Evolutionäre Systemintegration für die Entwicklung interaktiver Roboter

Thorsten Peter Spexard

Dipl.-Inform. Thorsten Peter Spexard
AG Angewandte Informatik
Technische Fakultät
Universität Bielefeld
email: tspexard@techfak.uni-bielefeld.de

Abdruck der genehmigten Dissertation zur Erlangung
des akademischen Grades Doktor-Ingenieur (Dr.-Ing.).
Der Technischen Fakultät der Universität Bielefeld
am 16.03.2010 vorgelegt von Thorsten Peter Spexard,
am 17.05.2010 verteidigt und genehmigt.

Gutachter:

Prof. Dr.-Ing. Franz Kummert, Universität Bielefeld
Dr.-Ing. Jannik Fritsch, Honda Research Institute Europe, Offenbach
Dr.-Ing. Marc Hanheide, University of Birmingham

Prüfungsausschuss:

Prof. Dr. rer. pol. Philipp Cimiano, Universität Bielefeld
Prof. Dr.-Ing. Franz Kummert, Universität Bielefeld
Prof. Dr. rer. nat. Ipke Wachsmuth, Universität Bielefeld
Dr.-Ing. Jannik Fritsch, Honda Research Institute Europe, Offenbach
Dr.-Ing. Marc Hanheide, University of Birmingham

Gedruckt auf alterungsbeständigem Papier nach ISO 9706

Evolutionäre Systemintegration für die Entwicklung interaktiver Roboter

Der Technischen Fakultät der Universität Bielefeld

zur Erlangung des Grades

Doktor-Ingenieur

vorgelegt von

Thorsten Peter Spexard

Bielefeld – 16.03.2010

Danksagung

Es mag vielleicht seltsam erscheinen, doch auch wenn das Schreiben einer Dissertation im Allgemeinen, genauso wie bei der hier vorliegenden Ausfertigung im Speziellen, einen gewissen Arbeitsaufwand darstellt, fällt es mir deutlich schwerer die passenden Worte für diese Danksagung zu finden, als zuvor für die Dissertationsschrift an sich. Der Grund hierfür liegt darin, dass viele Menschen auf ihre jeweils eigene Art zum Gelingen dieser Arbeit beigetragen haben und ich weder durch Formulierung noch durch Reihenfolge die Bedeutung dieser Beträge zueinander ins Verhältnis setzen möchte - oder etwa kann.

Von den vielen Wegbegleitern möchte ich zuerst meinen Eltern Danken, ohne deren stete Unterstützung und Zusprache ich wohl kaum ein Studium, geschweige eine Promotion gewagt hätte. Dennoch hätte ich die Promotion vermutlich nicht begonnen, wenn nicht auch aus fachlicher Richtung während des Diplomstudiums Zuspruch vorhanden gewesen wäre. Hier möchte ich vor allem Dr. Jannik Fritsch, Dr. Axel Haasch sowie Dr. Marcus Kleinehagenbrock danken, die mich durch Ihre erstklassige Betreuung und anhaltende Freundschaft darin bestärkten zu promovieren.

Während der Promotionszeit konnte ich - bedingt durch personelle Veränderungen - nicht nur auf den Rat sowohl eines Doktorvaters und eines praktischen Betreuers zurückgreifen, sondern auf jeweils zwei Sichtweisen. Diese erlaubten mir, ein breites Spektrum an Meinungen zu berücksichtigen. Ich danke Dr. Jannik Fritsch und Professor Gerhard Sagerer dafür, dass sie mich während der ersten Etappe dieser Arbeit begleitet haben, genauso wie Dr. Marc Hanheide und Professor Franz Kummert, welche den restlichen Weg mit mir beschritten. Egal ob fachlich oder persönlich und trotz eines stets ausgefüllten Terminplanes, war immer die Zeit für ein Gespräch vorhanden.

Diese Bereitschaft, sich gegenseitig zu unterstützen und der feste Wille, als Team unabhängig von Fachrichtung oder Status gemeinsam Ziele zu erreichen und dabei neue Maßstäbe zu setzen zeichnete die Kolleginnen und Kollegen aus, mit denen ich zusammen arbeiten durfte. Besonders in der Systemintegration ist man auf Kooperation angewiesen. Mit Euch gab es aber auch in den zeitlich knapperen Phasen von Präsentationen und Paper-Deadlines immer ein gutes Miteinander, ohne welches diese Arbeit kaum so gut gelungen wäre. Es hat Spaß gemacht mit Euch zu arbeiten und ich bin Euch sehr dankbar für diese Erfahrung und dafür, dass ich nicht nur Kollegen sondern auch Freunde gefunden habe.

Umso schwerer fiel es mir, meine Dissertation und somit auch meine Forschungsarbeit in dieser Arbeitsgruppe mit der Verteidigung abzuschließen. Ich danke in diesem Zusammenhang besonders den Gutachtern und Mitgliedern des Prüfungsausschusses Professor Philipp Cimiano, Professor Franz Kummert, Professor Ipke Wachsmuth, Dr. Jannik Fritsch und Dr. Marc Hanheide für die offenen und inspirierenden Gespräche genauso wie für das Entgegenkommen bei der Organisation und der sofortigen Zusage trotz ihrer vielfältigen Verpflichtungen.

Vor allem aber danke ich der Person, die mich seit Beginn meines Studiums stets begleitet hat und der ich diese Arbeit gerne widmen möchte, meiner wunderbaren Frau Meike. Ich danke Dir und liebe Dich für Dein Verständnis und Deine Geduld in so manch zeitlich begrenzter Situation.

Inhaltsverzeichnis

1. Integrierte Systeme für die Mensch-Roboter-Interaktion 1
1.1. Möglichkeiten der Mensch-Roboter-Interaktion . . . 3
1.2. Evolutionäre Systemintegration . . . 5
1.3. Zielsetzung . . . 6
1.4. Gliederung der Arbeit . . . 7

2. Methoden der Systemintegration in der Service-Robotik 9
2.1. Basiskonzepte und Modelle aktueller Systemarchitekturen . . . 9
2.1.1. Deliberative Kontrolle . . . 10
2.1.2. Reaktive Kontrolle . . . 11
2.1.3. Hybride Kontrolle . . . 13
2.1.4. Verhaltensbasierte Kontrolle . . . 14
2.1.5. Regelproduktionssysteme . . . 16
2.2. Robotersysteme in der Anwendung . . . 18
2.2.1. Marktorientierte und verfügbare Systeme . . . 18
2.2.2. Integrierte Systeme in der Forschung . . . 23
2.3. Zusammenfassung . . . 27

3. Entwicklungsbeispiele der Mensch-Roboter-Interaktion 29
3.1. Szenariobezogene Systemintegration . . . 29
3.1.1. Ein mobiler Entdecker auf Home-Tour . . . 30
3.1.2. Rezeptionist mit Körpersprache . . . 32
3.2. Demonstratorbezogene Systemintegration . . . 33
3.2.1. Der funktionale Assistenzroboter BIRON . . . 33
3.2.2. Der anthropomorphe Torso Roboter Barthoc . . . 35
3.3. Multimodale Umgebungswahrnehmung und Interaktionsfähigkeiten . . . 38
3.3.1. Multimodales Personen-Tracking . . . 38
3.3.2. Perzeptgenerierende Module . . . 41
3.4. Integration von Interaktion . . . 44
3.5. Zusammenfassung . . . 50

4. Entwicklungsgrundlagen für interaktive Roboter 51
4.1. Evolutionäre Systementwicklung statt Neubeginn . . . 51
4.2. Diskussion des Kommunikations-Frameworks . . . 53
4.2.1. Datenaustausch via *XCF* . . . 54

4.2.2. Flexible Schnittstellendefinition mit *XMLTIO* 55
4.2.3. Datenspeicherung im *Active Memory* 56
4.3. Remodellierung einer zentralen Kontrolle 57
4.4. Zusammenfassung 60

5. Evolutionäre Systemintegration 61
5.1. Das Active Control Memory Interface 62
5.2. Schnittstellenanalyse und Adaption 67
5.2.1. Design modulübergreifenden Datenaustausches 68
5.2.2. Bridging the Gap: Verknüpfung von Schnittstellen 71
5.3. Zusammenfassung 75

6. Generierung und Adaption des Systemverhaltens 77
6.1. Arbitrierung konkurrierender Aktionen 77
6.2. Sequenzmodell zur Koordination asynchroner Prozesse 81
6.3. Dynamische Verhaltensanpassung 84
6.4. Zusammenfassung 87

7. Analyse und Diskussion 89
7.1. Software-Integration und Systemwartbarkeit 89
7.2. Praktische Anwendung der Systemarchitektur 98
7.3. Zusammenfassung 105

8. Zusammenfassung und Ausblick 107

A. Systemimplementierung und -konfiguration 111
A.1. ACMI Regelbasis - BIRON 111
A.2. ACMI Regelbasis - Barthoc 118
A.3. ACMI Regelbasis - ARbInI 118
A.4. ACMI Klassendiagramm 119
A.5. TAME Spezifikation 120

Literaturverzeichnis 121

Abbildungsverzeichnis 131

1. Integrierte Systeme für die Mensch-Roboter-Interaktion

„Voraussagen sind schwierig, vor allem, wenn es um die Zukunft geht."

(Isaak Asimov)

Seit den frühen Anfängen der Computertechnologie wurden vielfältigste Aussagen über deren Entwicklung getroffen. Erste Schätzungen beliefen sich darauf, dass nur drei „Elektronengehirne" benötigt würden, um den weltweiten Bedarf zu decken. Fokussiert durch Sciencefiction, entstand in den frühen fünfziger Jahren der Eindruck, dass in absehbarer Zeit alle manuelle Arbeit durch intelligente Maschinen verrichtet werden könne (siehe Abbildung 1.1) und unsere Umwelt vernetzt und vollautomatisch auf alle menschlichen Bedürfnisse reagiere. Als zumindest die Vernetzung zu Beginn der neunziger Jahre eintrat, glaubten wiederum auch die größten Software-Konzerne nicht an deren weitreichenden Erfolg im privaten Umfeld. All diese Beispiele zeigen, dass es sehr schwierig ist, Prognosen über technologischen Fortschritt zu formulieren, die sich als zutreffend erweisen.

