteten HTN-Planers SHOP2 in einem Robotersystem beschrieben, das sich eine Verbesserung des Roboterverhaltens durch Lernen aus Erfahrungen zum Ziel gesetzt hat. Es konnte gezeigt werden, dass SHOP2 durch seine Zuverlässigkeit und Performanz grundsätzlich als Handlungsplaner für die Steuerung eines Roboters eingesetzt werden kann und sich insbesondere als Übergangskomponente oder Platzhalter für eine schnelle Integration in ein sich in der Entwicklung befindendes Robotersystem eignet. Zudem bietet die HTN-Planung weitere Vorteile für Roboter: Experten können zusätzliches Domänenwissen einbringen, um so die Kontrolle des Roboters genauer vorgeben zu können; Methoden können gelernt werden, damit sich der Roboter an neue Anforderungen anpassen kann; und die Aufgabenhierarchie macht die Pläne nachvollziehbar. Für die Ausführung der Pläne wurde ein auf einem Zustandsautomaten basierender Ansatz umgesetzt. Auf diese Weise konnten bereits zu einem frühen Zeitpunkt das Gesamtsystem in verschiedenen Szenarien demonstriert und Erfahrungen generiert werden, die als Eingabedaten für die lernenden Komponenten dienen.

Für die Kommunikation mit einem als Wissensbasis dienenden Blackboard wurde ein allgemeines Format für die Planungsdomäne und die Beschreibung des Zustands entwickelt, das insbesondere auch die Integration mit der Ausführungsüberwachung erleichtert, da diese neben den geplanten Aktionen auch deren Vor- und Nachbedingungen benötigt.

Allerdings hat der Einsatz von SHOP2 auch einige Schwächen. Klassische Planung und auch gebräuchliche HTN-Planer wie SHOP2 sind nicht an die speziellen Anforderungen von Robotern ausgelegt. Sie setzen Grundannahmen wie eine vollständige Beobachtbarkeit der Umgebung voraus, die für die meisten Robotersysteme nicht zutreffen. Es wurden daher zwei Verfahren vorgestellt, die dennoch in einem gewissen Maße einen Umgang mit Unsicherheit erlauben und die Robustheit des Gesamtsystems somit erhöhen, ohne Änderungen am Planer selbst zu erfordern. Hierbei wurde deutlich, dass die Planung und die Planausführung für Roboter nicht als zwei getrennte abgeschlossene Systeme betrachtet, sondern enger verknüpft werden sollten.

Ein weiterer Nachteil ist, dass SHOP2 ein reiner Handlungsplaner ist und primär kausales Wissen in Form von Operatoren und Methoden für seine Planung verwendet. Wie in den nächsten beiden Kapiteln erläutert wird, liegen in einer Roboterumgebung jedoch viele verschiedene Formen von Wissen vor. Diese sollten genutzt werden, um alle Anforderungen abdecken zu können.

Kapitel 4

Hybride Planung für Roboter

4.1 Motivation

Im vorherigen Kapitel wurden nur konventionelle Planungsverfahren eingesetzt, die rein kausales Wissen verwenden. Jedoch sind in den Umgebungen, in denen sich Roboter bewegen, häufig sehr unterschiedliche Arten von Wissen verfügbar. Betrachten wir das Beispiel eines Kellnerroboters. Zunächst benötigt der Roboter weiterhin kausales Wissen: Er muss die Folgen und Voraussetzungen seiner eigenen Aktionen kennen. Zudem muss er beispielsweise bestimmte Aktionen von Gästen, wie das Bestellen eines Kaffees, interpretieren können.

Gerade in Restaurants ist allerdings auch *temporales* Schlussfolgern von Bedeutung. Der Roboter sollte folgern können, dass Kaffee in einer bestimmten Zeitspanne serviert werden muss, damit er nicht zu sehr abkühlt oder der Gast ungeduldig wird. Daher sollte der Roboter im Falle mehrerer Aufgaben zunächst den Kaffee servieren, bevor er einen anderen Tisch abräumt. Ist der Kaffee hingegen noch nicht von einem anderen Akteur fertig aufgebrüht worden, sollte der Roboter bemerken, dass er in der Zwischenzeit mit dem Abräumen des Tisches beginnen und diese Tätigkeit später wiederum für das Servieren des Kaffees unterbrechen kann.

Als weitere Wissensform sollte der Roboter auch *Ressourcen* berücksichtigen. Ein PR2 kann mit seinen zwei Armen im Normalfall zwei Objekte transportieren, mit einem Tablett wären hingegen mehrere Objekte möglich. Zusammen mit temporalem Schlussfolgern können auf diese Weise mehrere Aufgaben zeitlich überlappend ausgeführt werden. Die Arme stellen eine physische Ressource dar. Zudem sind auch technische Ressourcen zu beachten. Beispielsweise sind die Aktionen zum Greifen oder Platzieren von Objekten häufig derart implementiert, dass der Roboter mit seinen zwei Armen nicht mehrere Objekte zur gleichen Zeit greifen oder abstellen kann, da die Bewegungsplanung für einen Arm nicht die Trajektorie des anderen Arms berücksichtigt.

Da autonome mobile Roboter sich in ihrer Umgebung bewegen und Objekte manipulieren, ist für sie auch *räumliches* Wissen von Bedeutung. Beispielsweise muss der Roboter berechnen, welche Posen nahe genug an einem Objekt liegen, um es greifen zu können. Beim Tischdecken kann der Roboter räumliches Wissen zudem nutzen, um Besteck und Geschirr korrekt zu platzieren [102].

Eng verbunden mit räumlichen und geometrischen Wissensrepräsentationsformen ist die Bewegungsplanung [94, Part II]. Unter Berücksichtigung der Roboterkinematik werden mit einem Bewegungsplaner Trajektorien von einer Ausgangspose des Roboters zu einer Zielpose geplant. Häufig werden unterschiedliche dedizierte Bewegungsplaner für bestimmte Aufgaben verwendet. Diese werden zur Planausführungszeit direkt in der Implementierung der low-level Roboterfähigkeiten aufgerufen. So wurde in RACE zur Navigation der standardmäßig von ROS zur Verfügung gestellte Pfadplaner [105] und ein weiterer Planer für die Berechnung der Armtrajektorien zum Greifen und Platzieren von Objekten eingesetzt. Aber auch diese primär für die Ausführung bestimmter Aktionen verwendeten Planer können für die Erstellung des Gesamtplans wichtige Informationen liefern. Beispielsweise kann ein Pfadplaner eine gute Abschätzung liefern, wie lange der Roboter für die Fahrt zu einer bestimmten Pose benötigen wird. Zudem können Pläne, die kinematisch nicht ausführbar sind, ausgeschlossen werden.

Wie im vorherigen Kapitel gezeigt, kann bereits mit einem rein kausalen Planer und hinreichendem Aufwand in der Modellierung der Domänenbeschreibung in vielen Fällen das gewünschte Roboterverhalten erzielt werden. So lassen sich die Kapazitäten bestimmter Ressourcen wie beispielsweise die Anzahl der Arme oder die Objekte, die auf ein Tablett passen, in SHOP2 auch mit einem ganzzahligen Parameter modellieren. Jedoch lässt sich damit kein Scheduling durchführen. Da SHOP2 nur vollständig geordnete Pläne erzeugt, fallen diese Limitierungen der Modellierung von Ressourcen bei diesem Planer weniger ins Gewicht. Für einen Planer, der Pläne mit parallel ausführbaren Aktionen generieren soll, ist jedoch ein geeignetes Scheduling unter Berücksichtigung der Ressourcen notwendig.

Räumliche Information lassen sich in einem rein symbolischen Planer durch die Aufteilung des Raumes in voneinander abgetrennte Bereiche und den Verbindungen zwischen diesen symbolisch darstellen. So wurden in RACE beispielsweise Bereiche vom Typ `ManipulationArea` definiert, von denen der Roboter Objekte auf einer zugeordneten `PlacingArea` auf einem Tisch manipulieren kann. Jedoch lässt dies nur begrenzte Möglichkeiten für die Planung zu. So kann beispielsweise nicht ermittelt werden, welche Roboterpose innerhalb der `ManipulationArea` für das Greifen eines Objekts geeignet ist, wenn sich weitere Objekte auf dem Tisch befinden und möglicherweise die Sicht auf das zu greifende Objekt einschränken.

Problematisch wird die Verwendung eines rein kausalen Planers vor allem dann, wenn die gegebene Problemstellung leicht von der idealen Ausgangssituation abweicht. Um ein robustes Roboterverhalten in komplexen Umgebungen zu erzielen, sollte daher das Wissen über die Umgebung hinreichend repräsentiert und mit speziellen Schlussfolgerungsmethoden genutzt werden. Wenn der Roboter keine Möglichkeiten für temporales Schlussfolgern besitzt, könnte er beispielsweise nach einer langen Wartezeit einen kalten Kaffee servieren, ohne auch nur eine Ahnung zu haben, warum der Gast unzufrieden ist und ihm kein Trinkgeld gibt.

Aber auch für die Planausführung und die Ausführungsüberwachung ist eine ausdrucksstarke Repräsentation von Bedeutung, weil die Pläne dadurch zusätzliche wichtige Informationen für diese enthalten können. Während der Plan ausgeführt wird, können zum Beispiel die temporalen Relationen überwacht werden, um sicherzustellen, dass der Kaffee auch wie geplant heiß serviert wird.

Aus diesen Gründen ist die hybride Planung im Allgemeinen und die hybride Planung für Roboter im Speziellen ein aktuelles Forschungsthema. Im nachfolgenden Abschnitt werden zunächst relevante Arbeiten in diesem Gebiet beschrieben. Anschließend wird der in dieser Arbeit verwendete Ansatz der hybriden Planung als Meta-CSP erläutert.

4.2 Stand der Forschung

Hybride Planung als Verknüpfung einzelner Formen von Wissen für bestimmte Problemstellungen wurde schon häufig in der Künstlichen Intelligenz und Robotik durchgeführt. So haben frühere Arbeiten versucht, partiell ordnende Planung und temporale Planung zu verbinden [34, 62]. Hieraus resultierte die zeitleistenbasierte Planung, die in vielen Planungssystemen wie HSTS [113], RAX-PS [83], Europa [6, 58], ASPEN [30], OMPS [59], ASPI-TRF [29] und EPSL [169] umgesetzt wird. Cialdea Mayer et al. geben hierzu einen Überblick und präsentieren eine Formalisierung für diesen Planungsansatz [31]. Eine Zeitleiste repräsentiert danach die zeitliche Entwicklung einzelner Merkmale, die meist als Zustandsvariablen modelliert werden, und der Planer modifiziert diese Zeitleisten, sodass sie seiner Domänenbeschreibung entsprechen.

Obwohl SHOP2 ein rein kausaler Planer ist, lassen sich mit diesem durch Multi-Timeline Preprocessing (MTP) [117] oder dem von Goldman [66] vorgeschlagenen Verfahren dennoch auch temporale Informationen in dem Plan repräsentieren. Beispielsweise ist MTP ein Vorverarbeitungsschritt, der eine gegebene Domänenbeschreibung um temporale Informationen ergänzt. Hierfür wird ausgenutzt, dass SHOP2 numerische Variablen repräsentieren kann und diese von Operatoren modifiziert werden können. Dabei kann auf Ausdrücke in der Programmiersprache LISP zurückgegriffen werden, wodurch komplexe Berechnungen möglich sind. MTP erweitert die Operatoren um zwei zusätzliche Parameter zur Repräsentation des Startzeitpunktes und der Dauer. Zusätzlich wird für jedes dynamische Prädikat jeweils ein Lese- und ein Schreibezeitpunkt repräsentiert. Der Startzeitpunkt und die Dauer eines Operators wird mittels Formeln in dessen Vorbedingungen ermittelt. Der Startzeitpunkt wird auf das Maximum aller Schreibezeitpunkte seiner Vorbedingungen gesetzt. Zudem werden die Lesezeitpunkte aller in dem Operator verwendeten Prädikate sowie die Schreibezeitpunkte der Prädikate der Effekte aktualisiert. Auf diese Weise lassen sich mit Hilfe von MTP auch mit SHOP2 Schlussfolgerungen über die Dauer und mögliche zeitgleiche Ausführungen von Aktionen treffen. Unter Berücksichtigung der Startzeitpunkte der Aktionen ist somit auch eine parallele Ausführung von Aktionen möglich. Durch die Verwendung der Lese- und Schreibzeitpunkte wird ausgeschlossen, dass sich Aktionen, die dasselbe Prädikat verändern, zeitlich überschneiden. Ressourcen werden hier jedoch nicht betrachtet. Zudem werden die temporalen Relationen zwischen den Vorbedingungen, den Effekten und den Aufgaben nur sehr grob modelliert.

Der HTN-Planer SIADEX [27] kann ebenfalls eine temporale Planung durchführen. Er verwendet dabei eine ausdrucksstarke temporale Repräsentation als Simple Temporal Problem (STP) [37] und kann damit sowohl qualitative als auch quantitative Informationen berücksichtigen. Zudem wird der Modellierungsaufwand im Vergleich zu MTP mit SHOP2 verringert, da die Unteraufgaben beispielsweise die temporalen Constraints der übergeordneten Aufgabe erben.

Aktuelle Ansätze für Roboter kombinieren Bewegungsplanung oder geometrische Planung mit Handlungsplanung und insbesondere mit hierarchischer Planung. Der Handlungsplaner erstellt hierbei einen abstrakten Plan, dessen Details von dem Bewegungsplaner aufgefüllt werden. Die Ansätze unterscheiden sich dabei unter anderem in der Stärke der Kopplung beziehungsweise Verknüpfung der Repräsentationen der verschiedenen Planer. So schlagen Dornhege et al. *Semantic Attachments* für die Verbindung eines symbolischen PDDL-Planers mit externen Modulen vor [44]. Hierfür werden die Werte bestimmter Prädikate während der Planung durch die Auswertung einer an sie geknüpften Prozedur ermittelt. Dieses Schema nutzen sie für die Integration eines geometrischen Planers in einen Handlungsplaner [45]. Hierbei sind die Planung und die zusätzlichen Prozeduren jedoch stark von einander getrennt und die externen Prozeduren sind aus Sicht des Planers eine Blackbox. Dies ist bei Wolfe et al. der Fall, wo ein Bewegungsplaner für die Planung der untersten Ebene eines HTN-Planes eingesetzt wird [178].

De Silva et al. beschreiben hingegen eine Schnittstelle zur etwas stärkeren Verzahnung von geometrischer Planung und HTN-Planung [152]. Sie verflechten die symbolische und geometrische Repräsentation beider Planer, indem diese dieselben Prädikate teilen. Des Weiteren diskutieren sie, wie das Backtracking der beiden Planer integriert werden kann. Lagriffoul et al. kombinieren SHOP2 mit einem Bewegungsplaner [93]. Dabei führen sie den Begriff *geometrisches Backtracking* als ein Hauptproblem der Kombination von Handlungs- und Bewegungsplanern ein. Die Vielzahl geometrischer Konfigurationen der Aktionen multipliziert sich über die Aneinanderreihung von Aktionen in einem Plan auf und führt zu einen sehr großen Suchraum. Bei der Wahl einer ungünstigen Konfiguration zu Beginn der Suche muss Backtracking über alle möglichen späteren Alternativen durchgeführt werden. Um die Zahl der Aufrufe des Bewegungsplaners zu reduzieren, führen sie zwischen den beiden Planern eine weitere Schicht ein. In dieser Schicht werden die möglichen geometrischen Konfigurationen für Aktionen mit Intervallen und Constraints zwischen diesen repräsentiert und mittels Constraint-Propagierung einschränkt. Ebenso integrieren Kaelbling und Lozano-Pérez Bewegungsplanung in einen hierarchischen Planer [85]. Wie bereits angesprochen, basiert dieser Ansatz im Gegensatz zu den meisten HTN-Planern auf einer sehr engen Integration zwischen Planung und Ausführung. Der Bewegungsplaner führt dabei zur Ausführungszeit eine Verfeinerung der unteren Ebenen des Plans durch, und auch die oberen Ebenen werden nicht initial vollständig geplant, sondern während der Ausführung inkrementell erweitert.

Mit ANML [155] existiert eine ausdrucksstarke Domänenbeschreibungssprache, die neben temporalen Relationen und dem Ressourcenverbrauch von Aktionen auch HTN-Methoden unterstützt. Der erste Planer, der einige Eigenschaften von ANML inklusive der HTN-Aufgabendekomposition integriert, jedoch laut eigener Dokumentation Ressourcenscheduling noch nicht voll unterstützt, ist FAPE [47], welcher dem im nächsten Kapitel beschriebenen CHIMP relativ ähnlich ist. Bei ihm werden zudem die Planung, Ausführung das Reparieren von Plänen sowie eine Neuplanung eng ineinander verschachtelt.

Weitere Ansätze kombinieren Handlungsplanung und ontologisches Schließen [12, 65, 154]. Hierzu gehört auch das bereits genannte Verfahren von Hartanto, welches das Planungsproblem sowie eine speziell an das Planungsproblem angepasste Domänenbeschreibung aus einer OWL-Ontologie generiert [70]. Awaad et al. erweitern die Arbeit von Hartanto dahingehend, dass in der

OWL-Ontologie zusätzliche Affordanzen repräsentiert und für die HTN-Planung und Planausführung genutzt werden können [3]. Damit lassen sich geeignete Ersatzgegenstände identifizieren, wenn die präferierte Objektkategorie in der Umgebung nicht vorhanden ist.

Schließlich sei erwähnt, dass der Begriff *hybride Planung* in der Literatur unterschiedliche Verwendung findet. Während damit in der vorliegenden Arbeit die Integration unterschiedlicher Wissensrepräsentationsformalismen gemeint ist, wird der Begriff auch spezieller für die Kombination von hierarchischer und Operator-basierter Planung verwendet [8, 16, 53, 87, 149].

4.3 Hybride Planung als Meta-CSP

Eine bisher noch nicht erwähnte Art der Planung ist die Planung durch Repräsentation als Constraint Satisfaction Problem (CSP). Ein Constraint-Netz besteht aus einer endlichen Menge von Variablen, einer Menge von Domänen dieser Variablen sowie einer Menge von Constraints, die die möglichen Werte der Variablen einschränken [36]. Gesucht ist eine Zuweisung von Werten aus den Domänen an die Variablen, die den Constraints genügt. Diese Formalisierung kann genutzt werden, um spezielle Probleme als Constraint-Netz zu formulieren und mit allgemeineren Verfahren für die Suche in Constraint-Netzen zu bearbeiten. Ein Überblick über diese Techniken gibt Rina Dechter in ihrem Buch [36]. Im Speziellen lassen sich diese Techniken auch für die Planung einsetzen. Dabei wird das Planungsproblem in einem Constraint-Netz repräsentiert und mit gewöhnlichen Constraint-Lösern versucht, einen Plan als gültige Lösung des Constraint-Netzes zu finden. Ein Überblick über diese Möglichkeit der Planung findet sich in [63, Kapitel 8] sowie [7, 114].

Constraint-Netze bieten zudem unterschiedliche Methoden des Schlussfolgerns für spezielle Problemstellungen. Ihr Einsatz für die Handlungsplanung ist im Vergleich dazu eher selten. So lassen sich beispielsweise temporale oder räumliche Beziehungen mit Constraints darstellen und Inferenzverfahren auf diese anwenden. Für Anwendungen mit hybriden Formen von Wissen ist es daher wünschenswert, verschiedene spezielle Constraint-Netze zu kombinieren. Hierfür bietet sich der Meta-CSP-Ansatz an, der in dieser Arbeit eingesetzt wird. Im Folgenden wird zunächst am Beispiel eines Optimierungsproblems in die Meta-CSP-Planung eingeführt, bevor anschließend auf das allgemeine Meta-CSP-Verfahren und seine Umsetzung im *Meta-CSP-Framework* eingegangen wird.

4.3.1 Der Meta-CSP-Ansatz am Beispiel eines Optimierungsproblems

Die Kombination mehrerer Constraint-Netze wurde von Cesta et al. für das *RCSPSP/max*-Problem (Resource Constraint Project Scheduling Problem with Generalized Precedence Relations) demonstriert [28]. Es liefert den Grundstein für das Verfahren hybriden Schlussfolgerns in verschiedenen Constraint-Netzen, wie es im Meta-CSP-Framework verallgemeinert und in der vorliegenden Arbeit verwendet wird. Dieser Ansatz kombiniert ein temporales Constraint-Netz mit Ressourcenanforderungen.

RCPSP/max ist ein Optimierungsproblem aus dem Bereich des Operations Research. In ihm ist eine Menge von Aktivitäten gegeben, die jeweils in einer bestimmten Dauer ausgeführt werden müssen. Zwischen diesen Aktivitäten bestehen zeitliche Ordnungsrelationen, die einzuhalten sind. Zudem beanspruchen die Aktivitäten während ihrer Ausführung jeweils begrenzte wiederverwendbare Ressourcen. Das Ziel des Optimierungsproblems ist die Erstellung eines Zeitplans für alle Aktivitäten, der die gegebenen Nebenbedingungen erfüllt und die Gesamtdauer der Ausführung aller Aktivitäten minimiert.

Cesta et al. verwenden dafür den Meta-CSP-Ansatz, bei dem das Problem als Hierarchie von Constraint-Netzen modelliert wird. Ein übergeordnetes *Meta-CSP* wird auf Basis eines untergeordneten *Grund-CSPs* definiert. Für das RCPSP/max-Problem dient ein temporales Constraint-Netz als Grund-CSP. In ihm werden die Start- und Endzeitpunkte der Aktivitäten und ihre temporalen Beziehungen repräsentiert. Das temporale Constraint-Netz entspricht dabei einem Simple Temporal Problem [37]. Auf Basis dieses Grund-CSPs sowie den zusätzlichen Informationen über die Ressourcen, die zur Verfügung stehen, und dem Verbrauch dieser Ressourcen durch Aktivitäten wird ein Meta-CSP definiert. Dessen Variablen sind Mengen von Aktivitäten, die zeitgleich eine Ressource überbeanspruchen. Solche Mengen von Variablen, die zusammen genommen einen Konflikt darstellen, werden auch als *Meta-Variablen* bezeichnet. Ihre Domänen sind in diesem Beispiel Mengen temporaler Ordnungsrelationen, die eine Aktivität vor eine andere Aktivität anordnen und sie so zeitlich trennen. Die Berechnung der Ressourcenkonflikte geschieht auf Basis eines Earliest Start Schedule (ESS), bei welchem nur die frühestmöglichen Startzeiten der Aktivitäten betrachtet werden. Als Meta-Variablen werden in der genannten Arbeit *Minimal Critical Sets* (MCS) in Betracht gezogen. Dies sind minimale Mengen von Aktivitäten, die eine Ressource überbeanspruchen. Sie sind minimal in dem Sinne, dass das Entfernen eines beliebigen Elements den Ressourcenkonflikt auflöst. Das hat den Vorteil, dass bereits die zeitliche Trennung zweier Aktivitäten durch Hinzufügen einer temporalen Ordnungsrelation zur Auflösung des Ressourcenkonflikts des MCSs führt. Für zwei Elemente eines MCS wird eine solche Ordnungsrelation ausgewählt und in das temporale Constraint-Netz eingefügt. Hierbei werden seine temporalen Auswirkungen durch das gesamte Constraint-Netz propagiert. Führt dies zu einer temporalen Inkonsistenz, wird jener Constraint wiederum entfernt und ein anderer Constraint zur Auflösung des Ressourcenkonflikts gewählt. Wenn dies hingegen erfolgreich gewesen ist, werden anhand des aktualisierten Grund-CSPs neue Meta-Variablen ermittelt und auf diese Weise fortgefahren, bis keine neuen Meta-Variablen gefunden werden. Letzteres bedeutet, dass kein Ressourcenkonflikt mehr vorliegt. Der Earliest Start Schedule der Aktivitäten stellt in diesem Fall einen gültigen Zeitplan des Optimierungsproblems dar. Anstatt in jedem Schritt die Gesamtheit aller MCSs zu ermitteln, verwenden die Autoren für ein polynomielles Sampling-Verfahren. Des Weiteren werden Heuristiken für die Wahl der Meta-Variablen und Meta-Werte vorgestellt, die Aktivitäten mit hohem Ressourcenverbrauch bevorzugen.

4.3.2 Allgemeine Beschreibung des Meta-CSP-Ansatzes

Der Meta-CSP-Ansatz zum hybriden Schließen in mehreren Constraint-Netzen wurde von Mansouri und Pecora an der Universität Örebro aufgegriffen, verallgemeinert und formalisiert [98,

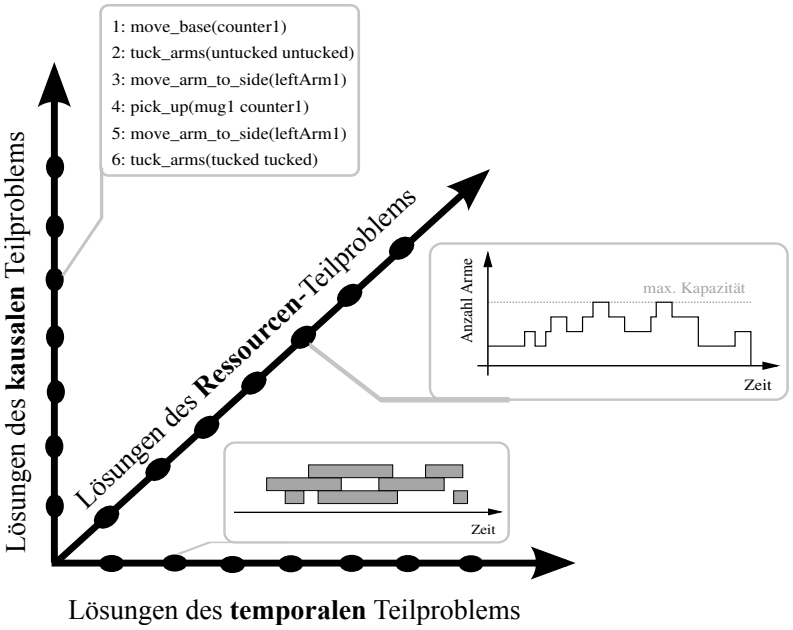


Abbildung 4.1: Veranschaulichung des hybriden Suchraums.

104]. Zudem wurde dort das Meta-CSP-Framework entwickelt, auf welches im nächsten Abschnitt eingegangen wird.

Dies adressiert allgemein das Problem, wie unterschiedliche Formen von Wissen und Anforderungen gemeinsam repräsentiert und mit ihnen geplant oder Inferenz betrieben werden kann. Es wird dazu eine gemeinsame constraintbasierte Repräsentation verwendet. Des Weiteren führt das Vorhandensein unterschiedlicher Wissensformen und Anforderungen zu einem potentiell hochdimensionalem Suchproblem, welches in Abbildung 4.1 veranschaulicht wird. Häufig werden die einzelnen Teilprobleme getrennt betrachtet und nacheinander gelöst. Hierdurch kann die Suche potentiell in viele Sackgassen laufen. Der Meta-CSP-Ansatz verfolgt hingegen eine andere Strategie: Anstatt die einzelnen Teilprobleme sequenziell und für sich abgeschlossen anzugehen, werden sie abwechselnd und iterativ gemeinsam von den speziellen Constraint-Reasonern bearbeitet. Die Teilschritte in der Suche eines Reasoners wirken sich so direkt auf die Suche der anderen Reasoner und umgekehrt aus. Auf diese Suchstrategie wird später in diesem Abschnitt noch genauer eingegangen. Zunächst wird jedoch die Repräsentation behandelt. Für die Integration der unterschiedlichen Reasoner wird das Problem wie im vorhergehenden Beispiel als Meta-CSP [104] definiert, welches Constraint Satisfaction Probleme einer höheren Abstraktionsebene sind. Ähnlich zu ihren Gegenstücken in CSPs gibt es Meta-Constraints, Meta-Variablen und Meta-Werte. Im Folgenden wird erläutert, worum es sich bei diesen handelt und wie sie für die Planung verwendet werden können.

Als Basis dient einem Meta-CSP ein *gemeinsames Constraint-Netz*, welches alle Aspekte des Gesamtproblems umfasst [129]. Es ist eine gemeinsame Repräsentation für verschiedene dedizierte Constraint-Löser. In gewisser Weise werden damit wiederum Aspekte eines klassischen

Blackboard-Systems aufgegriffen. Eine formale Definition des in dieser Arbeit verwendeten gemeinsamen Constraint-Netzes folgt im nächsten Abschnitt in Definition 5.1.1. An dieser Stelle soll dieses jedoch zunächst allgemeiner betrachtet werden.

Das gemeinsame Constraint-Netz besteht aus Variablen und Constraints eines oder mehrerer *Grund-CSPs*. Sie bilden die Grundlage für die Meta-CSPs und decken tiefer liegende Teilaspekte des Gesamtproblems ab. Diese können auch als unterschiedliche Sichten auf das Gesamtproblem verstanden werden. In dem Beispiel von Cesta et al., welches im letzten Abschnitt beschrieben wurde, war das temporale Constraint-Netz ein Grund-CSP; der Planer von Mansouri und Pecora [104], der das korrekte Anordnen von Objekten auf einem Tisch sicherstellt, hat ein temporales und ein räumliches Grund-CSP; das gemeinsame Constraint-Netz von CHIMP beinhaltet Variablen und Constraints eines temporales und ein symbolisches Grund-CSPs. Diese Grund-CSPs können wie üblich mit denen für sie vorgesehenen speziellen Constraint-Lösern für sich allein genommen bearbeitet werden. Auf diese Weise wird sichergestellt, dass die Teilaspekte, die dem Meta-CSP zu Grunde liegen, konsistent sind.

Auf Basis des gemeinsamen Constraint-Netzes kann ein Meta-CSP, das aus einer Menge von *Meta-Constraints* besteht, definiert werden. Meta-Constraints repräsentieren allgemeine Anforderungen auf dem gemeinsamen Constraint-Netz, welche zur Lösung des Gesamtproblems zu erfüllen sind. Anforderungen können zum Beispiel wie bei Cesta et al. die Einhaltung von Ressourcenkapazitäten oder, wie im nächsten Kapitel noch gezeigt wird, das Geplantsein aller Aufgaben eines Aufgabennetzes sein. Meta-Constraints bestehen aus einem gemeinsamen Constraint-Netz, einer Menge von Meta-Variablen sowie Domänen der Meta-Variablen. Abbildung 4.2 gibt einen schematischen Überblick über diesen hierarchischen Aufbau eines Meta-CSPs.

Nach Mansouri und Pecora [104] sind Meta-Constraints folgendermaßen definiert:

Definition 4.1. Ein *Meta-Constraint* ist ein Tripel (N, Ξ, Δ), für das gilt:

- N ist ein Constraint-Netz.
- $\Xi = \{\xi_1, \ldots, \xi_n\}$ ist eine Menge von Meta-Variablen, die jeweils Konflikte repräsentieren. Jede Meta-Variable ξ_i ist ein Constraint-Netz mit Constraints und Variablen aus N.
- $\Delta = \{\delta(\xi_1), \ldots, \delta(\xi_n)\}$ ist eine Menge von Domänen für die einzelnen Meta-Variablen. Dabei ist $\delta(\xi_i)$ wiederum jeweils eine Menge von Constraint-Netzen, die den von der Meta-Variable ξ_i repräsentierten Konflikt auflösen.

Meta-Variablen sind die Grundelemente eines Meta-Constraints. Sie sind Teile des gemeinsamen Constraint-Netzes N, für welche die Anforderungen, die von dem Meta-Constraint repräsentiert werden, nicht erfüllt sind. Somit handelt es sich bei ihnen wiederum um Constraint-Netze. Bei Cesta et al. waren dies Constraint-Netze jener Aktivitäten, die zeitgleich eine Ressource überbeanspruchen. Meta-Variablen werden allerdings nicht vorab explizit definiert, sondern ergeben sich zur Laufzeit prozedural aus der Ermittlung derjenigen Variablen und Constraints des gemeinsamen Constraint-Netzes, die in Konflikt mit der zu repräsentierenden Anforderung stehen.

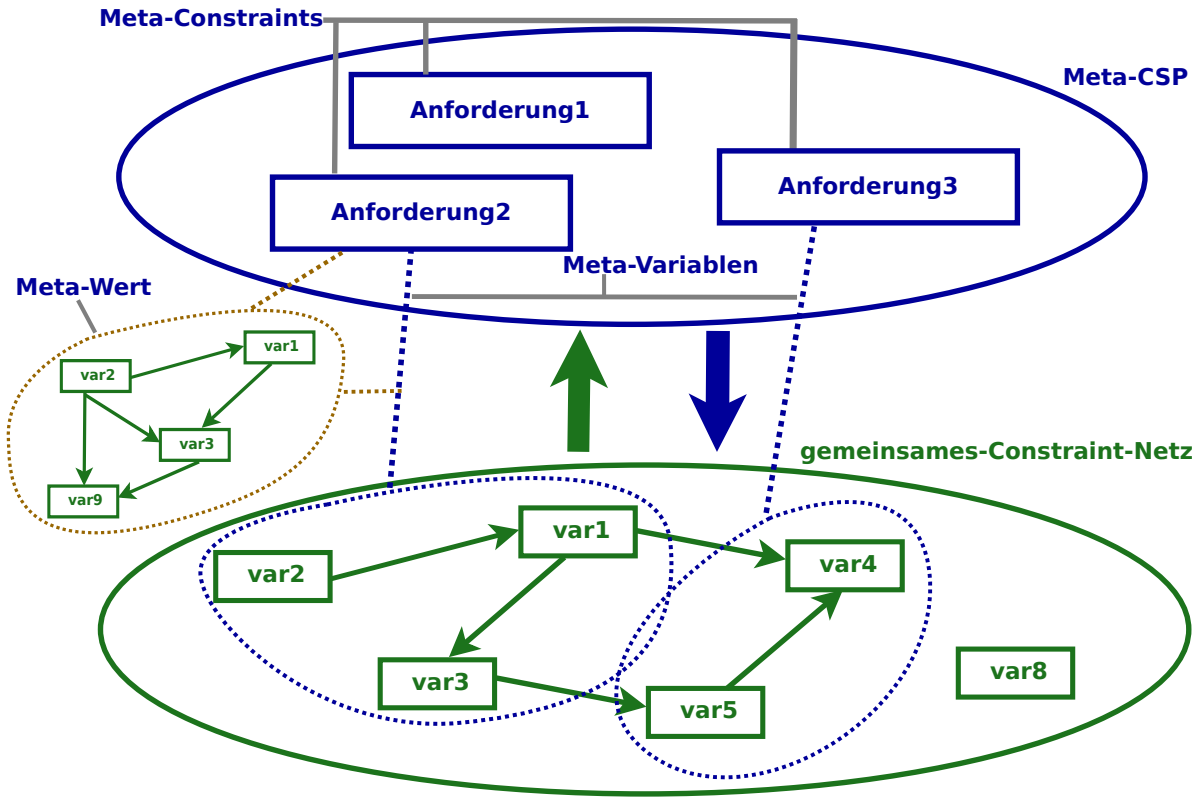


Abbildung 4.2: Überblick über die Elemente eines Meta-CSPs. Meta-Constraints repräsentieren Anforderungen, die im gemeinsamen Constraint-Netz überprüft werden. Meta-Variablen sind Teilmengen des gemeinsamen Constraint-Netzes und entsprechen Konflikten jener Anforderungen. Zum Auflösen dieser Konflikete werden Meta-Werte in das gemeinsame Constraint-Netz eingefügt.

Die Domänen der Meta-Variablen sind wiederum Constraint-Netze. Sie werden als *Meta-Werte* bezeichnet. Auch sie werden für gewöhnlich zur Laufzeit ermittelt. Ein Meta-Wert stellt eine potentielle Lösung des gegebenen Konflikts dar. Durch das Einfügen der Variablen und Constraints eines Meta-Wertes in das gemeinsame Constraint-Netz würde der Konflikt, der von der Meta-Variablen repräsentiert wird, aus Sicht der Anforderung, welche von dem Meta-Constraint repräsentiert wird, behoben. Im Unterschied zu gewöhnlichen Constraints legt jedoch nicht alleine der Meta-Constraint fest, ob eine Zuweisung eines Meta-Wertes an eine Meta-Variable gültig ist [103]. Stattdessen bedarf es nach dem Einfügen der Variablen und Constraints des Meta-Wertes in das gemeinsame Constraint-Netz einer zusätzlichen Konsistenzprüfung der Grund-CSPs. Zudem können sich daraus wiederum Konflikte in den anderen Meta-Constraints ergeben.

Durch die Verwendung mehrerer Meta-Constraints lassen sich verschiedene Anforderungen modellieren und integrieren. Auf diese Weise repräsentiert ein Meta-CSP das hybride Gesamtproblem.

Als Nächstes stellt sich die Frage nach geeigneten Suchverfahren für Lösungen eines Meta-CSPs. Das Ziel ist ein gemeinsames Constraint-Netz, das alle im Meta-CSP repräsentierten Anfor-

derungen erfüllt. Es wird nach einer Zuweisung von Meta-Werten zu Meta-Variablen gesucht, sodass sowohl die einzelnen Meta-Constraints erfüllt als auch die im gemeinsamen Constraint-Netz enthaltenen Grund-CSPs konsistent sind.

Die Zuweisung eines Meta-Wertes an eine Meta-Variable und das damit verbundene Einfügen des Meta-Wertes in das gemeinsame Constraint-Netz kann wiederum Auswirkungen auf die anderen Meta-Constraints haben und für sie zu neuen Konflikten beziehungsweise Meta-Werten führen. Wie bereits angesprochen werden die unterschiedlichen Meta-Constraints bei der Suche abwechselnd behandelt. Auf diese Weise kann frühzeitig festgestellt werden, ob die Wahl eines bestimmten Meta-Wertes zu unlösbaren Konflikten in anderen Meta-Constraints führt. Dabei können wiederum altbekannte Verfahren zur Suche in gewöhnlichen Constraint-Netzen angewendet werden. Speziell wird hierfür eine Backtracking-Suche eingesetzt, deren Variante für Meta-CSPs in Algorithmus 4.1 gegeben ist. Die Darstellung basiert auf Arbeiten von Mansouri und Pecora [104].

Algorithmus 4.1 Suche einer Lösung des Meta-CSPs mittels Backtracking.

```
 1: Prozedur BACKTRACK(M)
 2:     Ξ ← ERMITTLE-META-VARIABLEN(M)
 3:     falls Ξ ≠ ∅ dann
 4:         ξ ← WÄHLE(Ξ, h_var)
 5:         Werte ← δ(ξ)
 6:         solange Werte ≠ ∅ führe aus
 7:             w ← WÄHLE(Werte, h_val)
 8:             falls alle Grund-CSPs von (M ∪ w) konsistent sind, dann
 9:                 Ergebnis ← BACKTRACK(M ∪ w)
10:                 falls Ergebnis = wahr dann
11:                     gib zurück Ergebnis
12:                 Ende falls
13:             Ende falls
14:             Werte ← Werte \ w
15:         Ende solange
16:         gib zurück falsch
17:     Ende falls
18:     gib zurück wahr
19: Ende Prozedur
```

Der Algorithmus bekommt das gemeinsame Constraint-Netz M als Eingabe. Als Erstes werden die Meta-Variablen der Meta-Constraints auf Basis des gemeinsamen Constraint-Netzes ermittelt (Zeile 2). Sofern keine Konflikte existieren, ist das Gesamtproblem gelöst (Zeilen 3 und 18). Andernfalls wird eine Meta-Variable ξ ausgewählt, deren Konflikt es als Nächstes zu lösen gilt. Hierfür werden nacheinander mögliche Meta-Werte für sie ausgewählt und getestet (Zeile 7). Diese Entscheidungen werden mit Hilfe der Ordnungsheuristik h_{var} für die Meta-Variablen sowie h_{val} für die Meta-Werte getroffen. Nach der Auswahl eines Meta-Wertes w wird dieser in das gemeinsame Constraint-Netz eingefügt und eine Konsistenzprüfung aller Grund-CSPs durchge-

führt (Zeile 8). Im Erfolgsfall wird die Funktion `Backtrack` rekursiv mit dem aktualisierten gemeinsamen Constraint-Netz aufgerufen (Zeile 9). Wenn `Backtrack` das Ergebnis *wahr* liefert, wird dieser Wert wiederum direkt zurück gegeben. Wenn das Ergebnis hingegen den Wert *falsch* hat oder bereits die vorhergehende Konsistenzprüfung nicht erfolgreich war, wird auf gleiche Weise mit dem nächsten Meta-Wert fortgefahren. Sollten in diesem Fall auch die restlichen Meta-Werte zu keiner Lösung führen, wird als Ergebnis *falsch* zurück geliefert. An dieser Stelle unterscheidet sich die Darstellung des Algorithmus von den zitierten Arbeiten [95, 102, 104]. In jenen sind die Darstellungen fehlerhaft, da in ihnen das Ergebnis des rekursiven Funktionsaufrufs von `Backtrack` direkt zurück gegeben wird, anstatt die übrigen Möglichkeiten zu prüfen. In diesem Punkt entspricht die Backtracking-Suche für Meta-CSPs der Variante für gewöhnliche CSPs [145, Kapitel 6.3]. Es wird somit ein Backtracking über alle Meta-Werte jedoch nicht über die Wahl der Meta-Variablen durchgeführt.

Neben der Backtracking-Suche sind auch andere bekannte Suchverfahren für Constraint-Netze wie beispielsweise Backjumping [36] denkbar. Diese wurden jedoch bisher noch nicht ausreichend für Meta-CSPs untersucht und implementiert.

Meta-CSPs können auch selbst wiederum die Basis für andere Meta-CSPs sein, indem sie in diesen als Grund-CSP eingesetzt werden. Auf diese Weise kann eine ganze Hierarchie von Meta-CSPs erstellt werden. Bereits implementierte Meta-CSPs können als Bausteine zur Modellierung anderer Problemstellungen weiter verwendet werden. Die technische Implementierung und die Wiederverwendung von Meta-CSPs soll mit dem *Meta-CSP-Framework*, auf welches im folgenden Abschnitt eingegangen wird, erleichtert werden.

4.3.3 Das Meta-CSP-Framework

Die Repräsentation als Meta-CSP ist ein allgemeines Verfahren zum Lösen hybrider Problemstellungen. Einen wichtigen Schritt in der Umsetzung nimmt die Modellierung der Meta-Constraints und der zu Grunde liegenden Constraint-Netze ein. Anschließend gilt es, diese in Programmcode zu implementieren. Durch den allgemeinen Lösungsansatz für Meta-CSPs können für spezielle Problemstellungen bestehende Komponenten wie beispielsweise die Backtracking-Suche für Meta-CSPs oder auch spezielle Constraint-Löser wieder verwendet werden. Aus diesem Grund hat es sich das *Meta-CSP-Framework* [130] zum Ziel gesetzt, die Implementierung durch die Bereitstellung von Basisfunktionalitäten zu erleichtern. Geschrieben wurde es in der Programmiersprache Java und liegt unter der MIT-Lizenz als freie Software vor. Ein Großteil der Entwicklung des Frameworks erfolgte durch die Projektpartner in Örebro im Rahmen von RACE.

Das Framework gibt ein Grundgerüst in Form abstrakter Basisklassen für Meta-Constraints, Meta-Variablen, Constraints, Constraint-Löser et cetera vor. Dadurch müssen idealerweise für die Implementierung neuer Meta-Constraints lediglich wenige Methoden wie beispielsweise für die Ermittlung der Meta-Variablen und Meta-Werte implementiert werden. Diese werden im Rahmen der ebenso bereits vorhandenen Backtracking-Suche aufgerufen.

Darüber hinaus bietet das Meta-CSP-Framework eine Vielzahl von Constraint-Lösern für spezielle Problemtypen, die wiederum zur Beschreibung anderer Meta-CSPs genutzt werden können.

In der aktuellen Master-Version[1] `25c5eea` vom 14.09.2016 sind bereits einige Constraint-Löser sowohl für gewöhnliche CSPs als auch für Meta-CSPs vorhanden. Wie teilweise in [103] für eine frühere Version des Frameworks aufgeführt, existieren einfache Constraint-Löser für

- Instanzen eines Simple Temporal Problems (STP) [37];
- Probleme bestehend aus Constraints aus Allens Intervallalgebra [2];
- CSPs, die im Region Connection Calculus [140] in der Variante RCC-8 formuliert sind;
- CSPs, die aus Relationen des Dimensionally Extended nine-Intersection Model (DE-9IM) [32] bestehen;
- das Erfüllbarkeitsproblem der Aussagenlogik.

Zusätzlich sind Constraint-Löser höherer Ordnung vorhanden für

- erweiterte Allen Intervall Constraints mit metrischen Grenzen;
- das Temporal Constraint Satisfaction Problem (TCSP)[37];
- die Rectangle Algebra (RA) [5] sowie die Augmented Rectangle Algebra (ARA^+) [101] und die Block Algebra [4].

Zudem gibt es unterschiedliche Planer und Constraint-Löser für komplexere integrierte Problemstellungen. So existiert bereits ein Constraint-Löser für die in Abschnitt 4.3.1 beschriebenen RCPSPs und das auf ähnliche Weise funktionierende Scheduling von Zustandsvariablen. Außerdem gibt es ein System zur Kontexterkennung [131], das beispielsweise zur Erkennung häuslicher Tätigkeiten wie Kochen genutzt werden kann.