Was auf lange Sicht für den gesamten Bereich der Computerindustrie gilt, spiegelt sich nicht minder in deutlich kürzer gefassten Forschungsprojekten wieder. Genauer betrachtet wird hier der Entwicklungsprozess komplexer Systeme in Form von Service-Robotern, die den Anwender in alltäglichen Situationen unterstützen. Motiviert wird dieser Ansatz durch Gesellschaften, deren demographische Verläufe immer größere Herausforderung an die Versorgung älterer Menschen stellen. Der Einsatz von Service-Robotern weist eine vielversprechende Lösung auf, die dem jeweiligen Nutzer zudem hilft, seine Unabhängigkeit und Selbstständigkeit zu bewahren. Doch sind solche Systeme, die unter für Menschen konzipierten Bedingungen operieren und Alltagsaufgaben, wie den Einkauf oder einen Hausputz, verrichten können, immer noch Visionen, die sich in einem sehr wandelbaren Entwicklungsprozess befinden.

Wie kann nun bei der Entwicklung komplexer System verfahren werden, um sowohl flexibel auf neue Erkenntnisse aus der technischen Entwicklung, als auch aus Benutzerstudien reagieren zu können? Häufig wird für ein Szenario ein Prototyp entwickelt, jedoch werden diese Systeme nicht um neue Fähigkeiten erweitert, sondern „verstauben" nach einiger Zeit in den Archiven. Für ein System, welches kontinuierlich weiterentwickelt und dabei auch alternativen Ansätzen

(a) (b)

Abbildung 1.1.: Entwicklung in der Service-Robotik: Bereits 1931 begann die *Westinghouse Electric & Manufacturing Company* mit der kommerziellen Vermarktung von Robotern, die im Haushalt tätig werden sollten. Der 1939 vorgestellte Elektro (a) reagierte auf Sprachbefehle, ließ sich durch autonomes Vorwärtsschieben seiner Beine geradeaus fahren und bewegte Arme und Finger. Auch mehr als 40 Jahre später im Jahre 1981 waren die autonomen Fähigkeiten von Haushaltsrobotern sehr beschränkt, wie der hier von Peter Holland aus Hertfordshire entwickelte und ferngesteuerte Bügelroboter (b) zeigt.

Abbildung 1.2.: Moderne Systeme wir der Care-O-Bot 3 müssen Aufgaben selbstständig erfüllen und sich durch Lernen an ihre Umgebung anpassen.

gerecht wird, stellt sich die Frage, wie mit sich plötzlich ergebenden neuen Anforderungen umzugehen ist, ohne stetig alle bisherigen Komponenten zu überarbeiten. Besonders in der Forschung, deren Ziel es sein muss, offen auf neue Ansätzen zuzugehen, ist schwer abzusehen, welche Entwicklung sich in einem projektüblichen Zeitraum von vier Jahren abzeichnen wird.

Für komplexe integrierte Systeme, wie z. B. Service-Roboter, die in alltäglicher Umgebung operieren müssen, stellt sich zudem die Herausforderung, dass es für eine Forschungseinrichtung kaum zu bewältigen ist, alle notwendigen Fähigkeiten eines solchen Systems selber zu entwickeln. Die Kooperation mit Partnern ist damit ein wichtiger Bestandteil der Forschung und birgt die Herausforderung, dass die Systemanforderungen von Projektpartnern in geringerem Maße planbar sind. Dies gilt vor allem, wenn sich Kooperationen erst spontan ergeben und nicht bereits zu Projektbeginn geplant sind.

Um dennoch die Ergebnisse unterschiedlicher Forschungsschwerpunkte zusammen nutzen zu können, bedarf es eines sehr flexiblen Gesamtsystems, in welches Komponenten auf möglichst einfache Weise eingefügt werden. Sich als stabil erwiesene Module müssen auch bei weitreichenden Modifikationen unverändert bleiben können. Diese Arbeit beschäftigt sich mit der Entwicklung eines solchen Systems, welches im Bereich der Mensch-Roboter Interaktion auf verschiedenen Hardware-Plattformen und in unterschiedlichen Szenarien eingesetzt wird und somit seine generelle Anwendbarkeit demonstriert.

1.1. Möglichkeiten der Mensch-Roboter-Interaktion

Ziel der Forschung in der Mensch-Roboter-Interaktion ist es, komplexe Systeme für den Anwender so einfach wie möglich bedienbar zu machen. Assistenzsysteme sollen die Lebensqualität des Benutzers steigern. Ein besonderer Fokus wird hier auf Tätigkeiten im Haushalt gesetzt, die gerade älteren oder behinderten Menschen eine längere Autonomie in ihrer gewohnten Umgebung ermöglichen. Ohne eine einfache und flexible Handhabung bei der Anwendung im Alltag können sich derartige Systeme auch in technikoffenen Gesellschaften nicht etablieren [Fou08]. Im Gegensatz zu fachspezifischer Nutzung in Forschung oder Industrie ist es dem Benutzer nicht zuzumuten, als Voraussetzung für die Interaktion, detaillierte Computer- und Robotikkenntnisse zu erwerben. Außerdem ist es notwendig, dass Service-Systeme in einer alltäglichen Umgebung operieren können, ohne dass diese für z. B. einen Roboter angepasst werden muss. Lösungen, wie Bodenbeläge mit integrierten RF-ID-Tags mögen eine sehr effiziente Strategie für exakte Navigation in Gebäuden sein, erhöhen jedoch den Anschaffungsaufwand, der von hilfsbedürftigen Personen nicht generell geleistet werden kann. Damit Roboter in Privathaushalten von Nutzen für ihre Besitzer sind, müssen sie mit der jeweils vorhandenen Einrichtung zurechtkommen. Diese divergiert stark von Haushalt zu Haushalt. Somit ist es nicht möglich, den Roboter mit umfangreichem Weltwissen über sein neues Einsatzgebiet auszustatten. Selbst wenn bei einer ersten Begehung eine Fachkraft die notwendigen Parameter fest installierte, so dürfte der Anwender seine Einrichtung nicht mehr modernisieren oder müsste jeweils einen Techniker zur Wartung des Roboters einbestellen. Folglich muss der Roboter nicht nur mit vorhandener Einrichtung zurecht

kommen, sondern sich seiner Umgebung anpassen und lernen. Hierbei muss der Anwender dem Roboter dieses Wissen in natürlicher Weise, also wie einem menschlichen Gegenüber, präsentieren können. Einmal an den neuen Operationsbereich angepasst, sind bereits heute mehrere Szenarien denkbar, in der ein Roboter den Benutzer unterstützen kann:

Fetch and Carry ist ein häufig verwendetes Szenario der Service-Robotik, in der ein Roboter das von dem Benutzer benannte Objekt selbstständig finden und anschließend zu dem Anwender bringen muss. Auch wenn diese Aufgabe anfänglich einfach klingt, so beinhaltet sie gleich mehrere komplexe und noch nicht gelöste Aufgaben in der Robotik. Zu Beginn muss das System erkennen, welches Objekt der Benutzer meint. Für eine vereinfachte Bedienung ist es notwendig, dass sowohl Umgangssprache, als auch Gesten hierfür interpretiert werden können. Gesprochene Sprache ist jedoch unpräzise und bzw. oder unvollständig, was bei der Mensch-Mensch Interaktion durch Welt- und Kontextwissen der Kommunikationspartner kompensiert wird. So wird z. B. nicht genau zwischen Tasse und Becher oder diversen Arten von Tassen, wie Tee- oder Kaffeetassen unterschieden. Eine Aussage, wie *„Bring mir die Tasse.“*, bietet einen breiten Interpretationsraum. Ist das Objekt geklärt, muss der Roboter die Position des Objektes feststellen. Im einfachsten Fall ist diese zuvor in einer Repräsentation der Umgebung gespeichert, anderenfalls muss mittels Objektdetektion das Objekt in der Umgebung, beziehungsweise im wahrnehmbaren Bereich des Roboters, detektiert werden. Wurde die Position festgestellt, hat der Roboter autonom zu dem Objekt zu navigieren und dabei potentiellen statischen, aber auch beweglichen Hindernissen, wie Menschen oder Haustieren, auszuweichen. Am Objekt angelangt, muss dieses gegriffen werden, was erneut ein vollständiges Forschungsgebiet in der Robotik repräsentiert. So müssen nicht nur die Winkel der einzelnen Armgelenke für die korrekte Position des Greifers bestimmt werden, sondern abhängig von der Objektform ein passender Griff zum sicheren Erfassen des Objektes gewählt werden. Wurde das Objekt erfolgreich aufgenommen, so muss der Roboter zu seiner Ausgangsposition zurückkehren.

Objektsuche Einen Teil des Fetch & Carry Szenarios ohne Manipulation von Objekten stellt die Objektsuche dar. Hintergrund ist die alltägliche Suche nach einem bestimmten Gegenstand, sei es eine Brille, ein Autoschlüssel oder eine Fernbedienung. Jeder kennt das Problem, dass er schon einmal einen Gegenstand in der Wohnung verlegt hat. Betrachtet man die bereits angesprochenen demographische Entwicklung, zeigt sich deutlich der Nutzen von Assistenzsysteme, die dem Anwender helfen, sich an die Positionen bestimmter Objekte zu erinnern. Dies kann entweder durch Benennung der Position geschehen, *„Die Brille liegt auf dem Tisch in der Küche“* oder bei mobilen Systemen durch Führen des Benutzers zum gewünschten Ort.

Zutrittskontrolle und Überwachung Natürlich kann die Fähigkeit Menschen zu einen Bereich zu führen auch ausgebaut werden, um in Abwesenheit der Bewohner z. B. Handwerkern begrenzten Zutritt zu privaten Räumen zu gewähren. Der Roboter kann die Zielperson zu der vom Anwender bestimmten Stelle führen und die Tätigkeiten des Handwerkers mittels Videokameras aufzeichnen oder, wenn gewünscht, an moderne Mobiltelefone übertragen.