Neben gewöhnlichen Constraints und Meta-Constraints unterscheidet das Meta-CSP-Framework auch *Multi-Constraints*, welche sich aus mehreren anderen Constraints zusammensetzen. Beispielsweise könnte ein Multi-Constraint intern aus einem temporalen und einem symbolischen Constraint bestehen. Der übergeordnete Constraint-Löser würde in diesem Fall nur einen solchen Multi-Constraint in das gemeinsame Constraint-Netz einfügen, wodurch die internen Constraints auf die jeweiligen Grund-CSPs verteilt werden. Dadurch kann die Übersichtlichkeit des gemeinsamen Constraint-Netzes verbessert und die Implementierung der Meta-Constraint-Löser erleichtert werden. Der übergeordnete Constraint-Löser braucht somit nur einen Multi-Constraint, der intern aus mehreren speziellen Constraints in das gemeinsame Constraint-Netz einfügen beziehungsweise beim Backtracking wieder entfernen, anstatt separate Constraints für die unterschiedlichen Dimensionen erstellen zu müssen. Darüber hinaus werden *Multi-Variablen* verwendet, um die verschiedenartigen Variablen mehrerer spezieller Grund-CSPs in einer gemeinsamen Variablen zusammenzufassen und im gemeinsamen Constraint-Netz zu repräsentieren. Auf diese Weise lassen sich auch Variablen beziehungsweise Constraints gleichen Typs zu einer höherdimensionalen Variablen beziehungsweise einem höherdimensionalen Constraint komponieren. Zum Beispiel sind die Variablen und Constraints der Rectangle Algebra aus jeweils zwei Variablen und Constraints aus Allens Intervallalgebra implementiert.

[1] `https://github.com/FedericoPecora/meta-csp-framework`

4.3.4 Anwendungen des Meta-CSP-Frameworks

Das Meta-CSP-Framework wurde bereits für die Implementierung verschiedener hybrider Planungs- oder Schlussfolgerungssysteme eingesetzt. Die für die vorliegende Arbeit relevanteste Anwendung ist der bereits genannte Planer von Mansouri und Pecora [104]. Dieser Planer kombiniert temporales, geometrisches und kausales Planen sowie das Scheduling von Ressourcen. Das Planungssystem wurde für das Decken eines Tisches mit einem einarmigen und einem zweiarmigen Roboter demonstriert. Als Teil ihrer Aufgabe mussten die Roboter jeweils sicherstellen, dass sie das Besteck und Geschirr korrekt zueinander auf dem Tisch platzierten. Hierfür wurde in der genannten Arbeit mit ARA$^+$ zudem ein Kalkül für die Repräsentation von sowohl metrischen als auch qualitativen geometrischen Informationen entwickelt.

Das Meta-CSP-Framework wird derzeit auch für die Entwicklung realer Anwendungen eingesetzt. In [99] wird ein Ansatz vorgestellt, in dem die Koordinierung von Gabelstaplern in Industrieanwendungen in einem Meta-CSP modelliert wird. Die Planung umfasst die Zuweisung von Aufgaben an Gabelstapler, deren Koordinierung bis hin zur Bewegungsplanung. Die einzelnen Teilaspekte werden dabei in Form von Meta-Constraints modelliert. Als Grund-CSPs dienen ein temporales und ein räumliches Constraint-Netz.

Eine weitere reale Anwendung ist die Koordinierung von Maschinen für die Bohrung von Sprenglöchern im Tagebau [100]. Auch hier müssen den Maschinen wiederum Aufgaben in Form von Sprenglöchern, die zu bohren sind, zugewiesen und Trajektorien berechnet werden, die mehrere spezielle Nebenbedingungen der Anwendung berücksichtigen.

Des Weiteren wurde das Meta-CSP-Framework für die Konfigurationsplanung mehrerer Robotersysteme in einem Forschungsprojekt zum Einsatz von Robotern in der Altenpflege eingesetzt [42], und hybride Verfahren wurden ebenfalls für die Kontexterkennung von Aktivitäten in einer Haushaltsumgebung verwendet und jener Kontext für die Planung eines Roboters genutzt [131, 168].

Eng verbunden mit dem Meta-CSP-Verfahren sind auch Arbeiten von Köckemann, dessen Dissertation sich ausführlich mit dem Einsatz Constraint-basierter Planungsmethoden für Aufgaben, die die Interaktion mit und Berücksichtigung von Menschen involvieren, auseinandersetzt [92]. In dem von ihm vorgeschlagenen Framework für menschennahe Planung, *Human-aware Planner and Executive (HaPlEx)*, werden ebenfalls unterschiedliche Arten von Planern und Reasonern integriert. Neben heuristischer und temporaler Planung sowie Scheduling werden auch Prolog-Constraints für die Modellierung statischer Beziehungen zwischen Objekten sowie Kosten zur Modellierung der sozialen Akzeptanz von Aktionen integriert.

Kapitel 5

CHIMP: Ein hierarchischer hybrider Planer für mobile Roboter

Nachdem in den beiden vorhergehenden Kapiteln die Verwendung des HTN-Planers SHOP2 in einem Robotersystem sowie die hybride Planung mit dem Meta-CSP-Ansatz besprochen wurde, wird in diesem Kapitel der neue hierarchische hybride Planer CHIMP vorgestellt. Während SHOP2 bereits einige Vorteile für den Einsatz auf einem mobilen Roboter und insbesondere für das Lernen aus Erfahrungen in RACE bot, war dies auf rein kausales Planen beschränkt. Bei dem hybriden Planer von Mansouri und Pecora, der ebenfalls in RACE entwickelt wurde, ist hingegen die kausale Planung relativ eingeschränkt. Zudem führte die hybride Planung zu einem vergrößerten Suchraum. CHIMP kombiniert die Vorteile, die beide Verfahren für mobile Roboter haben, indem er eine hybride Basisrepräsentation als Meta-CSP verwendet und die HTN-Dekompositionsstrategie als weiteren Meta-Constraint implementiert. Abbildung 5.1 stellt dies schematisch dar.

Dieses Kapitel beschreibt die Implementierung von CHIMP als Meta-CSP. Dafür wird zunächst detailliert auf die zugrunde liegende Repräsentation der Variablen des Grund-CSPs eingegangen und anschließend die Umsetzung der HTN-Dekomposition als Meta-Constraint dargelegt. Die aktuelle Version von CHIMP ist unter `http://sebastianstock.github.io/chimp` frei verfügbar und steht unter der MIT-Lizenz.

5.1 Repräsentation

5.1.1 Fluenten

CHIMP verwendet *Fluenten* als Basisbausteine zur Repräsentation seines internen Zustands und seiner Pläne. Mit Fluenten werden Roboteraktionen, Aufgaben sowie Fakten über den Zustand des Roboters und seiner Umgebung repräsentiert. Sie bilden die Variablen des gemeinsamen Constraint-Netzes.

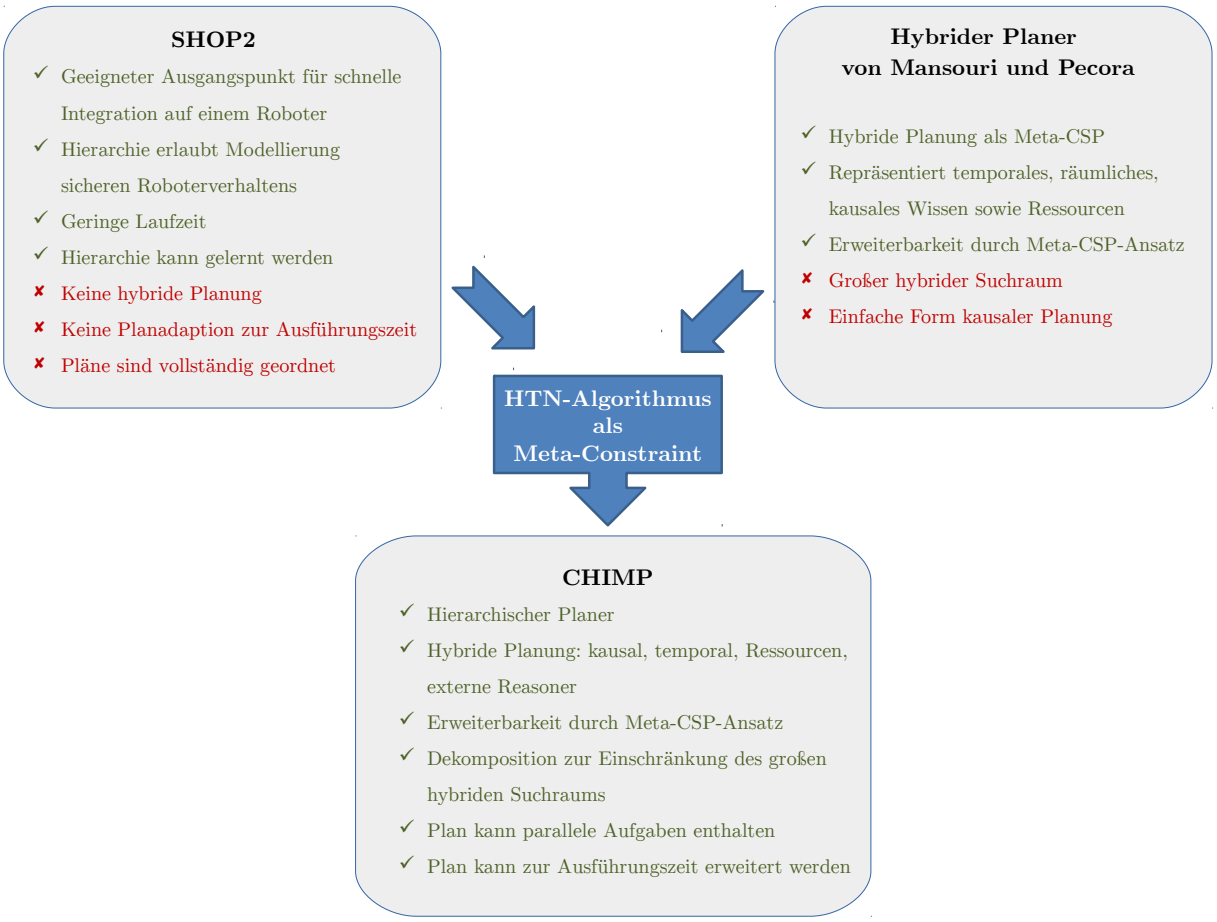


Abbildung 5.1: Durch die Implementierung der HTN-Aufgaben-Dekomposition als Meta-Constraint kombiniert CHIMP die Vorteile von SHOP2 und dem hybriden Planer von Mansouri und Pecora [102].

Definition 5.1. Ein *Fluent* f ist ein Tupel $(\mathrm{P}, \mathrm{S}, I, u)$, für das gilt:

- $\mathrm{P} \in \mathrm{D}_{\text{Aufgaben}} \cup \mathrm{D}_{\text{Zustände}}$ ist ein *Prädikatensymbol*, wobei $\mathrm{D}_{\text{Aufgaben}}$ eine endliche Menge von Aufgabensymbolen (f ist ein *Aufgabenfluent*) und $\mathrm{D}_{\text{Zustände}}$ eine endliche Menge von Zustandssymbolen (f ist ein *Zustandsfluent*) ist.

- $\mathrm{S} = \{\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}\}$ ist eine endliche Menge symbolischer Variablen mit Domänen aus $\mathrm{D}_1, \ldots, \mathrm{D}_\mathrm{n}$, welche jeweils den Typ der Variablen repräsentieren.

- $I = [I_s, I_e]$ ist ein *flexibles temporales Intervall*, innerhalb dessen P den Wert `wahr` hat. Dabei sind $I_s = [l_s, u_s], I_e = [l_e, u_e], l_{s/e}, u_{s/e} \in \mathbb{N}$ jeweils zulässige Intervalle der Start- und Endzeitpunkte des Fluenten.

- $u\colon \mathcal{R} \to \mathbb{N}$ bildet *wiederverwendbare Ressourcen* auf ihre *Nutzung* ab. Hierbei ist $\mathcal{R}$ die Menge aller Ressourcenbezeichner.

Das Prädikatensymbol und die symbolischen Variablen eines Fluenten werden im Folgenden in der Form $P(S_1, \ldots, S_n)$ geschrieben. Dabei sei S_i die i-te symbolische Variable von S. Zum Beispiel gibt der folgende Aufgabenfluent an, dass der Roboter zum Zeitpunkt 10 beginnt, den Becher `mug1` vom Tisch `table1` zu greifen, und diese Aufgabe in einem Zeitintervall zwischen den Zeitpunkten 30 und 50 abschließt. Dabei verwendet er eine Einheit der Ressource `arm`.

$$f_1 = (\text{Pick}(\text{mug1}, \text{table1}), [[10, 10], [30, 50]], u(\text{arm}) = 1)$$

Ebenso kann der Fakt, dass sich `mug1` ab dem Zeitpunkt 100 und mindestens bis zum Zeitpunkt 150 auf der Fläche `placingAreaEastRightCounter1` befindet, mit folgendem Zustandsfluenten angegeben werden:

$$f_2 = (\text{On}(\text{mug1}, \text{placingAreaEastRightCounter1}), [[100, 100], [150, \infty]], \emptyset)$$

Fluenten werden von CHIMP als Variablen des gemeinsamen Constraint-Netzes eingesetzt, welches in dieser Arbeit folgendermaßen definiert sei:

Definition 5.2. Ein *Constraint-Netz* ist ein Paar $(\mathcal{F}, \mathcal{C})$, das aus einer Menge *Fluenten* $\mathcal{F}$ und einer Menge *Constraints* $\mathcal{C}$ zwischen Fluenten aus $\mathcal{F}$ besteht.

Gemäß dem Meta-CSP-Ansatz kann ein solches gemeinsames Constraint-Netz verschiedene Arten von Constraints enthalten und dient somit als gemeinsame Repräsentationsform verschiedener spezieller Constraintlöser. CHIMP repräsentiert den Ausgangszustand inklusive der zu erfüllenden Aufgaben als Constraint-Netz und fügt während der Planung zusätzliche Variablen und Constraints in dieses hinzu. Somit repräsentiert CHIMP auch seinen internen Zustand und die resultierenden Pläne mittels Constraint-Netzen. Als Nächstes wird auf die verschiedenen Arten von Constraints und die sich daraus ergebenden speziellen Constraint-Netze eingegangen.

5.1.2 Symbolische Constraints

Die Domänen der symbolischen Variablen der Fluenten können mittels *symbolischer* Constraints eingeschränkt werden. $B_\text{S} = \{=, \neq, \in, \notin\}$ sei die Menge der symbolischen Constraints. Bei $=$ und $\neq$ handelt es sich um binäre symbolische Constraints zwischen zwei Fluenten. Mit $=$ werden die Domänen zweier symbolischer Variablen gleichgesetzt, während mit $\neq$ festgelegt wird, dass deren Elemente paarweise verschieden sind. Bei den anderen beiden Constraintarten handelt es sich hingegen um unäre Constraints. $\in$ und $\notin$ geben an, dass der Wert der symbolischen Variable ein Element beziehungsweise kein Element einer gegebenen Menge ist. Constraints vom Typ $\in$ werden unter anderem intern eingesetzt, um die symbolische Domäne einer Variable auf einen bestimmten Typ einzuschränken. Auf diese Weise kann beispielsweise ein Aufgabenfluent vom Typ `get_object` zur Repräsentation der Aufgabe, ein unbestimmtes Objekt vom Typ `Tasse` zu holen, verwendet werden, indem dessen symbolische Variable mit einem $\in$-Constraint auf die Menge aller Tassen $\{\text{mug1}, \text{mug2}, \text{mug3}\}$ eingeschränkt wird. Diese symbolische Variable kann während der Planung zum Beispiel mit einem $=$-Constraint zur ersten symbolischen Variable von `On(mug1, table2)` an den konstanten Wert `mug1` gebunden werden.

Damit kann jetzt der Spezialfall eines symbolischen Constraint-Netzes definieren werden:

Definition 5.3. Ein *symbolisches Constraint-Netz* ist ein Paar $N = (V, SC)$, für das gilt:

- $V = \{V_1, \ldots, V_n\}$ ist eine Menge *symbolischer Variablen*.
- $SC\colon V \times V \to 2^{B_S}$ ist eine *Abbildung*, welche die unären und binären symbolischen Constraints zwischen den Variablen angibt.

Da ein Fluent mehrere symbolische Variablen beinhalten kann, müssen für die symbolischen Constraints zusätzlich jeweils die Indizes der symbolischen Variablen, auf welche er sich bezieht, mit angegeben werden, damit die symbolischen Constraints direkt im gemeinsamen Constraint-Netz repräsentiert werden können.

5.1.3 Temporale Constraints

Die zeitliche Dimension der Fluenten kann mit temporalen Constraints eingeschränkt werden. Analog zu [102] verwendet dazu auch CHIMP binäre Relationen aus Allens Intervallalgebra (IA) [2]. Ein temporaler Constraint $(f_1\ r\ f_2)$ zwischen zwei Fluenten f_1 und f_2 schränkt deren flexible temporale Intervalle ein. Für die temporalen Intervalle eines Fluenten f wird in dieser Arbeit die Notation $I^{(f)}$ verwendet. Ein temporaler Constraint im gemeinsamen Constraint-Netz wird an die Intervalle $I^{(f_1)}$ und $I^{(f_2)}$ delegiert.

Allens Intervallalgebra definiert 13 atomare Relationen zur Beschreibung der zeitlichen Beziehung zweier Intervalle. Für diese werden im Folgenden die in der Literatur gebräuchlichen englischen Namen und Abkürzungen verwendet. Diese sind „before“ (b), „meets“ (m), „overlaps“ (o), „during“ (d), „starts“ (s), „finishes“ (f) sowie deren Inversen (zum Beispiel b^{-1}) und „equals“ (≡). An Stelle der klassischen IA-Relationen verwendet CHIMP eine Erweiterung mit zusätzlichen metrischen Grenzen. Es können dadurch sowohl qualitative als auch quantitative Informationen über Fluenten und ihre Beziehungen untereinander repräsentiert werden [111]. Die atomaren IA-Relationen sind schematisch in Abbildung 5.2 dargestellt. Beispielsweise gibt die Relation $f_1\ \{\mathrm{b}\}\ f_2$ an, dass der Endzeitpunkt des Fluenten f_1 kleiner zu sein hat, als der Startzeitpunkt des Fluenten f_2. Mit der optionalen Angabe eines Intervalls kann zudem die minimale und maximale Differenz dieser Zeitpunkte spezifiziert werden.

B_{IA} sei die Menge dieser 13 atomaren IA-Relationen. Beliebige Relationen in Allens Intervallalgebra ergeben sich nun als Disjunktion mehrerer Relationen aus B_{IA}. Zum Beispiel fordert die Disjunktion $\{\mathrm{m}, \mathrm{o}, \mathrm{f}^{-1}, \mathrm{d}^{-1}\}$ zwischen zwei Fluenten f_1 und f_2, dass f_1 bereits vor f_2 begonnen hat und frühestens zeitgleich mit dem Start von f_2 endet (m), innerhalb von f_2 endet (o), zeitgleich mit f_2 endet (f^{-1}) oder erst nach f_2 endet (d^{-1}). Somit eignet sich diese Disjunktion zu einer allgemeinen temporalen Repräsentation von Vorbedingungen. Eine alleinige atomare Relation ist dafür teilweise zu restriktiv. So könnte beispielsweise die Relation f^{-1} für die Repräsentation eines negativen Effekts genutzt werden. Der Zustandsfluent müsste hierbei genau zeitgleich mit dem Aufgabenfluenten enden. Auch wenn dies in der Theorie die richtige Wahl sein sollte, kann es in der Praxis jedoch zu Problemen in der Planausführung führen. Wenn die

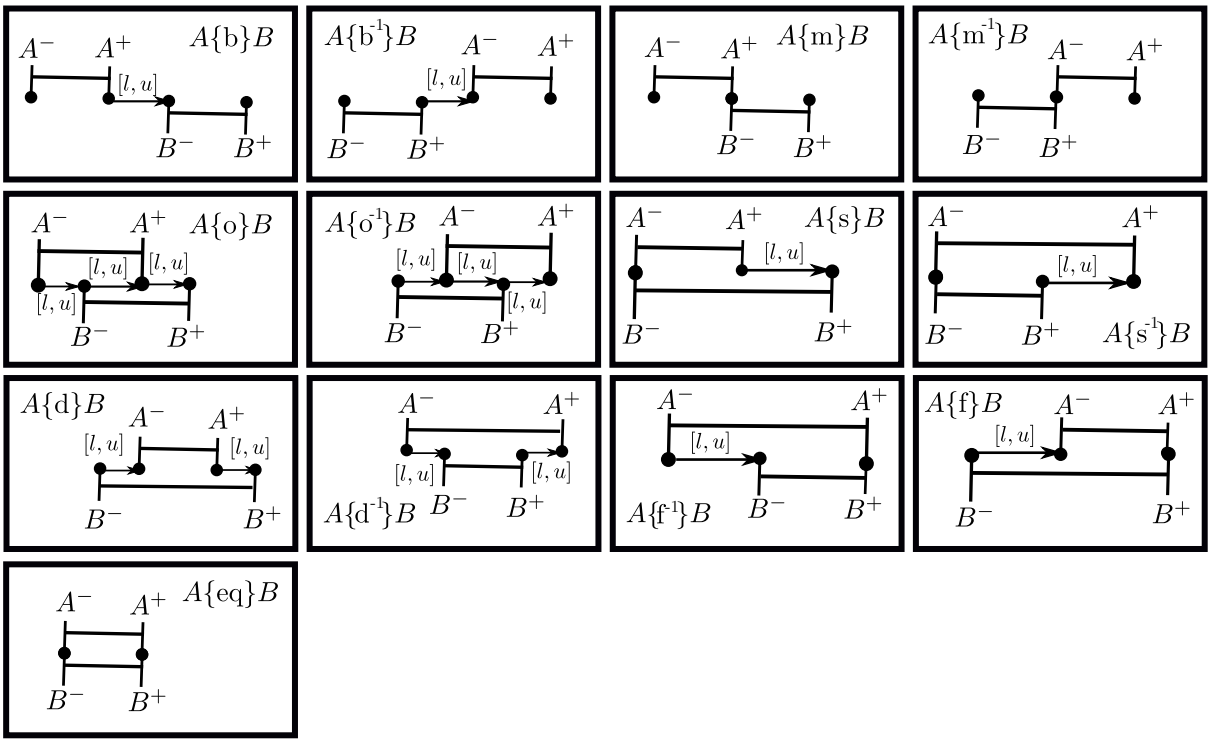


Abbildung 5.2: Die 13 Basisrelationen aus Allens Intervallalgebra mit zusätzlichen metrischen Grenzen. Die Darstellung wurde aus [101] übernommen und angepasst.

Zeitstempel der beiden Fluenten von zwei unterschiedlichen Modulen gesetzt werden, führt dies leicht zu geringen zeitlichen Unterschieden, wodurch die sehr strikte f^{-1}-Relation nicht mehr erfüllt ist.

Aus den 13 atomaren Relationen ergeben sich als Disjunktionen $2^{13} = 8192$ Relationen. Anhand einer Kompositionstabelle lassen sich Aussagen über die Komposition von Relationen zwischen mehreren Variablen ermitteln. Im Allgemeinen ist die Erfüllbarkeitsprüfung in Allens Intervallalgebra jedoch NP-schwer [171]. Aus diesem Grund werden in dieser Arbeit wie in [89, 102] nur *konvexe* IA-Relationen [97, 124] verwendet. Für diese Untermenge ist die Erfüllbarkeitsprüfung durch Berechnung der Pfadkonsistenz in polynomieller Zeit möglich [97]. Intuitiv handelt es sich bei den konvexen IA-Relationen um die Disjunktionen von IA-Relationen ohne Lücke. So ist beispielsweise die zuvor erwähnte Disjunktion $\{\mathrm{m}, \mathrm{o}, \mathrm{f}^{-1}, \mathrm{d}^{-1}\}$ konvex, während $\{\mathrm{m}, \mathrm{f}^{-1}\}$ hingegen nicht konvex ist.

Ein temporales Constraint-Netz, welches von CHIMP als weiteres Grund-CSP verwendet wird, ist folgendermaßen definiert:

Definition 5.4. Ein *temporales Constraint-Netz* ist ein Paar $N = (V, TC)$, für das gilt:

- $V = \{V_1, \ldots, V_n\}$ ist eine Menge von *Variablen*, welche flexible temporale Intervalle repräsentieren.

- $TC\colon V \times V \to 2^{B_{\mathrm{IA}}}$ ist ein *Abbildung*, welche die binären Constraints zwischen den Variablen angibt. Dabei seien $2^{B_{\mathrm{IA}}}$ nur die konvexen IA-Relationen.

Ein Constraint-Netz $M = (\mathcal{F}, \mathcal{C})$ *subsumiert* ein *temporales Constraint-Netz* $M_t = (V, TC)$, wenn gilt: $V = \{I^{(f)} \mid f \in \mathcal{F}\}$ und $TC = \{I^{(f_1)}\ r\ I^{(f_2)} \mid (f_1\ r\ f_2) \in \mathcal{C}\}$ [102].

5.1.4 Kausale Constraints

Temporale und symbolische Constraints dienen zur Repräsentation der jeweils spezifischen Wissensformen und bilden so die Basis für CHIMPs Wissensrepräsentation. Darüber hinaus verwendet CHIMP mit *kausalen* Constraints eine weitere Repräsentationsform, die hierarchisch über diesen angesiedelt ist. Kausale Constraints werden primär bei der HTN-Planung zur Repräsentation des kausalen Domänenwissens genutzt. Wie im nachfolgenden Abschnitt 5.1.5 beschrieben wird, bestehen sie intern teilweise aus zusätzlichen symbolischen und temporalen Constraints. $B_{\mathrm{C}} = \{\mathrm{dc}, \mathrm{pre}, \mathrm{opens}, \mathrm{closes}, \mathrm{planned}, \mathrm{ordering}, \mathrm{matches}, \mathrm{moveduration}\}$ sei die Menge der kausalen Relationen zwischen Fluenten. Diese Relationen haben folgende Bedeutungen:

t_1 dc t_2: Der Aufgabenfluent t_2 ist eine Teilaufgabe des Aufgabenfluenten t_1.

f pre t: Der Zustandsfluent f ist eine Vorbedingung des Aufgabenfluenten t.

t opens f: Der Zustandsfluent f ist ein positiver Effekt des Aufgabenfluenten t.

t closes f: Der Zustandsfluent f ist ein negativer Effekt des Aufgabenfluenten t.

planned t: Der Aufgabenfluent t wurde bereits geplant.

t_1 ordering t_2: Der Aufgabenfluent t_1 muss vor dem Aufgabenfluenten t_2 geplant werden. Diese Bedingung wirkt sich darauf aus, in welcher Reihenfolge die Aufgaben vom Planer behandelt werden. Sie hat keine Auswirkungen auf die Ausführung.

t_1 matches t_2: Der Aufgabenfluent t_1 wird mit dem Aufgabenfluenten t_2 unifiziert. Dadurch ist es möglich, dass unterschiedliche Aufgabenfluenten dieselben Teilaufgaben haben und sie sich somit teilen können.

moveduration t: Für den Aufgabenfluenten t vom Typ `!move_base` wurde bereits die minimale Fahrtdauer ermittelt.

Mit diesen Relationen kann ein kausales Constraint-Netz definiert werden:

Definition 5.5. Ein *kausales Constraint-Netz* ist ein Paar $N = (\mathcal{F}, CC)$, für das gilt:

- $\mathcal{F}$ ist eine Menge von *Fluenten*;
- $CC\colon \mathcal{F} \times \mathcal{F} \to 2^{B_{\mathrm{C}}}$ ist eine *Abbildung*, die angibt, welche kausalen Constraints zwischen Paaren von Fluenten existieren.

Damit *subsumiert* ein Constraint-Netz $M = (\mathcal{F}, \mathcal{C})$ ein *kausales Constraint-Netz* $M_c = (\mathcal{F}_C, CC)$, wenn gilt: $\mathcal{F}_C \subseteq \mathcal{F}$ und $CC = \{(f_1\ r\ f_2) \mid (f_1\ r\ f_2) \in \mathcal{C} \wedge r \in B_{\mathrm{C}}\}$.

5.1.5 Kombination von Constraints

Wie in Abschnitt 4.3.3 erwähnt, erlaubt das Meta-CSP-Framework das Zusammenfassen mehrerer Constraints in einem übergeordneten Multi-Constraint. Dies ermöglicht eine einfache Identifizierung sowie ein erleichtertes Hinzufügen und Entfernen semantisch zusammengehöriger Constraints. Stehen zwei Fluenten bezüglichen einer bestimmten Wissensform in Beziehung zueinander, so existieren häufig auch weitere Relationen in den anderen Repräsentationsformen zwischen ihnen. Dazu können kausale Constraints können mehrere temporale oder symbolische Constraints beinhalten.

Beispielsweise besteht zwischen einem Fluenten, der die Aktion, ein Objekt zu greifen, repräsentiert, und dem Zustandsfluenten, der das Halten des Objekts in der Hand des Roboters beschreibt, die rein kausale Beziehung, dass der zweite Fluent ein Effekt des ersten Fluenten ist. Der zeitliche Zusammenhang ist damit noch nicht genau spezifiziert, kann jedoch mit zusätzlichen temporalen Constraints angeben werden. Schließlich existiert auch noch die symbolische Beziehung zwischen den beiden Fluenten, dass es sich bei dem Objekt, das gegriffen wird, und jenem, das der Roboter in der Hand hält, um dasselbe Objekt handeln sollte. Die letzten beiden Constraints können dafür zu dem symbolischen Constraint hinzugefügt und in dem gemeinsamen Constraint-Netz repräsentiert werden.

CHIMPs gemeinsames Constraint-Netz besteht aus Fluenten sowie kausalen, temporalen und symbolischen Constraints zwischen den Fluenten. Die temporalen und symbolischen Constraints können hierbei eigenständig oder Teil eines kausalen Multi-Constraints sein. Betrachtet man nur eine bestimmte Art von Variablen und Constraints, ergeben sich verschiedene Sichten auf das gemeinsame Constraint-Netz. Die kausalen Constraints werden dabei allein von dem Meta-Constraint, der die HTN-Planung implementiert, verwendet. Für das temporale und symbolische Constraint-Netz wird hingegen eine Konsistenzprüfung mit speziellen Inferenzverfahren durchgeführt. Das gemeinsame Constraint-Netz ist somit diesen beiden Constraint-Netzen übergeordnet. Diese Sichtweise ist in Abbildung 5.3 schematisch dargestellt. Aber auch die temporalen Constraints und Variablen sind im Meta-CSP-Framework wiederum als Multi-Constraints beziehungsweise Multi-Variablen implementiert und intern als STP umgesetzt.

5.1.6 HTN-Repräsentation

Mit den zuvor definierten Constraint-Netzen kann CHIMP den internen Zustand des Roboters und den Umgebungzustand repräsentieren. Zudem wird mit ihnen auch die Domänenbeschreibung modelliert. Da CHIMP im Gegensatz zu SHOP2 auch Ressourcen sowie Zustandsvariablen berücksichtigt, sind diese anders als in Abschnitt 3.2.2 Teil der Domänenbeschreibung, die folgendermaßen definiert ist:

Definition 5.6. Eine *CHIMP-Planungsdomäne* ist ein 4-Tupel $\mathcal{D} = (\mathcal{O}, \mathcal{M}, \mathcal{RC}, \mathcal{SV})$, das aus einer Menge von Operatoren $\mathcal{O}$, einer Menge von Methoden $\mathcal{M}$, einer Menge von Ressourcen $\mathcal{RC}$ und einer Menge von Zustandsvariablen $\mathcal{SV}$ besteht.

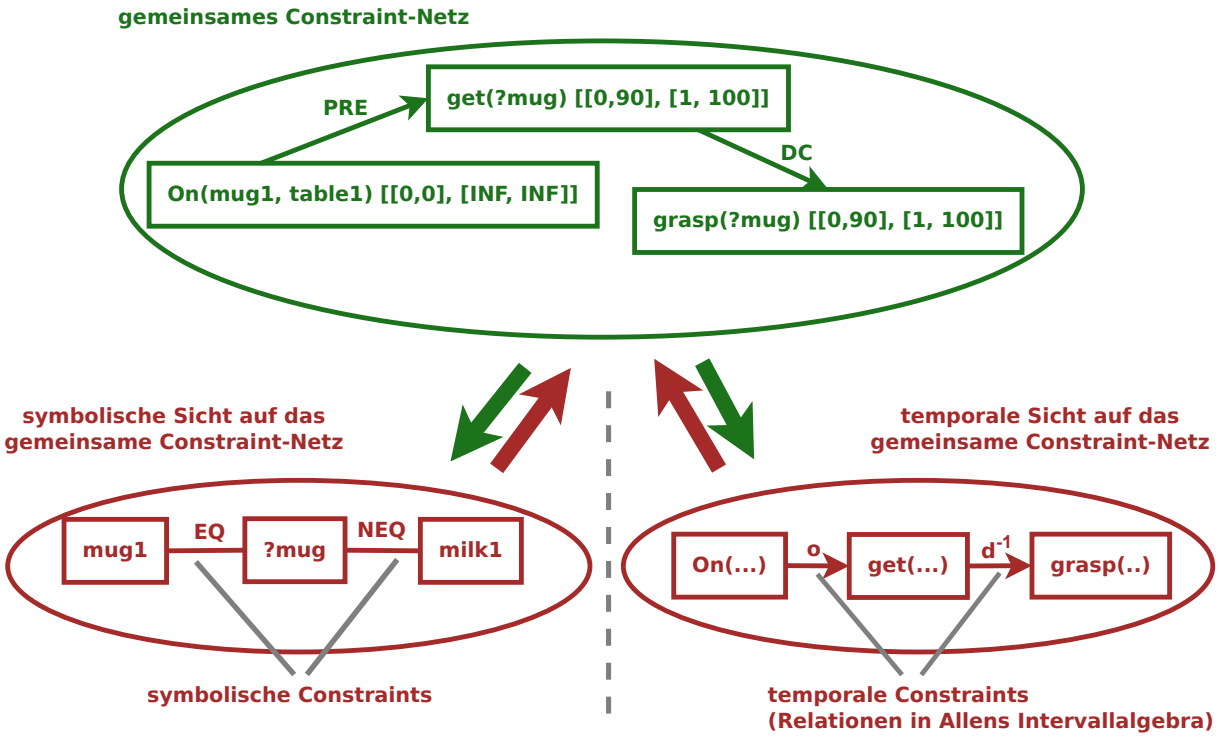


Abbildung 5.3: Das gemeinsame Constraint-Netz setzt sich aus einem symbolischen und einem temporalen Constraint-Netz zusammen. Zudem enthält es kausale Constraints, welche als Repräsentationsform für einen übergeordneten Meta-Constraint dienen.

Das Planungsproblem repräsentiert CHIMP mit Hilfe eines Constraint-Netzes:

Definition 5.7. Ein *CHIMP-Planungsproblem* ist ein Tripel $\mathcal{P} = ((\mathcal{F}, \mathcal{C}), \mathcal{T}, \mathcal{D})$. Dabei ist $(\mathcal{F}, \mathcal{C})$ ein Constraint-Netz, das aus Fluenten $\mathcal{F}$, welche den initialen Zustand und die Aufgaben beschreiben, und kausalen, temporalen sowie symbolischen Constraints $\mathcal{C}$ besteht. Weiterhin ist $\mathcal{T} \subseteq \mathcal{F}$ eine Menge von Aufgabenfluenten, die es zu erfüllen gilt, und $\mathcal{D}$ ist eine CHIMP-Planungsdomäne.

Fluenten werden somit auch für die Repräsentation der zu erfüllenden Aufgaben und der Ausgangssituation verwendet. Beispielsweise repräsentiert der Aufgabenfluent

$$f_G = (\text{serve-coffee}(\text{guest1}), [[0, \infty), [0, 600]], \emptyset)$$

die Aufgabe, dass dem Gast `guest1` spätestens bis zum Zeitpunkt 600 ein Kaffee zu servieren ist. Auf gleiche Weise lässt sich die Information, dass der Roboter zu Beginn den Becher `mug1` mit einem seiner Arme hält und dieses eine Einheit der Ressource `Arm` beansprucht, mit dem folgenden Zustandsfluenten ausdrücken:

$$f_I = (\text{Hold}(\text{mug1}), [[0, 0], [1, \infty)], u(\text{Arm}) = 1).$$

Die Planung erfolgt durch Modifikation des initialen Constraint-Netzes mittels Hinzufügen zusätzlicher Constraints und Fluenten, welche die Roboteraktionen, abstrakten Aufgaben, Vorbedingungen und Effekte repräsentieren. Dieses Constraint-Netz repräsentiert somit gleichzeitig auch den resultierenden Plan.

Planungsoperatoren sind folgendermaßen mit Hilfe von Prototypen für Fluenten und Constraints definiert:

Definition 5.8. Ein *CHIMP-Operator* ist ein Tripel $(f, (F, C), w)$, für das gilt:

- $f = (\mathrm{P}, \mathrm{S}, \mathrm{I}, \mathrm{u})$ ist ein Schema-Fluent einer primitiven Aufgabe, auf die der Operator angewendet werden kann.
- F ist eine Menge von Schema-Fluenten, welche die Vor- und Nachbedingungen repräsentieren.
- $C = CC \cup BC \cup TC$ sind Mengen von kausalen, symbolischen und temporalen Schema-Constraints zwischen Schema-Fluenten aus $F \cup \{f\}$.
- $w \in \mathbb{Z}$ ist ein Wichtungsfaktor.

An Stelle richtiger Fluenten und Constraints werden Schemata dieser verwendet, die keine konkreten Instanzen sind, sondern als Prototypen vorgeben, wie die konkreten Fluenten und Constraints beschaffen sein müssen. Diese Schema-Fluenten werden im Folgenden allerdings wie gewöhnliche Fluenten behandelt.

Die Vorbedingungen und Effekte werden mit den in Abschnitt 5.1.4 beschriebenen Relationen `pre` beziehungsweise `opens` und `closes` repräsentiert. Damit ergeben sich die Mengen der Vorbedingungsfluenten $F_p = \{f_i \mid f_i \in F \wedge (f_i \,\mathrm{pre}\, f) \in CC\}$, der Fluenten negativer Effekte $F_- = \{f_i \mid f_i \in F_p \wedge (f \,\mathrm{closes}\, f_i) \in CC\}$ sowie positiver Effekte $F_+ = \{f_i \mid f_i \in F \wedge (f \,\mathrm{opens}\, f_i) \in CC\}$. Wie in Kapitel 3 beginnen bei CHIMP die Namen von Operatoren beziehungsweise primitiven Aufgaben per Konvention mit einem Ausrufezeichen.

Mit der Wichtung kann angegeben werden, welche Operatoren bei der Expansion von Aufgaben präferiert werden sollen. Sie können in Werteordnungsheuristiken berücksichtigt werden. Hierauf wird in Abschnitt 5.4 genauer eingegangen. Ein Beispiel ist der folgende Operator:

$$
\begin{aligned}
f &= (!\mathrm{grasp}(?\mathrm{obj}_1),\, [[0,0],[4,\infty]], u(\mathrm{manCap}) = 1) \\
F_p &= \{f_1 = (\mathrm{On}(?\mathrm{obj}_2, ?\mathrm{loc}_1), \cdot, \cdot),\, f_2 = (\mathrm{RobotAt}(?\mathrm{loc}_2), \cdot, \cdot)\} \\
F_- &= \{f_1\} \\
F_+ &= \{f_3 = (\mathrm{Holding}(?\mathrm{obj}_3), \cdot, u(\mathrm{Arm}) = 1)\} \\
CC &= \{f_1 \,\mathrm{pre}\, f,\, f_2 \,\mathrm{pre}\, f,\, f \,\mathrm{closes}\, f_1,\, f \,\mathrm{opens}\, f_3,\, f \,\mathrm{planned}\, f\} \\
BC &= \{S_1^{(f)} = S_1^{(f_1)},\, S_2^{(f_1)} = S_1^{(f_2)},\, S_1^{(f_1)} = S_1^{(f_3)}\} \\
TC &= \{I^{(f)}\, \mathrm{o}^{-1}\, I^{(f_1)},\, I^{(f)}\, \mathrm{d}\, I^{(f_2)},\, I^{(f)}\, \mathrm{o}\, I^{(f_3)},\, I^{(f_1)}\, \mathrm{m}\, I^{(f_3)}\} \\
w &= 1
\end{aligned}
\tag{5.1}
$$

Dieser Operator kann auf Aufgaben mit dem Namen `!grasp` angewendet werden. Er repräsentiert damit die kausalen, symbolischen, temporalen sowie auf Ressourcenwissen basierenden Aspekte davon, was es für den Roboter bedeutet, ein Objekt zu greifen. Die beiden Fluenten f_1 und f_2 repräsentieren die Vorbedingungen und fordern, dass der Roboter bereits bei dem Objekt ist. Die symbolischen Constraints in BC stellen sicher, dass die symbolischen Variablen $?obj_1$ und $?obj_2$ sowie $?loc_1$ und $?loc_2$ jeweils identisch sind und somit dasselbe Objekt beziehungsweise denselben Ort repräsentieren. f_1 ist ein negativer Effekt des Operators. Das Objekt ist durch Anwendung des Operators nicht mehr an seinem bisherigen Ort. Stattdessen hält der Roboter es in einem seiner Greifer. Dieser positive Effekt wird von den Fluenten f_3 und dessen *opens*-Relation repräsentiert. Die genauen zeitlichen Zusammenhänge zwischen dem Fluenten und seinen Vorbedingungen und Effekten werden in der Menge temporaler Constraints TC angegeben. So muss das Objekt bereits vor der Ausführung des Operators an dem besagten Ort sein. Dies endet jedoch wieder zwischen dem Start- und Endzeitpunkt des Operators (*overlaps*). Des Weiteren ist es nicht mehr an jenem Ort, sobald der Roboter es in seinem Greifer hält (*meets*). Das Halten beginnt wiederum während der Ausführung der Aktion und endet erst nach der Aktion. Wie bereits erwähnt kann die Verwendung der *meets*-Relation während der Planausführung zu Problemen führen. Dessen Verwendung hängt damit auch davon ab, wie die Module, die die Start- und Endzeitpunkte der Fluenten schreiben, implementiert sind. Hier wäre gegebenenfalls die Verknüpfung mehrerer primitiven Relationen vorzuziehen, um für die Planausführung eine größere Flexibilität zuzulassen. Für zwei Fluenten wird ein Ressourcenverbrauch angegeben. Das Greifen selbst verwendet eine Einheit der Ressource `manCap` und der `Holding`-Fluent eine Einheit der Ressource `Arm`. Das Ausführen der zugehörigen Aktion dauert mindestens vier Sekunden. Die maximale Dauer ist auf den Wert ∞ gesetzt und damit unbegrenzt. Die Relation *planned* kennzeichnet schließlich den Aufgabenfluenten, auf den der Operator angewendet wird, derart, dass er bereits geplant wurde und nicht ein weiteres Mal betrachtet werden muss.

Neben den Operatoren benötigt CHIMP für die HTN-Planung *Methoden*, die folgendermaßen definiert werden:

Definition 5.9. Eine *CHIMP-Methode* ist ein Tripel $(f, (F, C), w)$, für das gilt:

- $f = (\mathrm{P}, \mathrm{S}, \mathrm{I}, \mathrm{u})$ ist ein Schema-Fluent für eine *komplexe Aufgabe*.
- F ist eine Menge von Schema-Fluenten, welche die Vorbedingungen und Unteraufgaben der Methode repräsentieren.
- $C = CC \cup BC \cup TC$ ist eine Menge von kausalen, symbolischen und temporalen Constraints zwischen Fluenten aus $F \cup \{f\}$.
- $w \in \mathbb{Z}$ ist ein Wichtungsfaktor.

Auch die Vorbedingungen von Methoden werden mit der kausalen Relation `pre` gekennzeichnet. Für die Teilaufgaben der Methoden wird die Relation `dc` verwendet. Damit können die beiden Mengen der Vorbedingungsfluenten $F_p = \{f_i \mid f_i \in F \wedge (f_i \,\mathrm{pre}\, f) \in CC\}$ und der Teilaufgabenfluenten $F_S = \{f_i \mid f_i \in F \wedge (f \,\mathrm{dc}\, f_i) \in CC\}$ definiert werden. Das folgende Beispiel zeigt eine Methode für die komplexe Aufgabe `get_object`, mit welcher der Roboter angewiesen wird, ein bestimmtes Objekt zu holen:

$$
\begin{aligned}
f &= (\text{get\_object}(?\text{mug}_1), [[0,0],[1,\infty]], \cdot) \\
F_p &= \{f_1 = (\text{RobotAt}(?\text{loc}_1), \cdot, \cdot), f_2 = (\text{On}(?\text{mug}_2, ?\text{loc}_2), \cdot, \cdot)\} \\
F_S &= \{f_3 = (!\text{grasp}(?\text{mug}_3), \cdot, \cdot)\} \\
CC &= \{f_1 \text{ pre } f, f_2 \text{ pre } f,\ f \text{ dc } f_3,\ \text{planned} f\} \\
BC &= \{S_1^{(f)} = S_1^{(f_2)},\ S_1^{(f)} = S_1^{(f_3)},\ S_1^{(f_1)} = S_2^{(f_2)}\} \\
TC &= \{I^{(f)} \equiv I^{(f_3)},\ I^{(f)} \text{d } I^{(f_1)},\ I^{(f)} \{\text{o, f}\}\ I^{(f_2)}\} \\
w &= 2
\end{aligned}
\tag{5.2}
$$

Als Vorbedingung wird vorausgesetzt, dass sich der Roboter bereits nahe dem gewünschten Objekt befindet. Daher hat diese Methode nur die einzige Teilaufgabe `!grasp` und fügt einen entsprechenden Fluenten zu dem Constraint-Netz hinzu. Da `get_object` in diesem Fall nur eine Teilaufgabe hat, sollten die Werte der temporalen Variablen beider Fluenten identisch sein. Dies wird mit der temporalen Relation $I^{(f)} \equiv I^{(f_3)}$ erzwungen.

Sieht eine Methode hingegen mehrere Teilaufgaben vor, können deren temporalen Relationen untereinander sowie zu der übergeordneten Aufgabe mit Hilfe der Relationen aus Allens Intervallalgebra feingranular spezifiziert werden. Diese Art von Nebenbedingungen sind allerdings allein für die temporale Reihenfolge der Aufgaben zur *Ausführungszeit* bindend. Zusätzlich kann mit den kausalen ordering-Relationen eine Ordnung vorgegeben werden, in welcher die Teilaufgaben expandiert werden sollen. In diesem Punkt unterscheidet sich CHIMP von SHOP2, welcher die Aktionen in exakt derselben Reihenfolge plant, in der sie auszuführen sind. Damit erlaubt es CHIMP, auch zwischen solchen Teilaufgaben eine Reihenfolge für die Planung festzulegen, zwischen denen keine rein temporale Beziehung besteht. Auf diese Weise wird für die Modellierung der Planungsdomäne eine weitere Möglichkeit gegeben, den Suchraum weiter einzuschränken.

Die nachfolgende Methode gibt ein Beispiel sowohl für die Verwendung solcher Ordnungsrelation als auch für temporale Relationen:

$$
\begin{aligned}
f &= (\text{get\_object}(?\text{mug}_1), [[0,0],[1,\infty]], \cdot) \\
F_p &= \{f_1 = (\text{RobotAt}(?\text{loc}_1), \cdot, \cdot), f_2 = (\text{On}(?\text{mug}_2, ?\text{loc}_2), \cdot, \cdot)\} \\
F_S &= \{f_3 = (\text{drive}(?\text{loc}_3), \cdot, \cdot), \\
&\qquad f_4 = (\text{assume\_manipulation\_pose}(?\text{loc}_4), \cdot, \cdot), \\
&\qquad f_5 = (!\text{grasp}(?\text{mug}_3), \cdot, \cdot)\} \\
CC &= \{f_1 \text{ pre } f, f_2 \text{ pre } f,\ f \text{ dc } f_3, f \text{ dc } f_4, f \text{ dc } f_5, f_3 \text{ ordering } f_4, f_4 \text{ ordering } f_5,\ \text{planned} f\} \\
BC &= \{S_1^{(f)} = S_1^{(f_2)},\ S_1^{(f)} = S_1^{(f_5)},\ S_1^{(f_1)} \neq S_2^{(f_2)},\ S_2^{(f_2)} = S_1^{(f_3)},\ S_1^{(f_3)} = S_1^{(f_4)}\} \\
TC &= \{I^{(f_1)} \{\text{m,o}\}\ I^{(f)},\ I^{(f_2)} \text{o } I^{(f)},\ I^{(f_3)} \text{b } I^{(f_4)},\ I^{(f_4)} \text{b } I^{(f_5)}, \\
&\qquad I^{(f_3)} \text{s } I^{(f)},\ I^{(f_4)} \text{d } I^{(f)},\ I^{(f_5)} \text{f } I^{(f)}\} \\
w &= 1
\end{aligned}
\tag{5.3}
$$

Es handelt sich dabei um eine weitere Methode für die Aufgabe `get_object`. Im Gegensatz zu der vorherigen Methode ist diese Methode anwendbar, wenn der Roboter nicht in der Nähe des Objekts ist ($S_1^{(f_1)} \neq S_2^{(f_2)}$). In diesem Fall dekomponiert die Methode die Aufgabe in die drei Teilaufgaben `drive`, `assume_manipulation_pose` und `!grasp`. Diese sollten jeweils nacheinander ausgeführt werden, weshalb zwischen ihnen temporale Constraints der Relation *before* gesetzt werden. Zudem wird definiert, dass die erste Teilaufgabe gleichzeitig mit `get_object` beginnt und die letzte zeitgleich mit ihr endet. Die Reihenfolge, in der die Teilaufgaben geplant werden, wird nun mit den kausalen Constraints (f_3 ordering f_4) sowie (f_4 ordering f_5) vorgegeben.

Wie die Beispiele zeigen, besteht eine solche Methode bereits aus einer Vielzahl von Constraints. Um den Modellierungsaufwand gering zu halten, werden einige von ihnen wie die kausalen `pre`- und `dc`-Constraints bereits automatisch beim Parsen der Domänenbeschreibung erzeugt und es werden Standardconstraints verwendet, wenn für eine kausale Relation keine temporalen Constraints explizit angegeben werden. So wird zwischen einer Aufgabe und ihren Teilaufgaben, sofern nicht anders angegeben, die zusammengesetzte konvexe Relation $\{\mathrm{s}, \mathrm{d}, \equiv, \mathrm{f}\}$ hinzugefügt. Gleichermaßen wird zwischen der Aufgabe und Vorbedingungen $\{\mathrm{m}^{-1}, \mathrm{o}^{-1}, \mathrm{f}, \mathrm{d}\}$, zu negativen Effekten $\{o^{-1}\}$ und zu positiven Effekten die Relation $\{o\}$ verwendet. In Anhang A wird das konkrete Format von CHIMPs Domänenbeschreibungssprache und Problembeschreibungssprache beschrieben.

5.2 Meta-Constraints

Nachdem CHIMPs Repräsentationformen definiert wurden, kann nun auf die verwendeten Meta-Constraints eingegangen werden. Die aktuelle Version von CHIMP verwendet vier Meta-Constraints. Das Hauptaugenmerk und der Hauptbeitrag der vorliegenden Arbeit liegt dabei auf jenem Meta-Constraint, der die HTN-Dekompositionsstrategie umsetzt. Darüber hinaus werden jeweils ein Meta-Constraint für das Scheduling von Ressourcen sowie Zustandsvariablen und ein weiterer für die Abschätzung der Fahrtzeit zwischen zwei Bereichen in der Restaurantumgebung eingesetzt. Allen vier Meta-Constraints dient das gemeinsame Constraint-Netz, welches aus Fluenten und den zuvor beschriebenen kausalen, temporalen und symbolischen Constraints besteht, als Basis. Dies wird schematisch in Abbildung 5.4 dargestellt. In den nachfolgenden Abschnitten werden die verschiedenen Meta-Constraints ausführlich beschrieben.

5.2.1 Der HTN Meta-Constraint

Der HTN-Meta-Constraint implementiert die HTN-Dekompositionsstrategie. Er ist somit für die rein kausale Planung zuständig und stellt die kausale Gültigkeit des Plans sicher. Die Implementierung der kausalen Planung als Meta-Constraint anstelle eines externen Moduls ermöglicht eine nahtlose Integration in das allgemeine hybride Planungsschema des Meta-CSP-Ansatzes und der damit verbundenen verschachtelten Suche eines Gesamtplans.

Bevor auf die Implementierung eingegangen und der Ablauf der Planung an einem Beispiel erläutert wird, soll zunächst eine formale Definition dieses Meta-Constraints gegeben werden:

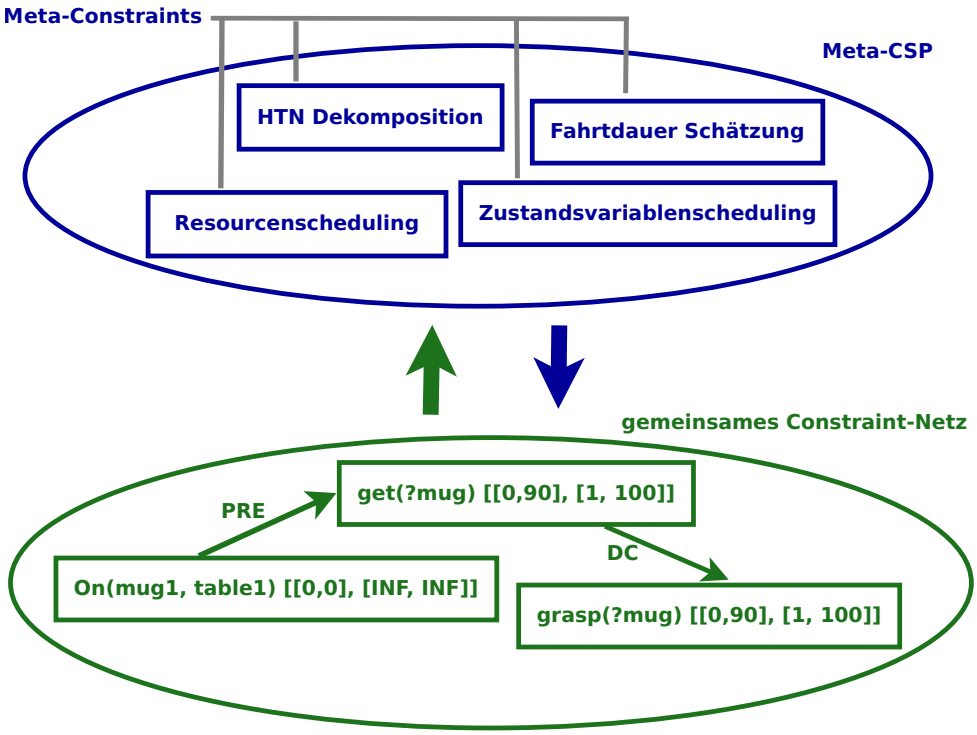


Abbildung 5.4: Überblick über das Meta-CSP, welches aus vier Meta-Constraints besteht, und das zu Grunde liegende gemeinsame Constraint-Netz.

Definition 5.10. Ein *HTN-Dekompositions-Meta-Constraint* ist ein Tripel $(M, \Xi_{HTN}, \Delta_{HTN})$, für das gilt:

- $M = (\mathcal{F}, \mathcal{C})$ ist ein Constraint-Netz, welches aus einer Menge von Fluenten $\mathcal{F}$ und einer Menge $\mathcal{C}$ von temporalen, symbolischen sowie kausalen Constraints besteht;
- $\Xi_{HTN} = \{\xi\}$ ist eine einelementige Menge bestehend aus einem Constraint-Netz, welches definiert ist als $\xi = \{f \mid f \in \mathcal{F} \wedge (\text{planned } f) \notin \mathcal{C} \wedge \forall f_j((f_j \text{ ordering } f) \in \mathcal{C}) \Rightarrow ((\text{planned } f_j) \in \mathcal{C})\}$, das heißt f ist ein ungeplanter Aufgabenfluent, dessen Vorgänger bereits geplant wurden ;
- $\Delta_{HTN} = \delta_{DC}(\xi) \cup \delta_U(\xi)$ ist eine Menge von Meta-Werten, die sich folgendermaßen zusammensetzt:
  - $\delta_{DC}(\xi)$ ist eine Menge von Constraint-Netzen, welche sich jeweils aus der Anwendung einer Methode oder eines Operators auf einen noch ungeplanten Aufgabenfluenten ergeben.
  - $\delta_U(\xi)$ ist eine Menge von Constraint-Netzen, welche jeweils aus einem noch ungeplanten und einem bereits geplanten Aufgabenfluenten sowie einem `matches`-Constraint zwischen diesen bestehen.

Das gemeinsame Constraint-Netz, das aus Fluenten besteht, ist wie üblich die Basis zur Ermittlung der Meta-Variablen und Meta-Werte. Zur Bestimmung der Meta-Variablen, also der Konflikte, wird dieses Constaint-Netz nach Aufgabenfluenten durchsucht, die selbst noch nicht geplant wurden, aber deren Vorgänger hingegen bereits alle geplant wurden. Die vorhergehenden Aufgabenfluenten sind mit einem kausalen `ordering`-Constraint zu jenem Aufgabenfluenten verbunden. Derartige Fluenten sind aus Sicht des HTN-Meta-Constraints Konflikte und werden zu jenem Constraint-Netz hinzugefügt, welches den Meta-Wert repräsentiert.

Für den HTN-Dekompositions-Meta-Constaint gibt es somit in jedem Schritt nur maximal eine Meta-Variable, die in ihrem Constraint-Netz alle Konflikte beinhaltet. Dies ist notwendig, um die Vollständigkeit der Planung bei Verwendung des Meta-CSP-Backtracking-Algorithmus zu gewährleisten, da das Backtracking abbricht, wenn zu einer gewählten Meta-Variable kein gültiger Meta-Wert gefunden werden kann, anstatt in diesem Fall einen anderen Meta-Wert zu prüfen. Wenn mehrere ungeplante Aufgaben vorliegen und diese jeweils auf separate Meta-Variablen verteilt würden, könnte ansonsten der Fall auftreten, dass der Planer sich für eine Aufgabe entscheidet, deren Vorbedingungen derzeit noch nicht erfüllbar wären, sondern dies erst durch die Unteraufgaben einer anderen ungeplanten Aufgabe würden. Die Planung dieser Aufgabe würde daher fehlschlagen und somit würde auch die Gesamtplanung ohne gefundenen Plan abbrechen.

Dennoch liefert ein einzelner Meta-Wert nur für genau einen ungeplanten Aufgabenfluenten eine Lösung. Die restlichen Aufgabenfluenten bleiben ungeplant und werden wiederum Elemente späterer Meta-Variablen. Die Auswirkungen dieses Vorgehens werden später in Abschnitt 6.1.3 genauer diskutiert.

Die Ermittlung aller Meta-Werte geschieht auf zwei Arten mittels Berechnung der Mengen δ_{DC} und δ_U und wird in Algorithmus 5.1 dargestellt. δ_{DC} ergibt sich aus der Anwendung von Methoden beziehungsweise Operatoren auf einen Aufgabenfluenten $a \in \xi$ und entspricht somit der gewöhnlichen Art der HTN-Planung. Dafür wird zunächst die Menge der *aktiven* Zustandsfluenten aus der Fluentenmenge $\mathcal{F}$ anhand der Werte ihrer temporalen Variablen herausgefiltert (Zeile 5). Der frühestmögliche Startzeitpunkt des Zustandsfluenten muss kleiner als der spätestmögliche Startzeitpunkt von a sein. Diese aktiven Fluenten kommen als Vorbedingungen in Frage. Dem Filtern liegt die Annahme zu Grunde, dass Vorbedingungen einer Methode beziehungsweise eines Operators immer zum Startpunkt der zugehörigen Aktion gültig sind. Sollen hingegen auch solche Vorbedingungen zulässig sein, die vorgeben, dass eine Anforderung beispielsweise bis fünf Sekunden vor der Aktion erfüllt sein soll, so ist dieser Schritt dementsprechend anzupassen.

Als Nächstes werden diejenigen Operatoren beziehungsweise Methoden, die für den jeweiligen Aufgabentypen anwendbar sind, aus der Domänenbeschreibung ausgewählt (Zeilen 7 und 12). Dies geschieht anhand des Prädikatnamens. Für jeden dieser Operatoren beziehungsweise Methoden wird nun versucht, ihn auf den Aufgabenfluenten anzuwenden. Hierbei werden alle Möglichkeiten der Anwendung ermittelt. Für die Vorbedingungen werden jeweils alle aktiven Zustandsfluenten mit passenden Prädikatnamen und Argumenten gesucht und entsprechende Constraints zwischen jenen Zustandsfluenten und dem Aufgabenfluenten generiert. Für jede dieser n Vorbedingungen gibt es somit eine Menge von Constraints, aus denen das kartesische Produkt berechnet wird. Die Elemente dieser Menge sind n-Tupel aus Vorbedingungsconstraints. Für jedes Tupel wird ein neues Constraint-Netz erzeugt, in welches jene Constraints sowie die zugehörigen

Fluenten hinzugefügt werden. Hinzu kommen noch weitere Constraints und Fluenten für die Dekompositionen beziehungsweise die Effekte. Für neuen Teilaufgaben werden dabei auch die entsprechenden ausgehenden `ordering`-Constraints der ursprünglichen Aufgabe übernommen. Jedes der auf diese Weise erzeugten Constraint-Netze ist eine potentielle Lösung des Konflikts und somit ein Meta-Wert.

Algorithmus 5.1 Berechnung aller Meta-Werte für eine Meta-Variable ξ auf Basis des gemeinsamen Constraint-Netzes $(\mathcal{F}, \mathcal{C})$.

1: **Prozedur** BerechneMeta-Werte(ξ, $(\mathcal{F}, \mathcal{C})$, $\mathcal{O}$, $\mathcal{M}$)
2: $werte \leftarrow \{\}$
3: $(\mathcal{F}_\xi, \mathcal{C}_\xi) \leftarrow \xi$
4: **für alle** a **in** $\mathcal{F}_\xi$ **führe aus**
5: $aktiv \leftarrow$ ErmittleAktiveZustandsfluenten$(a, (\mathcal{F}, \mathcal{C}))$
6: **falls** a primitiv ist **dann**
7: $operatoren \leftarrow \{o \mid o \in \mathcal{O} \wedge$ die Aufgabe von o ist mit a symbolisch unifizierbar$\}$
8: **für alle** o **in** $operatoren$ **führe aus**
9: $werte \leftarrow werte \cup \{(\mathcal{F}_o, \mathcal{C}_o) \mid (\mathcal{F}_o, \mathcal{C}_o)$ ist eine Instanziierung des Constraint-Netzes des Operators o, in dem die Schema-Fluenten der Vorbedingungen von o mit Fluenten aus $aktiv$ unifiziert worden sind$\}$
10: **Ende für**
11: **sonst**
12: $methoden \leftarrow \{m \mid m \in \mathcal{M} \wedge$ die Aufgabe von m ist mit a symbolisch unifizierbar$\}$
13: **für alle** m **in** $methoden$ **führe aus**
14: $werte \leftarrow werte \cup \{(\mathcal{F}_m, \mathcal{C}_m) \mid (\mathcal{F}_m, \mathcal{C}_m)$ ist eine Instanziierung des Constraint-Netzes der Methode m, in dem die Schema-Fluenten der Vorbedingungen von m mit Fluenten aus $aktiv$ unifiziert worden sind$\}$
15: **Ende für**
16: **Ende falls**
17: **Ende für**
18: **für alle** a **in** $\mathcal{F}_\xi$ **führe aus**
19: **für alle** f **in** $\mathcal{F}$, die bereits geplant wurden, **führe aus**
20: **falls** der Prädikatname und die Argumente von a und f übereinstimmen können **und** die temporalen Variablen von a und f übereinstimmen können **dann**
21: $\mathcal{F}_U \leftarrow \{a, t\}$
22: $\mathcal{C}_U \leftarrow \{(a$ {matches} $t),\ (\{planned\}\ a)\}$
23: $werte \leftarrow werte + (\mathcal{F}_U, \mathcal{C}_U)$
24: **Ende falls**
25: **Ende für**
26: **Ende für**
27: **gib zurück** $werte$
28: **Ende Prozedur**

Im Gegensatz zu δ_{DC} wird eine ungeplante Aufgabe mit δ_U nicht durch Anwendung von Operatoren oder Methoden geplant, sondern es wird versucht, sie mit einer bereits geplanten Aufgabe zu

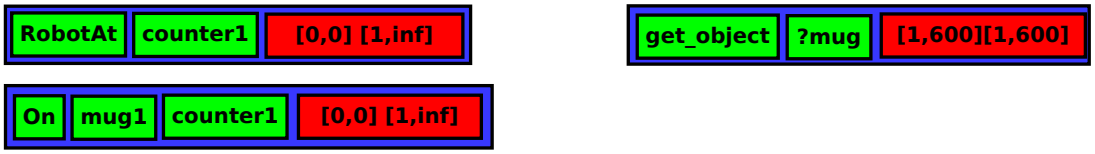


Abbildung 5.5: Constraint-Netz der Ausgangssituation. Abbildung zuerst veröffentlicht in [162], angepasste Version.

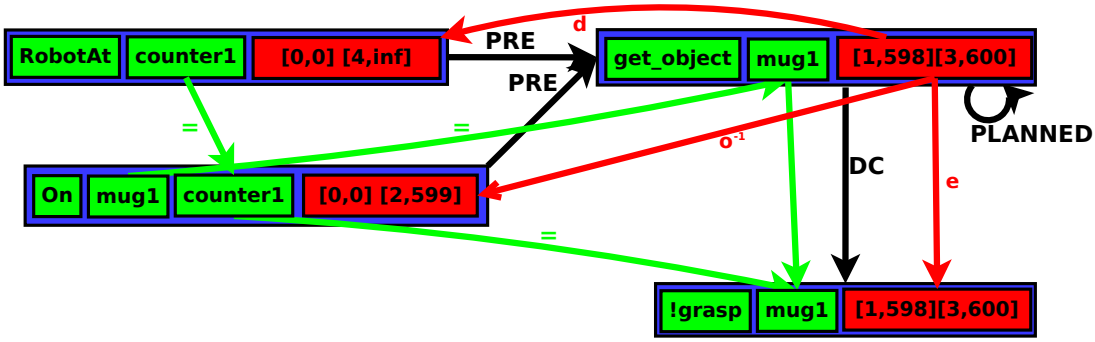


Abbildung 5.6: Constraint-Netz nach Anwendung einer Methode. Kausale Constraints sind schwarz, symbolische Constraints sind grün und temporale Constraints sind rot gekennzeichnet. Abbildung zuerst veröffentlicht in [162], angepasste Version.

unifizieren. Dies geschieht in den Zeilen 14 bis 24 von Algorithmus 5.1. Wie zuvor wird auch dies einzeln für jede ungeplante Aufgabe der Meta-Variable durchgeführt. Dazu wird über die bereits geplanten Aufgaben des gemeinsamen Constraint-Netzes $(\mathcal{F}, \mathcal{C})$ iteriert und zunächst anhand der symbolischen Variablen überprüft, ob diese symbolisch unifiziert werden können. Entsprechend werden zudem die temporalen Variablen geprüft. Falls beide Tests erfolgreich verlaufen, wird in den Zeilen 18 und 19 das Constraint-Netz $(\mathcal{F}_U, \mathcal{C}_U)$ als neuer Meta-Wert erzeugt. Ein Constraint vom Typ `matches` repräsentiert dabei die Unifikation. Zudem wird die entsprechende Aufgabe wie üblich mit einem `planned`-Constraint als geplant markiert. Der `matches`-Constraint enthält interne symbolische und temporale Constraints, die die Unifikation der jeweiligen symbolischen und der temporalen Variablen der beiden Aufgabenfluenten sicherstellen. Die eigentliche Unifikation geschieht somit erst bei der Anwendung eines solchen Meta-Wertes durch Hinzufügen der Constraints aus $\mathcal{C}_U$ und anschließende Constraint-Propagierung inklusive Konsistenzprüfung. Das Überprüfen der Bedingungen in den Zeilen 16 und 17 ist daher an dieser Stelle nicht zwingend notwendig, jedoch reduziert es die Anzahl potentieller Meta-Werte und das damit verbundene Backtracking.

Das Beispiel, das in den Abbildungen 5.5 bis 5.7 dargestellt ist, veranschaulicht den Ablauf der Planung mit dem HTN-Dekompositions-Meta-Constraint. Abbildung 5.5 zeigt die Ausgangssituation in Form eines Constraint-Netzes, das drei Fluenten und keine Constraints enthält. Die symbolischen Variablen der Fluenten sind grün und die temporalen Variablen rot markiert. Der Roboter befindet sich initial am Tresen `counter1` und das Objekt `mug1` ist auf jenem `counter1` platziert. Zudem ist die komplexe Aufgabe gegeben, eine beliebige Tasse bis spätestens zum Zeitpunkt 600 zu holen. Da die Aufgabe `get_object` noch nicht geplant wurde, wird sie

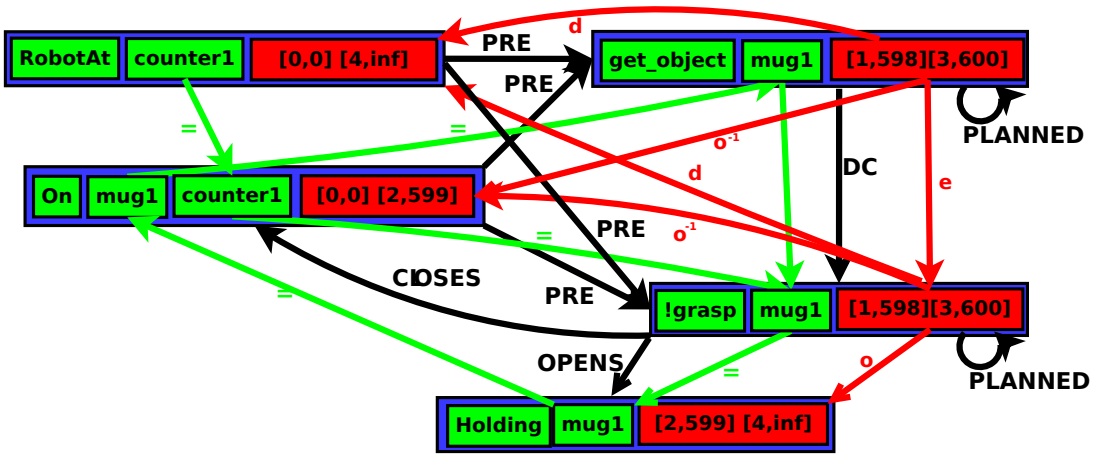


Abbildung 5.7: Constraint-Netz nach Anwendung eines Operators. Abbildung zuerst veröffentlicht in [162], angepasste Version.

von dem HTN-Dekompositions-Meta-Constraint als Konflikt identifiziert und somit als einzige Meta-Variable ein Constraint-Netz mit diesem Fluenten als Meta-Variable erzeugt. Zum Auflösen dieses Konflikts wird eine geeignete Methode auf den Aufgabenfluenten angewendet. Da sich der Roboter schon am Tresen `counter1` und damit nahe einer Tasse befindet, dekomponiert die Methode aus Gleichung 5.2 den Aufgabenfluenten `get_object` in die einzige Teilaufgabe `!grasp`, für welche ein neuer Aufgabenfluent erzeugt und einem neuen Meta-Wert hinzugefügt wird. Zudem werden diesem Meta-Wert neue Constraints hinzugefügt. Die verwendete Methode besitzt die beiden Vorbedingungen, dass sich eine Tasse auf einem Tisch oder Tresen befindet und der Roboter nahe zu diesem Ort positioniert ist. Daher wird `get_object` mit zwei kausalen Constraints vom Typ `PRE` mit den entsprechenden Zustandsfluenten `On` und `RobotAt` verbunden. Ebenso werden die in der Methodenbeschreibung vorgegebenen temporalen und symbolischen Constraints zwischen den Fluenten hinzugefügt. Während initial nicht vorgegeben war, welche Tasse der Roboter holen soll, ist das erste Argument von `get_mug` nun durch einen symbolischen Constraint an das erste Argument des Fluenten `On` gebunden und dadurch auf die Tasse `mug1` festgelegt.

Nachdem das Hinzufügen des Meta-Wertes in das gemeinsame Constraint-Netz sowie die Konsistenzprüfungen der zugrunde liegenden symbolischen und temporalen Constraint-Netze erfolgreich war, sucht wiederum der HTN-Dekompositions-Meta-Constraint nach neuen Konflikten. In diesem Fall wird der Aufgabenfluent `!grasp` als ein solcher identifiziert. Da es sich hierbei um eine primitive Aufgabe handelt, wird ein Operator angewendet. Dieser hat dieselben Vorbedingungen wie die zuvor verwendete Methode. Zudem wird ein neuer Zustandsfluent generiert, der den Effekt repräsentiert, dass der Roboter die Tasse nun in der Hand hält. Mit einem kausalen `opens`-Constraint wird dieser Fluent mit dem Aufgabenfluenten verbunden. Ebenso wird für den negativen Effekt, dass sich die Tasse nicht mehr auf `counter1` befindet, ein `closes`-Constraint erzeugt. Nach dem Hinzufügen dieses Meta-Wertes in das Grund-CSP wurden, wie in Abbildung 5.7 zu sehen ist, beim anschließenden Propagieren der neuen Constraints auch die Werte der temporalen Variablen aktualisiert.

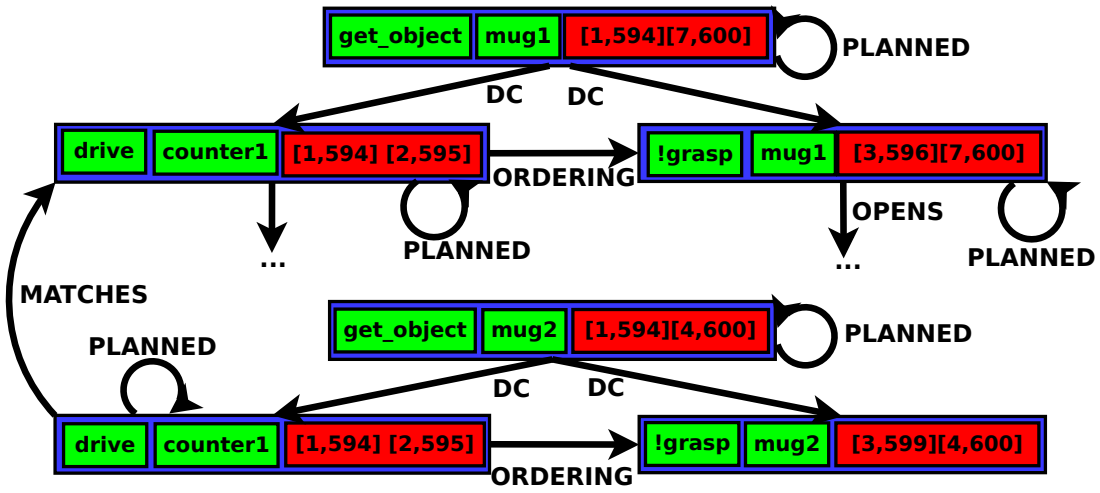


Abbildung 5.8: Constraint-Netz nach Unifikation der neuen mit der bereits geplanten `drive`-Aufgabe. Abbildung zuerst veröffentlicht in [162], angepasste Version.

Abbildung 5.8 zeigt ein Beispiel für die zweite Art der Ermittlung von Meta-Werten mittels Unifikation. In dieser Abbildung sind nur die Aufgabenfluenten und kausalen Constraints eingezeichnet. Hier waren die beiden komplexen Aufgaben gegeben, jeweils die Tassen `mug1` und `mug2` zu holen. Dies wurde zunächst vollständig für `mug1` geplant und anschließend `get_object(mug2)` in `drive` und `!grasp` dekomponiert. An dieser Stelle werden unterschiedliche Meta-Werte für die untere komplexe Aufgabe `drive` erzeugt. Wie zuvor ergibt sich ein Meta-Wert durch Anwendung einer Methode. Allerdings ist das Fahren zu `counter1` bereits in dem oberen Plan enthalten. Die zweite Art der Meta-Werte-Berechnung verwendet diese bereits geplante komplexe Aufgabe, indem sie den neuen Aufgabenfluenten `drive` durch einen `matches`-Constraint mit dem bereits geplanten unifiziert. Nach der Auswahl und Anwendung dieses Meta-Wertes werden wiederum das temporale und das symbolische Constraint-Netz aktualisiert und auf Konsistenz geprüft. Auf diese Weise wird der zeitliche Ablauf der beiden Zielaufgaben ineinander verwoben und die beiden Objekte werden zeitgleich mit einer einzigen Fahrt zum Tresen geholt.

Der Einsatz dieser Art der Unifikation hat mehrere Vorteile für die Planung. So braucht der komplette Teilbaum der zweiten Aufgabe nicht erneut geplant zu werden, wodurch der Verzweigungsfaktor und der Planungsaufwand gesenkt wird. Für das Beispiel hält sich dieser Vorteil noch in Grenzen, da die Domänenbeschreibung eine weitere Methode für die Aufgabe `drive` beinhaltet, die keine Teilaufgaben hat, falls sich der Roboter bereits an dem Zielort befindet. Für größere Aufgaben, für die eine solche Methode nicht existiert und bei denen sich erst in den Teilaufgaben feststellen lässt, ob sie bereits erfüllt sind, kann dies aber von Vorteil sein.

Wichtiger ist jedoch die Eigenschaft, dass dadurch die Erstellung kürzerer Pläne favorisiert und erleichtert wird. Mit der Größe des Plans und der darin befindlichen Zustandsfluenten steigt auch die Anzahl potentieller Kandidaten von Fluenten für die Vorbedingungen von Methoden und Operatoren und somit auch die Anzahl von Meta-Werten. Die Planungsdauer und die Planlänge sind dann von geeigneten Heuristiken zur Auswahl der Meta-Werte abhängig. Hierauf

geht Abschnitt 5.4 genauer ein. So kann es passieren, dass in dem zuvor beschriebenen Fall der Meta-Wert ohne Teilaufgaben aus anderen Gründen dennoch nicht verwendet wird. Favorisiert die Heuristik hingegen Meta-Werte, die eine Unifikation mit `matches`-Constraints beinhalten, können kürzere Pläne und das temporale Verweben komplexer Aufgaben begünstigt werden.

5.2.2 Meta-Constraints für das Scheduling von Ressourcen und Zustandsvariablen

Auch das Scheduling wiederverwendbarer Ressourcen und Zustandsvariablen wird mittels zweier Meta-Constraints durchgeführt. Es basiert auf dem Verfahren von Cesta et al., welches in Abschnitt 4.3.1 diskutiert wurde. Die Umsetzung basiert auf dem bereits im Meta-CSP-Framework vorhandenen Meta-Constraint für das Ressourcen-Scheduling. Dieser wurde an das von CHIMP verwendete gemeinsame Constraint-Netz und dessen Repräsentation von Ressourcen und Zustandsvariablen angepasst.

5.2.3 Abschätzung der Fahrtdauer

Neben kausalem, temporalem und Ressourcenwissen erfordern verschiedene Anforderungen häufig auch ganz spezielle Wissensformen oder externe Module, die ihr Wissen nur als Blackbox zur Verfügung stellen. Durch die Verwendung des Meta-CSP-Ansatzes lassen sich auch solche zusätzlichen Anforderungen und Wissensquellen durch die Implementierung weiterer Meta-Constraints in CHIMP integrieren. Als Beispiel dient hier die Abschätzung der Fahrtdauer, welche der Roboter für das Fahren von seiner aktuellen Position in einen bestimmten Bereich benötigen wird. Dies wird mit folgendem Meta-Constraint umgesetzt:

Definition 5.11. Ein *Fahrtdauer-Meta-Constraint* ist ein Tripel $(M, \Xi_{Fahrtdauer}, \Delta_{Fahrtdauer})$, für das gilt:

- $M = (\mathcal{F}, \mathcal{C})$ ist das gemeinsame Constraint-Netz, welches aus einer Menge von Fluenten $\mathcal{F}$ und einer Menge $\mathcal{C}$ von temporalen, kausalen und symbolischen Constraints besteht.
- $\Xi_{Fahrtdauer} = \{\xi_1, \ldots, \xi_n\}$ ist eine Menge von Constraint-Netzen, die jeweils aus einem einzigen Fluenten $f_i \in \mathcal{F}$ bestehen, für den gilt: f_i ist ein Aufgabenfluent vom Typ `!move_base`, für den die erwartete Fahrtdauer noch nicht bestimmt wurde und daher (moveduration f_i) $\notin \mathcal{C}$ ist.
- $\Delta_{Fahrtdauer}$ ist eine Menge von Meta-Werten. Für jede Meta-Variable $\xi_i \in \Xi$ ist deren Meta-Wert $\delta_{Fahrtdauer}(\xi_i) \in \Delta_{Fahrtdauer}$ ein Constraint-Netz, welches für den jeweiligen Aufgabenfluenten $f_i \in \xi_i$ einen Constraint (moveduration f_i) enthält.

Als Basis des Meta-Constraints dient wiederum das gemeinsame Constraint-Netz. Da hier nur das normale Fahren mittels des gewöhnlichen in ROS verwendeten Fahrtplaners betrachtet wird, sind nur die Aufgabenfluenten vom Typ `!move_base` relevant. Es wird für jeden Fluenten dieses

Typs geprüft, ob dessen Fahrtdauer schon abgeschätzt wurde, was mit Hilfe eines moveduration-Constraints markiert wird. Wenn dies nicht der Fall ist, wird für diesen Fluenten eine Meta-Variable erzeugt. Ein Meta-Wert für eine solche Meta-Variable ist ein Constraint-Netz, das lediglich einen moveduration-Constraint für den jeweiligen Fluenten enthält. Der moveduration-Constraint ist ein Multi-Constraint, der intern einen temporalen Constraint zur Repräsentation der minimalen Fahrtdauer beinhaltet. Diese minimale Fahrtdauer wird derzeit auf Basis einer Nachschlagetabelle ermittelt, die die Zeitdauer zwischen zwei Bereichen der Roboterumgebung angibt. Es ließe sich jedoch ebenso der Pfadplaner des Robotersystems dafür nutzen. Der Meta-Constraint dient somit als Wrapper, der die Schnittstelle zu einer solchen externen Komponente während der Planung aufruft und deren Ergebnisse wiederum in das gemeinsame Constraint-Netz einfügt. Auf gleiche Weise lassen sich auch die Ergebnisse anderer externer Planer und Reasoner integrieren. Dieses Verfahren ist damit ähnlich zu Semantic Attachments, die bereits in Abschnitt 4.2 erwähnt wurden.

Wenn der Planer für eine Meta-Variable den moveduration-Constraint des Meta-Wertes in das gemeinsame Constraint-Netz einfügt, wird anschließend wie üblich direkt eine Konsistenzprüfung des temporalen Grund-CSPs vorgenommen. Im Falle einer Inkonsistenz wird der Constraint wieder entfernt und Backtracking durchgeführt, da es zu jeder Meta-Variablen nur maximal einen Meta-Wert gibt.

5.3 Zusammenspiel der Meta-Constraints

CHIMPs vier Meta-Constraints sowie die darunter angesiedelten temporalen und symbolischen Constraint-Reasoner der Grund-CSPs bearbeiten das Planungsproblem gemeinsam mit dem zuvor beschriebenen Backtracking-Algorithmus 4.1. Hierbei werden Constraints und Variablen in das gemeinsame Constraint-Netz hinzugefügt und Konsistenzprüfungen der Grund-CSPs durchgeführt (siehe Abbildung 5.4). CHIMP hat mit dem Meta-CSP-Backtracking-Algorithmus einen Plan gefunden, sobald das gemeinsame Constraint-Netz den Anforderungen aller Meta-Constraints genügt und die vom gemeinsamen Constraint-Netz subsumierten symbolischen und temporalen Constraint-Netze jeweils konsistent sind.

Für das Zusammenspiel der Meta-Constraints ist die Reihenfolge, in der sie geprüft werden, von Bedeutung. CHIMP überprüft die Meta-Constraints für gewöhnlich in der Reihenfolge, die in Abbildung 5.9 dargestellt ist. Bei Bedarf lässt sich diese Reihenfolge jedoch auch variieren. Zunächst wird das Scheduling der Ressourcen und Zustandsvariablen durchgeführt, anschließend geprüft, ob ein `!move_base`-Fluent ohne Schätzung der Fahrtdauer existiert, und schließlich nach ungeplanten Aufgabenfluenten gesucht. Falls Meta-Variablen für einen dieser Meta-Constraints existieren, wird versucht, einen der Konflikte aufzulösen. Anschließend wird wieder mit dem Ressourcenscheduling-Meta-Constraint begonnen.

Der HTN-Meta-Constraint erzeugt durch das Hinzufügen neuer Teilaufgaben ständig neue Meta-Variablen für sich selbst, bis ein kausal gültiger Plan gefunden ist. Stünde er an erster Stelle, würde er dadurch andauernd erneut aufgerufen, und die restlichen Meta-Constraints kämen erst anschließend zum Zuge. Durch die Positionierung an letzter Stelle wechseln sich die Meta-

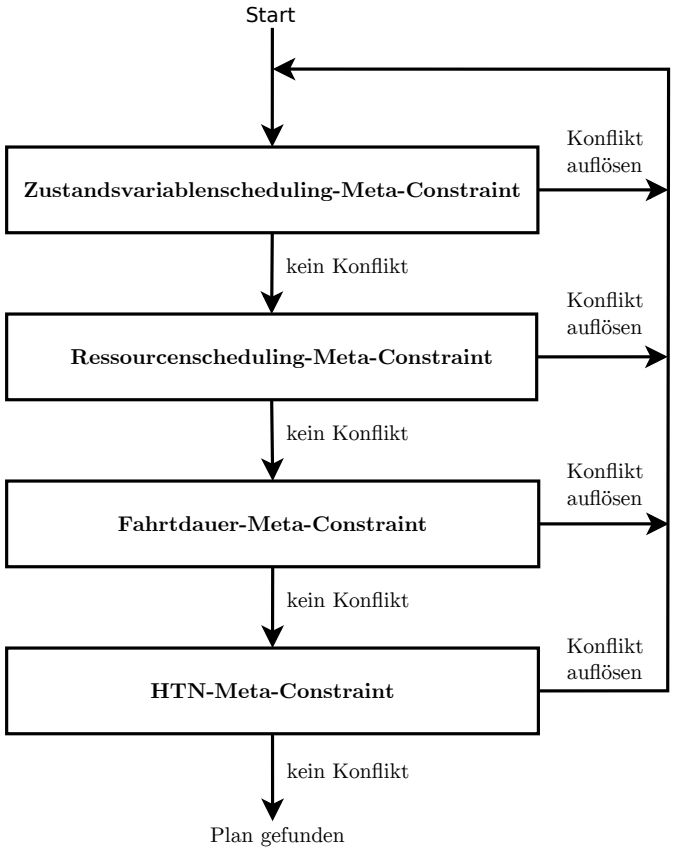


Abbildung 5.9: Reihenfolge, in der CHIMP seine Meta-Constraints ausgewertet.

Constraints hingegen ab, da sowohl die Meta-Constraints für das Scheduling als auch der Fahrtdauer-Meta-Constraint vergleichsweise wenige Änderungen vornehmen müssen.

Durch diese Verschachtelung der Meta-Constraints bearbeiten sie das Gesamtproblem gemeinsam. Nachdem der HTN-Meta-Constraint neue Teilaufgaben hinzufügt oder primitive Aufgaben geplant hat, stellen die Meta-Constraints für das Scheduling sicher, dass der Teilplan hinsichtlich der Ressourcenallokation gültig ist, bevor der HTN-Meta-Constraint fortfahren kann. Zudem werden auch Propagierungen und Kosistenzprüfungen der tiefer liegenden temporalen und symbolischen Constraint-Netze durchgeführt. Dadurch wird erreicht, dass das gemeinsame Constraint-Netz in jedem Schritt symbolisch und temporal konsistent ist.

Ebenso ist der Fahrtdauer-Meta-Constraint Teil dieses verschachtelten Planungsprozesses. Immer dann, wenn ein Operator auf eine Aufgabe vom Typ `!move_base` angewendet wurde und die Scheduling-Meta-Constraints konfliktfrei sind, wird der Fahrtdauer-Meta-Constraint überprüft, bevor mit der HTN-Planung fortgefahren wird. Damit wird sichergestellt, dass ein Pfad zwischen den beiden Roboterpositionen existiert und die Mindestfahrtdauer nicht zu lang ist. Andernfalls würde an dieser Stelle Backtracking durchgeführt. Die Ermittlung der Mindestfahrtdauer und dessen temporale Propagierung führt so zu einer Reduktion des kausalen Suchraums.

Aber auch die Strategie der hierarchischen Planung sorgt dafür, dass nur für solche Fluenten Scheduling durchgeführt oder die Fahrtdauer ermittelt werden muss, die aus kausaler Sicht Sinn ergeben. Hierbei hilft vor allem, dass der Suchraum des hierarchischen Planers durch seine Hierarchisierung bereits stark eingeschränkt ist. Dadurch können potentiell auch solche Reasoner in den Planungsprozess integriert werden, deren Laufzeit vergleichsweise hoch ist.

5.4 Ordnungsheuristiken

Mit Heuristiken für die Ordnung von Meta-Variablen und Meta-Werten kann der Verlauf der Suche beeinflusst werden. Auf die Heuristiken für das Ressourcenscheduling wurde bereits in Abschnitt 4.3.1 eingegangen. Sie basieren auf [28], wo weitere Details nachzulesen sind. Für den Fahrtdauer-Meta-Constraint sind hingegen keine Heuristiken notwendig, da zu jeder Meta-Variablen nur ein Meta-Wert existiert. Im Folgenden wird daher auf Ordnungsheuristiken für den HTN-Meta-Constraint eingegangen.

Da CHIMPs HTN-Meta-Constraint alle noch ungeplanten Aufgaben, die keinen ungeplanten Vorgänger haben, in einer einzigen Meta-Variablen repräsentiert, wird keine Variablenordnungsheuristik benötigt. Mit der Wahl der Werteordnungsheuristik kann hingegen Einfluss auf die Laufzeit genommen werden. Hierfür wurden unterschiedliche Ordnungsheuristiken entwickelt. Die Implementierung einer Ordnungsheuristik geschieht im Meta-CSP-Framework durch die Implementierung einer Methode, die zwei Meta-Werte, also zwei Constraint-Netze, vergleicht.

Es wurden mehrere Heuristiken implementiert. Hierfür wurden unterschiedliche Kriterien identifiziert, die zum Vergleich zweier Meta-Werte herangezogen werden können:

1. Bevorzugen der Unifikation zu einer bereits geplanten Aufgabe.
2. Die Anzahl der Teilaufgaben. (Die geringere Anzahl wird bevorzugt.)
3. Die frühesten Startzeitpunkte der geplanten Aufgaben. (Der frühere Wert wird bevorzugt.)
4. Die Summe der frühesten Startzeitpunkte der Vorbedingungen. (Der größere Wert wird bevorzugt.)
5. Die Dekompositionstiefen der geplanten Aufgaben. (Der größere Wert wird bevorzugt.)
6. Die Wichtungsfaktoren der Methoden. (Der größere Wert wird bevorzugt.)

In der Abschlusspräsentation von RACE und in [163] wurde eine Heuristik verwendet, die die Kriterien 1–4 berücksichtigt. Beim Vergleich zweier Meta-Werte überprüft die Heuristik zunächst, ob genau einer der Meta-Werte eine Unifikation beinhaltet, und siedelt in diesem Fall jenen Meta-Wert vor dem anderen an. Andernfalls bevorzugt sie den Meta-Constraint mit der kleineren Anzahl an Teilaufgaben durch Zählen der `dc`-Constraints. Wenn auch hier kein Unterschied zwischen den Meta-Variablen vorliegt, werden die frühesten Startzeitpunkte der Aufgabenfluenten, die mit dem Meta-Wert geplant werden, verglichen und der frühere Wert bevorzugt. In einem letzten Schritt wird schließlich noch ein Vergleich der Vorbedingungen durchgeführt

und dabei diejenigen Meta-Werte vorgezogen, deren Summe der frühesten Startzeitpunkte der Vorbedingungen größer ist.