Allen Anwendungen gemein ist die große Flexibilität, die ihnen zu Grunde liegt und dass der Roboter eine Vielzahl zum Teil benutzerabhängiger Informationen vorhalten muss. Diese müssen dem Roboter in seinem neuen Anwendungsbereich durch den Anwender vermittelt werden, wobei die sich die Art der Vermittlung an einer Führung mit einem menschlichen Interaktionspartner orientiert. Ein inzwischen international etabliertes Szenario bietet der *RoboCup@Home*[1], ein Wettbewerb, in dem Service-Roboter unterschiedlicher Institutionen für die Bewältigung von Alltagsaufgaben in einer Privatwohnung eingesetzt werden. Vorrangiges Ziel ist hierbei der Wissensaustausch der Entwickler zur Weiterentwicklung der Service-Robotik. Dementsprechend werden die zu absolvierenden Aufgaben jedes Jahr überarbeitet und mit einem höheren Grad an Komplexität ausgestattet, um stets neue Herausforderungen für die Entwickler zu bieten. Der RoboCup@Home stellt hiermit eine sehr gute Testumgebung dar, die auch im Verlauf dieser Arbeit Verwendung findet.

Flexibilität wird nicht nur bei der Benutzerinteraktion vorausgesetzt. Sie ist auch bei der Implementierung unumgänglich. Die beschriebenen Szenarien zeigen, dass eine Großzahl unterschiedlichster Forschungsgebiete miteinander verknüpft werden. In entsprechendem Maße müssen deren Ergebnisse, die häufig als autonom funktionierende Module vorliegen, miteinander verknüpft werden, was hohe Ansprüche an die Integration in ein Gesamtsystem und an deren zugrunde liegende Architektur stellt.

1.2. Evolutionäre Systemintegration

Welche Aspekte zeichnen eine Systemarchitektur, die die hier betonte evolutionäre Systemintegration ermöglicht, gegenüber anderen Ansätzen aus? Auch bei dem hier beschriebenen Ansatz definiert die Architektur bei der Systemintegration in erster Linie, welche Komponenten in den Roboter oder genereller das System aufgenommen werden und wie diese miteinander in Verbindung stehen, analog zu Architekturplänen im Bauwesen. In der Software-Domäne gehören hierzu statt Raumgröße und Leitungsanschlüssen eine Beschreibung der gelieferten Daten, wie aktuelle Position, Bewegungskommandos, etc. und die genauen Definitionen, wie diese Daten repräsentiert werden. Häufig werden einzelne Elemente über diese Definitionen fest miteinander verbunden. Datenleitungen werden, wie die unterschiedlichen Leitungen in Gebäuden, fest installiert und die Räume an den beiden Endpunkte einer Leitung so aneinander gekoppelt. Doch was geschieht, wenn das Gebäude, oder analog ein Robotersystem, erweitert werden soll, wenn sich an der geplanten Nutzung etwas verändert hat?

Das Ziel der evolutionären Systemintegration ist es, bereits bei der Architektur den Aspekt der Anpassungsfähigkeit gegenüber neuen Anforderungen zu berücksichtigen. Die Kopplung der einzelnen Räume soll möglichst gering sein, um diese Räume bei Bedarf neu anzuordnen oder durch Anbauten zu erweitern, ohne alle bestehenden Verbindungen im Haus erneuern zu müssen. Dennoch müssen Verbindungen geschaffen werden, um Räume miteinander zu koppeln und

[1] http://www.robocupathome.org

ein Gebäude als funktionale Einheit zu schaffen. Analog bildet der Datenaustausch die Grundlage für ein erfolgreiches Zusammenspiel einzelner Module mit ihren jeweiligen Anforderungen. Als Module, oder auch Komponenten, werden im Weiteren die Bestandteile eines Systems bezeichnet, die Funktionseinheiten bilden, wie zum Beispiel die Detektion eines Gesichtes oder Bewegungssteuerung der Hardware. Es handelt sich, sofern nicht explizit anders dargestellt, um autonome Programme, die jeweils eine Fähigkeit des Roboters realisieren. Für ein einfach zu wartendes und flexibel erweiterbares System muss der Modularität in einer Architektur große Beachtung geschenkt werden[Som00, Bas03]. Abhängigkeiten zwischen den einzelnen Komponenten sind so gering wie möglich zu halten, um bei Modifikation oder Austausch einzelner Elemente so wenig Anpassungen wie möglich an dem Rest des Systems vorzunehmen. Dies gilt vor allem in Bereichen, in denen sich mehrere Module gemeinsame Ressourcen teilen und eine Koordination zwischen den Modulzugriffen implementiert ist.

Für kommerzielle Produkte, die über einen längeren Zeitraum unverändert vertrieben werden, mag es eine effiziente Lösung sein, derartige Mechanismen in die einzelnen Komponenten zu integrieren, so dass diese mit der Information über andere beteiligte Komponenten ein optimiertes Ergebnis liefern. In der Forschung, wo auch alternative Ansätze schnell und unkompliziert evaluiert werden müssen, ist es wichtig, derartige Mechanismen in Komponenten auszulagern, die Bestandteil der Architektur und von den spezifischen Modulen unabhängig sind. Dies gilt besonders im Bereich interaktiver Robotik, in dem sowohl durch, als auch für Nutzerstudien häufig und schnell Modifikationen an einem bestehenden System durchzuführen sind. Eine übliche Lösung dieser Anforderung besteht in der Verwendung von Planern, die die Kontrolle über das Gesamtsystem ausüben und einzelne Komponenten über standardisierte Schnittstellen je nach Systemzustand aktivieren oder abschalten. Ein weiterer wichtiger Vorteil bei einem zentralen Kontrollmodul ist, dass es die Informationen über alle Systemkomponenten nutzen kann, um Unregelmäßigkeiten zu entdecken und diese, wenn möglich, zu beheben. Dennoch sind bisherige Kontrollstrategien sehr statisch und kaum in der Lage Robotersysteme hervorzubringen, die sich Veränderungen der Forschungsschwerpunkte ausreichend anpassen, so dass sie dem wissenschaftlichen Selektionsdruck standhalten.

1.3. Zielsetzung

Das Ziel meiner Arbeit ist die Entwicklung einer evolutionären Systemintegration auf Basis einer Architektur, die in Hinblick sowohl auf einzelne Module, als auch auf das durch den Benutzer wahrnehmbare Gesamtverhalten des Roboters genügend Anpassungsfähigkeit bietet, um ein System kontinuierlich weiterzuentwickeln. Auf der Ebene einzelner Module bedeutet dies, das Hinzufügen, Austauschen und Anpassen von Modulen zu erleichtern. Module erweitern den Roboter um einzelne Fähigkeiten, wie zum Beispiel Motorkontrolle oder Personendetektion. Komplexere Aktionen, wie das Folgen einer Person, sind Bestandteil des Gesamtverhaltens und werden durch Kombination von Fähigkeiten, wie Personenerkennung und Motorkontrolle, aufgebaut. Für diesen Aufbau wird ein zentrales Verfahren entwickelt, welches eine schnelle und einfache

Anpassung des Gesamtverhaltens ermöglicht, ohne jedoch eine direkte und statische Kontrolle auf die anderen Komponenten auszuüben. Stattdessen wird eine Koordination angestrebt, die nur im Konfliktfall in das laufende System eingreift und sehr flexibel ist.Das zu entwickelnde Integrationsverfahren muss ausreichend flexibel sein, um auf unterschiedlichen Hardware-Plattformen und deren individuellen Anwendungsgebieten einsetzbar zu sein. Ferner muss ein hoher Grad an Generalität erreicht werden, um der schnellen Entwicklung und Modifikation von Systembestandteilen, wie einzelnen Software-Modulen, gerecht zu werden. Um diese sehr weitreichende Thematik auf eine feste Basis zu stellen und in einem realen System zu implementieren, wird folgenden konkreten Fragestellungen nachgegangen:

- *Wie kann ein bestehendes System mit möglichst geringem Adaptionsaufwand um neu entwickelte Fähigkeiten erweitert werden?*
- *Wie können konkurrierende Anforderungen an begrenzte Ressourcen aufgelöst werden?*
- *Wie werden asynchrone Modulprozesse zu einer gewünschten Roboteraktion verknüpft?*
- *Wie kann ein bestehendes Systemverhalten einfach und schnell auf neue Anforderungen angepasst werden?*

Für die erfolgreiche Beantwortung dieser Fragen wird im Rahmen der präsentierten Arbeit auf eine Familie unterschiedlicher Systeme eingegangen, die sich aus einem Basis-Demonstrator und dessen Anwendungsszenario entwickelt hat. Teile dieser Familie sind Schwerpunkte zweier großer Forschungsprojekte, zum einen dem europaweiten und von der Europäischen Union geförderten Cognitive Robot Companion (COGNIRON), zum anderen dem von der DFG geförderten Sonderforschungsbereich 673, Alignment in Communication. Das hier beschriebene Integrationsverfahren muss sich nicht nur unter unterschiedlichen Bedingungen in realen Anwendungsfällen als erfolgreich erweisen, sondern auch durch seine Anwendung die oben gestellten Fragen für die Integrationsarbeit im jeweiligen Forschungsprojekt beantworten.

1.4. Gliederung der Arbeit

Die Arbeit gibt im folgenden 2. Kapitel einen Überblick bezüglich der unterschiedlichen existierenden Systemarchitekturen und deren individuelle Eigenschaften. Der Einsatz dieser Systemarchitekturen wird anschließend am Beispiel verschiedener Robotersysteme verdeutlicht und deren Nutzen für unterschiedliche Einsatzgebiete und Forschungsschwerpunkte diskutiert.

Bezugnehmend auf die Inhalte des 2. Kapitels werden in Kapitel 3 die zu Beginn dieser Arbeit vorhandene Ergebnisse vorgestellt, auf denen diese Arbeit aufbaut. Hierzu gehören die Beschreibung der verwendeten Hardware, genauso wie die bereits entwickelten Komponenten für die Mensch-Roboter-Interaktion in verschiedenen Szenarien.

Der Beschreibung der Voraussetzung folgt in Kapitel 4 die Diskussion der Anforderungen an die Systemintegration für interaktive Robotersysteme. Dabei wird auch die Wahl eines geeigneten Kommunikations-Frameworks als Basis der Systemintegration dargelegt.

Basierend auf den Überlegungen aus den Kapiteln 2 und 4 wird in Kapitel 5 das Modell einer neuen und in dieser Arbeit beschriebenen evolutionären Architektur zur Systemintegration (*EASI*) vorgestellt, welches mit seinem Schnittstellenkonzept besonders der notwendigen Flexibilität für Forschungsplattformen gerecht wird.

Der zweite Aspekt des entwickelten Architekturmodells bezieht sich auf die Koordination der Einzelkomponenten, die für ein möglichst modulares System asynchron agieren und voneinander unabhängig konzipiert sind. Wie aus diesen Einzelfunktionen ein konsistentes Systemverhalten generiert wird, wird in Kapitel 6 geschildert.

Das konzipierte und anschließend implementierte Gesamtsystem wird in Kapitel 7 diskutiert. Die Analyse berücksichtigt sowohl die theoretischen und technischen Aspekte, wie die Kopplung der Einzelkomponenten und Wartbarkeit der Systemkoordination, als auch die praktische Anwendbarkeit in verschiedenen Interaktionsszenarien außerhalb von Laborumgebungen.

Die Arbeit schließt in Kapitel 8 mit einer Zusammenfassung und einem Ausblick auf neue Forschungsfragen, die durch diese Arbeit möglich geworden sind.

2. Methoden der Systemintegration in der Service-Robotik

Die Entwicklung komplexer integrierter Systeme für die Interaktion in Alltagssituationen mit ungeschulten Benutzern ist eine herausfordernder Aufgabe, die besondere Ansprüche an die Systemintegration stellt. Diese Ansprüche resultieren aus der Unvorhersehbarkeit der Einsatzumgebung und des Verhaltens des Anwenders. Sehr häufig zeigt sich in durchgeführten Studien, dass vorausgehende Annahmen über Einsatzumgebung und Anwender nur teilweise zutreffen. Für einen derartigen iterativen Prozess aus Annahmen, Implementierung, Studie und Verbesserung ist eine einfache und schnelle Anpassung des Robotersystems erforderlich. Diese Anpassung ist bereits auf der Architekturebene zu berücksichtigen. Insbesondere wenn sich Details erst während der Realisierung ergeben oder sich durch neue Erkenntnisse Modifikationen bedingen, bedarf es einer klaren Konzeption, wie einzelne Fähigkeiten in ein Gesamtsystem eingebunden werden können.

2.1. Basiskonzepte und Modelle aktueller Systemarchitekturen

Analog zu der Errichtung von Bauwerken, in denen unterschiedliche Gewerke für das Gesamtbauwerk koordiniert zusammenarbeiten und sich als gemeinsamen Nenner an die Vorgaben des Architekten halten, wird auch in der Entwicklung von Software-Systemen der Begriff der *Architektur* verwendet. Da dieser Begriff in dem umfangreichen Gebiet der Software-Entwicklung unterschiedlichen Ausprägungen unterliegt, bezieht sich der Begriff in dieser Arbeit auf die Integration unterschiedlicher Software. Auch wenn sich die Architekturen für die jeweiligen Systeme unterscheiden, wie die individuellen Baupläne für jedes neue Gebäude, so existieren Gemeinsamkeiten [Bry00], die sich auf vier grundlegende Modelle zurückführen lassen. Diese Modelle werden als *deliberative*, *reaktive*, *hybride* oder *verhaltensbasierte (engl.: behaviour-based)* Architekturen bezeichnet. Welches Modell letztendlich Anwendung findet, hängt von dem gewünschten Anwendungsbereich ab, wie auch Gebäude für zum Beispiel private, gewerbliche oder gesellschaftliche Belange entworfen werden. Diese grundlegenden Architekturprinzipien sowie ihre Anwendungsgebiete werden im Folgenden vorgestellt und durch Anwendungsbeispiele anhand konkreter Architekturen und den daraus resultierenden Implementationen ergänzt.

2.1.1. Deliberative Kontrolle

Eine frühe Idee, komplexere Verhaltensweisen zu modellieren, besteht aus den folgenden Verarbeitungsschritten: Zu Beginn erfolgt die Generierung eines Planes. Dieser basiert auf der zu erfüllenden Aufgabe und den von den Sensoren aufgenommenen Umgebungsdaten. Mit Hilfe eines Planes wird eine komplexe Aufgabe, wie zum Beispiel das eingangs erwähnte Holen eines Objektes, in eine Sequenz von Einzelaktionen gegliedert, die von den jeweiligen Software-Modulen ausgeführt werden können. Nachdem der Plan erstellt wurde, erfolgt dessen Ausführung. Dieses auch unter der Bezeichnung *Sense-Plan-Act* [Nil82] bekannte Verfahren basiert auf der klassischen Vorgehensweise im Bereich der künstlichen Intelligenz (*KI*). Aus den gegebenen Daten, hier die Sensordaten, werden neue Informationen, wie zum Beispiel ein Plan zum Lösen einer Aufgabe, abgeleitet. Da dem Ableiten von Informationen und der Erstellung eines Planes in dieser Architektur eine große Bedeutung zukommt, wird eine Architektur (siehe Abb. 2.1), die diesem Konzept folgt als *deliberative Architektur* (*lat.: deliberare - überlegen, abwägen*) bezeichnet. Ein weiterer, wichtiger Bestandteil dieser Architektur sind fest vorgegebenen Repräsentationen der Umgebung, mit denen das System seine über die Sensoren aufgenommenen Informationen aus der realen Welt abgleicht. Dementsprechend wird diese Menge an festen und vorgegebenen Informationen als *Weltmodell* bezeichnet.

Es ist sehr einfach zu erkennen, dass die erfolgreiche Bewältigung einer Aufgabe durch ein deduktiv gesteuertes System sehr stark von dem verwendeten Weltmodell abhängt. Stimmen die Parameter des Weltmodells nicht oder nur ungenügend mit der Umgebung überein, reagiert das System nicht in gewünschter Weise. Besonders im Bereich autonom agierender Roboter, die mit Menschen interagieren, können die Konsequenzen bis zu Verletzungsrisiken reichen. Die zweite Herausforderung, vor die deduktive Systeme gestellt werden, ist die exakte Abarbeitung

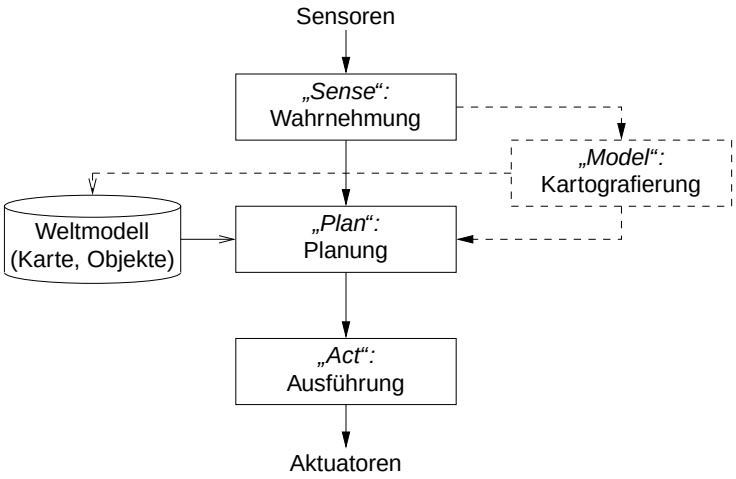


Abbildung 2.1.: In der deliberativen Systemarchitektur werden Sensorwerte mit einem vorgegebenen Weltmodell abgeglichen, um einen auf dem Abgleich basierenden Plan zur Erfüllung einer Aufgabe zu entwickeln und durchzuführen.

des Planes nach Abschluss der Planungsphase. Wurde ein Plan erstellt, so wird dieser in einem deduktiven System unabhängig von zwischenzeitlich neu eintretenden Ereignissen abgearbeitet. Die Auswertung der Sensoren und des Weltmodells erfolgt nur zu Beginn der Planung.

Betrachtet man beispielsweise die Aufgabe eines Roboters mit einer rein deliberativen Architektur, die Besucher eines Museums von einem Ausstellungsstück zu dem nächsten Exponat zu führen, so könnte das System einen optimalen Weg zum Ziel bestimmen. Es wäre aber nicht in der Lage, während der Fahrt auf plötzliche Hindernisse, wie passierende Besucher, zu reagieren, sondern liefe Gefahr, mit ihnen zu kollidieren. Es erweist sich als wichtig, auch während der Ausführung eines Planes dessen Anwendbarkeit kontinuierlich zu überprüfen. Die naheliegendste Lösung besteht darin, nach jedem Einzelschritt im Plan die aktuelle Situation zu prüfen und bei Veränderung den Plan neu zu kalkulieren. Dieses Verfahren kommt bei [Bur98] zum Einsatz und bietet eine höhere Flexibilität. Der Nachteil liegt vor allem bei dynamischen Umgebungen auf der Hand, wie das hier vorgestellte Museumsszenario zeigt. Bei jeder Veränderung muss erneut geplant werden. Dies schränkt die Aktionsgeschwindigkeit stark ein.

Auch die Verwendung des Weltmodells in einer deliberativen Architektur birgt ähnliche Herausforderungen. So sehr die Beteiligten bei der Entwicklung eines Systems auch bemüht sind, möglichst viele Eventualitäten zu berücksichtigen, wird es nicht gelingen alle möglichen Ereignisse zu modellieren, selbst wenn das Szenario eng begrenzt ist. Die Verwendung adaptiver Weltmodelle wie in [Thr91] bewirkt ebenfalls eine Neuplanung nach jedem erfolgreich absolvierten Einzelschritt unter Berücksichtigung des inzwischen eventuell veränderten Weltmodells. Es zeigt sich, dass besonders in dynamischen Umgebungen, welche den für Menschen alltäglichen Lebensraum ausmachen, rein deliberative Systeme wegen ihres hohen Planungsaufwands und der daraus resultierenden Reaktionszeit weniger geeignet sind.