Das fünfte und das sechste Kriterium wurde jeweils entwickelt, um die Auswahl der geplanten Aufgabe beziehungsweise der verwendeten Methode zielgerichteter beeinflussen zu können. Die Dekompositionstiefe wird für den Fall verglichen, dass die Meta-Variable mehrere zu planende Aufgaben enthält. Dabei wird durch die Wahl derjenigen Meta-Variable, die die Aufgabe mit der höchsten Dekompositonstiefe plant, eine Art Tiefensuche umgesetzt. Andersherum lässt sich über das Bevorzugen der Aufgabe mit der geringsten Tiefe auch eine Breitensuche realisieren. Mit dem Wichtungsfaktor wird für die Domänenmodellierung die Möglichkeit gegeben, zu spezifizieren, welche Methode oder welcher Operator bevorzugt werden soll, falls mehrere anwendbar sind. Dadurch kann die Suche für bestimmte Situationen beschleunigt werden, oder es lassen sich so auch Präferenzen für den zu erstellenden Plan festlegen. Beispielsweise könnte das Bringen einer Zuckerdose mit zwei Methoden modelliert werden. Mit der einen Methode holt der Roboter die Zuckerdose vom Tresen und mit der anderen Methode von einem anderen Tisch. Mit Wichtungsfaktoren lässt sich erstere Methode bevorzugen, ohne die Verwendung des Zuckers von einem Tisch auszuschließen.

Es sei allerdings angemerkt, dass diese Art Präferenzen im Vergleich zu Planungsverfahren, die sich explizit mit deren Optimierung beschäftigen, relativ rudimentär ist. Beispiele hierfür sind die Präferenzen in Form von weichen Constraints in PDDL3 [61] sowie deren von Sohrabi et al. [156] vorgeschlagene Erweiterung für HTN-Planung. Damit lassen sich beispielsweise auch bevorzugte Parameter oder Zustände spezifizieren. Zudem optimieren solche Planer ihre Pläne hinsichtlich dieser Präferenzen, während CHIMP den ersten gefundenen Plan verwendet. Erweiterungen von CHIMP in diese Richtung sind für zukünftige Arbeiten denkbar.

Neben diesen allgemeinen Heuristiken, die unabhängig von der Planungsdomäne sind, können zudem weitere Heuristiken implementiert werden, die auf die spezielle Domäne angepasst sind und so weiteres Hintergrundwissen berücksichtigen. Dies wurde bisher nicht betrachtet, könnte jedoch sinnvoll sein, um die Planung für konkrete Anwendungsfälle zu beschleunigen. Darüber hinaus könnten weitere domänenunabhängige Heuristiken für HTN-Planung aus der Literatur wie Beispielsweise jene, die auf Landmarken basieren [49], in zukünftigen Arbeiten in CHIMP integriert werden.

5.5 Planausführung

CHIMP kann nicht nur für die reine Plangenerierung eingesetzt werden, sondern er verfügt auch über Mechanismen, um die Pläne auf dem Roboter auszuführen. Hierfür verwendet er dasselbe Constraint-Netz, das schon für die Planung eingesetzt wurde und den resultierenden Plan darstellt.

Das Starten der Aktionen geschieht mittels eines auf Zeitleisten basierenden Verfahrens. Es entspricht jenem Verfahren, dass in [104] verwendet wird. Hierbei wird das gemeinsame Constraint-Netz mit den aktuellen temporalen Informationen aktualisiert. Dafür wird eine zusätzliche Va-

riable, welche die Zukunft repräsentiert, in das Constraint-Netz eingefügt. Beim Starten einer primitiven Aufgabe wird diese mit Hilfe eines temporalen `overlaps`-Constraints mit der Zukunft verbunden. Die temporal darauf folgenden, noch nicht gestarteten Aktivitäten liegen somit auch vollständig in der Zukunft. Wenn die primitive Aufgabe erfolgreich beendet wurde, wird dieser `overlaps`-Constraint wieder aus dem Constraint-Netz entfernt und stattdessen der Endzeitpunkt entsprechend gesetzt. Durch das Aktualisieren des temporalen Constraint-Netzes werden auch die Startzeiten der noch nicht ausgeführten Aktionen aktualisiert. Eine primitive Aufgabe wird auf dem Roboter ausgeführt, sobald ihre frühestmögliche Startzeit kleiner als die aktuelle Zeit ist. Auf diese Weise können auch mehrere Aktionen parallel ausgeführt werden.

Dadurch, dass CHIMP für die Planung und die Planausführung dieselbe Repräsentation in Form des gemeinsamen Constraint-Netzes verwendet, können dem Roboter während der Ausführung zusätzliche Aufgaben gegeben werden, woraufhin CHIMP versucht, diese mit den gegebenen Meta-Constraints zu planen und in den existierenden Plan zu integrieren. Hierfür braucht die Planausführung nicht unterbrochen zu werden. In Abschnitt 6.2.2 wird dies an einem Beispiel demonstriert.

Für die Planausführung sind einige verschiedene Erweiterungen denkbar, die teilweise bereits in Kapitel 3.1 diskutiert oder implementiert worden sind. Grundsätzlich sind jene Verfahren, die schon mit SHOP2 umgesetzt wurden, auch für die Ausführung eines CHIMP-Plans möglich. Hierbei ist CHIMPs ausdrucksstarke Repräsentation des Plans von Vorteil, da sie für die Ausführung weitere wichtige Informationen wie Beispielsweise die temporalen Relationen zwischen Aktionen liefern kann.

Eine naheliegende Erweiterung für die Planausführung ist eine konsistenzbasierte Ausführungsüberwachung (siehe Abschnitt 3.4.5). Dafür würden die vom Roboter zur Laufzeit beobachteten Fakten mit den im Plan erwarteten Vorbedingungen und Effekten verknüpft. Da der Plan bereits als Constraint-Netz vorliegt und die von Stefan Konecny in RACE entwickelten Komponenten zur konsistenzbasierten Ausführungsüberwachung ebenfalls im Meta-CSP-Framework entwickelt wurde, wäre keine zusätzliche Transformation zwischen den Repräsentationformen notwendig. Zudem müssen im Gegensatz zur Verwendung von SHOP2 keine zusätzlichen temporalen Information in den Plan eingefügt werden, da diese bereits Teil des Plans sind.

Auf diese Weise kann eine engere Kopplung der Planung und Ausführung beziehungsweise Ausführungsüberwachung erzielt werden, was aus bereits genannten Gründen vor allem für das Anwendungsgebiet der Robotik, in dem viele Unsicherheiten über den Zustand der dynamischen Umgebung vorliegen, von Bedeutung ist. Damit könnte der Plan zur Ausführungszeit angepasst werden oder Teile des Plans wie bei Kaelbling und Lozano-Perez [86] erst zur Ausführungszeit fertiggestellt werden. Beispiele hierfür sind das Lifting von Objekten und die Imagination, welche in Abschnitt 3.4.6 beschrieben wurden. Diese Umsetzungen ließen sich ebenso in CHIMP erzielen, mit CHIMP wäre jedoch auch eine noch engere Integration möglich.

Auch das Reparieren des Plans wäre eine mögliche Erweiterung, die bereits in anderen hierarchischen Planern umgesetzt wurde [9, 14, 173], und auch für CHIMP denkbar ist.

5.6 Diskussion

Der in diesem Kapitel vorgestellte Planer CHIMP kombiniert die Vorteile der HTN-Planung, wie sie beispielsweise von SHOP2 durchgeführt wird, und der hybriden Planung und Repräsentation als Meta-CSP. Dadurch kann CHIMP verschiedene Wissensrepräsentationsformen und Anforderungen für die Planung nutzen und darüber hinaus durch die Implementierung zusätzlicher Anforderungen in Form von Meta-Constraints noch erweitert werden.

CHIMPs HTN-Planungsstrategie, die in dem HTN-Meta-Constraint umgesetzt wurde, ist an SHOP2s PFD-Algorithmus angelehnt, welcher die Aufgaben in derselben Reihenfolge plant, in der sie später ausgeführt werden. Dadurch kennt SHOP2 in jedem Planungsschritt den vollständigen Zustand und kann so die Vorbedingungen direkt überprüfen. Da CHIMP Pläne mit parallel ausführbaren Aktionen planen sollte, war letzteres nicht direkt möglich. Er übernimmt jedoch das die vorwärts gerichtete Suchstrategie und das direkte Überprüfen aller Vorbedingungen bei der Anwendung eines Operators beziehungsweise einer Methode. Die Vorbedingungen und Effekte werden dabei in Form von kausalen, temporalen und symbolischen Constraints zu Zustandsfluenten repräsentiert und dauerhaft in das gemeinsame Constraint-Netz eingefügt. Auf diese Weise wird deren Korrektheit fortlaufend in den Grund-CSPs überprüft und sichergestellt.

In CHIMPs ausdrucksstarker Domänenbeschreibungssprache lassen sich die Aktionen und Anforderungen des Roboters feingranular modellieren und beispielsweise die genauen qualitativen sowie quantitativen temporalen Beziehungen zwischen Aufgaben, Vorbedingungen und Effekten abbilden. Diese lassen sich auch für die Planausführung auf dem Roboter nutzen und sollten daher an die Implementierung der unterschiedlichen Roboteraktionen und die Sensordatenverarbeitung angepasst sein. Der Planausführung werden somit wichtige Informationen zur Verfügung gestellt, die bei der zuvor beschriebenen Verbindung zwischen SHOP2 und der konsistenzbasierten Ausführungsüberwachung noch als zusätzliches Ausführungswissen bereit gestellt werden mussten. Durch die Verwendung relativ flexibler Standardconstraints kann der Modellierungsaufwand jedoch weiterhin gering gehalten werden. Unsicherheiten werden derzeit nicht explizit modelliert, jedoch lässt die Verwendung flexibler temporaler Intervalle Raum für temporale Varianzen in der Planausführung.

CHIMP repräsentiert seinen Plan in einem gemeinsamen Constraint-Netz, das alle für die Ausführung relevanten Informationen enthält und auch für das Starten der Roboteraktionen genutzt werden kann. Zudem können weitere Zielaufgaben während der Ausführung in diesen bestehenden Plan integriert und geplant werden. Dies wird im nächsten Kapitel an einem Beispiel auf dem Roboter demonstriert. Dadurch konnte im Vergleich zu der zuvor beschriebenen Variante mit SHOP2 und SMACH eine engere Verknüpfung zwischen Planung und Ausführung erzielt werden. Hier bestehen für zukünftige Arbeiten darüber hinaus Erweiterungsmöglichkeiten wie die Umsetzung einer konsistenzbasierten Ausführungsüberwachung oder das Reparieren von Plänen.

Die Verwendung des Meta-CSP-Frameworks führt allerdings auch zu einigen Einschränkungen. Eine Schwierigkeit war es, den HTN-Planungsprozess direkt auf den verwendeten Repräsentationsformalismus und den Meta-CSP-Backtracking-Algorithmus (siehe Algorithmus 4.1) zu übertragen und sich bei der Implementierung an die vom Meta-CSP-Framework vorgegebenen

Schnittstellen zu halten. Des Weiteren sieht jener Algorithmus mit Ausnahme der Backtrackingschritte kein Entfernen von Variablen oder Constraints des gemeinsamen Constraint-Netzes vor. Somit ist das Revidieren einer zuvor getroffenen Entscheidung, wie beispielsweise die Wahl einer Methode für eine Aufgabe, nur durch Backtracking zu diesem Entscheidungspunkt möglich. Dennoch überwiegt an dieser Stelle dank des Meta-CSP-Ansatzes der genannte Vorteil der Erweiterbarkeit zur Berücksichtigung zusätzlicher Anforderungen, den CHIMP gegenüber vergleichbaren aktuellen Planern wie FAPE hat. So konnte das im Meta-CSP-Framework vorhandene Ressourcenscheduling oder die Abschätzung der Fahrtdauer auf relativ einfache Weise integriert werden.

Kapitel 6

Experimente und Ergebnisse

In diesem Kapitel werden verschiedene Ergebnisse zu CHIMP beschrieben und diskutiert. Dazu werden zunächst einige Eigenschaften von CHIMP anhand der aus der Literatur bekannten theoretischen Hafenarbeiter-Domäne analysiert, zusätzliche Erweiterungsmöglichkeiten aufgezeigt und diese zum Teil auch implementiert. Anschließend wird auf zwei Szenarien der Restaurantdomäne eingegangen und die Planung und Ausführung auf einem realen PR2-Roboter demonstriert. Schließlich werden die Einsatzmöglichkeiten von CHIMP für die Planung einer planetaren Weltraummission als Anwendungsszenario eines aktuellen Forschungsprojekts beschrieben. Alle Laufzeitmessungen wurden auf einem Intel Core i7-4710MQ mit 2,50 GHz und 8 GB Arbeitsspeicher durchgeführt.

6.1 Die Hafenarbeiter-Domäne

Als weiteres Beispiel wird in diesem Abschnitt eine Variante der Hafenarbeiter-Domäne [63] verwendet, die in leicht abgewandelter Form auch als Demonstrationsbeispiel in [64] eingesetzt wird. Im Folgenden wird zunächst auf die Domäne und ihre Modellierung für CHIMP eingegangen. Auf dieser Basis werden verschiedene Eigenschaften von CHIMP diskutiert und dessen Erweiterungsmöglichkeiten demonstriert, um zu zeigen, wie CHIMP von zusätzlichen Wissensformen profitieren kann. Abschließend wird das Laufzeitverhalten für verschiedene Probleminstanzen verglichen.

6.1.1 Domänenmodellierung

In dieser Domäne wird ein Hafen modelliert, der aus mehreren Docks und Lagerbereichen für Container besteht. Container werden mit Kränen und Transportrobotern zwischen den Docks und den Lagerbereichen bewegt. Dazu werden die Container von einem Kran auf einen Roboter verladen, von diesem zu einem anderen Ort transportiert und mit einem Kran wieder in einem anderen Lagerbereich abgestellt. Die Container können gestapelt werden, aber ein Roboter kann

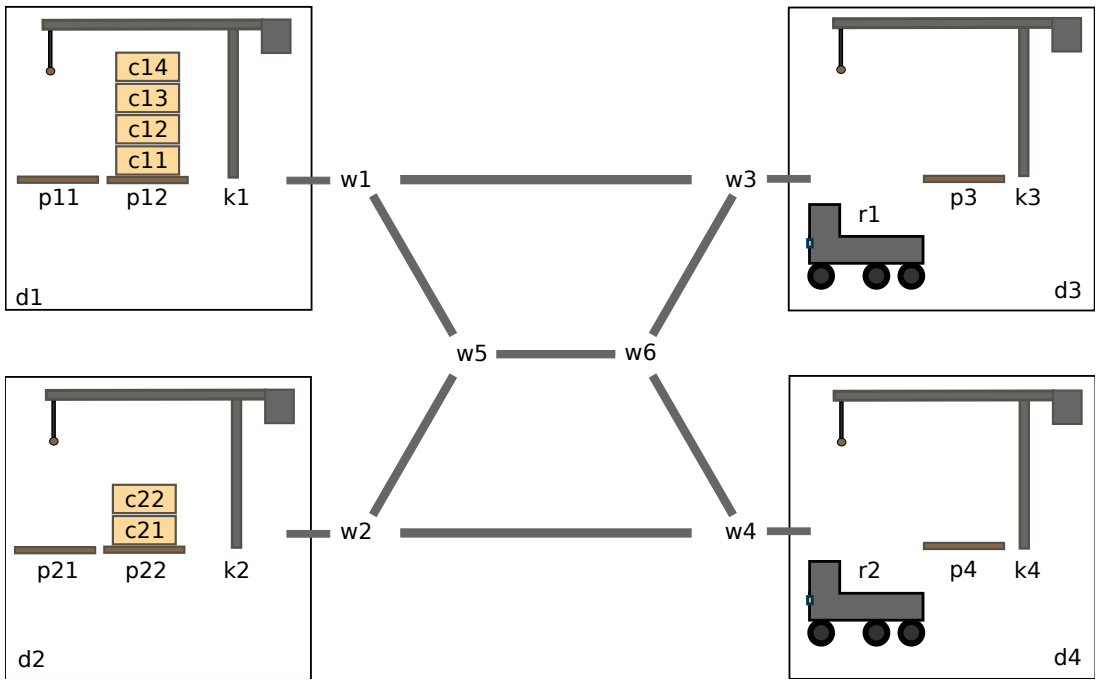


Abbildung 6.1: Beispiel einer Ausgangssituation in der Hafenarbeiter-Domäne. Darstellung in Anlehnung an [64].

immer nur einen Container transportieren, und es kann sich jederzeit immer nur ein Roboter in einem Dock befinden.

Abbildung 6.1 zeigt ein Beispiel für die Ausgangssituation eines Planungsproblems. Die beiden Transportroboter `r1` und `r2` befinden sich in den Docks `d3` und `d4`, die über Wege mit den Docks `d1` und `d2` verbunden sind. In diesen befinden sich gestapelte Container. Eine typische Planungsaufgabe ist es nun, die Container `c11` und `c21` in die Docks `d3` und `d4` zu befördern und auf die Stapel `p3` und `p4` abzustellen. Die Docks sind über verschiedene Straßen und Wegpunkte `w1` bis `w6` miteinander verbunden.

Die Domänenbeschreibung inklusive aller verwendeten Methoden und Operatoren wurde analog zu der ANML-Domänenbeschreibung erstellt, die bereits von Arthur Bit-Monnot für den Planer FAPE modelliert wurde[1], um so einen Vergleich beider Planer zu ermöglichen. Die vollständige Domänenbeschreibung für CHIMP ist in Anhang D.2 aufgelistet.

Die als Vorlage verwendete ANML-Domänenbeschreibung umfasste kausale und temporale Anforderungen, jedoch keine Ressourcen oder anderweitige Wissensformen. So war darin beispielsweise die Information, dass sich immer nur maximal ein Roboter in einem Dock befinden darf, mit dem Prädikat `d_occupant(?dock ?robot/free)`, dessen zweites Argument ein Roboter oder das Symbol `free` ist, modelliert, anstatt dafür eine Ressource zu verwenden. In der Domänenbeschreibung für CHIMP wurden zwei Abweichungen davon vorgenommen. Die Transitionen

[1] `https://github.com/athy/fape/tree/master/planning/domains/dwr` (Stand 02.06.2016)

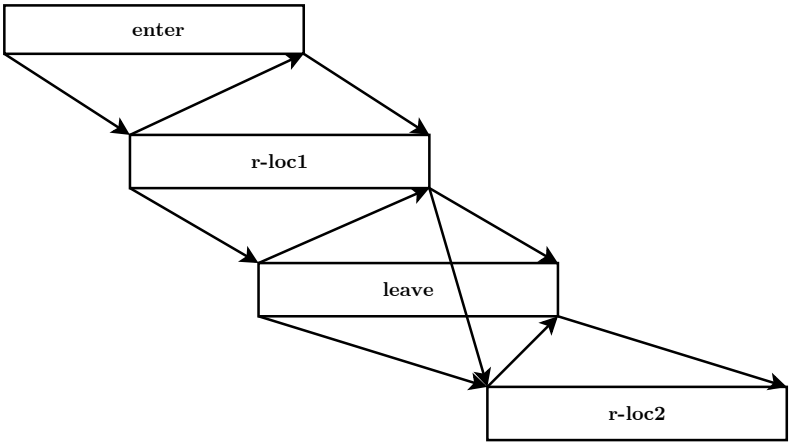


Abbildung 6.2: Beispiel möglicher temporaler Intervalle für das Befahren und sofortige wieder Verlassen eines Docks ohne Ressourcenscheduling. Die Pfeile geben temporale Ordnungsrelationen zwischen den Start- und Endzeitpunkten der Intervalle an.

zwischen zwei Zuständen wurden für CHIMP explizit mit der temporalen `meets`-Relation modelliert, wohingegen die ANML-Domäne einen Übergang zwischen den beiden Zuständen verwendet, der die gesamte Ausführungszeit der Aktion andauert. Welche dieser Varianten vorzuziehen ist, hängt letztendlich von der Umsetzung des realen Systems und der Art und Weise ab, wie die Zustände festgestellt und zwischen den verschiedenen Komponenten gesendet werden. Bei bestimmten Aktionen wie dem Befahren und anschließendem sofortigen Verlassen eines Docks kann dies dazu führen, dass der berechnete Startzeitpunkt des Verlassens noch vor dem Endzeitpunkt des Befahrens liegt. Dies ist vereinfacht in Abbildung 6.2 dargestellt. Der Effekt, dass sich der Roboter in dem Dock befindet (`r-loc1`), startet während des Befahrens und endet danach. Der Startzeitpunkt des anschließenden Verlassens hängt jedoch wiederum nur vom Startzeitpunkt von `r-loc1`, jedoch nicht von `enter` ab. Die zweite Aktion könnte also starten, sobald der Roboter in dem Dock ist. Um dieses zu vermeiden, wurde mittels einer Ressource festgelegt, dass ein Transportroboter nicht mehrere Aktionen parallel ausführen kann. Der Scheduler würde somit eine zusätzliche temporale Ordnungsrelation vom Endzeitpunkt von `enter` zum Startzeitpunkt von `leave` hinzufügen.

Für CHIMP wurden Zeitangaben für die minimale Dauer der Ausführen einer Aktion festgelegt, während dies in der Vorlage nur einen Zeitschritt umfasste. Zudem wurde die maximale Dauer für die Navigation zwischen zwei Wegpunkten begrenzt, um unnötige und redundante Wege zu vermeiden. Auf letzteres wird später noch ausführlicher eingegangen.

Listing 6.1 zeigt ein Beispiel für einen Plan für die zwei Aufgaben `robot_bring(r1 c13 p4)` und `robot_bring(r2 c21 p3)` ausgehend von der in Abbildung 6.1 dargestellten Situation. Die Planung dauerte 5,4 Sekunden. Das Abheben eines Containers mit einem Kran kann hierbei erfolgen, während sich der Roboter zu dem jeweiligen Dock bewegt.

```
1 !leave(r1 d3 w3)          [[1, 459999],  [10001, 469999]]
2 !unstack(k1 c14 p12)      [[1, 479999],  [10001, 489999]]
```

```
 3 !leave(r2 d4 w4)           [[1, 459999],  [10001, 469999]]
 4 !unstack(k2 c22 p22)       [[1, 479999],  [10001, 489999]]
 5 !move(r1 w3 w1)        [[10001, 469999],  [20001, 479999]]
 6 !stack(k1 c14 p11)     [[10001, 489999],  [20001, 499999]]
 7 !move(r2 w4 w2)        [[10001, 469999],  [20001, 479999]]
 8 !stack(k2 c22 p21)     [[10001, 489999],  [20001, 499999]]
 9 !enter(r1 d1 w1)       [[20001, 479999],  [40001, 499999]]
10 !enter(r2 d2 w2)       [[20001, 479999],  [40001, 499999]]
11 !unstack(k1 c13 p12)   [[40001, 499999],  [50001, 509999]]
12 !unstack(k2 c21 p22)   [[40001, 499999],  [50001, 509999]]
13 !put(k1 c13 r1)        [[50001, 509999],  [60001, 519999]]
14 !put(k2 c21 r2)        [[50001, 509999],  [60001, 519999]]
15 !leave(r1 d1 w1)       [[60001, 519999],  [70001, 529999]]
16 !leave(r2 d2 w2)       [[60001, 519999],  [70001, 529999]]
17 !move(r1 w1 w3)        [[70001, 529999],  [80001, 539999]]
18 !move(r2 w2 w5)        [[70001, 529999],  [80001, 539999]]
19 !move(r1 w3 w6)        [[80001, 539999],  [90001, 549999]]
20 !move(r2 w5 w1)        [[80001, 539999],  [90001, 549999]]
21 !move(r1 w6 w4)        [[90001, 549999], [100001, 559999]]
22 !move(r2 w1 w3)        [[90001, 549999], [100001, 559999]]
23 !enter(r1 d4 w4)      [[100001, 559999], [120001, 579999]]
24 !enter(r2 d3 w3)      [[100001, 559999], [120001, 579999]]
25 !take(k4 c13 r1)      [[120001, 579999], [130001, 589999]]
26 !take(k3 c21 r2)      [[120001, 579999], [130001, 589999]]
27 !stack(k4 c13 p4)     [[130001, 589999], [140001, 599999]]
28 !stack(k3 c21 p3)     [[130001, 589999], [140001, 599999]]
```

Listing 6.1: Die primitiven Aktionen eines Plans für das Bringen von `c13` nach `p4` mit Roboter `r1` und von `c21` nach `p3` mit `r2` in der in Abbildung 6.1 dargestellten Ausganssituation.

An dieser Problemstellung ist die Tatsache interessant, dass die beiden Aufgaben von einander abhängig sind, da sich immer nur ein Roboter in einem Dock befinden kann und das Zieldock jeden Roboters zu Beginn jeweils von dem anderen Roboter belegt ist. Somit kann der `ordering`-Constraint hierfür nicht eingesetzt werden und die Planung muss stattdessen ineinander verschachtelt erfolgen. Dies hat jedoch auch Auswirkungen auf den Suchraum, welche in Abschnitt 6.1.3 diskutiert werden.

Die Vorlage der Domänenbeschreibung sah nur die Aufgabe `bring` vor, die im Gegensatz zu `robot_bring` keinen Parameter für die Angabe des Roboters hat. In Kombination mit der Art, wie die `navigation`-Methode modelliert war, hat CHIMP in den getesteten Beispielen einen ungünstigen Suchpfad eingeschlagen und konnte auch nach zwei Minuten keinen Plan finden, wonach die Planung abgebrochen wurde. Mit dem im folgenden Abschnitt eingeführten Meta-Constraint für die Routenplanung, können auch für jene Aufgaben nach wenigen Sekunden Pläne gefunden werden.

6.1.2 Routenplanung als Meta-Constraint

Die Methoden für die Navigation waren in der als Ausgangsmaterial verwendeten FAPE-Domänenbeschreibung etwas unglücklich modelliert, da sie keine Informationen darüber enthalten, welche Wege zum Ziel führen. Auch hier kann CHIMP, wie schon in der RACE-Domäne, auf einfache Art mit einem weiteren Meta-Constraint ausgestattet werden, der speziell für die Navigationsaufgaben zuständig ist. Anders als in der RACE-Domäne führt dieser jedoch nicht nur eine Abschätzung der Fahrtdauer durch, sondern plant einen Pfad zwischen zwei gegeben Wegpunkten.

Die Routenberechnung geschieht dabei auf einfache Weise mit dem Algorithmus von Dijkstra [41]. Dessen Graph wird aus dem gemeinsamen Constraint-Netz durch Filtern nach Zustandsfluenten vom Typ `connected` generiert, die die Verbindung zweier Wegpunkte angeben. Die Meta-Variablen sind Constraint-Netze, die alle ungeplanten Aufgabenfluenten vom Typ `navigate` enthalten, zu denen kein `ordering`-Constraint von einem ungeplanten Vorgänger existiert. Die Meta-Werte entsprechen analog zur Aufgabendekomposition des HTN-Meta-Constraints Constraint-Netzen mit Aufgabenfluenten vom Typ `!move`, sowie Constraints zwischen diesen, sodass sie eine gültige Route repräsentieren. Diese Aufgabenfluenten werden anschließend wie gewöhnlich mit dem HTN-Meta-Constraint weiter geplant und dabei mit Zustandsfluenten für die Vorbedingungen verbunden, und es werden die zugehörigen Effekte erzeugt. In dieser Hinsicht übernimmt jener Meta-Constraint somit die Aufgaben einer HTN-Methode.

In der Reihenfolge der überprüften Meta-Constraints wurde dieser Meta-Constraint anstelle des Fahrtdauer-Meta-Constraints gesetzt, der nur für die RACE-Domäne verwendet wird. Er lässt sich ebenso auf einfache Weise dazuschalten oder deaktivieren und ist somit ein weiteres Beispiel dafür, wie zusätzliche Reasoner direkt in den Planungsprozess von CHIMP integriert werden und auf Basis derselben Repräsentation arbeiten können.

Es zeigte sich, dass dieser Meta-Constraint die Gesamtplanungsdauer teilweise deutlich verkürzen kann und dabei hilft, die im folgenden Abschnitt beschriebenen Probleme einzuschränken. So dauerte die Planung für das Beispiel in Listing 6.1 aus dem letzten Abschnitt nur 1,0 Sekunden anstatt der 5,4 Sekunden. Vor allem ist dieses Vorgehen auch für realistischere Problemstellungen mit mehr Wegpunkten und längeren Strecken zwischen den Docks realistischer.

Mit dem Navigations-Meta-Constraint lassen sich auch mehrere `bring`-Aufgaben ohne Angabe des Roboters planen. Die Planung dauerte für zwei Problembeispiele, in denen jeweils wiederum in der bekannten Ausgangssituation im ersten Fall die Container `c14` nach `p4` und `c22` nach `p4` beziehungsweise im zweiten Fall `c14` nach `p3` und `c22` nach `p3` gebracht werden sollten, 0,9 Sekunden beziehungsweise 3,6 Sekunden.

6.1.3 Auswirkungen der Meta-Variablendefinition auf den Suchraum und Einfluss des `ordering`-Constraints

Die Art, wie die Meta-Variablen des HTN-Meta-Constraints definiert sind, hat in Verbindung mit der Tiefensuche teilweise ungünstige Auswirkungen darauf, in welcher Reihenfolge Lösungen für

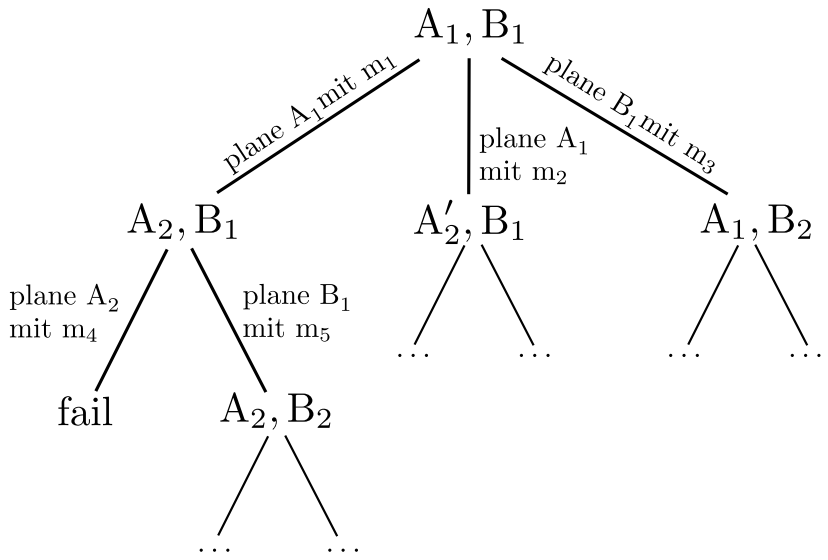


Abbildung 6.3: Beispiel für einen Suchbaum zweier Aufgaben A_1 und A_2. Die Knoten geben die Aufgabenfluenten der Meta-Variablen an und die Kanten die verwendeten Methoden.

Problemstellungen, die mehrere Zielaufgaben beinhalten, gesucht werden. Wie in Abschnitt 5.2.1 beschrieben, besteht ein Meta-Wert des HTN-Meta-Constraints aus all jenen Aufgabenfluenten, die noch nicht geplant wurden und zu denen kein `ordering`-Constraint von einer anderen ungeplanten Aufgabe existiert. Ein zugehöriger Meta-Wert expandiert jedoch immer nur genau eine dieser Aufgaben. Dadurch ergibt sich bei zwei Aufgaben eine Suchstruktur wie in Abbildung 6.3. Die erste Meta-Variable enthält die beiden ungeplanten Aufgabenfluenten A_1 und B_1. Angenommen, es wird zunächst A_1 mit der Methode m_1 expandiert, enthält die nächste Meta-Variable die Aufgaben A_2 (als Unteraufgabe von A_1) und wiederum B_1. Problematisch wird es nun, wenn die Dekomposition von A_1 mit m_1 keine gute Wahl war, sondern unabhängig von den Auswirkungen von B_1 in einen Sackgasse führt. In diesem Fall wird dennoch ausgehend von jener zweiten Meta-Variablen, die A_2 und B_1 enthält, weiter geplant, bis festgestellt wurde, dass sich A_2 auch nach Planung von B_1 und dessen Teilaufgaben nicht erfüllen lässt, um schließlich zu der ursprünglichen Meta-Variablen zurück zu kehren. Wenn die Aufgaben voneinander unabhängig sind, so ist diese Suche unnötig aufwendig, andererseits werden dabei für abhängige Aufgaben keine potentiellen Lösungsmöglichkeiten ausgeschlossen.

Durch die Angabe von `ordering`-Constraints zwischen Teilaufgaben kann der Domänenmodellierer die Reihenfolge, in denen die Teilaufgaben einer Aufgabe geplant werden, vorgeben und diesen Suchraum somit einschränken. Das oben genannte Problem besteht allerdings weiterhin, wenn bereits die Problembeschreibung mehrere Aufgaben ohne Angabe eines `ordering`-Constraints zwischen diesen enthält.

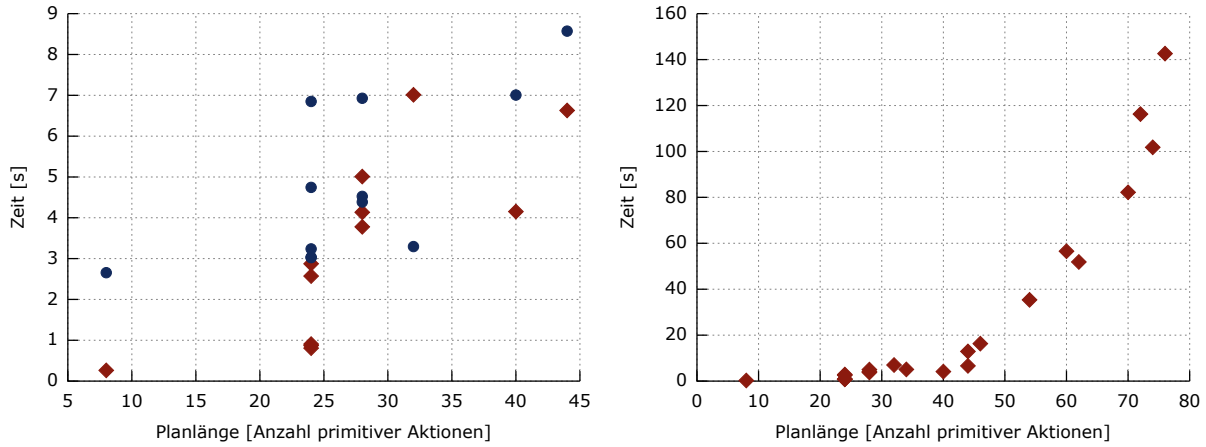
Ein Weg, hiermit umzugehen, ist es, dass der Planer zunächst annimmt, dass die Aufgaben voneinander unabhängig sind und somit streng sequentiell geplant werden können. Dafür wurde ein weiterer Meta-Constraint implementiert, dessen Meta-Variablen ungeplante Aufgabenfluenten oberster Ordnung (keine Teilaufgaben) sind, die keine ungeplanten Vorgänger haben. Die

Meta-Werte enthalten jeweils `ordering`-Constraints von einem ausgewählten Aufgabenfluenten zu allen anderen Aufgabenfluenten der Meta-Variablen. Zudem wird ein weiterer Meta-Wert erstellt, der keinen `ordering`-Constraint enthält, sondern nur eine Lösung der Meta-Variable markiert, sodass diese nicht erneut berechnet wird. Mit einer Werteordnungsheuristik wird für diesen Meta-Constraint sichergestellt, dass jener letzte Meta-Wert auch an letzter Stelle angewendet wird. Dieser Meta-Constraint ist eher als eine Erweiterung zu sehen und lässt sich durch CHIMPs modulare Struktur auf Wunsch aktivieren oder deaktivieren.

In der Hafenarbeiter-Domäne wurde dieser Umstand unter anderem durch die in der Vorlage unglücklich modellierten Methoden für die Navigation von einem Wegpunkt zu einem anderen deutlich. Für den in Abbildung 6.1 gezeigten Ausgangszustand trat bei der Planung der beiden Aufgaben, dass der Roboter `r1` den Container `c22` nach `p3` und der Roboter `r2` den Container `c14` nach `p4` transportieren sollte, ohne Navigations-Meta-Constraint die Situation auf, dass der Planer bei bestimmten Heuristiken für die Fahrt vom Wegpunkt `w3` nach `w2` in eine Sackgasse lief und beispielsweise mit der rekursiven `navigate`-Methode nach `w1`, anschließend wieder nach `w3` und dann nach `w6` fuhr, von wo es für den nächsten Aufgabenfluenten `!move(r1 w6 w5)` wegen der vorgegebenen maximalen Zeitdauer für die Navigation keine Lösungsmöglichkeit gab. Neben `!move(r1 w6 w5)` beinhaltete die Meta-Variable an dieser Stelle jedoch weiterhin den zweiten Aufgabenfluenten `robot_bring(r2 c14 p4)`. Mit letzterem wurde daher versucht weiter zu planen, was einen großen, unnötigen Backtracking-Aufwand erforderte, weshalb die Suche auch nach 20 Minuten noch zu keinem Plan führte und abgebrochen wurde. Da diese Navigation mehrmals in dem Plan vorkommt und diese Situation somit mehrfach auftritt, war dies in Kombination mit der Heuristik ein denkbar ungünstiges Problem für den Planer. Mit dem zusätzlichen Ordering-Meta-Constraint wurde hingegen nach 13 Sekunden ein Plan gefunden.

Andererseits gibt es in dieser Domäne auch Beispiele, in denen die Aufgaben der beiden Roboter nicht vollständig voneinander unabhängig sind: Wenn die Zielstapel der beiden Container vertauscht sind und der Roboter `r1` den Container `c14` zum Stapel `p4` statt zu `p3` und `r2` den Container `c22` zu `p3` statt zu `p4` bringen soll, so muss jeweils der andere Roboter das Dock, von dem aus er startet, zunächst verlassen, da sich immer nur ein Roboter in einem Dock befinden kann. Der Ordering-Meta-Constraint ist in diesem Fall also nicht hilfreich. Ohne Verwendung des Meta-Constraints für die Navigation führt dies wieder zu der langen Laufzeit wie zuvor bei der Wahl der ungünstigen Werteordnungsheuristik. Mit dem Navigations-Meta-Constraint lassen sich jedoch die Auswirkungen vergleichen. So dauerte die Planung für die oben genannte Aufgabe ohne den Ordering-Meta-Constraint bei fünf Durchläufen gemittelt 0,56 Sekunden und mit Ordering-Meta-Constraint 2,5 Sekunden. Dabei wurden im ersten Fall 37 Meta-Variablen und 40 Meta-Werte und im zweiten Fall 67 Meta-Variablen und 75 Meta-Werte benötigt.

Eine weitere Möglichkeit wäre es, ebenfalls in der Domänenbeschreibung angeben zu können, dass die Planung einer Aufgabe von anderen Aufgaben unabhängig ist und somit immer eine alleinige Meta-Variable darstellen kann. Dennoch ist dies ein Punkt, an dem CHIMP in zukünftigen Arbeiten noch verbessert werden kann. Ein Ansatzpunkt wäre, die Interaktionen zwischen den Aufgaben anhand der Domänenbeschreibung zu analysieren und dies in die Suche einzubeziehen. Zudem könnte auch wie von Bit-Monnot et al. eine Erreichbarkeitsanalyse durchgeführt werden, um den Suchraum einzuschränken [15].



(a) Zwei Zielaufgaben (b) Zwei bis sechs Zielaufgaben

Abbildung 6.4: Planungsdauer von CHIMP und FAPE in der Hafenarbeiter-Domäne in Abhängigkeit von der im Plan enthaltenen Operatoren. Die Laufzeiten von CHIMP sind als rote Quadrate und jene von FAPE als blaue Punkte eingezeichnet. Links bestanden die Probleme aus zwei und rechts aus zwei bis sechs `robot_bring`-Aufgaben. Zudem wurde die Tiefe der zu transportierenden Container variiert. Die Laufzeiten von FAPE sind nur in der linken Grafik eingezeichnet, da er für mehr als zwei Zielaufgaben jeweils innerhalb von 600 Sekunden keinen Plan finden konnte.

6.1.4 Laufzeitmessungen

Ein Vorteil der akademischen Hafenarbeiter-Domäne ist dessen Skalierbarkeit durch das Hinzufügen weiterer Container und das Stellen mehrerer Planungsaufgaben. Es wurden 22 Problemstellungen getestet, die je zwischen zwei und sechs `robot_bring`-Aufgaben umfassten. Zum Vergleich wurden auch mit dem Planer FAPE[2] [47] in der Master-Version `bf08629` vom 02.06.2016 entsprechende Problemstellungen getestet. Variiert wurden die Stapelhöhen, die Positionen der zu bewegenden Containern in den Stapeln sowie die Anzahl der Container, die zu transportieren waren, sodass jeweils zwischen zwei und sechs Zielaufgaben zu planen waren.

Für CHIMP wurde eine Heuristik verwendet, die die Dekompositionstiefe der geplanten Aufgabenfluenten, die Wichtungsfaktoren der Methoden sowie die Summe der frühesten Startzeitpunkte der Vorbedingungen berücksichtigt. Zudem wurde der im vorherigen Abschnitt beschriebene Meta-Constraint für die Routenplanung eingesetzt. Die Unifikation neuer Aufgaben zu bereits geplanten Aufgaben wurde hingegen deaktiviert, da sie für diese Planungsdomäne nicht hilfreich ist. CHIMP generierte dabei für jede Problemstellung einen Plan; FAPE fand hingegen nur für Probleme mit zwei Zielaufgaben einen Plan und war in den anderen Fällen auch nach einem Zeitlimit von 10 Minuten nicht erfolgreich. Die Laufzeiten von CHIMP und FAPE sind in den Abbildungen 6.4 in Abhängigkeit von der Planlänge als Anzahl der im Plan enthaltenen primitive Aktionen aufgetragen. Die Messwerte und weitere Daten wie die Anzahl der angewendeten Methoden, Meta-Werte und Meta-Variablen sind zudem im Anhang in Tabelle C.1 aufgelistet.

[2] `https://github.com/athy/fape`

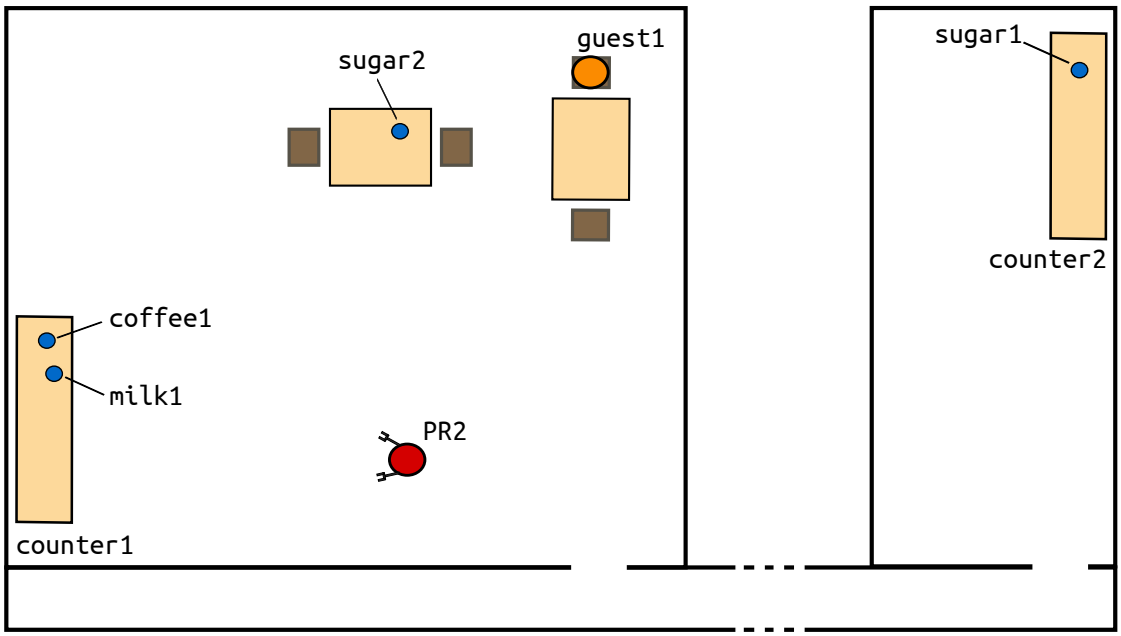


Abbildung 6.5: Schematische Darstellung der Restaurantumgebung und des initialen Zustands. Der Tresen `counter2` befindet sich in einer weit entfernten Küche am Ende des Flures.

CHIMP war dabei in allen bis auf einen Fall schneller als FAPE. Hierbei kommt auch der Vorteil, dass CHIMP mit dem Meta-Constraint für die Routenplanung erweitert werden konnte, zum Tragen. Deutliche relative Unterschiede, gibt es auch bei kurzen Plänen mit einer Länge von bis zu 25 Operatoren wie dies häufig bei Anwendungen mit Robotern der Fall ist. Die rechte Abbildung zeigt, dass CHIMP auch für umfangreichere Problemstellungen noch Pläne finden kann, die Laufzeit jedoch stark zunimmt. Dies liegt unter anderem an dem steigenden Aufwand für die Constraint-Propagierung und das Backtracking in größeren temporalen Constraint-Netzen und an der größeren Auswahl an Möglichkeiten, die der Planer bei der Bindung von Fluenten zu Vorbedingungen zur Verfügung hat.

6.2 Demonstration auf einem PR2 in der RACE-Domäne

Nach der eher theoretischen Hafenarbeiter-Domäne wird nun wieder auf den Einsatz von CHIMP in der Restaurantumgebung und die Ausführung der Pläne mit dem Roboter eingegangen.