2.1.2. Reaktive Kontrolle

Resultierend aus diesen Erfahrungen, stammt die Idee zu einer *reaktiven Architektur* wie sie in Abb. 2.2 dargestellt ist. Dem Motto „*The world is its own best model.*“ [Bro90] folgend, wird auf ein Weltmodell verzichtet und anstatt eines Planes einer neuen Sensorinformation eine Aktion zugeordnet, die umgehend ausgeführt wird. Die Zuordnung geschieht über vordefinierte Regeln, was dem biologischen Prinzip von *Stimulus und Antwort* nachempfunden ist. Im Gegensatz zu deliberativen Systemen ist die Komplexität von Aufgaben stark eingeschränkt, da keine Pläne aus einer Sequenz von Einzelaufgaben erstellt werden. In [Zie96] wird ein reaktiver Ansatz zum Greifen sich bewegender Objekte verwendet, wobei die Sensorwerte des Objektes über Regeln direkt mit der Geschwindigkeitssteuerung des Greifers verknüpft sind. Komplexere Aufgaben sind genauso innerhalb einer einzigen Regel zu formulieren, wie die Ansteuerung eines Aktuators, was bei den variierenden Ausgangsbedingungen, für die diese Architektur eingesetzt werden soll, schon anhand der möglichen Parametermenge schwierig zu realisieren ist.

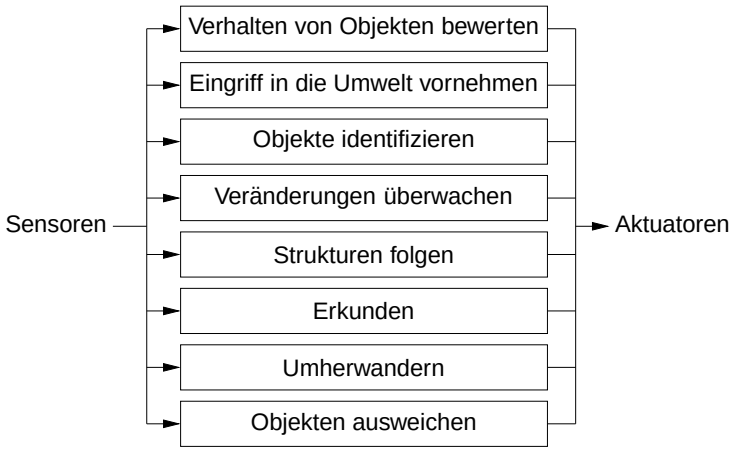


Abbildung 2.2.: Darstellung eines reaktiven Systems: Auf jede Sensoreingabe folgt eine Reaktion mittels einfacher Regelauswertung.

Einen höheren Grad an Flexibilität und Intelligenz verspricht die Verschachtelung von einzelnen reaktiven Zuordnungen in Form von mehrschichtigen Hierarchien (siehe Abb. 2.3). Dieses Vorgehen wurde in der *Subsumption Architektur* [Bro86] umgesetzt. Außerdem kann ohne zentrale Kontrolle durch geeignete Abstimmung der Verhalten untereinander ein effizienteres Gesamtverhalten des Systems erzeugt werden. Sollen einzelne Verhalten aktualisiert, ergänzt oder ausgetauscht werden, birgt der Subsumption-Ansatz das Risiko, dass alle verbleibenden Verhalten erneut auf die Modifikationen abgestimmt werden müssen. Das gleiche gilt auch bei erwünschter Veränderung des Systemverhaltens, selbst wenn ursprünglich kein Einzelverhalten verändert wurde.

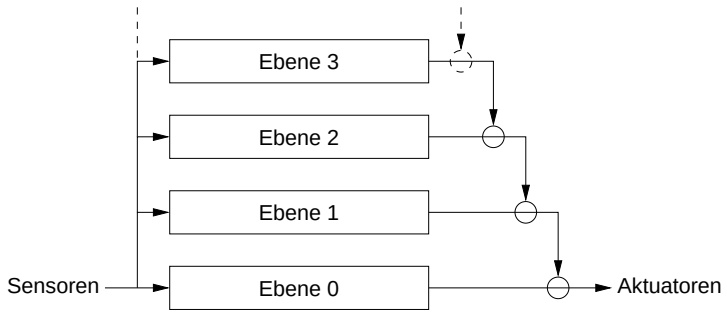


Abbildung 2.3.: In dem Subsumption-Ansatz wird auf explizite Kontrolle verzichtet, indem die einzelnen Komponenten so miteinander verschaltet werden, dass das gewünschte Systemverhalten implizit generiert wird.

2.1.3. Hybride Kontrolle

Betrachtet man Vor- und Nachteile der beiden genannten Ansätze, so lässt sich leicht erkennen, dass die geschickte Kombination einer deliberativen und einer reaktiven Architektur die zuvor beschriebenen Herausforderungen bewältigen kann. Für komplexe Aufgabenstellungen, wie autonome Navigation, werden deliberative Komponenten zur besseren Planung verwendet, während einzelne zeitkritische Funktionen, wie zum Beispiel Kollisionsvermeidung, durch reaktive Module ausgeführt werden. Dieses Prinzip hat sich auf sehr vielen Systemen im Bereich der Mensch-Roboter-Interaktion bewährt und wird wegen der Vermischung zweier Konzepte als *hybride Architektur* bezeichnet. Häufig werden hybride Architekturen im Modell auch als 3-Schichten-Architektur [Gat98] dargestellt (siehe Abb. 2.4), wobei je nach Granularität die Schichtenanzahl auch höher ausfallen kann. Die Darstellung stammt von der Vorstellung, die häufig den Hardware-Controllern näheren reaktiven Module in einer eigenen Schicht zu modellieren, ebenso wie die deliberativen Komponenten und diese beiden Schichten durch die sogenannte *intermediäre Schicht* zu verbinden. Die intermediäre Schicht beinhaltet Softwarekomponenten für den Datenaustausch, genauso wie Datenbanken und Elemente, die die gegebenenfalls widersprüchlichen Anweisungen von deliberativen und reaktiven Komponenten synchronisieren. In der als ATLANTIS (A Three-Layer Architecture for Navigating Through Intricate Situations) [Gat92] bekannt gewordenen Architektur wurde dieses Konzept erfolgreich zur Kontrolle mobiler Roboter umgesetzt. Es besteht aus einem reaktiven *Controller* für Sensor-Motor-Ansteuerung, dem *Deliberator*, in welchem Planung und Weltmodell integriert sind und dem *Sequenzer* zur Koordination der einzelnen sowohl deliberativen als auch reaktiven Prozessabläufe.

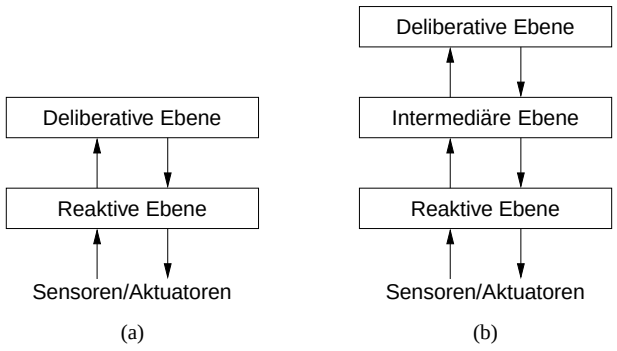


Abbildung 2.4.: Darstellung einer hybriden Zwei(a)- beziehungsweise Drei(b)-Schichten-Architektur. Die Vorteile von reaktiven und deliberativen Systemen werden verknüpft, wobei die Synchronisierung der Einzelprozesse häufig in der sog. *Intermediären Schicht* zusammen gefasst wird.

Einen ähnlichen, schichtweisen Aufbau bietet auch die *Architecture for Autonomy* [Ala98]. Sie setzt sich aus einer Entscheidungs-, Ausführungs- und Funktionsschicht zusammen. Während die Entscheidungsschicht Planer und Sequenzer beinhaltet, wird die Koordination der einzelnen Elemente der Funktionsschicht in der Ausführungsschicht realisiert. Hervorzuheben ist hier

die Idee eines *Generators of Modules* (GENOM), der für die einzelnen Funktionen, wie zum Beispiel Navigation, separate Module mit standardisierten Schnittstellen erzeugt. Diese müssen nur noch mit den eigentlichen Algorithmen gefüllt werden und können so von der Koordination angesteuert werden, wobei die Kommunikation über gemeinsam genutzte Speicherbereiche geschieht. So elegant dieses Vorgehen ist, müssen kommunizierende Prozesse auf dem gleichen Rechner agieren, um Daten austauschen zu können und es müssen nicht für dieser Architektur erstellte Funktionen mit Hilfe von GENOM reimplementiert werden.

2.1.4. Verhaltensbasierte Kontrolle

Ein weiteres Modell für ein System ist die in [Ark98] proklamierte *Verhaltensbasierte Architektur*, die genau wie der hybride Ansatz in vielen Systemen zum Einsatz kommt. Analog zum Reiz-Stimulus-Modell reaktiver Systeme, werden gewünschte Aktionen als einzelne Verhalten modelliert, die abhängig von eingehenden Umweltreizen ausgeführt werden (siehe Abb. 2.5). Dieses Vorgehen weist Ähnlichkeiten zu reaktiven Architekturen auf, darf aber nicht mit diesen verwechselt werden. Im Vergleich mit reaktiven Systemen können gleichzeitig mehrere Verhalten aktiviert sein, die untereinander konkurrieren. Die unterschiedlichen Verhaltensmuster dürfen zudem sich gegenseitig widersprechende Aktionen auslösen, wobei zu entscheiden ist, welches Verhalten „die Kontrolle“ über das System gewinnt. Wie in biologischen Systemen, werden durch die unterschiedlichen Umweltinformationen die vorhandenen Verhalten unterschiedlich stark stimuliert. Im einfachsten Fall kommt das Verhalten zur Ausführung, welches zum jeweiligen Zeitpunkt über den höchsten Stimulus verfügt. Betrachtet man das Beispiel der autonomen Navigation, so wären das Erreichen eines Ziels und das Ausweichen vor Hindernissen zwei Verhaltensweisen mit unterschiedlicher Komplexität. Solange kein Hindernis existiert oder es noch in größerer Entfernung liegt, wird das Ausweichverhalten kaum stimuliert und das System bewegt sich zu dem Ziel. Je näher der Roboter einem Hindernis kommt, desto stärker wird das Ausweichverhalten stimuliert, bis es die Kontrolle von der Zielverfolgung übernimmt.

Ein Beispiel für verhaltensbasierte Systeme zur Roboterkontrolle, stellt die JDE(C)-Architektur ([Cañ05]) dar. Das gewünschte Systemverhalten ist durch die Hierarchisierung von einzelnen Verhalten realisiert. In einer Baumstruktur konkurrieren jeweils die Kind-Verhalten eines Eltern-Verhaltens untereinander, wobei das Elternverhalten gleichzeitig die Kriterien für die Auswahl des auszuführenden Kindverhaltens vorgibt. Die Implementierung ist in einer Pluginstruktur realisiert, bei der jedes Verhalten in Form eines einzelnen Plug-Ins realisiert ist. Mit der Weiterentwicklung zu JDE-NEOC ([Cañ07]) wurden die ursprünglich als statische Bibliotheken zur Kompilierzeit eingebundenen Plug-Ins als zur Laufzeit einsetzbare, dynamische Bibliotheken reimplementiert. Dies ermöglicht, unterstützt durch ein grafisches Interface, die schnelle Vernetzung von Verhalten für flexible Systemerweiterungen. Jedoch müssen mit Aktualisierung einzelner Verhalten gegebenenfalls auch die jeweiligen Eltern-Verhalten bis hin zur obersten Verhaltens-Schicht in ihren Auswahlkriterien aktualisiert werden, da die Systemkontrolle implizit auf die jeweiligen Eltern-Verhalten aufgeteilt ist.

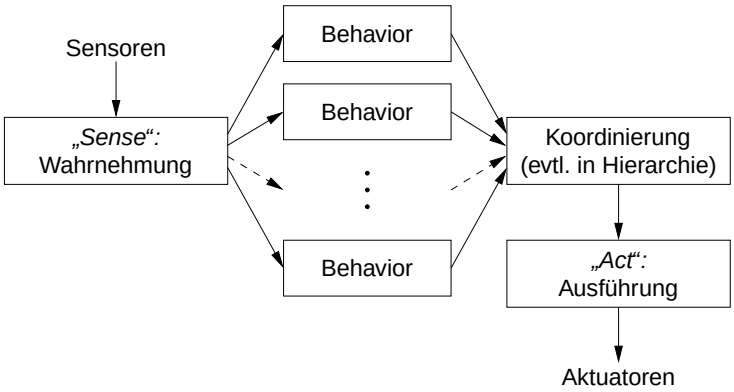


Abbildung 2.5.: In biologisch motivierten verhaltensbasierten Systemen erzeugen einzelne oder mehrere Prozesse jeweils autonome Verhaltensweisen (*Behaviors*) unterschiedlich hoher Komplexität. Da sich die einzelnen Behaviors widersprechen können, zum Beispiel in der Bewegungsrichtung, müssen sie vor der Ausführung koordiniert werden.

Kontrollparadigmen

Unter einer impliziten Kontrolle ist ein emergentes Systemverhalten zu verstehen, welches sich meist aus dem geeignet parametrisierten Zusammenspiel der beteiligten Komponenten ergibt. Ein Beispiel hierfür ist der bereits erwähnte Subsumption-Ansatz nach Brooks, in dem alle Verhalten intern so aufeinander abgestimmt werden, dass es zu keinem Konflikt kommt. Es wird folglich auch ganz auf eine zentrale Kontrolle verzichtet. Bei verhaltensbasierten Architekturen wird dies meist durch Gewichtungen der Reaktion innerhalb jedes verhaltengenerierenden Moduls mit Rücksicht auf alle anderen Verhalten abgestimmt. Dies ähnelt der biologischen Verarbeitungsweise stärker als Systeme mit zentraler Arbitrierung und führt zu einer gleichermaßen eleganten, wie auch effizienten Kontrolle, wie sie zum Beispiel in [Lóp07] als Subsumption Architektur für virtuelle Agenten vorgestellt wird. Eine generelle Herausforderung liegt jedoch in der starken Verknüpfung von Komponenten, die an der impliziten Kontrolle beteiligt sind. Die Modifikation einer solchen Architektur bedingt meist die Aktualisierung vieler Komponenten hinsichtlich ihrer Kontrollparameter.

Alternativ zur impliziten Kontrolle bietet sich die den deliberativen Architekturen nähere explizite Kontrolle in Form eines Planers oder eines Moduls zur *Arbitrierung*, wie bei verhaltensbasierten Systemen (siehe Abb. 2.6) üblich, an. Die Arbitrierung entscheidet bei widersprüchlichen Anweisungen, hier Verhalten, welche Anweisung zur Ausführung kommt. Hierzu verfügt die Arbitrierung zentral über Informationen, wie stark einzelne Verhalten zu gewichten sind und bei welcher Stimulusstärke sie über andere Verhalten dominieren. Proetzsch et al. stellen mit iB2C [Pro07] eine verhaltensbasierten Architektur vor, bei der die einzelnen Verhalten in Modulen mit standardisierten Schnittstellen realisiert sind. Für die eigentliche Datenein- und -ausgabe wird eine Vektorrepräsentation gewählt. Die Aktivität des Verhaltens kann über eine Aktivator- und eine Inhibitor-Schnittstelle beeinflusst werden. Resultierend aus der Aktivierung und den

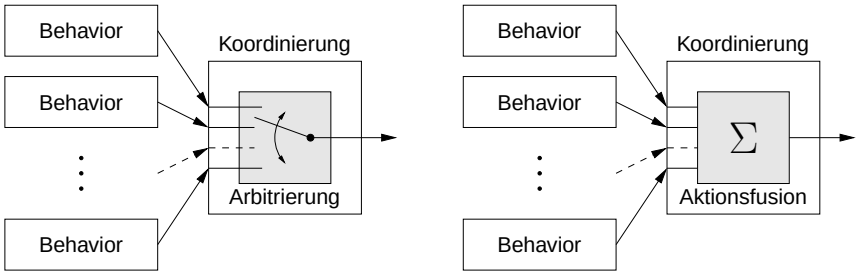


Abbildung 2.6.: Zur Koordination widersprüchlicher Behaviors in verhaltensbasierten Architekturen wird entweder ein Behavior zur Ausführung gebracht, während die Restlichen ignoriert werden (links), oder eine gewichtete Summe über die Behaviors gebildet.

Eingangsdaten werden über zwei weitere Schnittstellen Informationen über die tatsächliche Aktivität und Dringlichkeit der Ausführung des Verhaltens an die Arbitrierung mitgeteilt. iB2C verwendet eine gewichtete Arbitrierung aller Ausgabevektoren der vorhandenen Verhalten. Als Gewichte werden die jeweiligen Aktivitäten und Dringlichkeiten eines jeden an der Arbitrierung beteiligten Moduls verwendet. Als Voraussetzung für dieses Verfahren muss neben der eventuellen Reimplementation in ein Modul mit übereinstimmenden Schnittstellen und der Herleitung von Werten, wie Aktivität und Dringlichkeit, auch berücksichtigt werden, wie die Ein- und Ausgabedaten in eine gemeinsame vektorielle Repräsentation transformiert werden können und in wie weit dies für die unterschiedlichen Ressourcen sinnvoll ist. Eine gewichtete Summe als Arbitrierung für Sprachsynthese ist sicher genauso unpassend wie eine generelle Verrechnung von Aktuatorpositionen, so dass weder das eine noch das andere Ziel erreicht wird.

2.1.5. Regelproduktionssysteme

In wie fern Kontrollmechanismen implizit durch intelligente Verknüpfung einzelner Komponenten oder explizit durch externe Vorgaben realisiert sind, hängt stark von dem jeweiligen Anwendungsgebiet ab. Die Wahl der Systemarchitektur gibt grundlegende Vorgaben über die Möglichkeiten zur Kontrolle des Gesamtsystems, wie Planung, zentrale Kontrolle oder implizite Verhaltensgenerierung, vor. Doch sogenannte *Regelproduktionssysteme* finden sowohl in klassischen, als auch in aktuellen Architekturimplementationen verbreitet Anwendung. Ein Regelproduktionssystem besteht aus einem Satz an Regeln, einem Interpreter-Programm für die Regelauswertung und dem Arbeitsspeicher, in dem die Regeln und Interpreterergebnisse abgelegt werden. Vergleichbar mit reaktiven Systemarchitekturen stellt eine Regel eine wenn-dann-Beziehung dar, jedoch verknüpft sie nicht zwingend Sensordaten mit Aktuatoren. Jedes Ereignis, auf den der „wenn-Teil“ einer Regel zutrifft, bewirkt, dass die Regel ausgeführt wird. Analog zu biologischen, neuronalen Systemen wird das Ereignis auch als *Reiz* oder *Stimulus* und das Anwenden einer Regel als *Feuern* bezeichnet. Die Auswertung der Stimuli und die Anwendung der

entsprechenden Regeln wird von dem Interpreter durchgeführt. In wie weit das Systemverhalten, wie zum Beispiel Arbitrierung begrenzter Ressourcen, fester Bestandteil des Interpreters ist oder über die Regeln selbst definiert wird, ist hierbei offen.

Ishida et al. propagieren in [Ish95] ein Regelproduktionssystem, dessen Kontrolle sie auf Grund des „intelligenten“ Interpreters als implizit einstufen. Durch Verlagerung von Funktionen in den Interpreter verringert sich die Komplexität der Regeln. Daraus resultiert jedoch eine Einschränkung in der System- und Regelflexibilität. Eine parallele Auswertung von Regeln, selbst wenn diese sich nicht widersprechen, ist bei diesem Ansatz zum Beispiel nicht möglich.