6.2.1 Planung und Ausführung auf einem PR2-Roboter

In diesem Abschnitt wird die Verwendung von CHIMP auf einem PR2-Roboter für eine vergleichsweise umfangreiche Aufgabe demonstriert. Der Ausgangszustand ist in Abbildung 6.5 dargestellt, und Listing 6.2 zeigt eine angepasste Version der entsprechenden Problembeschrei-

```
1  (Problem
2  [...]
3   (Fluent c0 Connected(placingAreaEastRightCounter1
4     manipulationAreaEastCounter1 preManipulationAreaEastCounter1))
5   (Constraint Release[0,0](c0))
6   (Fluent c1 Connected(placingAreaWestLeftTable1
7     manipulationAreaNorthTable1 preManipulationAreaNorthTable1))
8   (Constraint Release[0,0](c1))
9  [...]
10  (Fluent f1 HasArmPosture(leftArm1 ArmTuckedPosture))
11  (Fluent f2 HasArmPosture(rightArm1 ArmTuckedPosture))
12  (Fluent f3 HasTorsoPosture(TorsoDownPosture))
13  (Fluent f4 RobotAt(floorAreaTamsRestaurant1))
14  (Fluent f6 Holding(rightArm1 nothing))
15  (Fluent f7 Holding(leftArm1 nothing))
16  (Fluent f8 On(coffeeJug1 placingAreaEastRightCounter1))
17  (Fluent f9 On(milkPot1 placingAreaEastRightCounter1))
18  (Fluent f10 On(sugarPot1 placingAreaEastRightCounterOS1))
19  (Fluent f11 On(sugarPot2 placingAreaWestRightTable1))
20  (Fluent i0 Type(Coffee coffeeJug1))
21 [...]
22  (Task t0 serve_coffee_to_guest(placingAreaNorthLeftTable2))
23  (Constraint Deadline[0,1200000](t0))
24 )
```

Listing 6.2: Auszug aus der Problembeschreibung für das Servieren eines heißen Kaffees mit Milch und Zucker.

bung. Der Tresen `counter1` befindet sich an einer Wand des Raumes, in dem der Roboter startet. Gegenüber des Tresens sind die beiden dicht beieinander stehenden Tische platziert. Ein Milchkännchen `milk1` und ein Kaffee `coffee1` befinden sich auf dem Tresen (Zeilen 16 und 17). Da der PR2 mit seinen Greifern keine reale Kaffeekanne manipulieren kann, wird für diesen Versuch eine Dose mit löslichem Kaffee verwendet. Eine Zuckerdose `sugar2` steht auf `table1` und eine weitere Zuckerdose `sugar1` steht auf dem zweiten Tresen `counter2`, der sich in einem weit entfernten Raum befindet (Zeilen 18 und 19). Der Roboter kennt alle Positionen der genannten Objekte. Zudem sitzt ein Gast an `table2`. Der Roboter wird angewiesen, einen heißen Kaffee mit Milch und Zucker zu servieren (Zeile 22).

Für dieses Szenario benötigt CHIMP seine ausgeprägten Wissensrepräsentations- und Schlussfolgerungsmöglichkeiten. Damit der Gast einen heißen Kaffee trinken kann, darf die Ausführung der Gesamtaufgabe nicht zu lange dauern. Dafür wurde der späteste Endzeitpunkte der Zielaufgabe `serve_coffee_to_guest` auf 20 Minuten gesetzt (Zeile 23). Die Anforderung, dass neben dem Kaffee auch Milch und Zucker serviert werden müssen, wurde in die Methoden der Zie-

```
1  !move_base(preManAreaEastCounter1)
2    [[   12, 464970], [30012, 494970]]
3  !move_torso(TorsoUpPosture)
4    [[30013, 498972], [34013, 502972]]
5  !tuck_arms(ArmUnTuckedPos ArmUnTuckedPos)
6    [[30013, 494971], [34013, 498971]]
7  !move_arm_to_side(leftArm1)
8    [[34014, 498972], [38014, 502972]]
9  !move_arm_to_side(rightArm1)
10   [[34014, 498972], [38014, 502972]]
11 !move_base_blind(manAreaEastCounter1)
12   [[38015, 502973], [42015, 506973]]
13 !pick_up_object(milkPot1 rightArm1)
14   [[42016, 506974], [46016, 510974]]
15 !pick_up_object(coffeeJug1 leftArm1)
16   [[46016, 510974], [50016, 514974]]
17 !move_base_blind(preManAreaEastCounter1)
18   [[50017, 514975], [54017, 518975]]
```

Listing 6.3: Die ersten 9 primitiven Aktionen eines Plans für das Servieren eines heißen Kaffees mit Milch und Zucker inklusive den flexiblen temporalen Intervallen.

laufgabe modelliert. Die vollständige Domänenbeschreibung ist in Listing D.1 in Anhang D.1 aufgeführt und besteht aus 11 Operatoren und 28 Methoden.

Nachdem dem Roboter diese Zielaufgabe gegeben wurde, erstellte CHIMP einen Plan, der 40 Aktionen beinhaltete, von denen die ersten 9 Aktionen mit ihren Prädikaten und flexiblen temporalen Intervallen in Listing 6.3 aufgeführt sind. Die temporalen Intervalle sind hier in Millisekunden angegeben. Es sei allerdings angemerkt, dass der vollständige Plan mit allen Aufgabenfluenten, Zustandsfluenten und Constraints zwischen diesen jedoch weitaus mehr Informationen als diese bloße Auflistung der Operatoren enthält. Bei der Demonstration mit dem realen Roboter zur RACE-Abschlusspräsentation Anfang 2015 dauerte die Planung auf jenem System, dass für die Laufzeittests in diesem Abschnitt verwendet wurde, 48 Sekunden, durch Optimierung des Planers und kleinere Änderungen der Domänenbeschreibung konnte die Planungsdauer mittlerweile jedoch deutlich auf reduziert werden und beträgt derzeit 1,4 Sekunden.

Der Plan ist partiell geordnet, und verschiedene Aktionen können parallel ausgeführt werden. So haben die Aktionen `!move_torso` und `!tuck_arms` (Zeilen 3 bis 6) sowie das Bewegen der Arme zur Seite (Zeilen 7 bis 10) dieselben frühesten Startzeiten und können jeweils gleichzeitig ausgeführt werden. Die Ausführungszeit des Gesamtplans wird somit im Vergleich zu dem in Kapitel 3 verwendeten Planer SHOP2 verkürzt. Das Greifen der Milch und des Kaffees kann hingegen wie bereits erwähnt nicht parallel ausgeführt werden. Dies wurde von CHIMP berücksichtigt und das Greifen der Milch (Zeilen 13, 14) mittels Ressourcenscheduling vor das Greifen des Kaffees (Zeilen 15, 16) geordnet.

Abbildung 6.6: Ausführung des Plans für das Servieren eines heißen Kaffees mit Milch und Zucker. Das linke Bild zeigt den PR2 vor dem Greifen der Milch und des Zuckers vom Tresen. In der Mitte greift der PR2 den Zucker von `table1`, und rechts stellt er ihn wieder auf `table2` ab, womit die Aufgabe erfolgreich ausgeführt wurde.

Nach dem Greifen der beiden Objekte fährt der Roboter zu `table2` und platziert sie vor dem Gast. Auch hier werden die entsprechenden Aktionen wiederum parallel ausgeführt. CHIMP hat somit das Bewegen des Kaffees und der Milch ineinander verschachtelt, ohne dafür speziell modellierte Methoden zum gleichzeitigen Bewegen zweier Objekte zu benötigen.

Anschließend muss dem Gast noch Zucker gebracht werden. Hierfür holt der Roboter die Zuckerdose `sugar2` vom benachbarten Tisch und bringt sie ebenfalls dem Gast. Anhand des temporalen Constraint-Netzes wird die Mindestdauer des Plans ermittelt und dadurch sichergestellt, dass der späteste Endzeitpunkt der Zielaufgabe nicht überschritten wird. Die Dauer des Fahrens zwischen zwei Bereichen wird dabei mit Hilfe des Fahrtdauer-Meta-Constraints ermittelt. So wird ausgeschlossen, dass der Roboter `sugar1` vom weit entfernten Tresen holt, da dies zu lange dauern und der Kaffee kalt würde.

Die Verwendung von CHIMP bietet in diesem Szenario im Vergleich zu SHOP2 somit zwei Vorteile: Zum einen wird die Gesamtausführungszeit durch die zeitgleiche Ausführung parallelisierbarer Aktionen verkürzt, und zum anderen werden temporal ungültige Pläne ausgeschlossen. Diese Vorteile rechtfertigen eine längere Planungsdauer, welche im Vergleich zu der Gesamtausführungszeit von circa 13 Minuten sehr gering ist.

Abbildung 6.6 zeigt drei Bilder der erfolgreichen Ausführung des Plans mit dem PR2. Es sei angemerkt, dass die im Plan vorgesehene zeitgleiche Ausführung bestimmter Aktionen bei den Tests mit dem realen PR2 auf Grund von Einschränkungen der Implementierung der ROS-Services in der Java-Version von ROS zunächst nicht möglich war. Dieses Problem konnte später jedoch umgangen und die zeitgleiche Ausführung erfolgreich in der Gazebo-Simulation demonstriert werden. Es bestand keine Möglichkeit mehr, dies auf dem realen Roboter zu testen. Da die be-

treffenden Komponenten in Simulation und Realität identisch sind, ist davon auszugehen, dass die zeitgleiche Ausführung somit auch auf dem realen PR2 funktionieren würde.

6.2.2 Erweiterung des Plans während der Ausführung

In dem vorhergehenden Demonstrationsszenario wurde der Plan initial geplant und anschließend vollständig auf dem Roboter ausgeführt. Wie bereits in Abschnitt 5.5 erwähnt wurde, kann CHIMP einen bestehenden Plan jedoch auch zur Ausführungszeit erweitern. Diese Fähigkeit wurde in der Publikation [163], die eine Vorabversion dieses Abschnitts darstellt, beschrieben. Da nach Ende von RACE kein Zugriff auf den realen Roboter mehr möglich war, wurde dies in einer Gazebo-Simulation durchgeführt, was auf der hier zu testenden Abstraktionsebene, in der es nicht explizit um gezielte Objektmanipulation oder Objekterkennung geht, keinen signifikanten Unterschied ausmacht.

Die initiale Situation dieses Szenarios entsprach dem vorherigen Beispiel aus Abschnitt 6.2.1. Nun bekam der Roboter allerdings zunächst die Aufgabe gestellt, lediglich einen Kaffee zum Tisch `table2` zu bringen und darauf zu platzieren. Dieses Ziel wurde ihm in Form des folgenden Aufgabenfluenten gegeben:

$$f_{G1} = (\text{move\_object}(\text{coffeeJug1 placingAreaNorthLeftTable2}),\ [[0, \infty), [0, \infty]],\ \emptyset).$$

Für diese vergleichsweise kleine Aufgabe erstellte CHIMP einen Plan, der 23 primitive Aktionen umfasste. Der PR2 begann direkt damit, diesen auszuführen, indem er als Erstes zum Tresen fuhr. Während der Fahrt wurde dem Roboter eine weitere Aufgabe f_{G2} gegeben: Er sollte ebenfalls das Milchkännchen, das sich auch auf jenem Tresen befand, zum Tisch `table2` bringen und dort abstellen:

$$f_{G2} = (\text{move\_object}(\text{milkPot1 placingAreaNorthLeftTable2}),\ [[0, \infty), [0, \infty]],\ \emptyset).$$

Dieser Aufgabenfluent wurde in CHIMPs gemeinsames Constraint-Netz, das den ersten Plan enthielt und ebenfalls für dessen Ausführung verwendet wurde, eingefügt. Da dieser neue Aufgabenfluent noch nicht als geplant markiert worden war, stellte er aus Sicht des HTN-Dekompositions-Meta-Constraints einen Konflikt dar, den es aufzulösen galt. An dieser Stelle fuhr CHIMP daher mit seiner gewöhnlichen Planung fort. Dazu teilte er zunächst die Aufgabe `move_object` in die Teilaufgaben `get_object` und `put_object` auf. Bei der Planung von `get_object` erkannte CHIMP, dass er den Effekt der ersten Aktion des initialen Planes, dass sich der Roboter nach dem Fahren in dem Bereich `preManipulationAreaCounter1` befindet, für eine der Vorbedingungen jener neuen `get_object`-Aufgabe verwenden kann. Diese Aufgabe wurde dadurch nur in die beiden Teilaufgaben `assume_manipulation_pose` und `!pick_up_object` dekomponiert. Eine gesonderte Fahrt zum Tresen war somit nicht notwendig. Im nächsten Schritt unifizierte CHIMP diese neue `assume_manipulation_pose`-Aufgabe mit der entsprechenden Aufgabe des ersten Plans. Auf diese Weise wurde ein Plan für das Transportieren des Milchkännchens in den bestehenden Plan eingeflochten. Dadurch kam `!pick_up_object` als einzige primitive Aufgabe des `get_object`-Zweiges hinzu. Wie schon im vorherigen Demonstrationsbeispiel wird sie durch das Ressourcenscheduling zeitlich vom Greifen des Kaffees separiert.

Mit der Aufgabe `put_object` wurde auf ähnliche Weise verfahren. Insgesamt bestand der aktualisierte Plan aus 25 komplexen Aufgaben und 18 Aktionen. Das Integrieren der neuen Aufgabe konnte in 0,6 Sekunden und damit noch während der Fahrt zum Tresen durchgeführt werden, ohne dass der Roboter dafür pausieren musste.

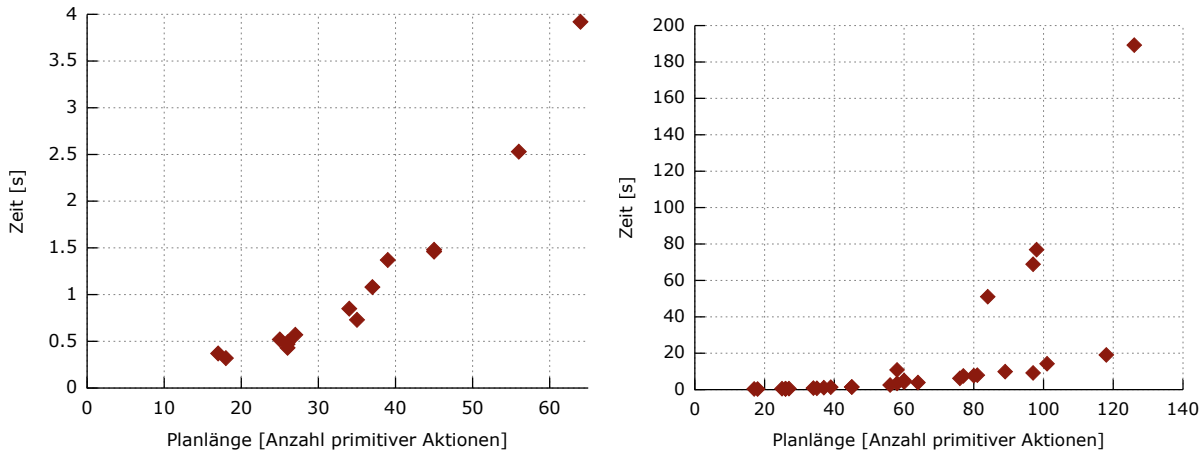
6.2.3 Laufzeitverhalten von CHIMP in der RACE-Domäne

Die RACE-Domäne wurde für verschiedene Probleminstanzen getestet. Dabei wurden sowohl die initialen Positionen der Objekte und des Roboters als auch die Anzahl der zu erfüllenden Aufgaben variiert. Die Zielaufgaben waren unterschiedliche Instanzen von `move_object`, also das Transportieren eines Objekts zu einem bestimmten Gebiet auf einem Tisch.

Eine Schwierigkeit bestand für den Planer darin, dass in den Problembeschreibungen keine temporalen oder `ordering`-Constraints vorgegeben wurden. Bei der `serve_coffee_to_guest`-Aufgabe in dem vorhergehenden Beispiel mussten auch mehrere Objekte transportiert werden, jedoch wurde dort mit `ordering`-Constraints, die Bestandteil der zur Zielaufgabe gehörigen Methode waren, vorgegeben in welcher diese geplant werden sollten. Auch bei diesen Tests gab es somit bei mehreren Zielaufgaben die zuvor in Abschnitt 6.1.3 beschriebenen Probleme durch den großen Suchraum. Doch auch diese ließen sich mit jenem Ordering-Meta-Constraint, der zunächst eine Unabhängigkeit der Zielaufgaben annimmt und dazu `ordering`-Constraints zwischen diesen hinzufügt, beheben.

In diesen Versuchen zeigte sich, dass auch das integrierte Ressourcenscheduling einen Einfluss auf die Reduzierung des Suchraums hat und die Suche beschleunigt. Neben den Operatoren, kann auch für Methoden ein Ressourcenverbrauch spezifiziert werden, der bei deren Anwendung der jeweiligen Aufgabe zugeordnet wird. Da ein PR2 mit seinen zwei Armen maximal zwei Objekte gleichzeitig transportieren kann, wurde die Kapazität auf den Wert zwei gesetzt. Nachdem die `move_object`-Methode dreimal auf entsprechende Aufgaben angewendet wurde, wird ein Ressourcenscheduling durchgeführt und sie temporal getrennt. Durch diese temporale Trennung fallen bei der weiteren Planung der Aufgaben einige frühere Zustandsfluenten als mögliche Vorbedingungen weg. Andernfalls würde erst zu einem späteren Zeitpunkt festgestellt, dass sich mit diesen Bindungen von Vorbedingungen zu Zustandsfluenten durch die begrenzte Anzahl an Armen keine Lösung ergibt und daher Backtracking durchgeführt werden müsste.

Abbildung 6.7 zeigt die benötigte Dauer der Planung für die verschiedenen Probleminstanzen in Abhängigkeit von der im gefundenen Plan enthaltenen Aktionen. Die verwendete Heuristik bevorzugte dabei Unifikationen, eine Hohe Dekompositionstiefe, eine Hohe Wichtung der Methoden oder Operatoren sowie neuere Zustandsfluenten in den Vorbedingungen. Für bis zu vier Aufgaben betrug die Planungsdauer stets unter 4 Sekunden und in den meisten Fällen unter 1,5 Sekunden (Abbildung 6.7(a)). Damit liegt sie unter der Zeitdauer, die der PR2 beispielsweise für das Hochfahren seines Torsos oder gar für das Greifen einer Tasse benötigt. Bei einer größeren Anzahl zu bewegender Objekte variierten die Laufzeiten der verschiedenen Problemstellungen stärker (Abbildung 6.7(b)). In drei Fällen konnte auch nach 5 Minuten kein Plan gefunden werden, was zum einen an dem größeren Suchraum und zum anderen an der mit der Planlänge einhergehenden Größe des temporalen Constraint-Netzes, dessen Constraint-Propagierung



(a) Zwei bis vier Zielaufgaben

(b) Zwei bis zehn Zielaufgaben

Abbildung 6.7: Planungdauer in Abhängigkeit von der im Plan enthaltenen Operatoren. Links bestanden die Probleme aus zwei bis vier und rechts aus zwei bis zehn `move_object`-Aufgaben.

somit aufwändiger wird, liegt. Dies schlägt sich vor allem beim Backtracking und dem damit verbundenen Entfernen von Constraints nieder, da in diesem Fall gesamte Constraint-Netz erneut berechnet werden muss. Die Varianzen in den Laufzeiten hängen damit auch stark davon ab, ob der Großteil des Backtrackings zu einem frühen oder späten Zeitpunkt in der Planung durchgeführt wird.

In Tabelle C.2 in Anhang C sind neben den gemessenen Laufzeiten auch weitere Indikatoren wie die Anzahl der im gemeinsamen Constraint-Netz enthaltenen Fluenten und Constraints sowie die Anzahl der bei der Planung betrachteten Meta-Variablen und Meta-Werte aufgeführt. Vor allem die Anzahl der angewendeten Meta-Variablen ist dabei in Relation zu der Länge des Plans ein Indikator für den benötigten Backtracking-Aufwand.

Beim Vergleich der Problemstellungen zeigte sich, dass wie zu erwarten ein höherer Planungsaufwand mit komplexeren Problemstellungen einherging, wie es beispielsweise der Fall war, wenn sich die zu bewegenden Objekte initial auf unterschiedlichen Tresen und Tischen befanden und zu unterschiedlichen Zielgebieten zu transportiert waren. Die Planungsdauer für bis zu acht zu bewegende Objekte betrug in vielen Fällen unter 10 Sekunden und war damit für die Anwendung auf einem Roboter immer noch im Rahmen, ohne eine zu lange Verzögerung hervorzurufen. Es sei zudem angemerkt, dass diese Ergebnisse für eine Große Anzahl an Objekten, in realen Umgebungen mit weiteren Akteuren, die die Umgebung verändern, und mit den gegebenen nur begrenzt zuverlässigen Fähigkeiten des Roboters eher theoretischer Natur sind. Hier wäre es ohnehin vorzuziehen, nur einen Plan für eine geringe Anzahl von Objekten zu erstellen und nach dessen Ausführung auf Basis aktualisierter Informationen über die Umgebung erneut zu planen.

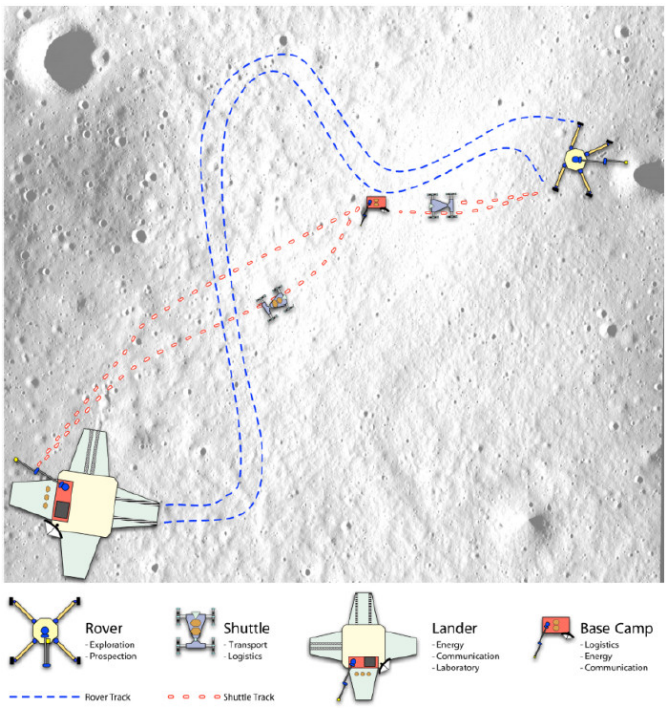


Abbildung 6.8: Schematische Darstellung einer logistischen Kette, die aus einem Lander, Rover, zwei Shuttles und einem Base-Camp besteht. Die Grafik stammt aus [157].

6.3 Modellierung einer Mission eines Weltraumszenarios

Da CHIMP domänenübergreifend einsetzbar ist, ist die beschriebene Restaurantdomäne nur ein Beispiel einer Vielzahl von Einsatzgebieten. Als weiteres Beispiel wurde im Rahmen des Projekts TransTerrA[3] am DFKI ein Szenario aus einer planetaren Weltraummission modelliert.

TransTerrA beschäftigt sich unter anderem mit der semiautonomen Exploration von Planeten oder Monden mit logistischen Ketten, die aus mehreren unterschiedlichen Robotern bestehen können. Neben der reinen Exploration sieht das Szenario auch den Aufbau einer Infrastruktur für spätere bemannte Missionen vor. Abbildung 6.8 zeigt eine schematische Darstellung einer solchen logistischen Kette mit den verwendeten Robotersystemen. Eine detaillierte Beschreibung des Szenarios wird von Sonsalla et al. gegeben [157]. Ein Rover, der für Explorationsaufgaben und das Sammeln von Gesteinsproben zuständig ist, wird von einem kleineren Shuttle unterstützt. Für die Überwindung längerer Strecken zwischen der Landestation und den Einsatzorten der Roboter werden zudem stationäre Base-Camps eingerichtet, die Energiespeicher oder Bodenproben lagern und als Kommunikationsmodule genutzt werden können. Der Lander dient als Basisstation zur Kommunikation mit der Bodenstation und als Depot für Base-Camps sowie

[3]Gefördert von der Raumfahrt-Agentur des Deutschen Zentrums für Luft- und Raumfahrt e. V. (DLR) mit Mitteln des Bundesministeriums für Wirtschaft und Technologie (BMWi) aufgrund eines Beschlusses des Deutschen Bundestages. Förderkennzeichen: 50RA1301

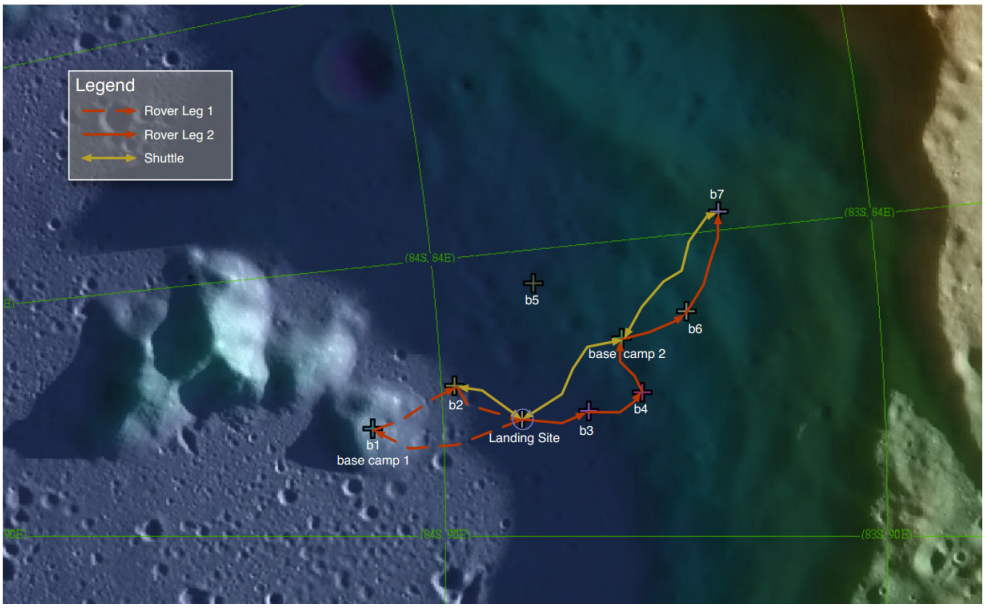


Abbildung 6.9: Beispiel einer vollständigen Mission aus TransTerrA mit einem Rover und einem Shuttle auf dem Mond. Die Pfade des Rovers sind in rot und die des Shuttles in gelb eingezeichnet. Die Grafik stammt aus [157].

portable, modulare Nutzlast-Elemente. Letztere können an die verschiedenen Robotersysteme angedockt werden und dadurch beispielsweise durch die Ausstattung mit einer Kamera zusätzliche Funktionalitäten bereitstellen. Sie können aber ebenso als Container für den Transport von Gesteinsproben eingesetzt werden.

Aufgrund jener Erweiterbarkeit der Robotersysteme wird in TransTerrA von Thomas Röhr ein spezieller Konfigurationsplaner entwickelt und eingesetzt, der den übergeordneten Ablauf der Mission plant [146]. Dabei wird der zeitlich-räumliche Ablauf geplant, an welchen Orten sich die einzelnen Roboter und Container zu bestimmten Zeitpunkten befinden und wie die Roboter konfiguriert sein müssen, um die vorgegebenen Missionsziele mit den gegebenen Ressourcen zu erreichen. Da diese Konfigurationsplanung keine Handlungsplanung durchführt, muss der Plan anschließend noch verfeinert werden, damit er auf den Robotern ausführbar wird. Insgesamt liegt somit ein vierstufiges Verfahren vor: Die Ziele der Mission werden von einem Experten vorgegeben, daraus erstellt der Konfigurationsplaner unter Berücksichtigung der zur Verfügung stehenden Roboter und Ressourcen einen räumlich-zeitlichen Ablauf der Mission, und schließlich wird mit einem Handlungsplaner wie CHIMP ein vollständiger Plan erstellt, der von den Robotern ausgeführt werden kann. Für die Ausführung des Planes soll schließlich PLEXIL [170] eingesetzt werden.

Die Beispielmission in TransTerrA umfasst das Einrichten einer logistischen Kette und das Sammeln von Gesteinsproben im Amundsen Krater des Mondes [157]. Abbildung 6.9 zeigt den groben Ablauf einer solchen Mission. Ausgehend von der Landestation soll der Rover zunächst an Position `b1` eine Basisstation errichten und dort sowie an Punkt `b2` Gesteinsproben sammeln. An `b2`

trifft sich der Rover zudem mit dem Shuttle, um Container für Gesteinsproben sowie Batterien auszutauschen. Anschließend nimmt der Rover an der Landestation eine weitere Basisstation auf, um zunächst an `b3` und `b4` weitere Gesteinsproben zu sammeln und die Basisstation auf dem Weg zu `b6` zu platzieren. Hierbei ist zu beachten, dass sich sowohl `b4` als auch `b6` dauerhaft in Dunkelheit befinden, die Basisstation mit ihren Solarmodulen jedoch nicht in völliger Dunkelheit aufgestellt werden sollte. Nach dem Errichten der Basisstation erfolgt ein weiteres Treffen mit dem Shuttle zum Austausch von Containern und Batterien. Anschließend sammelt der Rover weitere Gesteinsproben an den Orten `b6` und `b7`.

Für CHIMP wurde eine Domänenbeschreibung erstellt, die im Anhang in Listing D.3 aufgeführt ist, und erfolgreich anhand der Beispielmission getestet. Die Roboteraktionen umfassen das Sammeln von Gesteinsproben, das Fahren zu einem bestimmten Gebiet, den Austausch von Containern, Batterien oder Nutzlastelementen zwischen zwei Robotern sowie das Aufnehmen oder Platzieren von Basisstationen. Das Sammeln von Gesteinsproben und das Manipulieren der Basisstationen ist nicht mit einem Shuttle, sondern nur mit einem Rover möglich. Die folgenden primitiven Roboteraktionen wurden in Form von Operatoren modelliert:

`!move_to(?robot ?area)` Mit diesem Operator fährt der Roboter zu einem gegeben Ziel.

`!sample_regolith(?robot ?area ?container)` Mit diesem Operator sammelt der Rover in einem Gebiet, in dem er sich bereits befindet, Gesteinsproben und befördert sie in den leeren Container.

`!transfer_sample(?robot1 ?robot2 ?container)` Der Roboter `?robot1` übergibt einen Container an den anderen Robtoter `?robot2`.

`!transfer_battery(?robot1 ?robot2 ?battery)` Analog zum vorherigen Operator wird hiermit eine Batterie übergeben.

`!transfer_payload(?robot1 ?robot2 ?payload)` Mit diesem Operator wird ein Nutzlastelement übergeben.

`!pickup_basecamp(?robot1 ?camp)` Mit diesem Operator greift der Roboter ein Base-Camp, das sich an einem Lander oder in einem bestimmten Gebiet befindet.

`!place_basecamp(?robot1 ?camp ?area)` Als Gegenstück zum vorherigen Operator platziert der Roboter hiermit ein Base-Camp in einem bestimmten Gebiet.

Etwas problematisch gestaltete sich die Modellierung der Methoden für den Austausch aller vollen oder leeren Container zwischen Rover und Shuttle, da sich die Anforderung, alle Container zu übergeben, in dem gemeinsamen Constraint-Netz nicht direkt repräsentieren lässt. Bei der rekursiven Definition der Methode für die Übergabe aller gefüllten Sample (siehe Zeilen 327–354 in Listing D.3) war damit die Modellierung der Methode für die Rekursionsverankerung nicht direkt mit Fluenten möglich. Stattdessen wurde auf die Verwendung von Ressourcen zurückgegriffen, indem für die Methode, die keine Unteraufgaben hat, sondern lediglich als Rekursionsverankerung prüft, ob der Roboter zu dem jeweiligen Zeitpunkt keine leeren Samples mehr hat, ein maximaler Verbrauch der Ressource `FilledSampleStorageCapacityRover` beziehungsweise `FilledSampleStorageCapacityShuttle` festgesetzt wurde. Somit wendet der

HTN-Meta-Constraint zunächst die Methode an, und anschließend wird diese Bedingung durch den Ressourcenscheduler überprüft. Falls es zeitgleich für den Roboter anderen Fluenten mit einem leeren Container gibt, wird die Anwendung des Operators mittels Backtracking revidiert.

Listing D.4 im Anhang zeigt eine Problembeschreibung der Beispielmission. Sie enthält mehrere Zielaufgaben wie das Errichten von Base-Camps, das Sammeln von Gesteinsproben an einer bestimmten Position oder einem Treffen zwischen Rover und Shuttle zum Austausch von Containern. Listing 6.4 zeigt die primitiven Aktionen eines resultierenden Plans, der in 0,6 Sekunden generiert wurde.

```
1  !move_to(rover1 b1)
2  !place_basecamp(rover1 baseCamp1 b1)
3  !sample_regolith(rover1 b1)
4  !move_to(rover1 b2)
5  !sample_regolith(rover1 b2)
6  !move_to(shuttle1 b2)
7  !transfer_sample(rover1 shuttle1 sampleContainer3)
8  !transfer_sample(rover1 shuttle1 sampleContainer2)
9  !transfer_sample(shuttle1 rover1 sampleContainer4)
10 !transfer_sample(shuttle1 rover1 sampleContainer5)
11 !transfer_sample(shuttle1 rover1 sampleContainer6)
12 !transfer_battery(shuttle1 rover1 batteryPayload1)
13 !transfer_battery(rover1 shuttle1 batteryPayload2)
14 !move_to(rover1 landingSite1)
15 !move_to(shuttle1 landingSite1)
16 !pickup_basecamp(rover1 baseCamp2)
17 !transfer_sample(shuttle1 lander1 sampleContainer2)
18 !move_to(rover1 b3)
19 !transfer_sample(shuttle1 lander1 sampleContainer3)
20 !sample_regolith(rover1 b3)
21 !move_to(rover1 b4)
22 !sample_regolith(rover1 b4)
23 !move_to(rover1 b6)
24 !place_basecamp(rover1 baseCamp2 b6)
25 !move_to(shuttle1 b6)
26 !transfer_sample(rover1 shuttle1 sampleContainer5)
27 !transfer_sample(rover1 shuttle1 sampleContainer6)
28 !move_to(shuttle1 landingSite1)
29 !sample_regolith(rover1 b6)
30 !transfer_sample(shuttle1 lander1 sampleContainer6)
31 !move_to(rover1 b7)
32 !transfer_sample(shuttle1 lander1 sampleContainer5)
33 !sample_regolith(rover1 b7)
```

Listing 6.4: Die Aktionen eines Plans für das Weltraumszenario, in dem der Rover Gesteinsproben sammeln sowie zwei Basisstation errichten muss und dabei von einem Shuttle unterstützt wird.

Eine endgültige Entscheidung über den Einsatz von CHIMP in TransTerrA sowie die vollständige Integration in das Gesamtsystem stehen noch aus. Eine Erweiterung für CHIMP, die in Zukunft weiter verfolgt werden soll, ist die Implementierung eines Schedulers für konsumierbare Ressourcen wie Energie. Hierfür könnte analog zu dem in [40] präsentierten Verfahren vorgegangen werden, welches eine Erweiterung des aktuell in CHIMP und dem Meta-CSP-Framework genutzten Scheduling wiederverwendbarer Ressourcen ist. Auf diese Weise ließe sich der Verlauf der Batterien der Roboter bei der Planung berücksichtigen und die Pläne gegebenenfalls anpassen. Zudem ließen sich die konkreten Positionen, an denen Basisstationen platziert werden, geometrisch repräsentieren und entsprechend optimieren. Bei letzterem handelt es sich jedoch um ein isoliertes Problem der optimalen Positionierung von Sensoren beziehungsweise Stationen, das vom eigentlichen Planungsproblem losgelöst ist beziehungsweise diesem vorangestellt werden kann, und wird daher hier nicht weiter berücksichtigt.

Kapitel 7

Zusammenfassung und Ausblick

Die vorliegende Arbeit hat sich mit planbasierter Robotersteuerung und der Frage, wie sich Planungsverfahren für die besonderen Anforderungen von Robotern einsetzen lassen, auseinander gesetzt. Zur Erfüllung des ersten Teilziels der Arbeit wurde zunächst die Integration eines Standard-HTN-Planers in ein Robotersystem, dessen Ziel das Lernen aus Erfahrungen ist, demonstriert und beschrieben. Dabei wurde SHOP2 als Planer eingesetzt und die Pläne mit einem auf einem Zustandsautomaten basierenden Ansatz ausgeführt, sodass bereits zu einem relativ frühen Zeitpunkt in dem Projekt RACE ein lauffähiges Robotersystem demonstriert werden konnte. In RACE konnte damit gezeigt werden, dass HTN-Planung besondere Vorteile für die planbasierte Robotersteuerung bietet: Die Modellierung von Methoden erlaubt die Vorgabe eines sicheren Roboterverhaltens, sodass unerwünschte Aktionen besser vermieden werden können; durch die resultierende Dekompositionshierarchie hat der menschliche Benutzer zudem die Möglichkeit, die Pläne nachzuvollziehen und Fehler zu erkennen; die Planungsdauer ist im Vergleich zu der Gesamtausführungszeit gering; und wie in RACE können Methoden der Hierarchie schließlich auf Basis von Erfahrungen gelernt und damit das Gesamtsystem an spezielle Situationen angepasst werden.

Anhand dieses ersten Systems wurde zunächst versucht, einige Schwächen aufzuzeigen und zu umgehen, ohne den Planer selbst modifizieren zu müssen, um damit eine Übertragbarkeit auf andere Rahmenbedingungen, die beispielsweise den Einsatz eines anderen Planungssystems beinhalten, zu gewährleisten. Dafür wurden Wege aufgezeigt, wie die Robustheit des Systems durch die enge Verknüpfung des Planers mit einer konsistenzbasierten Ausführungsüberwachung sowie der als Imagination bezeichneten Simulation einzelner Teilaufgaben des Plans vor deren realer Ausführung erhöht werden kann.

Anschließend legte die Arbeit den Fokus auf die hybride Planung, das heißt die Verwendung unterschiedlicher Wissensrepräsentations- und Schlussfolgerungsmechanismen für die Planung, als wichtige Anforderung für planbasierte Robotersteuerung. Dabei wurde auf Vorarbeiten der Projektpartner von der Universität Örebro zurückgegriffen. Durch die Kombination des Meta-CSP-Ansatzes und der HTN-Planung wurde auch die Planung komplexer Roboteraufgaben ermöglicht. Der HTN-Dekompositionsalgorithmus wurde dafür als Meta-Constraint modelliert und

lässt sich dadurch direkt in dem allgemeinen Meta-CSP-Formalismus nutzen. Auf diese Weise werden die Vorteile beider Planungs- beziehungsweise Schlussfolgerungsverfahren kombiniert. Umgesetzt und demonstriert wurde dies in dem neuen Planer CHIMP, der mit Hilfe des noch in der Entwicklung befindlichen Meta-CSP-Frameworks implementiert wurde. Neben den genannten Vorteilen der HTN-Planung kann CHIMP dadurch neben der kausalen Planung auch temporales Wissen, symbolisches Wissen und Wissen über Ressourcen nutzen und durch den Meta-CSP-Ansatz um weitere Wissensformen erweitert werden, was in dem Restaurantszenario am Beispiel der Abschätzung der Fahrtdauer und in der Hafenarbeiter-Domäne mit dem Meta-Constraint für die Routenplanung demonstriert wurde. Zudem können die von CHIMP generierten Pläne parallel ausführbare Aktionen beinhalten und während der Ausführung auf dem Roboter erweitert werden.

Die genannten Eigenschaften von CHIMP setzen somit vielfältige Anforderungen um, die für einen Planer für Roboter relevant sind. Die in der Einleitung genannten Ziele dieser Arbeit wurden somit umgesetzt. Dennoch hat sich darüber hinaus eine Vielzahl von Erweiterungsmöglichkeiten ergeben, die in zukünftigen Arbeiten untersucht und implementiert werden können:

Engere Verknüpfung zwischen Planung und Ausführung: Wie bereits angesprochen ist die Planausführung und deren Überwachung ein bedeutender Teil der Robotersteuerung. In Roboterumgebungen gibt es eine Vielzahl von Ursachen für das Fehlschlagen der Ausführung eines Plans, die sich grob unter den Oberbegriffen inkorrekte Umgebungsinformationen sowie fehleranfällige primitive Roboteraktionen zusammenfassen lassen. Aufgrund der Häufigkeit von Ausführungsfehlern können sehr umfangreiche Pläne selten ohne jegliche Anpassungen ausgeführt werden. Daher soll der Fokus für die zukünftige Entwicklung von CHIMP stärker auf einer engeren Integration zwischen der Planung selbst und der Planausführung inklusive Ausführungsüberwachung liegen. Für SHOP2 wurden in Abschnitt 3.4 verschiedene Möglichkeiten der Verbindung zwischen Planung und Planausführung beschrieben, die sich auch für CHIMP einsetzen lassen. Hierzu gehören die konsistenzbasierte Ausführungsüberwachung, die Imagination und die Abstraktion von konkreten Objektinstanzen (Lifting). Für CHIMP bietet sich zudem die Möglichkeit an, diese Ansätze im Vergleich zu SHOP2 noch enger mit CHIMP zu verknüpfen. Beispielsweise basiert die konsistenzbasierte Ausführungsüberwachung auf einem gemeinsamen Constraint-Netz, das dem von CHIMP ähnlich ist. Auch das Lifting ließe sich wie zuvor für SHOP2 beschrieben umsetzen, mit CHIMP könnte die konkrete Wahl eines Objekts und die konkrete Bindung der symbolischen Variable jedoch eventuell auch so lange wie möglich offen gelassen werden. So könnte ein erster Plan Variablen enthalten, die nicht vollständig instanziiert sind und erst später während der Ausführung gebunden werden. Eine Schwierigkeit, die es in diesem Fall bei der konkreten Implementierung noch zu lösen gilt, ist allerdings die Umsetzung der entsprechenden Vorbedingungen im Constraint-Netz, ohne dadurch gleichzeitig auch jene Variablenbindung zu erzwingen. Ebenso wäre das spätere Planen bestimmter High-Level-Aufgaben wie es von Kaelbling und Lozano-Perez für die Verbindung von Handlungs- und Bewegungsplanung umgesetzt wird [86], eine vielversprechende Erweiterung in diese Richtung. Darüber hinaus könnte auch versucht werden, bei Ausführungsfehlern Teile des Plans unter Verwendung der bestehenden Dekompositionshierarchie zu reparieren, anstatt vollständig neu zu planen.

Umgang mit Unsicherheit: Durch die Verwendung von flexiblen temporalen Intervallen können die Pläne von CHIMP in einem gewissen Maße mit Unsicherheiten über die Dauer von Aktionen oder den Zeitpunkten, wann Vor- und Nachbedingungen erfüllt sein müssen, umgehen. Darüber hinaus gibt es weitere Formen von Unsicherheiten über die Umgebung, die derzeit nicht explizit repräsentiert werden können. Bei diesen handelt es sich vor allem um Informationen, die unter anderem durch die Dynamik der Umgebung oder Ungenauigkeiten in der Wahrnehmung insbesondere der Objekterkennung bedingt sind. In zukünftigen Arbeiten könnte daher untersucht werden, wie jene Unsicherheiten explizit in CHIMP modelliert und bei der Planung verwendet werden können.

Verwendung weiterer Konzepte der POCL-Planung: CHIMP verwendet derzeit für die HTN-Planung die Grundidee des Planungsalgorithmus von SHOP2, die Aufgaben in derselben Reihenfolge zu planen, in der sie später ausgeführt werden. Es hat sich gezeigt, dass diese Strategie beim Vorhandensein mehrerer voneinander unabhängigen Aufgaben ungünstig ist, wenn die Planungsreihenfolge nicht durch einen `ordering`-Constraint angegeben wurde. Daher bietet sich für die Zukunft eine Erweiterung von CHIMP an, die Konzepte zur Verknüpfung von HTN-Planung und der POCL-Planung, die in [149] beschrieben werden, aufzugreifen und zu integrieren. Während CHIMP nur mittels Dekomposition sowie der Anwendung von Operatoren auf primitive Aufgaben plant, könnten so zusätzliche Zielzustände allein mit Operatoren geplant werden. Zudem sollen auch die Interaktionen zwischen Aufgaben stärker in der Planung betrachtet werden, um so den großen Suchraum für mehrere voneinander unabhängige Aufgaben einzuschränken und ungültige Pfade auszuschließen.

Optimierung der Pläne: CHIMP terminiert derzeit, sobald ein Plan gefunden wurde, der den Anforderungen, die in Form von Meta-Constraints definiert werden, genügt und das temporale sowie symbolische Constraint-Netz konsistent sind. Ein weiter Punkt, der in Zukunft genauer betrachtet werden sollte, ist die Optimierung der Pläne.

Integration oder Anbindung semantischer Karten: Das RACE-Blackboard, welches vorrangig als Wissensspeicher diente, war ein Kompromiss, der die Anforderungen aller Softwarekomponenten, die in RACE beteiligt waren, berücksichtigte. Die Art der Daten, die darin gespeichert wurden, und die Wissensrepräsentationsform war primär auf die Anforderungen der lernenden Komponenten ausgerichtet. Bereits bei der relativ geringen Anzahl an Objekten und Umgebungsinformationen, die in den Demonstrationsszenarien von RACE im Blackboard gespeichert wurden, waren Anfragen an das Blackboard zum Teil bereits vergleichsweise laufzeitintensiv. So dauerten die Anfragen für die Erstellung des initialen Zustands für SHOP2 häufig länger als die Planung selbst.

Als Teil der stärkeren Integration zwischen der Planung mit CHIMP und der Planausführung soll in Zukunft auch eine semantische Karte eng mit CHIMP verknüpft werden. Hierfür bietet sich das Framework SEMAP [38] an, das die Entwicklung und Verwendung semantischer Karten ermöglicht. Es verbindet große geometrische Karten mit semantischem Wissen, indem es intern die geometrische Datenbank PostGIS verwendet, um Daten zu speichern und Anfragen über die geometrischen Relationen von Objekten zu erlauben.

Anhang A

Domänen- und Problembeschreibungsformat von CHIMP

Für CHIMP wurden Domänen- und Problembeschreibungssprachen entwickelt, die an dessen Eigenschaften und Repräsentationsmechanismen orientiert sind. Diese werden in den folgenden beiden Abschnitten erläutert.