Ein Ansatz, der genau auf die Koordination paralleler Aufgaben ausgerichtet ist, beschreibt die EPIC(Executive Process-Interactive Control)-Architektur von Kieras [Kie95]. Das ebenfalls auf Regelauswertung basierende System kann prinzipiell beliebig viele Regeln simultan anwenden, wobei es nur durch die zum Anwendungszeitpunkt verfügbaren Sensoren und Aktuatoren limitiert ist. Die Regeldefinitionen sind nur ein Teil des Modells, welches EPIC benötigt. So sind auch Spezifikationen über die metrische Beschaffenheit der Umgebung oder die zeitlichen Abstände von Eingangssignalen zu definieren, was die Architektur sehr domänenabhängig werden lässt und umfangreiche Modifikationen für Erweiterung auf neue Aufgabenfelder bedingt. Außerdem wurde EPIC für die Modellierung menschlicher kognitiver Prozesse entwickelt, um so dem natürlichen Vorbild einer kognitiven Architektur [Cla99] möglichst nahe zu kommen. Die Verwendung menschenfremder Ein- und Ausgabemodalitäten, wie zum Beispiel Laser-basierte Distanzmessungen sind somit ausgeschlossen, genauso wie der Zugriff auf native Datentypen zur beliebigen Weiterverarbeitung durch externe Anwendungen.

Eine speziell auf die Robotik fokussierte Architektur stellt ADAPT [Ben04]) dar. Basierend auf EPIC sowie auf ACT-R [And93] und SOAR [Leh98] verfolgt der ADAPT Ansatz zwei grundlegende Ideen: erstens, situationsabhängige aktive Perzeption und zweitens, echte Parallelität einzelner Roboteraktionen mit kontextabhängiger Arbitrierung von Konflikten. Anstatt alle eingehenden Informationen auszuwerten und erst die Ergebnisse zu vergleichen, werden situationsabhängig Perzepte gefiltert. Beim Überqueren einer Straße ist es wichtiger die Aufmerksamkeit auf sich nähernde PKW zu fokussieren, anstatt nach Interaktionspartnern Ausschau zu halten. Trotz dieser Priorisierung finden in realen Umgebungen mehrere wichtige Ereignisse gleichzeitig statt, welche bei ADAPT entsprechend parallel verarbeitet werden, was in ACT-R nicht realisierbar ist. Mit ACT-R gemein ist jedoch die Verwendung eines Langzeitgedächtnisses zur Speicherung der Verknüpfungen zwischen Perzepten und Aktionen. Außerdem werden wie bei EPIC einfache Sensordaten über einen Arbeitsspeicher verwaltet und ausgetauscht. SOAR ermöglicht eine gewisse Parallelität, indem Aufgaben in mehrere sogenannte *Problem Spaces* aufgeteilt sind, wobei in jedem Problem Space eine eigene Regel zur Ausführung gelangen kann. Durch sorgfältiges Design dieser Problem Spaces werden Konflikte bei den parallel aktivierten Regeln vermieden. Das Hauptaugenmerk bei ADAPT liegt auf der integrierten Kombination von Perzeption, Problemlösung und der Verwendung natürlicher Sprache. Durch diese Komposition wird die direkte Auswirkung von Umweltreizen auf die interne Repräsentation des Systems beobachtbarer als in den zugrundeliegenden Architekturansätzen, jedoch wird auch die Koppelung der Einzelkomponenten untereinander verstärkt.

Die genannten Beispiele geben nur einen kleinen Überblick bezüglich der verschiedenen entwickelten Ansätze. Sie liefern gute Ergebnisse für die jeweiligen Einsatzgebiete und beschreiben ein großes Feld an möglichen Lösungsstrategien für unterschiedliche, klar vordefinierte Anforderungen. Eine große Herausforderung stellt die Interaktionsfähigkeit von Robotersystemen dar, die in einer auf den Menschen zugeschnittenen und nur schwer zu spezifizierenden Umwelt agieren müssen. Im Folgenden wird eine Auswahl an sich aktuell in der Entwicklung befindenden Systemen präsentiert, die zwar unterschiedliche Aufgabenbereiche abdecken, denen jedoch gemein ist, dass sie erfolgreich im privateren Umfeld des Menschen operieren müssen.

2.2. Robotersysteme in der Anwendung

Die System-Architektur stellt ein abstrahiertes Modell dar, anhand dessen die Entwicklung komplexer Systeme geplant und koordiniert wird. Trotz gewissenhafter Planung und weitreichender theoretischer Analyse wird erst durch die Implementation realer Systeme und deren alltäglichen Einsatz die Güte der zugrundeliegenden Architektur sichtbar. Im Folgenden werden mehrere Systeme vorgestellt, die sowohl unter wissenschaftlichen als auch kommerziellen Gesichtspunkten entwickelt wurden und werden. Hierdurch zeigt sich das breite Spektrum an Anwendungsszenarien und die Variation an eingesetzten Demonstratoren. Die präsentierten Ansätze, mit ihren teilweise durch die Entwickler offengelegten Implementierungen, zeigen die praktische Anwendung der zuvor theoretisch geschilderten Architekturmodelle. So ergibt sich ein guter Überblick bezüglich aktueller Ergebnisse in der interaktiven Robotik, der als eine Grundlage für die praktische Umsetzung dieser Arbeit dient.

2.2.1. Marktorientierte und verfügbare Systeme

Roboter sind längst nicht mehr nur Gegenstand der Sciencefiction oder Forschungsobjekte in Laboratorien, sondern gewinnen auch aus marktwirtschaftlicher Sicht eine immer größere Aufmerksamkeit. Auch wenn die derzeitig eingesetzten Systeme in ihrem Erscheinungsbild und ihren Fähigkeiten noch weit von einem menschlichen Vorbild entfernt sind, so haben erste Exemplare bereits Einzug in unseren Alltag gefunden. Sie fungieren entweder als separate, intelligente Haushaltsgeräte oder sind vernetzt in unsere Umgebung integriert.

Intelligente Umgebung

Das bekannteste Beispiel für einen autonomen Roboter, der kommerziell erhältlich ist und im Haushalt hilft, ist der sogenannte Roomba (siehe Abb. 2.7(a)). Bei Roomba handelt es sich um einen autonom agierenden Staubsauger, welcher laut Herstellerangaben bereits in über eine Million Haushalten eingesetzt werde. Modellabhängig verfügt Roomba über Kontaktsensoren, visuelle Wahrnehmung mittels Infrarot, Radencoder für Odometrie und ein Mikrofon. Auch wenn

(a) Roomba 500 (b) Dimi (c) iCat

Abbildung 2.7.: Roboter als Bestandteil einer intelligenten Umgebung in Form von Haushaltsgeräten (a) oder als Kommunikationsschnittstelle (b, c).

das System in seiner beabsichtigten Anwendung autonom agiert, verfügt es über eine drahtlose Schnittstelle zur Kommunikation mit einem Personal Computer. Hierdurch kann der Roomba nicht nur als Staubsauger, sondern mit einem Anschaffungspreis von mehreren hundert Euro und seinen Sensoren auch als kostengünstige Roboterplattform genutzt werden [Tri07].

Ebenfalls als Bestandteil der Haushaltselektronik entwickelt werden die sogenannten *Smart Companions* der Firma Philips (siehe Abb. 2.7(b),2.7(c)). Ziel ist es, ein möglichst intuitives Interface für komplexe Haustechnik zu schaffen, die weitestgehend vor dem Anwender verborgen ist. Im Gegensatz zu einer intelligenten Umgebung ohne eine zentrale physikalische Repräsentation [Bul05], wurden mit Dimi [Bre05] und der iCat [vB04] künstliche Kommunikationspartner geschaffen. Diese sollen den Zugriff auf komplexe Informationen für den Benutzer intuitiver gestalten. Mittels natürlicher Sprache und Gesten können beispielsweise Informationen aus dem Internet oder digitalen Photoalben abgerufen und Anweisungen für das Aktivieren einzelner Haushaltsgeräte gegeben werden. Während Dimi als Prototyp für die Entwicklung nicht der Öffentlichkeit zugänglich gemacht wird, ist die iCat seit 2005 für Forschungseinrichtungen erhältlich. Die iCat zeichnet sich vor allem durch die Fähigkeit aus, Mimiken zu generieren und diese situationsbezogen als einfach zu interpretierendes Kommuniktionsmerkmal einzusetzen. Dies setzt voraus, dass die aktuelle Situation korrekt erkannt wurde.

Eine entsprechende situationsbezogene Software-Architektur präsentiert Croley in [Cro03]. Basierend auf den Handlungen von Personen in intelligenten Umgebungen werden Rückschlüsse bezüglich des gewünschten Systemverhaltens gezogen, ohne dass explizite Anweisungen seitens der Benutzer gegeben werden. Jede Situation ist als Verknüpfung von beobachtbaren Ereignissen definiert. Ein beobachtbares Ereignis definiert sich zum Beispiel durch die Anwesenheit einer bestimmten Person in Nähe eines künstlichen Kommunikationspartners. Verknüpft mit den Ereignissen, dass die Person vor kurzem die Wohnung betreten hat und nicht in Begleitung weiterer Personen ist, können unaufgefordert in Abwesenheit eingegangene private Nachrichten wiedergegeben werden. Ereignisse werden von einzelnen Modulen oder Modulketten detektiert, indem ein Modul Daten von Sensoren oder vorangehenden Modulen erhält und diese auswertet.

Bei jeder Auswertung werden Informationen mit einer zentralen *Supervisor*-Komponente ausgetauscht, die den Verarbeitungsprozess anhand von Regeln kontrolliert. Diese Regeln definieren, welche Situationen aus den Ereignissen abzuleiten sind und wie das System darauf reagiert. Auch wenn intelligente Umgebungen eine gute Voraussetzung für komplexe Sensorik und Situationserkennung liefern [Saf07], so sind sie nur mit hohem Aufwand zu installieren und auf einzelne, meist kaum vernetzte Haushaltsgeräte begrenzt. Für komplexere Aufgaben liegt es nahe, die bereits vorhandene und auf den Menschen zugeschnittene Infrastruktur zu verwenden, was wiederum komplexere autonome Plattformen erfordert.

Autonome Plattformen

Auch für autonome Plattformen hat sich inzwischen ein Markt verkäuflicher Roboter entwickelt. Aufgrund der noch hohen Kosten greifen jedoch meist nur Forschungseinrichtungen und größere Institute auf dieses Angebot zurück. Somit sind standardisierte Plattformen erhältlich, auf denen die Software für komplexe Interaktionsaufgaben entwickelt wird.