A.1 Format der Domänenbeschreibung

Eine Domänenbeschreibung für CHIMP umfasst die Definition von *Methoden*, *Operatoren*, *Zustandsvariablen* und *Ressourcen* sowie eine Auflistung der Prädikatennamen. Im Folgenden wird die verwendete Syntax beschrieben.

Alle Fluenten haben eine feste maximale Anzahl symbolischer Variablen. Diese Anazhl muss einmal mit dem Schlüsselwort `MaxArgs`, gefolgt von einem ganzzahigen Wert max, festgelegt werden:

(MaxArgs max)

Wiederverwendbare Ressourcen werden mit dem Schlüsselwort `Resource`, gefolgt von dem Namen n der Ressource und ihrer Kapazität k, angegeben:

(Resource n k)

Die Definition von Zustandsvariablen können zusätzliche Parameter enthalten. Da die Domänenbeschreibung nicht ausschließlich auf die Verwendung von Zustandsvariablen ausgelegt ist, sondern auf Prädikaten beruht, muss spezifiziert werden, welche Anteile des Prädikates die Zustandsvariable und welche deren Wert angeben. Zum Beispiel können die Positionen der Arme

```
1 (MaxArgs 5)
2 (Resource objectManipulationCapacity 1)
3 (Resource armManipulationCapacity 1)
4 (Resource navigationCapacity 1)
5 (StateVariable Holding 1 leftArm1 rightArm1)
6 (StateVariable HasArmPosture 1 leftArm1 rightArm1)
```

Listing A.1: Anfang einer Domänenbeschreibung. Die maximale Parameteranzahl der Fluenten, drei Ressourcen und zwei Zustandsvariablen werden angegeben.

des Roboters mit Zustandsvariablen angegeben werden, da sich ein Arm immer nur in einer Position befinden kann. Zur Definition der Zustandsvariablen müssen die Namen der beiden Arme ebenfalls in der Definition der Zustandsvariablen angegeben werden. Somit können nach dem Schlüsselwort `StateVariable` nicht nur der Name n des Prädikats, sondern auch Indizes p_i gefolgt von möglichen Werten v_i spezifiziert werden:

$$(StateVariable\ n\ p_1\ v_1\ \ldots\ p_k\ v_k)$$

Listing A.1 zeigt ein Beispiel für den ersten Teil einer CHIMP-Domänenbeschreibung. Die Maximale Parameteranzahl der Fluenten wird auf den Wert 5 gesetzt, und es werden die drei Ressourcen `objectManipulationCapacity`, `armManipulationCapacity` sowie `navigationCapacity` jeweils mit einer Kapazität 1 definiert. Zudem werden zwei Zustandsvariablen definiert: Eine Zustandsvariable für Fluenten vom Typ `Holding` mit dem Wert `leftArm1` als erstem Parameter, und eine weitere für `rightArm1`. Der zweite Parameter von `Holding` wird nicht aufgeführt – dies ist der Wert der Zustandsvariablen.

Die Definitionen der Operatoren und Methoden umfassen alle Elemente, die in Abschnitt 5.1 eingeführt wurden. Formal gesehen bestehen Operatoren und Methoden aus einem Prototypen für Aufgabenfluenten, auf welche sie angewendet werden können, und einem Constraint-Netz, das die Vorbedingungen und Effekte oder Dekompositionen repräsentiert. In CHIMPs Domänenbeschreibungsformat sind Operatoren folgendermaßen aufgebaut:

$$(:\text{operator}\ [w]\ h\ P_1 \ldots P_j\ N_1 \ldots N_k\ E_1 \ldots E_l\ V_1 \ldots V_m\ C_1 \ldots C_n\ u_1 \ldots u_o)$$

Dabei ist

- $w \in \mathbb{Z}$ ein optionaler Wichtungsfaktor,
- h der Kopf des Operators,
- P_i ein Vorbedingungsausdruck,
- N_i ein Ausdruck für einen negativen Effekt,
- E_i ein Ausdruck für einen positiven Effekt,
- V_i eine Restriktion einer der symbolischen Variablen,

- C_i ein temporaler, symbolischer Constraint,
- u_i ein Ressourcenverbrauch.

Der Wichtungsfaktor kann in Heuristiken verwendet werden, um die Meta-Werte nach der Wichtung der verwendeten Operatoren oder Methoden zu priorisieren. Wenn kein Wichtungsfaktor angegeben ist, wird standardmäßig der Wert 1 verwendet.

Die primitive Aufgabe, auf die der Operator angewendet werden kann, muss im Kopf des Operators spezifiziert werden. Dieser besteht aus dem Schlüsselwort `Head` und einem Schema-Fluenten f.

$$(\text{Head } f)$$

Der Schema-Fluent besteht aus einem Prädikatnamen und einer Liste von Bezeichnern für symbolische Variablen, welche mit einem Fragezeichen beginnen.

Vorbedingungen haben folgende Form:

$$(\text{Pre } id\ f)$$

Dabei ist id ein Bezeichner für die Vorbedingung und f ist ein Schema für einen Fluenten, der mit dem Aufgabenfluenten verbunden wird und auf diese Weise eine Vorbedingung repräsentiert.

Die positiven Effekte der Operatoren werden auf ähnliche Weise wie Vorbedingungen definiert und mit dem Schlüsselwort `Add` eingeleitet:

$$(\text{Add } id\ f)$$

Negative Effekte müssen bereits als Vorbedingungen aufgeführt sein. Daher folgt nach dem Schlüsselwort `Del` nur der Bezeichner der dazugehörigen Vorbedingung:

$$(\text{Del } id)$$

Die Werte und Typen der symbolischen Variablen eines Fluenten können eingeschränkt werden, oder es kann angegeben werden, dass die Werte zweier symbolischer Variablen gleich beziehungsweise unterschiedlich sind. Für Werterestriktionen werden die Schlüsselwörter `Values` und `NotValues` gefolgt von einer Liste von Konstanten verwendet:

$$(\text{Values} \mid \text{NotValues } var\ n_1 \dots n_n)$$

Diese setzen die Domäne der jeweiligen symbolischen Variable var auf die Konstanten $n_1 \dots n_n$ beziehungsweise schließen diese aus. Auf ähnliche Weise kann der Typ t festgelegt werden:

$$(\text{Type} \mid \text{NotType } var\ t)$$

Welche Instanzen zu einem Objekttyp gehören wird dabei in der Problembeschreibung festgelegt. Mit `VarDifferent` kann zudem für zwei symbolische Variablen festgelegt werden, dass ihre Werte

unterschiedlich sind. Intern repräsentiert CHIMP dies mit einem symbolischen $\neq$-Constraint zwischen den beiden Variablen.

(VarDifferent var_1 var_2)

Temporale Constraints können folgendermaßen angeben werden:

(Constraint aic)

aic ist dabei eine Relation aus Allens Intervallalgebra mit optionalen metrischen Grenzen oder die Angabe einer Zeitdauer. Die Angabe der zugehörigen Fluenten geschieht mittels der Bezeichner der Vorbedingungen, Effekte oder Teilaufgaben oder den Bezeichner `task` für den Aufgabenfluenten selbst. Wenn zwischen einer Vorbindung und der Aufgabe keine temporale Relation explizit angegeben ist, verwendet CHIMP automatisch die relativ allgemeine Relation $\{\mathrm{m}, \mathrm{o}, \mathrm{f}^{-1}, \mathrm{d}^{-1}\}$. Ebenso setzt er zwischen Teilaufgaben und der Aufgabe die Relation $\{\mathrm{d}, \equiv, \mathrm{s}, \mathrm{f}\}$, zwischen die Aufgabe und einem positiven Effekt $\{\mathrm{o}\}$ und zu einem negativen Effekt $\{\mathrm{o}^{-1}\}$.

Schließlich kann auch für den Aufgabenfluent ein Ressourcenverbrauch definiert werden:

(ResourceUsage r l)

Dabei ist r der Name der Ressource und l ist ein ganzzahliger Wert für den Ressourcenverbrauch.

Listing A.2 zeigt ein Beispiel für einen CHIMP-Operator, der auf Aufgabenfluenten vom Typ `!pick_up_object` angewendet werden kann. Er hat einen Parameter für das Objekt und einen weiteren für den Arm, mit welchem der Roboter das Objekt greifen soll. Die Vorbedingungen sind in den Zeilen 3 bis 6 aufgeführt. Zunächst wird die Position des Objekts benötigt (Zeile 3). In den Zeilen 4 und 5 wird gefordert, dass sich der Roboter in einer `ManipulationArea` befindet, von der aus das Gebiet, in dem das Objekt positioniert ist, erreichbar ist. Des Weiteren darf der Roboter mit jenem Arm nicht schon etwas Anderes halten (Zeilen 6, 7). Die `p1` und `p4` sind zugleich negative Effekte (Zeilen 8, 9). Als einzigen positiven Effekt hält der Roboter das Objekt anschließend mit dem jeweiligen Arm (Zeile 10). Mit den temporalen Constraints wird genau spezifiziert, wie sich die Fluenten und die Aufgabe temporal zueinander verhalten. Für die negativen Effekte werden hier ein `overlaps`-Relationen verwendet (Zeilen 11, 14), wodurch diese Fluenten während der Ausführung der zugehörigen Aktion aufhören wahr zu sein. Zudem sind beide Vorbedingungen durch eine `meets`-Relation mit dem Effekt, dass der Roboter das Objekt hält, verbunden. Diese Fluenten sind somit nicht mehr erfüllt, sobald der Roboter das Objekt mit seinem Arm hält. Für die Ausführung kommt es hierbei auf die Implementierung der Komponenten an, die diese Fluenten erzeugen und beenden. Wenn dies für die jeweiligen Fluenten von unterschiedlichen Komponenten durchgeführt wird, führt diese Relation zu einer Überspezifikation, da die Start- und Endzeitpunkte sich so leicht um einige Millisekunden unterscheiden können. Bei der Ausführungsüberwachung könnte dies zu einem ungewollt festgestellten Fehlverhalten führen. Die anderen beiden Vorbedingungen müssen hingegen während der gesamten Ausführung der Aktion erfüllt sein. Zeile 17 gibt schließlich für die Ausführung der Roboteraktion eine Mindestdauer von 4 Sekunden an, währenddessen eine Einheit der Ressource `objManCapacity` beansprucht wird (Zeile 18).

```
1  (:operator 1
2    (Head !pick_up_object(?obj ?arm))
3    (Pre p1 On(?obj ?objArea))
4    (Pre p2 RobotAt(?manArea))
5    (Pre p3 Connected(?objArea ?manArea ?preManArea))
6    (Pre p4 Holding(?arm ?nothing))
7    (Values ?nothing nothing)
8    (Del p1)
9    (Del p4)
10   (Add e1 Holding(?arm ?obj))
11   (Constraint OverlappedBy(task,p1))
12   (Constraint During(task,p2))
13   (Constraint During(task,p3))
14   (Constraint OverlappedBy(task,p4))
15   (Constraint Meets(p1,e1))
16   (Constraint Meets(p4,e1))
17   (Constraint Duration[4000,INF](task))
18   (ResourceUsage objManCapacity 1)
19 )
```

Listing A.2: Beispiel eines CHIMP-Operators für das Greifen eines Objekts.

Methoden werden auf ähnliche Weise definiert. Sie sind Ausdrücke der Form:

$$(:\text{method } [w]\ h\ P_1 \ldots P_i\ S_1 \ldots S_j\ V_1 \ldots V_k\ C_1 \ldots C_l\ O_1 \ldots O_m)$$

Dabei ist

- $w \in \mathbb{Z}$ ein optionaler Wichtungsfaktor,
- h der Methodenkopf,
- P_i ein Vorbedingungsausdruck,
- S_i ein Ausdruck für eine Teilaufgabe,
- V_i eine Restriktion einer der symbolischen Variablen,
- C_i ein temporaler oder symbolischer Constraint,
- O_i eine Ordnungsrelation.

Der optionale Wichtungsfaktor, der Methodenkopf, die Vorbedingungen, die symbolischen Restriktionen und die temporalen und symbolischen Constraints werden auf gleiche Weise wie bei Operatoren definiert. Teilaufgaben werden mit dem Schlüsselwort `Sub` gefolgt von einem Bezeichner und einem Prototypen für einen Aufgabenfluenten angegeben:

$$(\text{Sub } id\ f)$$

```
1  (:method 1
2    (Head get_object(?object))
3    (Pre p1 RobotAt(?robotArea))
4    (Pre p2 Connected(?objectArea ?manArea ?preManArea))
5    (Pre p3 On(?object ?objectArea))
6    (VarDifferent ?robotArea ?prArea)
7    (Sub s1 drive_robot(?prArea))
8    (Sub s2 assume_manipulation_pose(?mArea))
9    (Sub s3 !pick_up_object(?object ?arm))
10   (Constraint Before(s1,s2))
11   (Constraint Before(s2,s3))
12   (Constraint Starts(s1,task))
13   (Constraint Finishes(s3,task))
14   (Ordering s1 s2)
15   (Ordering s2 s3)
16 )
```

Listing A.3: Beispiel einer Methode für das Holen eines Objekts.

Die Ordnungsconstraints, die lediglich für den Planer, jedoch nicht für die Planausführung relevant sind, werden mit dem Schlüsselwort `Ordering` und zwei Bezeichnern für die beiden Teilaufgaben angegeben:

$$(\text{Ordering } id_1\ id_2)$$

Listing A.3 zeigt ein Beispiel einer Methode für das Holen eines Objekts. Mit den Vorbedingungen wird gefordert, dass sich der Roboter in einem Bereich befindet, der nicht nahe dem Objekt ist (Zeilen 2 bis 6). Die Methode hat drei Teilaufgaben (Zeilen 7 bis 9). Zunächst soll der Roboter zu dem Objekt fahren, sich in eine für die Objektmanipulation geeignete Position bringen und schließlich das Objekt greifen. Diese drei Teilaufgaben sollen alle nacheinander ausgeführt (Zeilen 10 bis 13) und auch in dieser Reihenfolge geplant werden (Zeilen 14, 15).

Neben den Roboteraktionen können auch andere Fluenten Ressourcen verwenden:

$$(\text{FluentResourceUsage (Usage } r\ l)\ (\text{Fluent } t)\ (\text{Param } i\ c))$$

Dabei ist r der Name der Ressource, l ist die beanspruchte Kapazität, f ist der Prädikatenname des Fluenten, i ist der Index eines Parameter und c ist ein symbolischer Wert für jenen Parameter. Listing A.4 zeigt hierfür ein Beispiel, in dem das Halten eines Objekts mit dem linken Arm eine Einheit der Ressource `leftArm1Ressource` beansprucht.

A.2 Format der Problembeschreibung

Zusätzlich zur Domänenbeschreibung kann CHIMPs Problembeschreibung in einer separaten Datei spezifiziert werden. Eine Problembeschreibung wird stets mit dem Schlüsselwort `Problem`

```
1 (FluentResourceUsage
2   (Usage leftArm1Resource 1)
3   (Fluent Holding)
4   (Param 2 leftArm1)
5 )
```

Listing A.4: Beispiel für die Ressourcenverwendung eines Fluenten vom Typ `Holding`.

eingeleitet. Sie umfasst eine Auflistung aller zur Verfügung stehenden Symbole und Objektinstanzen sowie eine Spezifikation des Ausgangszustands und Planungsproblems.

Nach dem Schlüsselwort `ArgumentSymbols` werden alle als Argumente der Prädikate verfügbaren Symbole aufgelistet. Zudem können zu jedem Objekttyp t alle zugehörigen Instanzen i_i angeben werden:

$$(\text{Instances } t\ i_1 \dots\ i_n)$$

Die Aufgabenfluenten werden mit dem Schlüsselwort `Fluent`, einem Bezeichner und einer Grundinstanz des Prädikats angegeben:

$$(\text{Fluent } id\ f)$$

Ebenso wird für Aufgabenfluenten das Schlüsselwort `Task` verwendet:

$$(\text{Task } id\ f)$$

Temporale Constraints können auf gleiche Weise wie in der Domänenbeschreibung eingesetzt werden.

Listing A.5 zeigt einen kurzen Ausschnitt aus einer beispielhaften Problembeschreibung für das Transportieren eines Kaffees mit einem PR2. Die Zeilen 2 bis 7 geben die initialen Konfigurationen der Arme und des Torsos an. Dabei wird auch die Startzeit dieser Fluenten mit temporalen Constraints festgesetzt. Des Weiteren befinden sich eine Kaffeekanne und ein Milchkännchen auf dem Tresen (Zeilen 8 bis 11). Zielaufgabe des Planungsproblems ist das Transportieren der Kaffeekanne `coffeeJug1` auf das Gebiet `placingAreaLeftTable2`. Diese Aufgabe muss nach spätestens 600 Sekunden fertiggestellt sein.

```
1  (Problem
2   (Fluent f1 HasArmPosture(leftArm1 ArmTuckedPosture))
3   (Constraint Release[0,0](f1))
4   (Fluent f2 HasArmPosture(rightArm1 ArmTuckedPosture))
5   (Constraint Release[0,0](f2))
6   (Fluent f3 HasTorsoPosture(TorsoDownPosture))
7   (Constraint Release[0,0](f3))
8   (Fluent f8 On(coffeeJug1 placingAreaEastRightCounter1))
9   (Constraint Release[0,0](f8))
10  (Fluent f9 On(milkPot1 placingAreaEastRightCounter1))
11  (Constraint Release[0,0](f9))
12  (Task t0 move_object(coffeeJug1 placingAreaNorthLeftTable2))
13  (Constraint Deadline[0,600000](t0))
14 )
```

Listing A.5: Auszug aus einer Problembeschreibung.

Anhang B

Syntax von SHOP2

In diesem Anhang wird kurz auf die spezielle Syntax, des in Kapitel 3 verwendeten Planers SHOP2 [117] eingegangen.

Ein SHOP2-Operator kann mit folgender Syntax definiert werden [67]:

$$(: \text{operator } h\ P\ D\ A\ [c])$$

Der Kopf h besteht aus einem Symbol einer primitiven Aufgabe, auf die der Operator angewendet werden kann, sowie einer Parameterliste. In SHOP2 beginnen alle Symbole primitiver Aufgaben per Konvention mit einem Ausrufezeichen und Variablennamen mit einem Fragezeichen. Es folgt ein logischer Ausdruck P, der die Vorbedingungen repräsentiert. Um den Operator anwenden zu können, muss dieser Ausdruck mit dem Wert `wahr` evaluiert werden. Ein *logischer Ausdruck* kann auf verschiedene Weise formuliert werden: Er kann ein logisches Atom oder eine Konjunktion, Disjunktion, Implikation oder universelle Quantifikation logischer Ausdrücke sein. Zudem kann er auch die Zuweisung einer in der Programmiersprache Lisp formulierten Ausdrucks an eine Variable oder eine boolesche Lisp Funktion sein. Dies führt zu einer großen Ausdrucksstärke in den Formulierungen der Vorbedingungen. Ein *logisches Atom* besteht wiederum aus einem Prädikatensymbol sowie einer Liste von Termen. Bei letzteren handelt es sich um Variablensymbole, Konstanten oder Zahlen. D und A sind Listen, die die negativen und positiven Effekte des Operators repräsentieren. Zumeist bestehen sie aus logischen Atomen, welche durch Anwendung des Operators aus dem Weltzustand entfernt oder zu ihm hinzugefügt werden. Sie können aber auch Sicherungskonditionen beinhalten, die das Entfernen eines logischen Atoms verhindern, bis die Sicherungskondition widerrufen wurde. Als dritte Möglichkeit können auch universelle Quantifizierungen für das Hinzufügen oder Entfernen mehrerer logischer Operatoren, die eine bestimmte Bedingung erfüllen, verwendet werden. Als letzten Parameter kann die Operatordefintion schließlich einen optionalen Lisp-Ausdruck für den Kostenwert des Operators enthalten. Andernfalls wird standardmäßig der Kostenwert 1 verwendet. Kommentare beginnen immer mit einem Semikolon.

Listing B.1 zeigt einen SHOP2-Operator für das Verschränken der Arme eines zweiarmigen Roboters. Der Operator kann auf primitive Aufgaben, die das Aufgabensymbol `!tuck_arms`

```
(:operator (!tuck_arms ?left_arm_goal ?right_arm_goal)
 (AND ; Vorbedingungen:
  (eval (member '?left_arm_goal
                '(ArmTuckedPosture ArmUnTuckedPosture)))
  (eval (member '?right_arm_goal
                '(ArmTuckedPosture ArmUnTuckedPosture)))
  (ArmPosture leftArm1 ?left_posture)
  (ArmPosture rightArm1 ?right_posture)
 )
 ( ; Liste negativer Effekte:
  (ArmPosture leftArm1 ?left_posture)
  (ArmPosture rightArm1 ?right_posture)
 )
 ( ; Liste positiver Effekte:
  (ArmPosture leftArm1 ?left_arm_goal)
  (ArmPosture rightArm1 ?right_arm_goal)
 )
)
```

Listing B.1: Beispiel eines SHOP2-Operators für das Verschränken der Arme.

besitzen, angewendet werden. Als Parameter hat er zwei Variablen für die Zielposen des linken und rechten Armes, als Vorbedingung wird eine Konjunktion von vier logischen Ausdrücken angeben. Die ersten beiden Ausdrücke stellen mit Hilfe von Lisps `member`-Funktion sicher, dass es sich bei den Parametern um gültige Zielposen für die dazugehörige Roboteraktion handelt, indem sie jeweils den Wert `ArmTuckedPosture` oder `ArmUnTuckedPosture` besitzen. Zusätzlich werden die aktuellen Armposen mit den folgenden beiden logischen Ausdrücken aus dem internen Zustand des Planers abgefragt. Die Armposen werden benötigt, um sie zur Liste der negativen Effekte hinzuzufügen und somit zu entfernen. Die neuen Armposen werden schließlich durch die Auflistung in den positiven Effekten zum internen Zustand des Planers nach Anwendung des Operators hinzugefügt.

Ähnlich zu den Operatoren können die SHOP2-Methoden definiert werden. Ihre Syntax hat die folgende Form [67]:

$$(:\text{method}\ h\ [n_1]\ C_1\ T_1\ [n_2]\ C_2\ T_2 \ldots [n_k]\ C_k\ T_k)$$

Wie die Operatoren verfügen auch Methoden über einen Kopf h, der nun ein Symbol einer zusammengesetzten Aufgabe enthält. Bei den C_i handelt es sich um *logische Vorbedingungen*. Auch diese können logische Ausdrücke sein. Zusätzlich können mit ihnen die möglichen Variablenbindungen der Vorbedingungen sortiert werden oder SHOP2 dazu zwingen, ausschließlich die erste Möglichkeit der Variablenbindung zu verwenden, die den logischen Ausdruck erfüllt. Auf diese Weise können dem Planer somit weitere Heuristiken gegeben oder der Suchraum kann eingeschränkt werden. Bei den T_i handelt es sich jeweils um Aufgabenlisten für die Teilaufgaben. Aufgabenlisten sind rekursiv als Aufgaben oder Listen von Aufgabenlisten definiert.

Standardmäßig sind sie geordnet, können mit dem Schlüsselwort `:unordered` jedoch auch zu ungeordneten Mengen werden. Dadurch können Kombinationen von geordneten und ungeordneten Teilaufgaben verwendet werden. Weiterhin ist die Reihenfolge der unterschiedlichen Listen von Vorbedingungen und Aufgaben von Bedeutung. Sie entspricht einer aus Programmiersprachen bekannten if-else-Verzweigung. Wenn SHOP2 versucht, eine Methode anzuwenden, versucht er zunächst, die Vorbedingungen in C_1 zu erfüllen. Ist dies möglich, wird die erste Liste von Teilaufgaben T_1 angewendet. Nur wenn C_1 nicht erfüllt werden kann, werden die Vorbedingungen in C_2 geprüft. Dies ist auch beim Backtracking der Fall. Wenn mehrere Arten existieren, die Vorbedingungen in C_i zu erfüllen, kommt der Planer während des Backtrackings zurück zu diesem Entscheidungspunkt und versucht, mit den Alternativen weiter zu planen. Die Vorbedingungen in C_{i+1} werden jedoch nicht mehr überprüft. Wenn hingegen mehrere Methoden für ein Aufgabensymbol existieren, werden auch alle vom Planer berücksichtigt. Zur Erleichterung der Fehlersuche können die Paare von Vorbedingungslisten und Aufgabenlisten noch mit einem optionalen Namen n_i versehen werden.

Listing B.2 zeigt ein Beispiel für eine SHOP2-Methode. Sie kann für die Aufgabe, ein gegebenes Objekt mit einem bestimmten Arm zu holen, angewendet werden. Die Methode besteht aus zwei alternativen Paaren von Vorbedingungs- und Aufgabenlisten. Die ersten Vorbedingungen fragen die aktuelle Position des Objekts ab und prüfen, ob sich der Roboter bereits nahe dieser Position befindet. Ist dies der Fall, dekomponiert die Methode die übergeordnete Aufgabe in eine einzige Teilaufgabe zum direkten Greifen des Objekts. Andernfalls wird der zweite Vorbedingungsausdruck überprüft. Dieser fragt wiederum die Positionen des Objekts und des Roboters ab. Die zweite Aufgabenliste besteht aus zwei Aufgaben, die geordnet sind: Der Roboter muss zunächst zu dem Objekt fahren und es anschließend greifen.

```
1  (:method (get_object_w_arm ?object ?arm)
2   ( ; Precondition:  Der Roboter befindet sich nahe dem Objekt.
3    (Instance On ?on)
4    (HasPhysicalEntity ?on ?object)
5    (HasArea ?on ?fromPlacingArea)
6    (HasManipulationArea ?fromPlacingArea ?fromManArea)
7    (HasPreManipulationArea ?fromManArea ?fromPreArea)
8    (Instance RobotAt ?robotAt)
9    (HasRobot ?robotAt trixi1)
10   (HasArea ?robotAt ?fromPreArea)
11  )
12  ( ; Teilaufgaben
13    (grasp_object_w_arm ?object ?arm)
14  )
15  ( ; Vorbedingungen: Der Roboter ist noch weit vom Objekt entfernt.
16   (Instance On ?on)
17   (HasPhysicalEntity ?on ?object)
18   (HasArea ?on ?fromPlacingArea)
19   (HasManipulationArea ?fromPlacingArea ?fromManArea)
20   (HasPreManipulationArea ?fromManArea ?fromPreArea)
```

```
21    (Instance RobotAt ?robotAt)
22    (HasRobot ?robotAt trixi1)
23    (HasArea ?robotAt ?robotArea)
24    (different ?robotArea ?fromPreArea)
25   )
26   ( ; Teilaufgaben
27    (drive_robot ?fromPreArea)
28    (grasp_object_w_arm ?object ?arm)
29   )
30 )
```

Listing B.2: Beispiel einer SHOP2-Methode zum holen eines Objekts mit einem bestimmten Arm.

Logische Atome, die Elemente logischer Ausdrücke sein können, müssen nicht explizit in den internen Weltzustand des Planers eingefügt werden, sondern es können dafür alternativ auch *Axiome* definiert werden. In SHOP2 sind Axiome generalisierte Versionen von Horn-Klauseln [117]. Sie bestehen aus einem Kopf und einer Liste logischer Ausdrücke. Der Kopf kann als logisches Atom verwendet werden und wird als `wahr` ausgewertet, wenn einer der logischen Ausdrücke in dem aktuellen internen Zustand des Planers erfüllbar ist. In der Methode aus Listing B.2 wird in Zeile 24 das Axiom `different` verwendet. Listing B.3 zeigt die Definition dieses Axioms und seines Gegenstücks `same`, welches zuerst definiert wird. Dieses hat den Wert `wahr`, wenn seine beiden Parameter identisch sind. Die Liste logischer Ausdrücke dieses Axioms ist leer. `same` wird wiederum in dem Axiom `different` verwendet, welches `wahr` ist, wenn `same` für seine Parameter nicht erfüllbar ist. Die Negation mit `not` wird als *Negation as Failure* behandelt. Das heißt, `not(a)` ist erfüllt, wenn `a` nicht bewiesen werden kann [117].

```
1 (:- (same ?x ?x) ())
2
3 (:- (different ?x ?y) ((not (same ?x ?y))))
```

Listing B.3: Beispiel zweier SHOP2-Axiome.

Neben der Domänenbeschreibung, die eher selten modifiziert wird, benötigt SHOP2 eine Beschreibung des eigentlichen Planungsproblems. Sie hat die folgende Syntax [67]:

$$(\text{defproblem } \textit{problem-name } \textit{domain-name } (a_1\ a_2\ \ldots\ a_n)\ T)$$

Auf die Symbole für den Namen des Problems und der Domäne folgen grundinstanzierte logische Atome a_i, die den initialen Zustand beschreiben, sowie eine Liste T der zu erfüllenden Aufgaben.

Anhang C

Weitere Daten

C.1 Ergebnisse der Laufzeitmessung der RACE-Domäne

Die nachfolgenden Tabellen enthalten die Ergebnisse der Laufzeitmessungen aus den Abschnitten 6.1.4 und 6.2.3. Diese sind in den Abbildungen 6.4 beziehungsweise 6.7 aufgetragen.

Tabelle C.1: Testergebnisse von CHIMP und FAPE (letzte Spalte) in der Hafenarbeiter-Domäne für verschiedene Probleminstanzen. Die Spalten geben den Index der Probleminstanz, die Laufzeit, jeweils die im Plan enthaltenen Operatoren, Methoden, Fluenten und Constraints sowie die ermittelten Meta-Variablen und angewendeten Meta-Variablen an.

Nr.	Zeit [s]	Op.	Meth.	Fluenten	Constr.	Meta-Var.	Meta-Wer.	FAPE Zeit[s]
1	0,91	24	18	166	521	39	43	3,13
2	0,81	24	18	174	529	39	43	4,88
3	0,26	8	6	110	254	15	15	2,75
4	6,63	44	28	296	1001	69	73	8,67
5	4,15	40	26	274	909	63	67	7,10
6	5,01	28	18	194	574	79	87	7,04
7	0,87	24	18	186	541	39	43	6,94
8	7,01	32	18	178	587	67	82	3,40
9	4,13	28	18	178	558	79	87	3,15
10	3,78	28	18	178	558	79	87	4,49
11	2,57	24	16	156	466	67	75	4,63
12	2,87	24	16	160	470	67	75	3,36
13	5,10	34	26	228	732	61	71	>600,00
14	12,92	44	34	270	917	83	96	>600,00
15	35,37	54	39	325	1156	103	119	>600,00
16	16,32	46	35	281	972	87	103	>600,00
17	56,55	60	42	358	1294	115	131	>600,00
18	51,87	62	46	356	1307	128	153	>600,00
19	82,21	70	50	400	1491	140	164	>600,00
20	101,79	74	52	422	1583	146	171	>600,00
21	116,28	72	54	398	1507	159	193	>600,00
22	142,61	76	56	420	1599	165	199	>600,00

Tabelle C.2: Testergebnisse von CHIMP in der RACE-Domäne für verschiedene Probleminstanzen, in denen unterschiedlich viele `move_base`-Aufgaben zu planen waren. Die Spalten geben den Index der Probleminstanz, die Anzahl der `move_base`-Aufgaben, jeweils die im Plan enthaltenen Operatoren, Methoden, Fluenten und Constraints sowie die ermittlten Meta-Variablen und angewendten Meta-Variablen an.

Nr.	Aufg.	Zeit [s]	Op.	Meth.	Fluenten	Constr.	Meta-Var.	Meta-Wer.
1	2	0,32	18	25	89	307	45	58
2	2	0,43	26	35	116	409	63	96
3	2	0,73	35	42	143	513	80	141
4	2	0,57	27	32	116	410	62	91
5	2	0,37	17	20	82	275	38	48
6	2	0,52	25	30	109	377	56	84
7	2	0,85	34	37	136	481	73	125
8	2	0,49	26	27	109	378	55	77
9	2	0,47	26	35	116	409	63	96
10	3	1,08	37	46	152	567	86	154
11	3	1,48	45	56	179	669	104	212
12	3	1,46	45	56	179	669	104	212
13	4	2,53	56	67	218	833	127	300
14	4	1,37	39	50	164	627	92	175
15	4	3,92	64	77	245	935	145	378
16	5	3,42	58	71	227	894	133	332
17	5	10,81	58	71	227	891	135	394
18	6	4,21	60	75	239	961	139	348
19	6	7,53	77	92	293	1167	174	515
20	6	>300,00						
21	6	5,04	60	75	242	964	139	328
22	6	6,20	76	87	286	1135	167	483
23	6	51,04	84	97	313	1236	187	681
24	8	7,95	81	100	314	1309	186	547
25	8	>300,00						
26	8	76,92	98	117	368	1514	223	849
27	8	68,90	97	112	361	1482	216	828
28	8	>300,00					–	
29	8	7,69	80	95	307	1277	179	543
30	8	9,17	97	112	361	1483	214	741
31	8	9,87	89	110	341	1411	204	645
32	10	14,20	101	120	382	1639	226	800
33	10	19,08	118	137	436	1845	261	971
34	10	189,25	126	147	463	1946	281	1169

Anhang D

Eingesetzte Domänen- und Problembeschreibungen

Im Folgenden werden die Domänenbeschreibungen der RACE-Domäne (Listing D.1), der DWR-Domäne (Listing D.2) und des Weltraumszenarios (Listing D.3) für CHIMP und eine beispielhafte Problembeschreibung des Weltraumszenarios (Listing D.4) für CHIMP aufgeführt. Auf diese wurde bereits im vorhergehenden Text verwiesen.

D.1 RACE-Domänenbeschreibung für CHIMP

```
1  (HybridHTNDomain RACEDomain)
2  (MaxArgs 5)
3
4  (PredicateSymbols On RobotAt Holding HasArmPosture HasTorsoPosture
5    Connected Type
6    !move_base !move_base_blind !place_object !pick_up_object
7    !move_arm_to_side !move_arms_to_carryposture !tuck_arms !move_torso
8    !observe_objects_on_area
9    adapt_torso torso_assume_driving_pose adapt_arms arms_assume_driving_pose
10   drive_robot move_both_arms_to_side assume_manipulation_pose
11   leave_manipulation_pose grasp_object get_object put_object
12   move_object serve_coffee_to_guest arm_to_side
13   serve_coffee_to_guest_test assume_manipulation_pose_wrapper
14   not_test Future Eternity
15 )
16
17 (Resource objManCapacity 1)
18 (Resource armManCapacity 1)
19 (Resource navigationCapacity 1)
20 (Resource leftArm1 1)
21 (Resource rightArm1 1)
```

```
22 (Resource objMoveCapacity 2)
23
24 (StateVariable RobotAt 2 n)
25 (StateVariable HasArmPosture 1 leftArm1 rightArm1)
26 (StateVariable Holding 1 leftArm1 rightArm1)
27 (StateVariable On 1 mug1 mug2 mug3)
28
29 ### !move_base
30 (:operator
31  (Head !move_base(?toArea))
32  (Pre p1 RobotAt(?fromArea))
33  (Constraint OverlappedBy(task,p1))
34  (Add e1 RobotAt(?toArea))
35  (Constraint Meets(task,e1))
36  (Del p1)
37  (ResourceUsage navigationCapacity 1)
38 )
39
40 ### !move_base_blind
41 (:operator
42  (Head !move_base_blind(?mArea))
43  (Pre p1 RobotAt(?preArea))
44  (Del p1)
45  (Pre p2 Connected(?plArea ?mArea ?preArea))
46  (Constraint Duration[4000,INF](task))
47  (Add e1 RobotAt(?mArea))
48  (Constraint Meets(task,e1))
49  (Constraint Meets(p1,task))
50  (ResourceUsage (Usage navigationCapacity 1))
51 )
52
53 (:operator
54  (Head !move_base_blind(?preArea))
55  (Pre p1 RobotAt(?mArea))
56  (Pre p2 Connected(?plArea ?mArea ?preArea))
57  (Constraint Duration[4000,INF](task))
58  (Add e1 RobotAt(?preArea))
59  (Constraint Meets(task,e1))
60  (Constraint Meets(p1,task))
61  (Del p1)
62  (ResourceUsage (Usage navigationCapacity 1))
63 )
64
65 ### !tuck_arms
66 (:operator
67  (Head !tuck_arms(?leftGoal ?rightGoal))
68  (Pre p1 HasArmPosture(?leftArm ?oldLeft))
69  (Pre p2 HasArmPosture(?rightArm ?oldRight))
70  (Del p1)
```

```
71   (Del p2)
72   (Add e1 HasArmPosture(?leftArm ?leftGoal))
73   (Add e2 HasArmPosture(?rightArm ?rightGoal))
74   (Values ?leftArm leftArm1)
75   (Values ?rightArm rightArm1)
76   (Values ?leftGoal ArmTuckedPosture ArmUnTuckedPosture)
77   (Values ?rightGoal ArmTuckedPosture ArmUnTuckedPosture)
78   (ResourceUsage (Usage armManCapacity 1))
79   (Constraint Duration[4000,INF](task))
80  )
81
82  ### !move_torso
83  (:operator
84   (Head !move_torso(?newPosture))
85   (Pre p1 HasTorsoPosture(?oldPosture))
86   (Constraint OverlappedBy(task,p1))
87   (Del p1)
88   (Add e1 HasTorsoPosture(?newPosture))
89   (Constraint Duration[4000,INF](task))
90  )
91
92  ### !pick_up_object
93  (:operator
94   (Head !pick_up_object(?obj ?arm))
95   (Pre p1 On(?obj ?fromArea))
96   (Pre p2 RobotAt(?mArea))
97   (Pre p3 Connected(?fromArea ?mArea ?preArea))
98   (Pre p4 Holding(?arm ?nothing))
99   (Values ?nothing nothing)
100  (Del p1)
101  (Del p4)
102  (Add e1 Holding(?arm ?obj))
103  (Constraint OverlappedBy(task,p1))
104  (Constraint During(task,p2))
105  (Constraint During(task,p3))
106  (Constraint OverlappedBy(task,p4))
107  (Constraint Meets(p4,e1))
108  (Constraint Duration[4000,INF](task))
109  (ResourceUsage (Usage armManCapacity 1))
110  (ResourceUsage (Usage objManCapacity 1))
111 )
112
113 ### !place_object
114 (:operator
115  (Head !place_object(?obj ?arm ?plArea))
116  (Pre p1 Holding(?arm ?obj))
117  (Pre p2 RobotAt(?mArea))
118  (Pre p3 Connected(?plArea ?mArea ?preArea))
119  (Pre p4 HasArmPosture(?arm ?armPosture))
```

```
120   (Values ?armPosture ArmToSidePosture)
121   (Del p1)
122   (Add e1 Holding(?arm ?nothing))
123   (Values ?nothing nothing)
124   (Add e2 On(?obj ?plArea))
125   (Constraint OverlappedBy(task,p1))
126   (Constraint During(task,p2))
127   (Constraint During(task,p3))
128   (Constraint Meets(p1,e1))
129   (Constraint Duration[4000,INF](task))
130   (ResourceUsage
131      (Usage leftArm1ManCapacity 1)
132      (Param 2 leftArm1))
133   (ResourceUsage
134      (Usage rightArm1ManCapacity 1)
135      (Param 2 rightArm1))
136   (ResourceUsage (Usage objManCapacity 1))
137  )
138
139 ### !move_arm_to_side
140  (:operator
141   (Head !move_arm_to_side(?arm))
142   (Pre p1 HasArmPosture(?arm ?oldPosture))
143   (Del p1)
144   (Add e1 HasArmPosture(?arm ?newPosture))
145   (Values ?oldPosture ArmUnTuckedPosture ArmCarryPosture ArmUnnamedPosture)
146   (Values ?newPosture ArmToSidePosture)
147   (ResourceUsage (Usage armManCapacity 1))
148   (Constraint Duration[4000,INF](task))
149   (Constraint OverlappedBy(task,p1))
150   (Constraint Overlaps(task,e1))
151  )
152
153  (:operator
154   (Head !move_arm_to_side(?arm)) # leftArm1
155   (Pre p1 HasArmPosture(?arm ?oldPosture))
156   (Pre p2 HasArmPosture(?otherArm ?otherPosture))
157   (Del p1)
158   (Add e1 HasArmPosture(?arm ?newPosture))
159   (Values ?arm leftArm1)
160   (Values ?otherArm rightArm1)
161   (Values ?oldPosture ArmTuckedPosture)
162   (Values ?otherPosture ArmUnTuckedPosture ArmCarryPosture
163                         ArmUnnamedPosture ArmToSidePosture)
164   (Values ?newPosture ArmToSidePosture)
165   (ResourceUsage (Usage armManCapacity 1))
166   (Constraint Duration[4000,INF](task))
167   (Constraint OverlappedBy(task,p1))
168   (Constraint Overlaps(task,e1))
```

```
169 )
170
171 (:operator
172  (Head !move_arm_to_side(?arm)) # rightArm1
173  (Pre p1 HasArmPosture(?arm ?oldPosture))
174  (Pre p2 HasArmPosture(?otherArm ?otherPosture))
175  (Del p1)
176  (Add e1 HasArmPosture(?arm ?newPosture))
177  (Values ?arm rightArm1)
178  (Values ?otherArm leftArm1)
179  (Values ?oldPosture ArmTuckedPosture)
180  (Values ?otherPosture ArmUnTuckedPosture ArmCarryPosture
181                        ArmUnnamedPosture ArmToSidePosture)
182  (Values ?newPosture ArmToSidePosture)
183  (ResourceUsage (Usage armManCapacity 1))
184  (Constraint Duration[4000,INF](task))
185  (Constraint OverlappedBy(task,p1))
186  (Constraint Overlaps(task,e1))
187 )
188
189 ### !move_arms_to_carryposture
190 (:operator
191  (Head !move_arms_to_carryposture())
192  (Pre p1 HasArmPosture(?leftArm ?oldLeft))
193  (Pre p2 HasArmPosture(?rightArm ?oldRight))
194  (Pre p3 HasTorsoPosture(?torsoPosture))
195  (Del p1)
196  (Del p2)
197  (Add e1 HasArmPosture(?leftArm ?newPosture))
198  (Add e2 HasArmPosture(?rightArm ?newPosture))
199  (Values ?leftArm leftArm1)
200  (Values ?rightArm rightArm1)
201  (Values ?newPosture ArmCarryPosture)
202  (Values ?torsoPosture TorsoUpPosture TorsoMiddlePosture)
203  (ResourceUsage (Usage armManCapacity 1))
204  (Constraint Duration[4000,INF](task))
205  (Constraint OverlappedBy(task,p1))
206  (Constraint OverlappedBy(task,p2))
207  (Constraint Overlaps(task,e1))
208  (Constraint Overlaps(task,e2))
209 )
210
211 ### !observe_objects_on_area
212 (:operator
213  (Head !observe_objects_on_area(?plArea))
214  (Pre p1 RobotAt(?robotArea))
215  (Pre p2 Connected(?plArea ?robotArea ?preArea))
216  (Constraint During(task,p1))
217  (Constraint During(task,p2))
```

```
218   (Constraint Duration[4000,INF](task))
219  )
220
221
222  (FluentResourceUsage
223    (Usage leftArm1 1)
224    (Fluent Holding)
225    (Param 2 leftArm1)
226  )
227
228  (FluentResourceUsage
229    (Usage rightArm1 1)
230    (Fluent Holding)
231    (Param 2 rightArm1)
232  )
233
234
235  ### adapt_torso
236  (:method
237   (Head adapt_torso(?newPose))
238   (Pre p1 HasTorsoPosture(?oldPose))
239   (VarDifferent ?newPose ?oldPose)
240   (Sub s1 !move_torso(?newPose))
241   (Constraint Equals(s1,task))
242  )
243
244  (:method
245   (Head adapt_torso(?posture))
246   (Pre p1 HasTorsoPosture(?posture))
247   (Constraint Duration[10,INF](task))
248   (Constraint During(task,p1))
249  )
250
251  ### torso_assume_driving_pose
252  (:method   # holding nothing
253   (Head torso_assume_driving_pose())
254    (Pre p1 Holding(?leftArm ?nothing))
255    (Pre p2 Holding(?rightArm ?nothing))
256    (Values ?nothing nothing)
257    (Values ?leftArm leftArm1)
258    (Values ?rightArm rightArm1)
259    (Sub s1 adapt_torso(?newPose))
260    (Values ?newPose TorsoDownPosture)
261    (Constraint Equals(s1,task))
262  )
263
264  (:method # holding something
265   (Head torso_assume_driving_pose())
266    (Pre p1 Holding(?arm ?obj))
```

```
267    (NotValues ?obj nothing)
268    (Sub s1 adapt_torso(?newPose))
269    (Values ?newPose TorsoMiddlePosture)
270    (Constraint Equals(s1,task))
271  )
272
273  ### adapt_arms
274  (:method
275   (Head adapt_arms(?posture))
276   (Pre p1 HasArmPosture(?leftArm ?posture))
277   (Pre p2 HasArmPosture(?rightArm ?posture))
278   (Values ?leftArm leftArm1)
279   (Values ?rightArm rightArm1)
280   (Constraint Duration[3,INF](task))
281   (Constraint During(task,p1))
282   (Constraint During(task,p2))
283  )
284
285  (:method  # tuck arms
286   (Head adapt_arms(?posture))
287   (Pre p1 HasArmPosture(?arm ?currentposture))
288   (Values ?posture ArmTuckedPosture)
289   (NotValues ?currentposture ArmTuckedPosture)
290   (Sub s1 !tuck_arms(?posture ?posture))
291   (Constraint Equals(s1,task))
292  )
293
294  (:method  # to carryposture
295   (Head adapt_arms(?posture))
296   (Pre p1 HasArmPosture(?arm ?currentposture))
297   (Values ?posture ArmCarryPosture)
298   (NotValues ?currentposture ArmCarryPosture)
299   (Sub s1 !move_arms_to_carryposture())
300   (Constraint Equals(s1,task))
301  )
302
303  ### arms_assume_driving_pose
304  (:method    # holding nothing
305   (Head arms_assume_driving_pose())
306    (Pre p1 Holding(?leftArm ?nothing))
307    (Pre p2 Holding(?rightArm ?nothing))
308    (Values ?nothing nothing)
309    (Values ?leftArm leftArm1)
310    (Values ?rightArm rightArm1)
311    (Sub s1 adapt_arms(?newPose))
312    (Values ?newPose ArmTuckedPosture)
313    (Constraint Equals(s1,task))
314  )
315
```

```
316 (:method    # holding something
317  (Head arms_assume_driving_pose())
318   (Pre p1 Holding(?arm ?obj))
319   (NotValues ?obj nothing)
320   (Sub s1 adapt_arms(?newPose))
321   (Values ?newPose ArmCarryPosture)
322   (Constraint Equals(s1,task))
323 )
324
325 ### drive_robot
326 (:method    # already there
327  (Head drive_robot(?toArea))
328  (Pre p1 RobotAt(?toArea))
329  (Constraint During(task,p1))
330  (Constraint Duration[10,INF](task))
331 )
332
333 (:method    # not at manipulationarea
334  (Head drive_robot(?toArea))
335  (Pre p1 RobotAt(?fromArea))
336  (VarDifferent ?toArea ?fromArea)
337  (NotType ?fromArea ManipulationArea)
338  (Sub s1 torso_assume_driving_pose())
339  (Constraint Starts(s1,task))
340  (Sub s2 arms_assume_driving_pose())
341  (Sub s3 !move_base(?toArea))
342  (Ordering s1 s2)
343  (Ordering s2 s3)
344  (Constraint Before(s1,s3))
345  (Constraint Before(s2,s3))
346 )
347
348 (:method    # at manipulationarea
349  (Head drive_robot(?toArea))
350  (Pre p1 RobotAt(?fromArea))
351  (VarDifferent ?toArea ?fromArea)
352  (Type ?fromArea ManipulationArea)
353  (Pre p2 Connected(?plArea ?fromArea ?preArea))
354  (Sub s0 !move_base_blind(?preArea))
355  (Constraint Starts(s0,task))
356  (Sub s1 torso_assume_driving_pose())
357  (Sub s2 arms_assume_driving_pose())
358  (Sub s3 !move_base(?toArea))
359  (Constraint Finishes(s3,task))
360  (Ordering s0 s1)
361  (Ordering s1 s2)
362  (Ordering s2 s3)
363  (Constraint Before(s0,s1))
364  (Constraint Before(s0,s2))
```

```
365   (Constraint Before(s1,s3))
366   (Constraint Before(s2,s3))
367  )
368
369  ### move_both_arms_to_side
370  (:method          # both are tucked
371   (Head move_both_arms_to_side())
372   (Pre p1 HasArmPosture(?leftArm ?oldLeftPosture))
373   (Pre p2 HasArmPosture(?rightArm ?oldRightPosture))
374   (Values ?oldLeftPosture ArmTuckedPosture)
375   (Values ?oldRightPosture ArmTuckedPosture)
376   (Values ?leftArm leftArm1)
377   (Values ?rightArm rightArm1)
378   (Sub s1 !tuck_arms(?lUntuckedPosture ?rUntuckedPosture))
379   (Values ?lUntuckedPosture ArmUnTuckedPosture)
380   (Values ?rUntuckedPosture ArmUnTuckedPosture)
381   (Sub s2 !move_arm_to_side(?leftArm))
382   (Sub s3 !move_arm_to_side(?rightArm))
383   (Ordering s1 s2)
384   (Ordering s2 s3)
385   (Constraint Starts(s1,task))
386   (Constraint Before(s1,s2))
387   (Constraint Before(s1,s3))
388  )
389
390  (:method # don't untuck if not both are tucked
391   (Head move_both_arms_to_side())
392   (Pre p1 HasArmPosture(?arm ?oldPosture))
393   (NotValues ?oldPosture ArmTuckedPosture)
394   (Values ?arm leftArm1 rightArm1)
395   (Values ?leftArm leftArm1)
396   (Values ?rightArm rightArm1)
397   (Sub s1 arm_to_side(?leftArm))
398   (Sub s2 arm_to_side(?rightArm))
399   (Ordering s1 s2)
400   (Constraint Before(s1,s2))
401  )
402
403  ### arm_to_side
404  (:method # arm is not at side
405   (Head arm_to_side(?arm))
406   (Pre p1 HasArmPosture(?arm ?armPosture))
407   (NotValues ?armPosture ArmToSidePosture)
408   (Sub s1 !move_arm_to_side(?arm))
409   (Constraint Equals(s1,task))
410  )
411
412  (:method # arm is at side
413   (Head arm_to_side(?arm))
```

```
414   (Pre p1 HasArmPosture(?arm ?armPosture))
415   (Values ?armPosture ArmToSidePosture)
416   (Constraint During(task,p1))
417  )
418
419  ### assume_manipulation_pose
420  (:method 10
421   (Head assume_manipulation_pose(?manArea))
422   (Pre p1 HasArmPosture(?leftArm ?leftPosture))
423   (Pre p2 HasArmPosture(?rightArm ?rightPosture))
424   (Pre p3 RobotAt(?manArea))
425   (Values ?leftArm leftArm1)
426   (Values ?rightArm rightArm1)
427   (Values ?leftPosture ArmToSidePosture)
428   (Values ?rightPosture ArmToSidePosture)
429   (Values ?torsoPosture TorsoUpPosture)
430   (Sub s1 adapt_torso(?torsoUpPosture))
431   (Constraint Equals(s1,task))
432  )
433
434  (:method
435   (Head assume_manipulation_pose(?manArea))
436   (Pre p1 RobotAt(?preArea))
437   (Pre p2 Connected(?plArea ?manArea ?preArea))
438   (Sub s1 adapt_torso(?torsoUpPosture))
439   (Values ?torsoUpPosture TorsoUpPosture)
440   (Sub s2 move_both_arms_to_side())
441   (Sub s3 !move_base_blind(?manArea))
442   (Ordering s1 s2)
443   (Ordering s2 s3)
444   (Constraint Before(s1,s3))
445   (Constraint Before(s2,s3))
446   (Constraint Starts(s1,task))
447   (Constraint Finishes(s3,task))
448  )
449
450  (:method 8
451   (Head assume_manipulation_pose(?manArea))
452   (Pre p1 RobotAt(?manArea))
453   (Pre p2 Connected(?plArea ?manArea ?preArea))
454   (Pre p3 HasArmPosture(?arm ?armPosture))
455   (NotValues ?armPosture ArmToSidePosture)
456   (Sub s0 !move_base_blind(?preArea))
457   (Sub s1 adapt_torso(?torsoUpPosture))
458   (Values ?torsoUpPosture TorsoUpPosture)
459   (Sub s2 move_both_arms_to_side())
460   (Sub s3 !move_base_blind(?manArea))
461   (Ordering s0 s1)
462   (Ordering s1 s2)
```

```
463   (Ordering s2 s3)
464   (Constraint Starts(s0,task))
465   (Constraint Finishes(s3,task))
466   (Constraint Before(s0,s1))
467   (Constraint Before(s0,s2))
468   (Constraint Before(s1,s3))
469   (Constraint Before(s2,s3))
470  )
471
472  ### leave_manipulation_pose
473  (:method
474   (Head leave_manipulation_pose(?manArea))
475   (Pre p1 RobotAt(?manArea))
476   (Pre p2 Connected(?plArea ?manArea ?preArea))
477   (Sub s1 !move_base_blind(?preArea))
478   (Constraint Equals(s1,task))
479  )
480
481  ### grasp_object
482  (:method
483   (Head grasp_object(?object))
484   (Pre p1 RobotAt(?preArea))
485   (Pre p2 Connected(?plArea ?manArea ?preArea))
486   (Pre p3 On(?object ?plArea))
487   (Values ?arm leftArm1 rightArm1)
488   (Sub s1 assume_manipulation_pose(?manArea))
489   (Sub s3 !pick_up_object(?object ?arm))
490   (Ordering s1 s3)
491   (Constraint Before(s1, s3))
492   (Constraint Starts(s1,task))
493   (Constraint Finishes(s3,task))
494  )
495
496  ### get_object
497  (:method 10
498   (Head get_object(?object))
499   (Pre p0 Connected(?plArea ?manArea ?preArea))
500   (Pre p1 On(?object ?plArea))
501   (Sub s1 assume_manipulation_pose(?manArea))
502   (Sub s2 !pick_up_object(?object ?arm))
503   (Ordering s1 s2)
504   (Constraint Before(s1,s2))
505   (Constraint Starts(s1,task))
506   (Constraint Finishes(s2,task))
507  )
508
509  (:method
510   (Head get_object(?object))
511   (Pre p1 RobotAt(?robotArea))
```

```
512  (Pre p2 Connected(?plArea ?manArea ?preArea))
513  (Pre p3 On(?object ?plArea))
514  (VarDifferent ?robotArea ?preArea)
515  (Sub s1 drive_robot(?preArea))
516  (Sub s2 assume_manipulation_pose(?manArea))
517  (Sub s3 !pick_up_object(?object ?arm))
518  (Ordering s1 s2)
519  (Ordering s2 s3)
520  (Constraint Before(s1,s2))
521  (Constraint Before(s2,s3))
522  (Constraint Starts(s1,task))
523  (Constraint Finishes(s3,task))
524 )
525
526 ### put_object
527 (:method
528  (Head put_object(?object ?plArea))
529  (Pre p1 Holding(?arm ?object))
530  (Pre p2 RobotAt(?robotArea))
531  (Pre p3 Connected(?plArea ?manArea ?preArea))
532  (VarDifferent ?robotArea ?preArea)
533  (VarDifferent ?robotArea ?manArea)
534  (Sub s1 drive_robot(?preArea))
535  (Sub s2 assume_manipulation_pose(?manArea))
536  (Sub s3 !place_object(?object ?arm ?plArea))
537  (Ordering s1 s2)
538  (Ordering s2 s3)
539  (Constraint Before(s1,s2))
540  (Constraint Before(s2,s3))
541  (Constraint Starts(s1,task))
542  (Constraint Finishes(s3,task))
543 )
544
545 (:method
546  (Head put_object(?object ?plArea))
547  (Pre p1 Holding(?arm ?object))
548  (Pre p2 RobotAt(?preArea))
549  (Pre p3 Connected(?plArea ?manArea ?preArea))
550  (Sub s1 assume_manipulation_pose(?manArea))
551  (Sub s2 !place_object(?object ?arm ?plArea))
552  (Ordering s1 s2)
553  (Constraint Starts(s1,task))
554  (Constraint Finishes(s2,task))
555  (Constraint Before(s1,s2))
556 )
557
558 (:method
559  (Head put_object(?object ?plArea))
560  (Pre p1 Holding(?arm ?object))
```

```
561   (Pre p2 RobotAt(?manArea))
562   (Pre p3 Connected(?plArea ?manArea ?preArea))
563   (Sub s1 !place_object(?object ?arm ?plArea))
564   (Constraint Equals(s1,task))
565  )
566
567  ### move_object
568  (:method
569   (Head move_object(?object ?toArea))
570   (Pre p1 On(?object ?fromArea))
571   (Sub s1 get_object(?object))
572   (Sub s2 put_object(?object ?toArea))
573   (Ordering s1 s2)
574   (Constraint Before(s1,s2))
575   (Constraint Starts(s1,task))
576   (Constraint Finishes(s2,task))
577   (Constraint Duration[20000,INF](task))
578   (ResourceUsage (Usage objMoveCapacity 1))
579  )
580
581  #### serve a hot coffee with milk and sugar
582  (:method
583   (Head serve_coffee_to_guest(?placingArea))
584   (Pre p0 Type(?coffeetype ?coffee))
585   (Values ?coffeetype Coffee)
586   (Values ?placingArea placingAreaEastLeftTable1 placingAreaWestLeftTable1
587                        placingAreaNorthLeftTable2 placingAreaSouthLeftTable2
588                        placingAreaNorthRightTable2)
589   (Pre p1 Type(?milktype ?milk))
590   (Values ?milktype Milk)
591   (Values ?sugar sugarPot1 sugarPot2)
592   (Sub s1 move_object(?coffee ?placingArea))
593   (Sub s2 move_object(?milk ?placingArea))
594   (Sub s3 move_object(?sugar ?placingArea))
595   (Ordering s1 s2)
596   (Ordering s2 s3)
597   (Constraint Before(s1,s3))
598   (Constraint Before(s2,s3))
599  )
```

Listing D.1: Vollständige in den Experimenten eingesetzte Beschreibung der RACE-Domäne für CHIMP.

D.2 Hafenarbeiter-Domänenbeschreibung für CHIMP

```
1  (HybridHTNDomain DWRDomain)
2  (MaxArgs 5)
3
4  (PredicateSymbols
5    # Static relations
```

```
 6   adjacent  # adjacent(dock waypoint)
 7   connected # connected(waypoint1 waypoint2)
 8    # Predicates
 9   r_loc       # robot is at location
10   d_occupant  # d_occupant(dock robot)
11   k_attached  # k_attached(crane dock)
12   p_ondock    # p_ondock(pile dock)
13   p_available # p_available(pile true/false)
14   k_grip      # k_grip(crane container)
15   c_in        # c_in(container pile)
16   c_on        # c_on(container container)
17   p_top       # p_top(pile container)
18   r_freight   # r_freight(robot container)
19    # Operators:
20   !leave      # !leave(robot dock waypoint) leaves a dock to a waypoint
21   !enter      # !enter(robot dock waypoint) enters a dock from a waypoint
22   !move       # !move(robot waypoint waypoint)
23   !stack      # ?crane holding ?container stacks it on top of ?pile
24   !unstack    # empty ?crane unstacks ?container from ?pile
25   !put        # ?crane holding ?conrainer puts it on ?robot which was empty
26   !take       # empty ?crane takes ?conrainer from ?robot
27    # Methods
28   load        # load ?container from ?pile onto ?robot
29   unload      # unload ?container from ?robot onto ?pile
30   uncover     # ?container in ?pile is rearranged onto the top of ?pile
31   navigate    # ?robot navigates between two waypoints
32   goto        # ?robot goes to ?dock
33   bring       # bring ?container to ?pile
34   robot_bring # robot brings container to pile
35 )
36
37 (StateVariable r_loc 1 r1 r2)
38 (StateVariable d_occupant 1 d1 d2 d3 d4)
39 (StateVariable k_attached 1 k1 k2 k3 k4)
40 (StateVariable p_ondock 1 p11 p12 p21 p22 p3 p4)
41 (StateVariable p_available 1 p11 p12 p21 p22 p3 p4)
42 (StateVariable k_grip 1 k1 k2 k3 k4)
43 (StateVariable c_in 1 c11 c12 c21 c22)
44 (StateVariable c_on 1 c11 c12 c21 c22)
45 (StateVariable p_top 1 p11 p12 p21 p22 p3 p4)
46 (StateVariable r_freight 1 r1 r2)
47
48 (Resource r1Activity 1)
49 (Resource r2Activity 1)
50 (Resource r1Goto 1)
51 (Resource r2Goto 1)
52
53 ####  OPERATORS ################
54
```

```
55 # leave a dock
56 (:operator
57  (Head !leave(?robot ?dock ?waypoint))
58  (Constraint Duration[10000,INF](task))
59  (Pre p0 adjacent(?dock ?waypoint))
60  (Constraint During(task,p0))
61  (Pre p1 r_loc(?robot ?dock))
62  (Del p1)
63  (Pre p2 d_occupant(?dock ?robot))
64  (Del p2)
65  (Add e1 r_loc(?robot ?waypoint))
66  (Constraint Meets(p1,e1))
67  (Add e2 d_occupant(?dock ?free))
68  (Constraint Meets(p2,e2))
69  (Values ?free free)
70  (ResourceUsage (Usage r1Activity 1) (Param 1 r1))
71  (ResourceUsage (Usage r2Activity 1) (Param 1 r2))
72 )
73
74 # enter a dock
75 (:operator
76  (Head !enter(?robot ?dock ?waypoint))
77  (Constraint Duration[20000,INF](task))
78  (Pre p0 adjacent(?dock ?waypoint))
79  (Constraint During(task,p0))
80  (Pre p1 r_loc(?robot ?waypoint))
81  (Del p1)
82  (Pre p2 d_occupant(?dock ?free))
83  (Del p2)
84  (Add e1 r_loc(?robot ?dock))
85  (Constraint Meets(p1,e1))
86  (Add e2 d_occupant(?dock ?robot))
87  (Constraint Meets(p2,e2))
88  (Values ?free free)
89  (ResourceUsage (Usage r1Activity 1) (Param 1 r1))
90  (ResourceUsage (Usage r2Activity 1) (Param 1 r2))
91 )
92
93 # move from waypoint1 to waypoint2
94 (:operator
95  (Head !move(?robot ?wp1 ?wp2))
96  (Constraint Duration[10000,20000](task))
97  (Pre p0 connected(?wp1 ?wp2))
98  (Constraint During(task,p0))
99  (Pre p1 r_loc(?robot ?wp1))
100 (Del p1)
101 (Add e1 r_loc(?robot ?wp2))
102 (Constraint Meets(p1,e1))
103 (ResourceUsage (Usage r1Activity 1) (Param 1 r1))
```

```
104  (ResourceUsage (Usage r2Activity 1) (Param 1 r2))
105 )
106
107 # ?crane holding ?container stacks it on top of ?pile
108 (:operator
109  (Head !stack(?crane ?container ?pile))
110  (Constraint Duration[10000,INF](task))
111  (Pre p0 k_attached(?crane ?dock))
112  (Pre p1 p_ondock(?pile ?dock))
113  (Pre p2 p_available(?pile ?true))
114  (Values ?true true)
115  (Pre p3 k_grip(?crane ?container))
116  (Del p3)
117  (Add e3 k_grip(?crane ?empty))
118  (Values ?empty empty)
119  (Constraint Meets(p3,e3))
120  (Pre p4 c_in(?container ?crane))
121  (Del p4)
122  (Add e4 c_in(?container ?pile))
123  (Constraint Meets(p4,e4))
124  (Pre p5 c_on(?container ?empty))
125  (Del p5)
126  (Add e5 c_on(?container ?prevtop))
127  (Constraint Meets(p5,e5))
128  (Pre p6 p_top(?pile ?prevtop))
129  (Del p6)
130  (Add e6 p_top(?pile ?container))
131  (Constraint Meets(p6,e6))
132 )
133
134 # empty ?crane unstacks ?container from ?pile
135 (:operator
136 (Head !unstack(?crane ?container ?pile))
137 (Constraint Duration[10000,INF](task))
138 (Pre p0 k_attached(?crane ?dock))
139 (Pre p1 p_ondock(?pile ?dock))
140 (Pre p2 p_available(?pile ?true))
141 (Values ?true true)
142 (Pre p3 k_grip(?crane ?empty))
143 (Del p3)
144 (Values ?empty empty)
145 (Add e3 k_grip(?crane ?container))
146 (Constraint Meets(p3,e3))
147 (Pre p4 c_in(?container ?pile))
148 (Del p4)
149 (Add e4 c_in(?container ?crane))
150 (Constraint Meets(p4,e4))
151 (Pre p5 c_on(?container ?nextop))
152 (Del p5)
```

```
153 (Add e5 c_on(?container ?empty))
154 (Constraint Meets(p5,e5))
155 (Pre p6 p_top(?pile ?container))
156 (Del p6)
157 (Add e6 p_top(?pile ?nextop))
158 (Constraint Meets(p6,e6))
159 )
160
161 # ?crane holding ?conrainer puts it on ?robot which was empty
162 (:operator
163  (Head !put(?crane ?container ?robot))
164  (Constraint Duration[10000,INF](task))
165  (Pre p0 k_attached(?crane ?dock))
166  (Pre p1 k_grip(?crane ?container))
167  (Del p1)
168  (Add e1 k_grip(?crane ?empty))
169  (Values ?empty empty)
170  (Constraint Meets(p1,e1))
171  (Pre p2 r_freight(?robot ?empty))
172  (Del p2)
173  (Add e2 r_freight(?robot ?container))
174  (Constraint Meets(p2,e2))
175  (Pre p3 c_in(?container ?crane))
176  (Del p3)
177  (Add e3 c_in(?container ?robot))
178  (Constraint Meets(p3,e3))
179  (Pre p4 r_loc(?robot ?dock))
180  (Constraint During(task,p4))
181  (ResourceUsage (Usage r1Activity 1) (Param 3 r1))
182  (ResourceUsage (Usage r2Activity 1) (Param 3 r2))
183 )
184
185 # empty ?crane takes ?conrainer from ?robot
186 (:operator
187  (Head !take(?crane ?container ?robot))
188  (Constraint Duration[10000,INF](task))
189  (Pre p0 k_attached(?crane ?dock))
190  (Pre p1 k_grip(?crane ?empty))
191  (Del p1)
192  (Add e1 k_grip(?crane ?container))
193  (Values ?empty empty)
194  (Constraint Meets(p1,e1))
195  (Pre p2 r_freight(?robot ?container))
196  (Del p2)
197  (Add e2 r_freight(?robot ?empty))
198  (Constraint Meets(p2,e2))
199  (Pre p3 c_in(?container ?robot))
200  (Del p3)
201  (Add e3 c_in(?container ?crane))
```

```
202   (Constraint Meets(p3,e3))
203   (Pre p4 r_loc(?robot ?dock))
204   (Constraint During(task,p4))
205   (ResourceUsage (Usage r1Activity 1) (Param 3 r1))
206   (ResourceUsage (Usage r2Activity 1) (Param 3 r2))
207  )
208
209
210  ##################### METHODS #####################################
211
212  # load ?container from ?pile onto ?robot
213  (:method
214   (Head load(?container ?robot ?pile))
215   (Pre p0 k_attached(?crane ?dock))
216   (Pre p1 p_ondock(?pile ?dock))
217   (Pre p2 p_top(?pile ?container))
218   (Pre p3 r_loc(?robot ?dock))
219   (Constraint During(task,p3))
220   (Pre p4 p_available(?pile ?true))
221   (Values ?true true)
222   (Constraint During(task,p4))
223   (Sub s1 !unstack(?crane ?container ?pile))
224   (Sub s2 !put(?crane ?container ?robot))
225   (Constraint BeforeOrMeets(s1,s2))
226   (Constraint Starts(s1,task))
227   (Constraint Finishes(s2,task))
228   (Ordering s1 s2)
229  )
230
231  # unload ?container from ?robot onto ?pile
232  (:method
233   (Head unload(?container ?robot ?pile))
234   (Pre p0 k_attached(?crane ?dock))
235   (Pre p1 p_ondock(?pile ?dock))
236   (Pre p2 c_in(?container ?robot))
237   (Pre p3 r_loc(?robot ?dock))
238   (Constraint During(task,p3))
239   (Pre p4 p_available(?pile ?true))
240   (Values ?true true)
241   (Constraint During(task,p4))
242   (Sub s1 !take(?crane ?container ?robot))
243   (Sub s2 !stack(?crane ?container ?pile))
244   (Constraint Starts(s1,task))
245   (Constraint Finishes(s2,task))
246   (Constraint BeforeOrMeets(s1,s2))
247   (Ordering s1 s2)
248  )
249
250  # ?container in ?pile is rearranged onto the top of ?pile
```

```
251 (:method 10
252  (Head uncover(?container ?pile))
253  (Pre p0 c_in(?container ?pile))
254  (Pre p1 p_top(?pile ?container))
255 )
256
257 (:method 1
258  (Head uncover(?container ?pile))
259  (Pre p0 c_in(?container ?pile))
260  (Pre p1 p_top(?pile ?prevtop))
261  (VarDifferent ?prevtop ?container)
262  (Pre p2 k_attached(?crane ?dock))
263  (Pre p3 p_ondock(?pile ?dock))
264  (Pre p4 p_ondock(?otherp ?dock))
265  (VarDifferent ?otherp ?pile)
266  (Pre p5 p_available(?pile ?true))
267  (Pre p6 p_available(?otherp ?true))
268  (Values ?true true)
269  (Sub s1 !unstack(?crane ?prevtop ?pile))
270  (Sub s2 !stack(?crane ?prevtop ?otherp))
271  (Sub s3 uncover(?container ?pile))
272  (Constraint BeforeOrMeets(s1,s2))
273  (Constraint BeforeOrMeets(s2,s3))
274  (Constraint Starts(s1,task))
275  (Constraint Finishes(s3,task))
276  (Ordering s1 s2)
277  (Ordering s2 s3)
278 )
279
280 # ?robot navigates between two waypoints
281 (:method 10
282  (Head navigate(?robot ?wp1 ?wp2))
283  (Pre p0 connected(?wp1 ?wp2))
284  (Sub s1 !move(?robot ?wp1 ?wp2))
285 )
286
287 (:method 0
288  (Head navigate(?robot ?wp1 ?wp2))
289  (Pre p0 connected(?wp1 ?wp3))
290  (VarDifferent ?wp2 ?wp3)
291  (Sub s1 !move(?robot ?wp1 ?wp3))
292  (Sub s2 navigate(?robot ?wp3 ?wp2))
293  (Constraint BeforeOrMeets(s1,s2))
294  (Ordering s1 s2)
295  (Constraint Duration[1,30000](task))
296 )
297
298 # ?robot goes to ?dock
299 (:method 10
```

```
300  (Head goto(?robot ?dock))
301  (Pre p0 r_loc(?robot ?dock))
302 )
303
304 (:method 1
305  (Head goto(?robot ?dock))
306  (Pre p0 r_loc(?robot ?from))
307  (VarDifferent ?from ?dock)
308  (Pre p1 adjacent(?from ?wp1))
309  (Pre p2 adjacent(?dock ?wp2))
310  (Sub s1 !leave(?robot ?from ?wp1))
311  (Sub s2 navigate(?robot ?wp1 ?wp2))
312  (Sub s3 !enter(?robot ?dock ?wp2))
313  (Constraint BeforeOrMeets(s1,s2))
314  (Constraint BeforeOrMeets(s2,s3))
315  (Constraint Starts(s1,task))
316  (Constraint Finishes(s3,task))
317  (Ordering s1 s2)
318  (Ordering s2 s3)
319  (ResourceUsage (Usage r1Goto 1) (Param 1 r1))
320  (ResourceUsage (Usage r2Goto 1) (Param 1 r2))
321 )
322
323 # bring ?container to ?pile
324 (:method 10
325  (Head bring(?container ?pile))
326  (Pre p0 c_in(?container ?pile))
327  (Constraint During(task,p0))
328  (Constraint Duration[1,1](task))
329 )
330
331 (:method 5
332  (Head bring(?container ?pile))
333  (Pre p0 k_attached(?crane ?dock))
334  (Pre p1 p_ondock(?pile ?dock))
335  (Pre p2 p_ondock(?otherp ?dock))
336  (VarDifferent ?pile ?otherp)
337  (Pre p3 c_in(?container ?otherp))
338  (Pre p4 p_available(?pile ?true))
339  (Pre p5 p_available(?otherp ?true))
340  (Values ?true true)
341  (Sub s1 uncover(?container ?otherp))
342  (Sub s2 !unstack(?crane ?container ?otherp))
343  (Sub s3 !stack(?crane ?container ?pile))
344  (Constraint BeforeOrMeets(s1,s2))
345  (Constraint BeforeOrMeets(s2,s3))
346  (Constraint Starts(s1,task))
347  (Constraint Finishes(s3,task))
348  (Ordering s1 s2)
```

```
349  (Ordering s2 s3)
350 )
351
352 # bring to other dock
353 (:method
354  (Head bring(?container ?pile))
355  (Type ?robot Robot)
356  (Pre p0 p_ondock(?pile ?todock))
357  (Pre p1 p_ondock(?otherp ?fromdock))
358  (VarDifferent ?fromdock ?todock)
359  (Pre p2 c_in(?container ?otherp))
360  (Pre p3 p_available(?pile ?true))
361  (Values ?true true)
362  (Pre p4 p_available(?otherp ?true))
363  (Sub s1 goto(?robot ?fromdock))
364  (Sub s2 uncover(?container ?otherp))
365  (Sub s3 load(?container ?robot ?otherp))
366  (Sub s4 goto(?robot ?todock))
367  (Sub s5 unload(?container ?robot ?pile))
368  (Constraint BeforeOrMeets(s1,s3))
369  (Constraint BeforeOrMeets(s2,s3))
370  (Constraint BeforeOrMeets(s3,s4))
371  (Constraint BeforeOrMeets(s4,s5))
372  (Ordering s1 s2)
373  (Ordering s2 s3)
374  (Ordering s3 s4)
375  (Ordering s4 s5)
376  (ResourceUsage (Usage rBring 1))
377 )
378
379 # bring ?container to ?pile with robot
380 (:method 10
381  (Head robot_bring(?robot ?container ?pile))
382  (Type ?robot Robot)
383  (Pre p0 c_in(?container ?pile))
384  (Constraint During(task,p0))
385  (Constraint Duration[1,1](task))
386 )
387
388 (:method 5
389  (Head robot_bring(?robot ?container ?pile))
390  (Type ?robot Robot)
391  (Pre p0 k_attached(?crane ?dock))
392  (Pre p1 p_ondock(?pile ?dock))
393  (Pre p2 p_ondock(?otherp ?dock))
394  (VarDifferent ?pile ?otherp)
395  (Pre p3 c_in(?container ?otherp))
396  (Pre p4 p_available(?pile ?true))
397  (Pre p5 p_available(?otherp ?true))
```

```
398  (Values ?true true)
399  (Sub s1 uncover(?container ?otherp))
400  (Sub s2 !unstack(?crane ?container ?otherp))
401  (Sub s3 !stack(?crane ?container ?pile))
402  (Constraint BeforeOrMeets(s1,s2))
403  (Constraint BeforeOrMeets(s2,s3))
404  (Ordering s1 s2)
405  (Ordering s2 s3)
406 )
407
408 # bring to other dock
409 (:method 1
410  (Head robot_bring(?robot ?container ?pile))
411  (Type ?robot Robot)
412  (Pre p0 p_ondock(?pile ?todock))
413  (Pre p1 p_ondock(?otherp ?fromdock))
414  (VarDifferent ?fromdock ?todock)
415  (Pre p2 c_in(?container ?otherp))
416  (Pre p3 p_available(?pile ?true))
417  (Values ?true true)
418  (Pre p4 p_available(?otherp ?true))
419  (Sub s1 goto(?robot ?fromdock))
420  (Sub s2 uncover(?container ?otherp))
421  (Sub s3 load(?container ?robot ?otherp))
422  (Sub s4 goto(?robot ?todock))
423  (Sub s5 unload(?container ?robot ?pile))
424  (Constraint BeforeOrMeets(s1,s3))
425  (Constraint BeforeOrMeets(s2,s3))
426  (Constraint BeforeOrMeets(s3,s4))
427  (Constraint BeforeOrMeets(s4,s5))
428  (Ordering s1 s2)
429  (Ordering s2 s3)
430  (Ordering s3 s4)
431  (Ordering s4 s5)
432  (ResourceUsage (Usage r1RobotBring 1) (Param 1 r1))
433  (ResourceUsage (Usage r2RobotBring 1) (Param 1 r2))
434 )
```

Listing D.2: Domänenbeschreibung des Hafenarbeiterbeispiels für CHIMP aus Abschnitt 6.1.

D.3 Domänen- und Problembeschreibungen des Weltraumszenarios

```
1 (HybridHTNDomain TransTerraDomain)
2 (MaxArgs 3)
3
4 (PredicateSymbols At RobotAt Attached ContainerAt BatteryAt
5  !create_attached_fluent !!check_empty !move_to !sample_regolith
6  !transfer_sample !transfer_battery !transfer_payload !pickup_basecamp
```

```
 7   !tuck_arms !place_basecamp deploy_basecamp take_samples get_basecamp
 8   transfer_all_samples transfer_filled_containers transfer_empty_containers
 9   transfer_charged_batteries transfer_discharged_batteries
10   rendezvous rendezvous_meet rendezvous_exchange_batteries
11   rendezvous_exchange_samples deposit_samples scenario_test)
12
13  (Resource FilledSampleStorageCapacityRover 5)
14  (Resource FilledSampleStorageCapacityShuttle 5)
15  (Resource EmptySampleStorageCapacityShuttle 5)
16  (Resource ChargedBatteryStorageCapacityShuttle 5)
17  (Resource DischargedBatteryStorageCapacityRover 5)
18
19  (FluentResourceUsage
20   (Usage FilledSampleStorageCapacityRover 1)
21   (Fluent ContainerAt)
22   (Param 2 rover1)
23   (Param 3 filled)
24  )
25
26  (FluentResourceUsage
27   (Usage FilledSampleStorageCapacityShuttle 1)
28   (Fluent ContainerAt)
29   (Param 2 shuttle1)
30   (Param 3 filled)
31  )
32
33  (FluentResourceUsage
34   (Usage EmptySampleStorageCapacityShuttle 1)
35   (Fluent ContainerAt)
36   (Param 2 shuttle1)
37   (Param 3 empty)
38  )
39
40  (FluentResourceUsage
41   (Usage ChargedBatteryStorageCapacityShuttle 1)
42   (Fluent BatteryAt)
43   (Param 2 shuttle1)
44   (Param 3 charged)
45  )
46
47  (FluentResourceUsage
48   (Usage DischargedBatteryStorageCapacityRover 1)
49   (Fluent BatteryAt)
50   (Param 2 rover1)
51   (Param 3 discharged)
52  )
53
54  ############ Operators ###############
55
```

```
56 #### Auxiliry operators:
57 (:operator
58  (Head !!check_empty())
59  (ResourceUsage SampleStorageCapacityRover 5)
60  (Constraint Duration[2,INF](task))
61 )
62
63 #### Real operators:
64 ### Move to a specified position
65 (:operator
66  (Head !move_to(?robot ?toArea))
67  (Pre p1 RobotAt(?robot ?fromArea))
68  (Constraint OverlappedBy(task,p1))
69  (Add e1 RobotAt(?robot ?toArea))
70  (Constraint Meets(task,e1))
71  (Del p1)
72 )
73
74 ### Take regolith samples from at a specified area
75 (:operator
76  (Head !sample_regolith(?robot ?area))
77  (Values ?robot rover1)
78  (Pre p1 RobotAt(?robot ?area))
79  (Constraint During(task,p1)) # robot has to stay there
80  (Pre p2 ContainerAt(?samplecontainer ?robot ?empty))
81  (Values ?empty empty)
82  (Del p2)
83  (Add e1 ContainerAt(?samplecontainer ?robot ?filled))
84  (Values ?filled filled)
85  (Constraint Meets(p2,e1))
86  (ResourceUsage ManipulationCapacityRover 1)
87 )
88
89 ### Transfer samples from one robot1 to robot2
90 # robots need to be at the same area/position
91 (:operator
92  (Head !transfer_sample(?robot1 ?robot2 ?container))
93  (Pre p1 RobotAt(?robot1 ?area))
94  (Pre p2 RobotAt(?robot2 ?area))
95  (Pre p3 ContainerAt(?container ?robot1 ?level))
96  (Del p3)
97  (Constraint During(task,p1))
98  (Constraint During(task,p2))
99  (Add e1 ContainerAt(?container ?robot2 ?level))
100  (Constraint Meets(task,e1))
101 )
102
103 ### Transfer samples from one robot1 to robot2
104 # robots need to be at the same area/position
```

```
105 (:operator
106  (Head !transfer_battery(?robot1 ?robot2 ?battery))
107  (Pre p1 RobotAt(?robot1 ?area))
108  (Pre p2 RobotAt(?robot2 ?area))
109  (Pre p3 BatteryAt(?battery ?robot1 ?status))
110  (Del p3)
111  (Constraint During(task,p1))
112  (Constraint During(task,p2))
113  (Add e1 BatteryAt(?battery ?robot2 ?status))
114  (Constraint Meets(task,e1))
115 )
116
117 ### Exchange payload item
118 (:operator
119  (Head !transfer_payload(?robot1 ?robot2 ?payload))
120  (Pre p1 RobotAt(?robot1 ?area))
121  (Pre p2 RobotAt(?robot2 ?area))
122  (Pre p3 Attached(?payload ?robot1))
123  (Del p3)
124  (Constraint During(task,p1))
125  (Constraint During(task,p2))
126  (Add e1 Attached(?payload ?robot2))
127  (Constraint Meets(task,e1))
128 )
129
130 ### Pick up a BaseCamp from area
131 (:operator
132  (Head !pickup_basecamp(?robot ?camp))
133  (Values ?robot rover1)
134  (Type ?camp BaseCamp)
135  (Pre p1 RobotAt(?robot1 ?area))
136  (Constraint During(task,p1))
137  (Pre p2 At(?camp ?area))
138  (Del p2)
139  (Constraint Meets(p2,task))
140  (Add e1 Attached(?camp ?robot))
141  (Constraint Meets(task,e1))
142  (ResourceUsage ManipulationCapacityRover 1)
143 )
144
145 ### Pick up a BaseCamp from lander
146 (:operator
147  (Head !pickup_basecamp(?robot ?camp))
148  (Values ?robot rover1)
149  (Type ?camp BaseCamp)
150  (Pre p1 RobotAt(?robot1 ?area))
151  (Constraint During(task,p1))
152  (Pre p2 Attached(?camp ?lander))
153  (Del p2)
```

```
154   (Pre p3 RobotAt(?lander ?area))
155   (Constraint Meets(p2,task))
156   (Add e1 Attached(?camp ?robot))
157   (Constraint Meets(task,e1))
158   (ResourceUsage ManipulationCapacityRover 1)
159  )
160
161  ### Place a BaseCamp
162  (:operator
163   (Head !place_basecamp(?robot ?camp ?area))
164   (Values ?robot rover1)
165   (Type ?camp BaseCamp)
166   (Pre p1 RobotAt(?robot1 ?area))
167   (Constraint During(task,p1))
168   (Pre p2 Attached(?camp ?robot))
169   (Del p2)
170   (Add e1 At(?camp ?area))
171   (Constraint Meets(task,e1))
172   (ResourceUsage ManipulationCapacityRover 1)
173  )
174
175  ############ Methods   ################
176
177  ### Deploy basecamp at a given position
178  # case1:
179  # basecamp is attached to the robot
180  # robot is not at the goal position
181  (:method
182   (Head deploy_basecamp(?robot ?toArea))
183   (Pre p1 RobotAt(?robot ?fromArea))
184   (VarDifferent ?toArea ?fromArea)
185   (Pre p2 Attached(?camp ?robot))
186   (Type ?camp BaseCamp)
187   (Sub s1 !move_to(?robot ?toArea))
188   (Constraint Starts(s1,task))
189   (Sub s2 !place_basecamp(?robot ?camp ?toArea))
190   (Constraint Finishes(s2,task))
191   (Ordering s1 s2)
192   (Constraint Before(s1,s2))
193  )
194
195  # case2:
196  # basecamp is attached to the robot
197  # robot is already at the position
198  (:method
199  (Head deploy_basecamp(?robot ?area))
200   (Pre p1 RobotAt(?robot ?area))
201   (Pre p2 Attached(?camp ?robot))
202   (Type ?camp BaseCamp)
```

```
203  (Sub s1 !place_basecamp(?robot ?camp ?toArea))
204  (Constraint Equals(s1,task))
205 )
206
207 ### Take regolith samples ##################
208
209 # case 1: robot is already at the position
210 (:method
211  (Head take_samples(?robot ?area))
212  (Pre p1 RobotAt(?robot ?area))
213  (Sub s1 !sample_regolith(?robot ?area))
214  (Constraint Equals(s1,task))
215 )
216
217 # case 2: robot is not at the position
218 (:method
219  (Head take_samples(?robot ?toArea))
220  (Pre p1 RobotAt(?robot ?fromArea))
221  (VarDifferent ?toArea ?fromArea)
222  (Sub s1 !move_to(?robot ?toArea))
223  (Constraint Starts(s1,task))
224  (Sub s2 !sample_regolith(?robot ?toArea))
225  (Constraint Finishes(s2,task))
226  (Ordering s1 s2)
227  (Constraint Before(s1,s2))
228 )
229
230 ### Shuttle rendezvous ################
231 (:method
232  (Head rendezvous(?robot1 ?robot2))
233  (VarDifferent ?robot1 ?robot2)
234  (Sub s1 rendezvous_meet(?robot1 ?robot2))
235  (Sub s2 rendezvous_exchange_samples(?robot1 ?robot2))
236  (Constraint Before(s1,s2))
237  (Ordering s1 s2)
238  (Sub s3 rendezvous_exchange_batteries(?robot1 ?robot2))
239  (Constraint Before(s2,s3))
240  (Ordering s2 s3)
241 )
242
243 (:method  # meet at the same location
244  (Head rendezvous_meet(?robot1 ?robot2))
245  (Pre p1 RobotAt(?robot1 ?r1Area))
246  (Pre p2 RobotAt(?robot2 ?r2Area))
247  (VarDifferent ?r1Area ?r2Area)
248  (Sub s1 !move_to(?robot2 ?r1Area))
249  (Constraint Equals(s1,task))
250 )
251
```

```
252 (:method
253  (Head rendezvous_meet(?robot1 ?robot2))
254  (VarDifferent ?robot1 ?robot2)
255 )
256
257 (:method  # exchange samples
258 (Head rendezvous_exchange_samples(?robot1 ?robot2))
259  (Sub s1 transfer_filled_containers(?robot1 ?robot2))
260  (Sub s2 transfer_empty_containers(?robot2 ?robot1))
261  (Constraint Before(s1,s2))
262  (Ordering s1 s2)
263 )
264
265 (:method
266 (Head rendezvous_exchange_batteries(?robot1 ?robot2))
267  (Sub s1 transfer_charged_batteries(?robot2 ?robot1))
268  (Sub s2 transfer_discharged_batteries(?robot1 ?robot2))
269  (Constraint Before(s1,s2))
270  (Ordering s1 s2)
271 )
272
273 ### Get a basecamp from the lander #####
274 # case 1: robot is at a different area
275 (:method
276  (Head get_basecamp(?robot ?camp))
277  (Type ?robot Rover)
278  (Type ?camp BaseCamp)
279  (Pre p0 Attached(?camp ?lander))
280  (Type ?lander Lander)
281  (Pre p1 RobotAt(?robot ?robotArea))
282  (Pre p2 RobotAt(?lander ?landerArea))
283  (VarDifferent ?robotArea ?landerArea)
284  (Sub s1 !move_to(?robot ?landerArea))
285  (Constraint Starts(s1,task))
286  (Sub s2 !pickup_basecamp(?robot ?camp))
287  (Constraint Finishes(s2,task))
288  (Ordering s1 s2)
289  (Constraint Before(s1,s2))
290 )
291
292 # case 2: robot is already near the lander
293 (:method
294  (Head get_basecamp(?robot ?camp))
295  (Type ?robot Rover)
296  (Type ?camp BaseCamp)
297  (Pre p0 Attached(?camp ?lander))
298  (Type ?lander Lander)
299  (Pre p1 RobotAt(?robot ?area))
300  (Pre p2 RobotAt(?lander ?area))
```

```
301   (Sub s1 !pickup_basecamp(?robot ?camp))
302   (Constraint Equals(s1,task))
303  )
304
305  ### Transfer all samples
306  (:method
307   (Head transfer_all_samples(?robot1 ?robot2))
308   (Pre p0 ContainerAt(?container ?robot1 ?level))
309   (Type ?container SampleContainer)
310   (Pre p1 RobotAt(?robot1 ?robotArea))
311   (Pre p2 RobotAt(?robot2 ?robotArea))
312   (Sub s1 !transfer_sample(?robot1 ?robot2 ?container))
313   (Constraint Starts(s1,task))
314   (Sub s2 transfer_all_samples(?robot1 ?robot2))
315   (Constraint Finishes(s2,task))
316   (Ordering s1 s2)
317   (Constraint Before(s1,s2))
318  )
319
320  (:method
321   (Head transfer_all_samples(?robot1 ?robot2))
322   (Values ?robot1 rover1)
323   (ResourceUsage SampleStorageCapacityRover 5)
324   (Constraint Duration[2,INF](task))
325  )
326
327  ### Transfer filled samples
328  (:method
329   (Head transfer_filled_containers(?robot1 ?robot2))
330   (Pre p0 ContainerAt(?container ?robot1 ?filled))
331   (Values ?filled filled)
332   (Pre p1 RobotAt(?robot1 ?robotArea))
333   (Pre p2 RobotAt(?robot2 ?robotArea))
334   (Sub s1 !transfer_sample(?robot1 ?robot2 ?container))
335   (Constraint Starts(s1,task))
336   (Sub s2 transfer_filled_containers(?robot1 ?robot2))
337   (Constraint Finishes(s2,task))
338   (Ordering s1 s2)
339   (Constraint Before(s1,s2))
340  )
341
342  (:method
343   (Head transfer_filled_containers(?robot1 ?robot2))
344   (Values ?robot1 rover1)
345   (ResourceUsage FilledSampleStorageCapacityRover 5)
346   (Constraint Duration[2,INF](task))
347  )
348
349  (:method
```

```
350  (Head transfer_filled_containers(?robot1 ?robot2))
351  (Values ?robot1 shuttle1)
352  (ResourceUsage FilledSampleStorageCapacityShuttle 5)
353  (Constraint Duration[2,INF](task))
354 )
355
356 ### Transfer empty containers
357 (:method
358  (Head transfer_empty_containers(?robot1 ?robot2))
359  (Pre p0 ContainerAt(?container ?robot1 ?empty))
360  (Values ?empty empty)
361  (Pre p1 RobotAt(?robot1 ?robotArea))
362  (Pre p2 RobotAt(?robot2 ?robotArea))
363  (Sub s1 !transfer_sample(?robot1 ?robot2 ?container))
364  (Constraint Starts(s1,task))
365  (Sub s2 transfer_empty_containers(?robot1 ?robot2))
366  (Constraint Finishes(s2,task))
367  (rdering s1 s2)
368  (Constraint Before(s1,s2))
369 )
370
371 (:method
372  (Head transfer_empty_containers(?robot1 ?robot2))
373  (Values ?robot1 shuttle1)
374  (ResourceUsage EmptySampleStorageCapacityShuttle 5)
375  (Constraint Duration[2,INF](task))
376 )
377
378 ### Transfer charged batteries
379 (:method
380  (Head transfer_charged_batteries(?robot1 ?robot2))
381  (Pre p0 BatteryAt(?battery ?robot1 ?charged))
382  (Values ?charged charged)
383  (Pre p1 RobotAt(?robot1 ?robotArea))
384  (Pre p2 RobotAt(?robot2 ?robotArea))
385  (Sub s1 !transfer_battery(?robot1 ?robot2 ?battery))
386  (Constraint Starts(s1,task))
387  (Sub s2 transfer_charged_batteries(?robot1 ?robot2))
388  (Constraint Finishes(s2,task))
389  (Ordering s1 s2)
390  (Constraint Before(s1,s2))
391 )
392
393 (:method
394  (Head transfer_charged_batteries(?robot1 ?robot2))
395  (Values ?robot1 shuttle1)
396  (ResourceUsage ChargedBatteryStorageCapacityShuttle 5)
397  (Constraint Duration[2,INF](task))
398 )
```

```
399
400 ### Transfer discharged batteries
401  (:method
402   (Head transfer_discharged_batteries(?robot1 ?robot2))
403   (Pre p0 BatteryAt(?battery ?robot1 ?discharged))
404   (Values ?discharged discharged)
405   (Pre p1 RobotAt(?robot1 ?robotArea))
406   (Pre p2 RobotAt(?robot2 ?robotArea))
407   (Sub s1 !transfer_battery(?robot1 ?robot2 ?battery))
408   (Constraint Starts(s1,task))
409   (Sub s2 transfer_discharged_batteries(?robot1 ?robot2))
410   (Constraint Finishes(s2,task))
411   (Ordering s1 s2)
412   (Constraint Before(s1,s2))
413  )
414
415  (:method
416   (Head transfer_discharged_batteries(?robot1 ?robot2))
417   (Values ?robot1 rover1)
418   (ResourceUsage DischargedBatteryStorageCapacityRover 5)
419   (Constraint Duration[2,INF](task))
420  )
421
422  (:method
423   (Head deposit_samples(?robot ?lander))
424   (Pre p1 RobotAt(?robot ?robotArea))
425   (Pre p2 RobotAt(?lander ?landerArea))
426   (VarDifferent ?robotArea ?landerArea)
427   (Sub s1 !move_to(?robot ?landerArea))
428   (Constraint Starts(s1,task))
429   (Sub s2 transfer_filled_containers(?robot ?lander))
430   (Constraint Finishes(s2,task))
431   (Ordering s1 s2)
432   (Constraint Before(s1,s2))
433  )
434
435  (:method
436   (Head scenario_test())
437   (Sub s1 rendezvous(?rover ?shuttle))
438   (Values ?rover rover1)
439   (Values ?shuttle shuttle1)
440   (Constraint Starts(s1,task))
441   (Sub s2 deposit_samples(?shuttle ?lander))
442   (Values ?lander lander1)
443   (Constraint Finishes(s2,task))
444   (Ordering s1 s2)
445   (Constraint Before(s1,s2))
446  )
```

Listing D.3: Vollständige Domänenbeschreibung des Weltraumszenarios aus Abschnitt 6.3.

```
(Problem

(ArgumentSymbols rover1 shuttle1 baseCamp1 baseCamp2 lander1
  b1 b2 b3 landingSite1 b4 b5 b6 b7
  sampleContainer1 sampleContainer2 sampleContainer3
  sampleContainer4 sampleContainer5 sampleContainer6
  payload1 batteryPayload1 batteryPayload2
  filled empty charged discharged)
(Instances BaseCamp baseCamp1 baseCamp2)
(Instances SampleContainer sampleContainer1 sampleContainer2
  sampleContainer3 sampleContainer4 sampleContainer5)
(Instances Rover rover1)
(Instances Shuttle shuttle1)
(Instances Lander lander1)

(Fluent f0 RobotAt(shuttle1 landingSite1))
(Constraint Release[0,0](f0))
(Fluent f1 RobotAt(rover1 landingSite1))
(Constraint Release[0,0](f1))
(Fluent f2 Attached(baseCamp1 rover1))
(Constraint Release[0,0](f2))
(Fluent f3 ContainerAt(sampleContainer1 rover1 empty))
(Constraint Release[0,0](f3))
(Fluent f4 ContainerAt(sampleContainer2 rover1 empty))
(Constraint Release[0,0](f4))
(Fluent f5 ContainerAt(sampleContainer3 rover1 empty))
(Constraint Release[0,0](f5))
(Fluent f51 ContainerAt(sampleContainer4 shuttle1 empty))
(Constraint Release[0,0](f51))
(Fluent f52 ContainerAt(sampleContainer5 shuttle1 empty))
(Constraint Release[0,0](f52))
(Fluent f53 ContainerAt(sampleContainer6 shuttle1 empty))
(Constraint Release[0,0](f53))
(Fluent f6 BatteryAt(batteryPayload1 shuttle1 charged))
(Constraint Release[0,0](f6))
(Fluent f7 BatteryAt(batteryPayload2 rover1 discharged))
(Constraint Release[0,0](f7))
(Fluent f8 Attached(baseCamp2 lander1))
(Constraint Release[0,0](f8))
(Fluent f9 RobotAt(lander1 landingSite1))
(Constraint Release[0,0](f9))

(Task t1 deploy_basecamp(rover1 b1))
(Constraint Release[0,1](t1))

(Task t2 take_samples(rover1 b1))
(Constraint Before(t1,t2)) (Ordering t1 t2)

(Task t3 take_samples(rover1 b2))
```

```
50 (Constraint Before(t2,t3)) (Ordering t2 t3)
51
52 (Task t4 rendezvous(rover1 shuttle1))
53 (Constraint Before(t3,t4)) (Ordering t3 t4)
54
55 (Task t5 get_basecamp(rover1 baseCamp2))
56 (Constraint Before(t4,t5)) (Ordering t4 t5)
57
58 (Task t6 deposit_samples(shuttle1 lander1))
59 (Constraint Before(t4,t6)) (Ordering t5 t6)
60
61 (Task t7 take_samples(rover1 b3))
62 (Constraint Before(t5,t7)) (Ordering t6 t7)
63
64 (Task t8 take_samples(rover1 b4))
65 (Constraint Before(t7,t8)) (Ordering t7 t8)
66
67 (Task t9 deploy_basecamp(rover1 b6))
68 (Constraint Before(t8,t9)) (Ordering t8 t9)
69
70 (Task t10 rendezvous(rover1 shuttle1))
71 (Constraint Before(t8,t10)) (Ordering t9 t10)
72
73 (Task t11 deposit_samples(shuttle1 lander1))
74 (Constraint Before(t10,t11)) (Ordering t10 t11)
75
76 (Task t12 take_samples(rover1 b6))
77 (Constraint Before(t10,t12)) (Ordering t11 t12)
78
79 (Task t13 take_samples(rover1 b7))
80 (Constraint Before(t12,t13)) (Ordering t12 t13)
81 )
```

Listing D.4: Beispiel eines Planungsproblems des Weltraumszenarios aus Abschnitt 6.3.

Literaturverzeichnis

[1] Alami, Rachid; Chatila, Raja; Fleury, Sara; Ghallab, Malik; Ingrand, Félix: An architecture for autonomy. In: *The International Journal of Robotics Research* 17 (1998), Nr. 4, S. 315–337

[2] Allen, James F.: Maintaining knowledge about temporal intervals. In: *Communications of the ACM* 26 (1983), Nr. 11, S. 832–843

[3] Awaad, Iman; Kraetzschmar, Gerhard K.; Hertzberg, Joachim: Finding Ways to Get the Job Done: An Affordance-Based Approach. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2014

[4] Balbiani, Philippe; Condotta, Jean-Francois; Cerro, Luis F.: A tractable subclass of the block algebra: constraint propagation and preconvex relations. In: *Progress in Artificial Intelligence*. Springer, 1999, S. 75–89

[5] Balbiani, Philippe; Condotta, Jean-Francois; Del Cerro, Luis F.: A new tractable subclass of the rectangle algebra. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)* Bd. 1, 1999

[6] Barreiro, Javier; Boyce, Matthew; Do, Minh; Frank, Jeremy; Iatauro, Michael; Kichkaylo, Tatiana; Morris, Paul; Ong, James; Remolina, Emilio; Smith, Tristan u. a.: EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. In: *Proc. International Competition on Knowl. Eng. for Planning and Scheduling (ICKEPS)*, 2012

[7] Barták, Roman; Salido, Miguel A.; Rossi, Francesca: Constraint satisfaction techniques in planning and scheduling. In: *Journal of Intelligent Manufacturing* 21 (2010), Nr. 1

[8] Bechon, Patrick; Barbier, Magali; Infantes, Guillaume; Lesire, Charles; Vidal, Vincent: HiPOP: Hierarchical Partial-Order Planning. In: *Proc. STAIRS*, 2014

[9] Bechon, Patrick; Barbier, Magali; Lesire, Charles; Infantes, Guillaume; Vidal, Vincent: Using hybrid planning for plan reparation. In: *Proc. European Conference on Mobile Robots (ECMR)*, 2015

[10] Beetz, Michael; Mösenlechner, Lorenz; Tenorth, Moritz: CRAM?A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2010

[11] Beetz, Michael; Tenorth, Moritz; Winkler, Jan: OPEN-EASE – A Knowledge Processing Service for Robots and Robotics/AI Researchers. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2015

[12] Behnke, Gregor; Bercher, Pascal; Biundo, Susanne; Glimm, Birte; Ponomaryov, Denis; Schiller, Marvin: Integrating ontologies and planning for cognitive systems. In: *Proc. 28th Int. Workshop on Description Logics (DL)*, 2015

[13] Ben-Kiki, Oren; Evans, Clark; Ingerson, Brian: *YAML Ain't Markup Language (YAML (tm)) Version 1.2.* 2009. – `http://www.yaml.org/spec/1.2/spec.pdf`; Zuletzt besucht am 24.01.2016

[14] Bidot, Julien; Schattenberg, Bernd; Biundo, Susanne: Plan repair in hybrid planning. In: *KI 2008: Advances in Artificial Intelligence.* Springer, 2008, S. 169–176

[15] Bit-Monnot, Arthur; Smith, David E.; Do, Minh: Delete-free Reachability Analysis for Temporal and Hierarchical Planning. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2016

[16] Biundo, Susanne; Schattenberg, Bernd: From Abstract Crisis to Concrete Relief ? A Preliminary Report on Combining State Abstraction and HTN Planning. In: *Proc. European Conference on Planning (ECP-01)*, 2001

[17] Blum, Avrim L.; Furst, Merrick L.: Fast planning through planning graph analysis. In: *Artificial Intelligence* 90 (1997), Nr. 1, S. 281–300

[18] Bohren, Jonathan: *smach - ROS Wiki.* 2016. – `http://wiki.ros.org/smach`; Zuletzt besucht am 04.02.2016

[19] Bohren, Jonathan; Cousins, Steve: The SMACH high-level executive [ROS News]. In: *Robotics & Automation Magazine, IEEE* 17 (2010), Nr. 4, S. 18–20

[20] Bohren, Jonathan; Rusu, Radu B.; Jones, E. G.; Marder-Eppstein, Eitan; Pantofaru, Caroline; Wise, Melonee; Mösenlechner, Lorenz; Meeussen, Wim; Holzer, Stefan: Towards autonomous robotic butlers: Lessons learned with the PR2. In: *Proc. International Conference on Robotics and Automation (ICRA)*, IEEE, 2011

[21] Bollini, Mario; Barry, Jennifer; Rus, Daniela: BakeBot: Baking Cookies with the PR2. In: *Proc. The PR2 workshop: results, challenges and lessons learned in advancing robots with a common platform, IROS*, 2011

[22] Bonet, Blai; Geffner, Héctor: Planning as heuristic search. In: *Artificial Intelligence* 129 (2001), S. 5–33

[23] Bouguerra, Abdelbaki; Karlsson, Lars; Saffiotti, Alessandro: Monitoring the execution of robot plans using semantic knowledge. In: *Robotics and Autonomous Systems* 56 (2008), Nr. 11, S. 942–954

[24] Brenner, Michael; Nebel, Bernhard: Continual planning and acting in dynamic multiagent environments. In: *Autonomous Agents and Multi-Agent Systems* 19 (2009), Nr. 3, S. 297–331

[25] Brooks, Rodney A.: A robust layered control system for a mobile robot. In: *IEEE Journal of Robotics and Automation* 2 (1986), Nr. 1, S. 14–23

[26] Brooks, Rodney A.: Intelligence Without Reason. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1991

[27] Castillo, Luis A.; Fernández-Olivares, Juan; Garcia-Perez, Oscar; Palao, Francisco: Efficiently Handling Temporal Knowledge in an HTN Planner. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2006

[28] Cesta, Amadeo; Oddi, Angelo; Smith, Stephen F.: A Constraint-based Method for Project Scheduling with Time Windows. In: *Journal of Heuristics* 8 (2002), January, Nr. 1, S. 109–136

[29] Cesta, Amedeo; Fratini, Simone: The timeline representation framework as a planning and scheduling software development environment. In: *Proc. 27th Workshop of the UK Planning and Scheduling SIG*, 2008

[30] Chien, Steve; Rabideau, G.; Knight, R.; Sherwood, Robert; Engelhardt, Barbara; Mutz, Darren; Estlin, T.; Smith, Benjamin; Fisher, F.; Barrett, T.; others: ASPEN – Automated planning and scheduling for space mission operations. In: *Proc. Space Ops*, 2000

[31] Cialdea Mayer, Marta; Orlandini, Andrea; Umbrico, Alessandro: A formal account of planning with flexible timelines. In: *Proc. 21st International Symposium on Temporal Representation and Reasoning (TIME)*, IEEE, 2014

[32] Clementini, Eliseo; Di Felice, Paolino; Oosterom, Peter van: A small set of formal topological relationships suitable for end-user interaction. In: *Proc. Advances in Spatial Databases*, Springer, 1993

[33] Coradeschi, Silvia; Saffiotti, Alessandro: An introduction to the anchoring problem. In: *Robotics and Autonomous Systems* 43 (2003), S. 85–96

[34] Currie, Ken; Tate, Austin: O-Plan: The Open Planning Architecture. In: *Artificial Intelligence* 52 (1991), Nr. 1, S. 49–86

[35] D'Andrea, Raffaello: Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. In: *IEEE Transactions on Automation Science and Engineering* 4 (2012), Nr. 9, S. 638–639

[36] Dechter, Rina: *Constraint Processing*. Morgan Kaufmann, 2003

[37] Dechter, Rina; Meiri, Itay; Pearl, Judea: Temporal Constraint Networks. In: *Artif. Intell.* 49 (1991), Nr. 1-3, S. 61–95

[38] Deeken, Henning; Wiemann, Thomas; Lingemann, Kai; Hertzberg, Joachim: SEMAP-a semantic environment mapping framework. In: *Proc. European Conference on Mobile Robots (ECMR)*, IEEE, 2015

[39] desJardins, Marie; Durfee, Edmund H.; Ortiz Jr, Charles L.; Wolverton, Michael J.: A survey of research in distributed, continual planning. In: *AI Magazine* 20 (1999), Nr. 4, S. 13–22

[40] Diaz, Daniel; Cesta, Amedeo; Oddi, Angelo; Rasconi, Riccardo; R-Moreno, Maria D.: Efficient energy management for autonomous control in rover missions. In: *IEEE Computational Intelligence Magazine* 8 (2013), Nr. 4, S. 12–24. `http://dx.doi.org/10.1109/MCI.2013.2279558`. – DOI 10.1109/MCI.2013.2279558

[41] Dijkstra, Edsger W.: A note on two problems in connexion with graphs. In: *Numerische mathematik* 1 (1959), Nr. 1, S. 269–271

[42] Di Rocco, Maurizio; Pecora, Federico; Saffiotti, Alessandro: When robots are late: Configuration planning for multiple robots with dynamic goals. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2013

[43] Doherty, Patrick; Kvarnström, Jonas; Heintz, Fredrik: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. In: *Autonomous Agents and Multi-Agent Systems* 19 (2009), Nr. 3, S. 332–377

[44] Dornhege, Christian; Eyerich, Patrick; Keller, Thomas; Trüg, Sebastian; Brenner, Michael; Nebel, Bernhard: Semantic attachments for domain-independent planning systems. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2009

[45] Dornhege, Christian; Gissler, Marc; Teschner, Matthias; Nebel, Bernhard: Integrating symbolic and geometric planning for mobile manipulation. In: *Proc. IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR)*, 2009

[46] Dornhege, Christian; Hertle, Andreas: Integrated Symbolic Planning in the Tidyup-Robot Project. In: *Proc. of AAAI Spring Symposium: Designing Intelligent Robots*, 2013

[47] Dvorak, Filip; Arthur, Bit-Monnot; Ingrand, Felix; Ghallab, Malik: A Flexible ANML Actor and Planner in robotics. In: *Proc. Planning and Robotics Workshop at ICAPS*, 2014

[48] Einig, Lasse; Klimentjew, Denis; Rockel, Sebastian; Zhang, Liwei; Zhang, Jianwei: Parallel plan execution and re-planning on a mobile robot using state machines with htn planning systems. In: *Proc. International Conference on Robotics and Biomimetics (ROBIO)* IEEE, 2013

[49] Elkawkagy, Mohamed; Bercher, Pascal; Schattenberg, Bernd; Biundo, Susanne: Improving Hierarchical Planning Performance by the Use of Landmarks. In: *Proc. National Conference on Artificial Intelligence (AAAI)*, 2012

[50] Erman, Lee D.; Hayes-Roth, Frederick; Lesser, Victor R.; Reddy, D. R.: The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. In: *ACM Computing Surveys (CSUR)* 12 (1980), Nr. 2, S. 213–253

[51] Erol, Kutluhan; Hendler, James; Nau, Dana S.: HTN Planning: Complexity and Expressivity. In: *Proc. National Conference on Artificial Intelligence (AAAI)*, 1994

[52] Erol, Kutluhan; Hendler, James; Nau, Dana S.: UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In: *Proc. International Conference on AI Planning Systems (AIPS)* Bd. 94, 1994

[53] Estlin, Tara A.; Chien, Steve A.; Wang, Xuemei: An argument for a hybrid HTN/operator-based approach to planning. In: *Proc. European Conference on Planning*, Springer, 1997

[54] Fikes, Richard E.; Hart, Peter E.; Nilsson, Nils J.: Learning and executing generalized robot plans. In: *Artificial intelligence* 3 (1972), S. 251–288

[55] Fikes, Richard E.; Nilsson, Nils J.: STRIPS: A new approach to the application of theorem proving to problem solving. In: *Artificial intelligence* 2 (1971), Nr. 3-4, S. 189–208

[56] Firby, R. J.: An investigation into reactive planning in complex domains. In: *Proc. National Conference on Artificial Intelligence (AAAI)* Bd. 87, 1987

[57] Fox, Dieter; Burgard, Wolfram; Thrun, Sebastian: The dynamic window approach to collision avoidance. In: *IEEE Robotics Automation Magazine* 4 (1997), Nr. 1, S. 23–33

[58] Frank, Jeremy; Jónsson, Ari K.: Constraint-based attribute and interval planning. In: *Constraints* 8 (2003), Nr. 4, S. 339–364

[59] Fratini, Simone; Pecora, Federico; Cesta, Amadeo: Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. In: *Archives of Control Sciences* 18 (2008), Nr. 2, S. 231–271

[60] Georgeff, Michael P.; Ingrand, Francois F.: Decision-making in an Embedded Reasoning System. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1989

[61] Gerevini, Alfonso; Long, Derek: Preferences and soft constraints in PDDL3. In: *Proc. ICAPS workshop on planning with preferences and soft constraints*, 2006

[62] Ghallab, Malik; Laruelle, Herve: Representation and Control in IxTeT, a Temporal Planner. In: *Proc. International Conference on AI Planning Systems (AIPS)*, 1994

[63] Ghallab, Malik; Nau, Dana; Traverso, Paolo: *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004. – ISBN 1–55860–856–7

[64] Ghallab, Malik; Nau, Dana; Traverso, Paolo: *Automated Planning and Acting*. New York, NY : Cambridge University Press, 2016. – ISBN 978–1–107–03727–4

[65] Gil, Yolanda: Description logics and planning. In: *AI Magazine* 26 (2005), Nr. 2

[66] Goldman, Robert P.: Durative Planning in HTNs. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2006

[67] Goldman, Robert P.; Nau, Dana S.; Kutur, Ugur: Documentation for SHOP2. (2012), September

[68] Guizzo, Eric: Three engineers, hundreds of robots, one warehouse. In: *IEEE spectrum* 45 (2008), Nr. 7, S. 26–34

[69] Hanheide, Marc; Göbelbecker, Moritz; Horn, Graham S.; Pronobis, Andrzej; Sjöö, Kristoffer; Aydemir, Alper; Jensfelt, Patric; Gretton, Charles; Dearden, Richard; Janicek, Miroslav; others: Robot task planning and explanation in open and uncertain worlds. In: *Artificial Intelligence* (2015)

[70] Hartanto, Ronny: *A hybrid deliberative layer for robotic agents: fusing DL reasoning with HTN planning in autonomous robots*. Bd. 6798. Springer Science & Business Media, 2011

[71] Hawes, Nick; Wyatt, Jeremy: Engineering intelligent information-processing systems with CAST. In: *Advanced Engineering Informatics* 24 (2010), Nr. 1, S. 27–39

[72] Heintz, Fredrik; Kvarnström, Jonas; Doherty, Patrick: Bridging the sense-reasoning gap: DyKnow-Stream-based middleware for knowledge processing. In: *Advanced Engineering Informatics* 24 (2010), Nr. 1, S. 14–26

[73] Helmert, Malte: The Fast Downward Planning System. In: *J. Artif. Intell. Res.(JAIR)* 26 (2006), S. 191–246

[74] Hertzberg, Joachim: *Planen: Einführung in die Planerstellungsmethoden der künstlichen Intelligenz*. Mannheim [u.a.] : BI-Wissenschaftsverlag, 1989 (Reihe Informatik ; 65)

[75] Hertzberg, Joachim; Lingemann, Kai; Nüchter, Andreas: *Mobile Roboter: Eine Einführung aus Sicht der Informatik*. Berlin : Springer Vieweg, 2012

[76] Hertzberg, Joachim; Zhang, Jianwei; Zhang, Liwei; Rockel, Sebastian; Neumann, Bernd; Lehmann, Jos; Dubba, Krishna S. R.; Cohn, Anthony G.; Saffiotti, Alessandro; Pecora, Federico; Mansouri, Masoumeh; Konecny, Stefan; Günther, Martin; Stock, Sebastian; Seabra Lopes, Luis; Oliveira, Miguel; Lim, Gi H.; Kasaei, Hamidreza; Mokhtari, Vahid; Hotz, Lothar; Bohlken, Wilfried: The race project. In: *KI-Künstliche Intelligenz* 28 (2014), Nr. 4, S. 297–304

[77] Hoffmann, Jörg; Nebel, Bernhard: The FF planning system: Fast plan generation through heuristic search. In: *Journal of Artificial Intelligence Research* (2001), S. 253–302

[78] Hoffmann, Jörg; Brafman, Ronen: Contingent planning via heuristic forward search with implicit belief states. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)* Bd. 2005, 2005

[79] Hoffmann, Jörg; Brafman, Ronen: Conformant planning via heuristic forward search: A new approach. In: *Artificial Intelligence* 170 (2006), Nr. 6, S. 507–541

[80] Hofmann, Till; Niemueller, Tim; Classen, Jens; Lakemeyer, Gerhard: Continual Planning in Golog. In: *Proc. National Conference on Artificial Intelligence (AAAI)*, 2016

[81] Hsiao, Kaijen; Kaelbling, Leslie P.; Lozano-Perez, Tomas: Grasping POMDPs. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2007

[82] Jones, Joseph L.: Robots at the tipping point: the road to iRobot Roomba. In: *IEEE Robotics & Automation Magazine* 13 (2006), Nr. 1, S. 76–78

[83] Jonsson, Ari K.; H., Morris P.; Muscettola, Nicola; Rajan, Kanna; Smith, Ben: Planning in Interplanetary Space: Theory and Practice. In: *Proc. International Conference on AI Planning Systems (AIPS)*, 2000

[84] Kaelbling, Leslie P.; Littman, Michael L.; Cassandra, Anthony R.: Planning and acting in partially observable stochastic domains. In: *Artificial Intelligence* 101 (1998), S. 99–134

[85] Kaelbling, Leslie P.; Lozano-Pérez, Tomás: Hierarchical Planning in the Now. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2011

[86] Kaelbling, Leslie P.; Lozano-Pérez, Tomás: Integrated task and motion planning in belief space. In: *International Journal of Robotics Research* 32 (2013), Nr. 9-10

[87] Kambhampati, Subbarao; Mali, Amol; Srivastava, Biplav: Hybrid planning for partially hierarchical domains. In: *Proc. National Conference on Artificial Intelligence (AAAI)*, 1998

[88] Koenig, Nathan; Howard, Andrew: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *Proc. International Conference on Robotics and Systems (IROS)* Bd. 3, 2004

[89] Konečný Š.; Stock, S.; Pecora, F.; Saffiotti, A.: Planning Domain + Execution Semantics: a Way Towards Robust Execution? In: *Proc. AAAI Spring Symposium: Qualitative Representations for Robots*, 2014

[90] Kunze, Lars; Dolha, Mihai E.; Beetz, Michael: Logic programming with simulation-based temporal projection for everyday robot object manipulation. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2011

[91] Kurniawati, Hanna; Hsu, David; Lee, Wee S.: SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In: *Proc. Robotics: Science and Systems*, 2008

[92] Köckemann, Uwe: *Constraint-based Methods for Human-aware Planning*, Örebro University, School of Science and Technology, Örebro University, Sweden, Diss., 2016

[93] Lagriffoul, Fabien; Dimitrov, Dimitar; Saffiotti, Alessandro; Karlsson, Lars: Constraint Propagation on Interval Bounds for Dealing with Geometric Backtracking. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2012

[94] LaValle, S. M.: *Planning Algorithms*. Cambridge, U.K. : Cambridge University Press, 2006

[95] Lehmann, Jos; Neumann, Bernd; von Riegen, Stephanie; Hotz, Lothar; Mansouri, Masoumeh; Pecora, Federico; Stock, Sebastian: Deliverable D1.3 – Hand-coded Non-hybrid Knowledge Contents / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2012. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=153`. Hamburg, Germany, 2012. – Public Deliverable

[96] Levesque, Hector J.; Reiter, Raymond; Lesperance, Yves; Lin, Fangzhen; Scherl, Richard B.: GOLOG: A logic programming language for dynamic domains. In: *The Journal of Logic Programming* 31 (1997), Nr. 1, S. 59–83

[97] Ligozat, Gerard: A New Proof of Tractability for ORD-Horn Relations. In: *Proc. AAAI Workshop on Spatial and Temporal Reasoning*, 1996

[98] Mansouri, Masoumeh: *A Constraint-Based Approach for Hybrid Reasoning in Robotics*, Örebro University, School of Science and Technology, Örebro University, Sweden, Diss., 2016

[99] Mansouri, Masoumeh; Andreasson, Henrik; Pecora, Federico: Towards Hybrid Reasoning for Automated Industrial Fleet Management. In: *Proc. IJCAI Workshop on Hybrid Reasoning*, 2015

[100] Mansouri, Masoumeh; Andreasson, Henrik; Pecora, Federico: Hybrid reasoning for multi-robot drill planning in open-pit mines. In: *Acta Polytechnica* 56 (2016), Nr. 1, S. 47–56. http://dx.doi.org/10.14311/APP.2016.56.0047. – DOI 10.14311/APP.2016.56.0047

[101] Mansouri, Masoumeh; Pecora, Federico: A Representation for Spatial Reasoning in Robotic Planning. In: *Proc. IROS Workshop on AI-based Robotics*, 2013

[102] Mansouri, Masoumeh; Pecora, Federico: More Knowledge on the Table: Planning with Space, Time and Resources for Robots. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2014

[103] Mansouri, Masoumeh; Pecora, Federico: Deliverable D1.6 – Final report on the integrated infrastructure for causal, spatial and temporal knowledge / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2015. http://race.informatik.uni-hamburg.de/wordpress/?page_id=159. Hamburg, Germany, 2015. – Public Deliverable

[104] Mansouri, Masoumeh; Pecora, Federico: A robot sets a table: a case for hybrid reasoning with different types of knowledge. In: *Journal of experimental and theoretical artificial intelligence* (2016), S. 1–21

[105] Marder-Eppstein, Eitan; Berger, Eric; Foote, Tully; Gerkey, Brian; Konolige, Kurt: The Office Marathon: Robust Navigation in an Indoor Office Environment. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2010

[106] Marder-Eppstein, Eitan; Lu, David V.; Ferguson, Michael: *navigation - ROS Wiki*. 2016. – http://www.ros.org/wiki/navigation; Zuletzt besucht am 30.01.2016

[107] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D.: PDDL - The Planning Domain Definition Language / Yale Center for Computational Vision and Control,. 1998 (TR-98-003). – Forschungsbericht

[108] McGann, Conor; Py, Frederic; Rajan, Kanna; Ryan, John P.; Henthorn, Richard: Adaptive Control for Autonomous Underwater Vehicles. In: *Proc. National Conference on Artificial Intelligence (AAAI)*, 2008

[109] McGann, Conor; Py, Frederic; Rajan, Kanna; Thomas, Hans; Henthorn, Richard; McEwen, Rob: A deliberative architecture for AUV control. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2008

[110] Meeussen, Wim; Lindzey, Laura: *pr2_tuck_arms_action - ROS Wiki*. 2016. – `http://wiki.ros.org/pr2_tuck_arms_action`; Zuletzt besucht am 31.01.2016

[111] Meiri, Itay: Combining qualitative and quantitative constraints in temporal reasoning. In: *Artificial Intelligence* 87 (1996), Nr. 1, S. 343–385

[112] Mokhtari, Vahid; Lopes, Luís S.; Pinho, Armando J.: Experience-Based Planning Domains: an Integrated Learning and Deliberation Approach for Intelligent Robots. In: *Journal of Intelligent & Robotic Systems* (2016), S. 1–21. `http://dx.doi.org/10.1007/s10846-016-0371-y`. – DOI 10.1007/s10846–016–0371–y

[113] Muscettola, Nicola: HSTS: Integrating Planning and Scheduling / Robotics Institute. Pittsburgh, PA, 1993 (CMU-RI-TR-93-05). – Forschungsbericht

[114] Nareyek, Alexander; Freuder, Eugene C.; Fourer, Robert; Giunchiglia, Enrico; Goldman, Robert P.; Kautz, Henry; Rintanen, Jussi; Tate, Austin: Constraints and AI planning. In: *IEEE Intelligent Systems* 20 (2005), Nr. 2

[115] Nau, Dana; Au, Tsz-Chiu; Ilghami, Okhtay; Kuter, Ugur; Munoz-Avila, Hector; Murdock, J. W.; Wu, Dan; Yaman, Fusun: Applications of SHOP and SHOP2. In: *IEEE Intelligent Systems* 20 (2005), März, Nr. 2, S. 34–41. – ISSN 1541–1672

[116] Nau, Dana; Cao, Yue; Lotem, Amnon; Munoz-Avila, Hector: SHOP: Simple hierarchical ordered planner. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)* Bd. 2, Morgan Kaufmann Publishers Inc., 1999

[117] Nau, Dana S.; Au, Tsz-Chiu; Ilghami, Okhtay; Kuter, Ugur; Murdock, J W.; Wu, Dan; Yaman, Fusun: SHOP2: An HTN planning system. In: *J. Artificial Intell. Research* 20 (2003), S. 379–404

[118] Neumann, Bernd; Hotz, Lothar; Rost, Pascal; Lehmann, Jos: A robot waiter learning from experiences. In: *Proc. International Workshop on Machine Learning and Data Mining in Pattern Recognition*, 2014

[119] Neumann, Bernd; Lehmann, Jos; Konecny, Stefan; Pecora, Federico; Stock, Sebastian; Hertzberg, Joachim: Deliverable D4.1 – Basic Versions of Prediction, Expectation about Own Actions, and Plan Execution Monitoring / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2013. http://race.informatik.uni-hamburg.de/wordpress/?page_id=181. Hamburg, Germany, 2013. – Public Deliverable

[120] Nguyen, Hai; Ciocarlie, Matei; Hsiao, Kaijen; Kemp, Charles C.: ROS Commander (ROSCo): Behavior Creation for Home Robots. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2013

[121] Nii, H. P.: The blackboard model of problem solving and the evolution of blackboard architectures. In: *AI magazine* 7 (1986), Nr. 2, S. 38

[122] Nilsson, Nils J.: Shakey the robot. 1984 (323). – Forschungsbericht

[123] Nilsson, Nils J.: *The quest for artificial intelligence*. Cambridge University Press, 2009

[124] Nökel, Klaus: *Convex relations between time intervals*. Springer, 1989

[125] Off, Dominik; Zhang, Jianwei: Continual HTN Planning and Acting in Open-ended Domains - Considering Knowledge Acquisition Opportunities. In: *Proc. International Conference on Agents and Artificial Intelligence (ICAART)*, 2012

[126] Okada, Kei; Kakiuchi, Yohei; Azuma, Haseru; Mikita, Hiroyuki; Murase, Kazuto; Inaba, Masayuki: Task Compiler: Transferring High-level Task Description to Behavior State Machine with Failure Recovery Mechanism. In: *Proc. ICRA Workshop on Combining Task and Motion Planning*, 2013

[127] Oliveira, Miguel; Lim, Gi H.; Seabra Lopes, Luis; Hamidreza Kasaei, S.; Tome, Ana M.; Chauhan, Aneesh: A perceptual memory system for grounding semantic representations in intelligent service robots. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2014

[128] Ong, Sylvie C.; Png, Shao W.; Hsu, David; Lee, Wee S.: Planning under uncertainty for robotic tasks with mixed observability. In: *International Journal of Robotics Research* 29 (2010), Nr. 8, S. 1053–1068

[129] Pecora, Federico: Is Model-Based Robot Programming a Mirage? A Brief Survey of AI Reasoning in Robotics. In: *KI-Künstliche Intelligenz* 28 (2014), Nr. 4, S. 255–261

[130] Pecora, Federico: *The Meta-CSP Framework*. 2016. – http://metacsp.org; Stand 14.09.2016

[131] Pecora, Federico; Cirillo, Marcello; Dell'Osa, Francesca; Ullberg, Jonas; Saffiotti, Alessandro: A constraint-based approach for proactive, context-aware human support. In: *Journal of Ambient Intelligence and Smart Environments* 4 (2012), Nr. 4, S. 347–367

[132] Pecora, Federico; Mansouri, Masoumeh; Rockel, Sebastian; Stock, Sebastian; Günther, Martin; Seabra Lopes, Luis; Tome, Ana M.; Neumann, Bernd; von Riegen, Stephanie; Hotz, Lothar; Dubba, Sandeep: Deliverable D1.1 – Relevant Knowledge Representation Tools and Formalisms / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2012. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=147`. Hamburg, Germany, 2012. – Public Deliverable

[133] Pecora, Federico; Mansouri, Masoumeh; von Riegen, Stephanie; Hotz, Lothar: Deliverable D1.2 – Software Framework for Hybrid Knowledge Representation and Reasoning / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2013. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=150`. Hamburg, Germany, 2013. – Public Deliverable

[134] Pednault, Edwin: Formulating multiagent, dynamic-world problems in the classical planning framework. In: *Proc. Timberline Oregon Workshop on reasoning about actions and plans* (1986), S. 47–82

[135] Peter Bonasso, R.; James Firby, R.; Gat, Erann; Kortenkamp, David; Miller, David P.; Slack, Mark G.: Experiences with an architecture for intelligent, reactive agents. In: *Journal of Experimental & Theoretical Artificial Intelligence* 9 (1997), Nr. 2-3, S. 237–256

[136] Pettersson, Ola: Execution monitoring in robotics: A survey. In: *Robotics and Autonomous Systems* 53 (2005), Nr. 2, S. 73–88

[137] Pineau, Joelle; Montemerlo, Michael; Pollack, Martha; Roy, Nicholas; Thrun, Sebastian: Towards robotic assistants in nursing homes: Challenges and results. In: *Robotics and Autonomous Systems* 42 (2003), Nr. 3, S. 271–281

[138] Quigley, Morgan; Conley, Ken; Gerkey, Brian P.; Faust, Josh; Foote, Tully; Leibs, Jeremy; Wheeler, Rob; Ng, Andrew Y.: ROS: an open-source Robot Operating System. In: *Proc. of ICRA Workshop on Open Source Software*, 2009

[139] RACE Consortium: *Robustness by Autonomous Competence Enhancement - Annex I - Description of Work*. 2011. – Grant agreement for Collaborative project, FP7-ICT-2011-7-287752

[140] Randell, David A.; Cui, Zhan; Cohn, Anthony G.: A Spatial Logic based on Regions and Connection. In: *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 1992

[141] Rockel, Sebastian: *Probabilistic Online Prediction of Robot Action Results based on Physics Simulation*, Universität Hamburg, Hamburg, Deutschland, Diss., 2016

[142] Rockel, Sebastian; Klimentjew, Denis; Zhang, Liwei; Zhang, Jianwei: An hyperreality imagination based reasoning and evaluation system (HIRES). In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2014

[143] Rockel, Sebastian; Konecny, Stefan; Stock, Sebastian; Hertzberg, Joachim; Pecora, Federico; Zhang, Jianwei: Integrating physics-based Prediction with Semantic Plan Execution Monitoring. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2015

[144] Rockel, Sebastian; Neumann, Bernd; Zhang, Jianwei; Dubba, Krishna S. R.; Cohn, Anthony G.; Konečný, Štefan; Mansouri, Masoumeh; Pecora, Federico; Saffiotti, Alessandro; Günther, Martin; Stock, Sebastian; Hertzberg, Joachim; Tomé, Ana M.; Pinho, Amando J.; Seabra Lopes, Luis; Riegen, Stephanie von; Hotz, Lothar: An Ontology-based Multi-level Robot Architecture for Learning from Experiences. In: *Proc. AAAI Spring Symposium: Designing Intelligent Robots: Reintegrating AI II*, 2013

[145] Russell, Stuart; Norvig, Peter: *Artificial intelligence : a modern approach*. 3rd revised edition. Upper Saddle River, NJ : Prentice Hall, 2010. – ISBN 978–0–13–207148–2

[146] Röhr, Thomas; Kirchner, Frank: Spatio-Temporal Planning for a Reconfigurable Multi-Robot System. In: *Proc. 4th Workshop on Planning and Robotics (PlanRob) at ICAPS*. London, United Kingdom, 2016

[147] Sacerdoti, Earl D.: Planning in a hierarchy of abstraction spaces. In: *Artificial intelligence* 5 (1974), Nr. 2, S. 115–135

[148] Sacerdoti, Earl D.: The nonlinear nature of plans. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)* Morgan Kaufmann Publishers Inc., 1975

[149] Schattenberg, Bernd: *Hybrid Planning And Scheduling*, Ulm University, Institute of Artificial Intelligence, Diss., 2009. – URN: urn:nbn:de:bsz:289-vts-68953

[150] Scheuren, Stephan; Stiene, Stefan; Hartanto, Ronny; Hertzberg, Joachim; Reinecke, Max: Spatio-temporally constrained planning for cooperative vehicles in a harvesting scenario. In: *KI-Künstliche Intelligenz* 27 (2013), Nr. 4, S. 341–346

[151] Siciliano, Bruno (Hrsg.); Khatib, Oussama (Hrsg.): *Springer Handbook of Robotics*. 2008 edition. Berlin : Springer, 2008. – ISBN 978–3–540–23957–4

[152] Silva, Lavindra de; Pandey, Amit K.; Alami, Rachid: An interface for interleaved symbolic-geometric planning and backtracking. In: *Proc. International Conference on Robotics and Systems (IROS)*, 2013

[153] Simmons, Reid; Apfelbaum, David: A task description language for robot control. In: *Proc. International Conference on Robotics and Systems (IROS)* Bd. 3, 1998

[154] Sirin, Evren: *Combining description logic reasoning with AI planning for composition of web services*, Department of Computer Science, University of Maryland, Diss., 2006

[155] Smith, David E.; Cushing, William: The ANML Language. In: *Proc. Scheduling and Planning Applications Workshop at ICAPS*, 2008

[156] Sohrabi, Shirin; McIlraith, Sheila A.; Baier, Jorge A.: HTN Planning with Preferences. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2009

[157] Sonsalla, Roland U.; Cordes, Florian; Christensen, Leif; Planthaber, Steffen; Albiez, Jan; Scholz, Ingo; Kirchner, Frank: Towards a Heterogeneous Modular Robotic Team in a Logistics Chain for Extended Extraterrestial Exploration. In: *Proc. 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2014

[158] Stock, Sebastian; Bauer, Jan; Hertzberg, Joachim; Lehmann, Jos; Rockel, Sebastian; Konecny, Stefan; Mansouri, Masoumeh; Pecora, Federio: Deliverable D4.3 – Concept and basic implementation of experience-based attention and semantic planning and execution / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2014. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=186`. Hamburg, Germany, 2014. – Public Deliverable

[159] Stock, Sebastian; Günther, Martin; Hertzberg, Joachim: Generating and Executing Hierarchical Mobile Manipulation Plans. In: *Proc. 41st International Symposium on Robotics (ISR)/Robotik*, 2014

[160] Stock, Sebastian; Günther, Martin; Hertzberg, Joachim; Lehmann, Jos; Rockel, Sebastian; Konecny, Stefan; Mansouri, Masoumeh; Pecora, Federio; Mokhtari, Vahid; Seabra Lopes, Luis; Pinho, Armando: Deliverable D4.4 – Consolidated implementation of goal-directed action using conceptualised experiences / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2015. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=188`. Hamburg, Germany, 2015. – Public Deliverable

[161] Stock, Sebastian; Hertzberg, Joachim; Lehmann, Jos; Neumann, Bernd; Bohlken, Wilfried; Hotz, Lothar; Rockel, Sebastian; Konecny, Stefan: Deliverable D4.2 – Consolidated Versions of Prediction, Expectation and Plan Execution / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2013. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=184`. Hamburg, Germany, 2013. – Public Deliverable

[162] Stock, Sebastian; Mansouri, Masoumeh; Pecora, Federico; Hertzberg, Joachim: Hierarchical hybrid planning in a mobile service robot. In: *Proc. 38th German Conference on Artificial Intelligence (KI2015)*, 2015

[163] Stock, Sebastian; Mansouri, Masoumeh; Pecora, Federico; Hertzberg, Joachim: Online Task Merging with a Hierarchica Hybrid Task Planner for Mobile Service Robots. In: *Proc. International Conference on Robotics and Systems (IROS)*. Hamburg, 2015

[164] Tate, Austin: Generating Project Networks. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1977

[165] Tate, Austin; Drabble, Brian; Kirby, Richard: O-Plan2: an Open Architecture for Command, Planning and Control. In: *Intelligent Scheduling*. Morgan Kaufmann, 1994, S. 213–239

[166] Tenorth, Moritz; Beetz, Michael: KnowRob: A knowledge processing infrastructure for cognition-enabled robots. In: *The International Journal of Robotics Research* 32 (2013), Nr. 5, S. 566–590

[167] Thrun, Sebastian; Burgard, Wolfgang; Fox, Dieter: *Probabilistic Robotics*. Cambridge, Mass : The MIT Press, 2005. – ISBN 978–0–262–20162–9

[168] Tomic, Stevan; Pecora, Federico; Saffiotti, Alessandro: Too cool for school – adding social constraints in human aware planning. In: *Proc. International Workshop on Cognitive Robotics (CogRob)*, 2014

[169] Umbrico, Alessandro; Orlandini, Andrea; Cialdea Mayer, Marta: Enriching a Temporal Planner with Resources and a Hierarchy-Based Heuristic. In: *AI*IA 2015, Advances in Artificial Intelligence*. Springer International Publishing, 2015, S. 410–423

[170] Verma, Vandi; Estlin, Tara; Jónsson, Ari; Pasareanu, Corina; Simmons, Reid; Tso, Kam: Plan execution interchange language (PLEXIL) for executable plans and command sequences. In: *Proc. 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005

[171] Vilain, Marc B.; Kautz, Henry A.: Constraint Propagation Algorithms for Temporal Reasoning. In: *Aaai* 86 (1986), S. 377–382

[172] von Riegen, Stephanie; Hotz, Lothar; Stock, Sebastian; Pecora, Federico; Sebara Lopes, Luis; Kasaei, Hamidreza: Deliverable D1.4 – Hand-coded Hybrid Knowledge Contents / Das RACE-Projekt (FP7-ICT-2011-7-287752). Version: 2013. `http://race.informatik.uni-hamburg.de/wordpress/?page_id=153`. Hamburg, Germany, 2013. – Public Deliverable

[173] Warfield, Ian; Hogg, Chad; Lee-Urban, Stephen; Munoz-Avila, Héctor: Adaptation of Hierarchical Task Network Plans. In: *Proc. FLAIRS Conference*, 2007

[174] Weser, Martin; Off, Dominik; Zhang, Jianwei: HTN robot planning in partially observable dynamic environments. In: *Proc. International Conference on Robotics and Automation (ICRA)*, 2010

[175] Wilkins, David E.: *Practical planning : extending the classical AI planning paradigm*. San Mateo, CA : Morgan Kaufmann, 1988 (The Morgan Kaufmann series in representation and reasoning). – 205 S S.

[176] Wilkins, David E.: Can AI planners solve practical problems? In: *Computational intelligence* 6 (1990), Nr. 4, S. 232–246

[177] Wisspeintner, Thomas; Van Der Zant, Tijn; Iocchi, Luca; Schiffer, Stefan: RoboCup@ Home: Scientific competition and benchmarking for domestic service robots. In: *Interaction Studies* 10 (2009), Nr. 3, S. 392–426

[178] Wolfe, Jason; Marthi, Bhaskara; Russell, Stuart J.: Combined Task and Motion Planning for Mobile Manipulation. In: *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2010

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Osnabrück, Oktober 2016