Mit der Zielsetzung, ein vollständiges System für den Privatanwender anzubieten, ist seit 2005 der Service-Roboter „Wakamaru“ (siehe Abb. 2.8(a)) von Mitsubishi Heavy Industries erhältlich. Das System verfügt über zwei Videokameras (eine davon omnidirektional), ein Mikrofon, Kontaktschalter auf Bodenhöhe sowie mehrere Ultraschall- und Infrarotsensoren. Die mitgelieferte Software [Shi06] ermöglicht multiple Personendetektion durch Bewegungserkennung im

(a) Wakamaru (b) Robovie R2 (c) Care-O-bot 3

Abbildung 2.8.: Mobile und autonome Plattformen die bereits auf dem Markt verfügbar sind (a,b) oder kurz vor der Einführung stehen (c). Während die Arme von Wakamaru und Robovie primär sozialen Aspekten dienen, ist der Manipulator von Care-O-bot für alltagsnahe Greifaufgaben konzipiert.

omnidirektionalen Kamerabild und anschließender Ausrichtung einer im Kopf positionierten Kamera mit angeschlossener Gesichtsdetektion. Das System soll bis zu zehn Personen erfolgreich unterscheiden und nach erfolgtem Training bis zu zwei Personen als seine Besitzer identifizieren können. Für die autonome Navigation müssen neben Odometriedaten Infrarotmarker als Landmarken in dem Arbeitsumfeld positioniert werden. Ultraschallsensoren und Bumper werden zur Kollisionsvermeidung und Detektion verwendet. Der Roboter bewegt sich mit Hilfe zweier angetriebener Räder sowie Stützrädern fort und ist nur für ebene Bereiche geeignet. Eingesetzt wird das System zur Umgebungsüberwachung und zur Kommunikation von hinterlassenen Nachrichten oder zur Erinnerung an Termine. Die Arme dienen ausschließlich der Bereicherung der Kommunikation mit einfacher Gestik.

Einen sehr ähnlichen Aufbau weist die Plattform Robovie auf, die in verschiedenen Konfigurationen angeboten wird. Abbildung 2.8(b) zeigt das verbreitete Modell „R2“. Ähnlich Wakamaru verfügt auch Robovie über eine omnidirektionale Kamera für periphere visuelle Wahrnehmung, zwei Kameras für Detailbetrachtung im Kopf und ein Mikrofon. Auch bei Robovie dienen die Arme der Vermittlung von Körpersprache und nicht zum Greifen von Objekten. Das zur Zeit im Test befindliche System Robovie IV [Mit06] verfügt über einen RF-Leser, je einen Laser Range Finder für Front- und Rückseite und zwei Gyro-Sensoren, deren Werte für Rauschunterdrückung gemittelt werden. Der größte Unterschied besteht jedoch in der Verwendung von 56 taktilen Sensoren an der Oberfläche des Roboters, die es dem System ermöglichen, Berührungen und Berührungsverläufe wie mit einer künstlichen Haut zu erfassen und zu interpretieren. Seitens der Plattformentwickler existieren bereits ein sprachbasiertes Dialogsystem sowie Personendetektion und -tracking durch videobasierte Gesichtsdetektion und laserbasierte Beinpaardetektion. Mit diesen Informationen ist es dem System möglich, Interaktionspartnern zu folgen. Komplexere Software, die die Kollisionsvermeidung bei gleichzeitiger Verwendung beider Arme [Sah04] realisiert, wird von den jeweiligen Forschungsgruppen selbst und nach ihren jeweiligen Bedürfnissen auf diesen Plattformen entwickelt.

Für die Manipulation von Objekten und somit nicht nur für Kommunikation und Überwachung bestimmt ist die dritte Version der Care-O-bot Reihe [Par08]. Care-O-bot III (siehe Abb. 2.8(c)) verfügt über einen Arm mit Drei-Finger-Greifer und taktilen Sensoren in den Fingern als Tastsinnersatz. Für weitere Möglichkeiten zur Wahrnehmung der Umgebung werden drei Laser Range Finder, zwei Farbvideokameras und ein infrarotbasierter Time-of-Flight Sensor auf der Roboterbasis benutzt. Die Basis besitzt einen omnidirektionalen Antrieb, so dass der Roboter seitlich fahren kann, um auch in beengten Bereichen navigieren zu können. Hierzu wird seitens der Entwickler an Software zur autonomen Navigation gearbeitet, genauso wie an Ansätzen für 3D-Objekterkennung, Objektmanipulation, Mensch-Maschine Interaktion und an einem Framework, um die jeweiligen Komponenten miteinander zu verknüpfen. Somit ist das System in der Lage, Karten bezüglich einer neuen Umgebung in Interaktion mit dem Anwender zu erstellen, wobei die Interaktion primär über ein Touchscreen, welches auch als Tablett von dem Roboter verwendet wird, realisiert ist. Zwar verfügt Care-O-bot über eine Sprachsynthese, Spracheingaben sind jedoch nicht unterstützt. Ist eine Karte aufgebaut, kann das System autonom zwischen zwei Punkten navigieren. Um einfache Bringdienste zu realisieren, existieren neben der Armansteuerung auch Objektdetektion und -identifikation. Diese basieren auf Punktwolken mit

sechsdimensionalen Feature-Vektoren, bestehend aus den drei Farbwerten der Videokamerabilder und den Punktkoordinaten des Time-of-Flight Sensors. Für die Wiedererkennung werden die Punktwolken, sich im Sichtbereich befindlicher Objekte, mit den bereits bekannten verglichen. Die Tiefen-Informationen werden nicht nur für die Objekterkennung verwendet, sondern ermöglichen auch, die Bewegungen eines Menschen in allen drei Dimensionen zu verfolgen, um daraus Gesten als weitere Interaktionsmodalität für die Mensch-Maschine-Kommunikation abzuleiten.

Androide Roboter

Um eine möglichst vollständige, intuitive Kommunikation unter Berücksichtigung der menschlichen Körpersprache zu erreichen, wird als eine Strategie die Menschenähnlichkeit von Robotern weiter erhöht, mit dem Ziel, die wahrnehmbaren Unterschiede zu minimieren. Aktuelle Systeme sind, aus einiger Entfernung oder durch das Bild eine Videokamera betrachtet, kaum mehr von ihrem menschlichen Vorbild zu unterscheiden. Hier beginnt die Domäne der sogenannten Androiden, dem Menschen zum Verwechseln ähnlich erscheinende Roboter.

Die Tendenz, Roboter menschenähnlicher zu gestalten, zeigt sich an dem südkoreanischen humanoiden Laufroboter Hubo [Oh06]. Dieses System wurde 2005 von Hanson Robotics mit einem menschlichen Kopf in Anlehnung an Albert Einstein versehen (siehe Abb. 2.9(a)). Während 41 Freiheitsgrade für Beine, Arme und Torso zur Verfügung stehen, werden weitere 16

(a) Albert HUBO (b) Repliee-Q2 (c) Geminoid (d) HRP-4C

Abbildung 2.9.: Die Androiden der heutigen Forschung (a-d) sehen dem Menschen bereits sehr ähnlich, verfügen aber kaum über autonome Verhaltensweisen, sondern werden noch größtenteils von den Entwicklern gesteuert.

Freiheitsgrade im Gesicht zur Erzeugung realistischer Mimiken verwendet. Durch den technischen Körper entsteht der Eindruck eines Druckanzuges. Befremdlich sind vor allem die unpassenden Proportionen des im Verhältnis zum Kopf zu kleinen Körpers.

Auf eine möglichst realitätsnahe Umsetzung setzt die Forschungsgruppe um Hiroshi Ishiguro. Mit dem in Abbildung 2.9(b) dargestellten Demonstrator *Repliee-Q2* wurden die Durchschnittsmaße einer asiatischen Frau umgesetzt. Repliee verfügt über 42 Aktuatoren in Oberkörper, Gesicht und Armen. Der Android ist somit nicht in der Lage, sich fort zu bewegen. Dafür sind auch kleinere und unbewusste Bewegungsabläufe wie Liedschlag und Atmung realisiert und geben dem System im Sitzen eine sehr realistische Erscheinung. Um diese Ähnlichkeit noch stärker in den Vordergrund zu stellen, wurde mit *Geminoid* [Nis07] ein Android nach dem Aussehen Hiroshi Ishiguros selbst entwickelt, der seinem Entwickler auf einiger Distanz zum Verwechseln ähnlich sieht (siehe Abb. 2.9(c)). Tatsächlich wurde das System bereits als Tele-Präsenz von Professor Ishiguro in Veranstaltungen verwendet. Die Steuerung erfolgt bei beiden Systemen über vordefinierte Bewegungsabläufe oder das Ganzkörper-Tracking eines Menschen in Echtzeit. Forschungsziel sind primär Verhaltensstudien menschlicher Interaktionspartner mit dem Androiden [Ish07]. Die Ergebnissen sollen bei der Verbesserung zukünftiger Androiden hinsichtlich ihrer Akzeptanz in der Allgemeinheit dienen.

Der Allgemeinheit durch die Medien während der 8. Japanischen Modewoche (März 2009) in Tokio vorgestellt wurde HRP-4C (siehe Abb. 2.9(d)). Ebenfalls nach den durchschnittlichen Maßen und Proportionen einer japanischen Frau modelliert ist der Androide 1.58 Meter groß. Er wiegt 43 Kilogramm und kann sich mit 39 Aktuatoren nicht nur menschenähnlich bewegen, sondern mit acht zusätzlichen Aktuatoren im Gesicht auch einfache Mimiken produzieren. Bislang sind alle Bewegungen vorgegeben oder werden zur Laufzeit von den Entwicklern gesteuert. Angestrebt ist jedoch ein autonomes Verhalten basierend auf optischen und akustischen Sensordaten von HRP-4C. Auch wenn die technische Leistung bei der Schaffung immer realistischer wirkender Androiden beeindrucken, so sind autonome Interaktionsfähigkeiten auf diesen Systemen bislang kaum existent.

2.2.2. Integrierte Systeme in der Forschung

Für Systeme im Alltagseinsatz sind autonome Interaktionsfähigkeiten eine Grundvoraussetzung, schließlich kann nicht jeder Roboter von einem Menschen ferngesteuert werden, sondern hat die ihm zugeteilte Aufgabe selbstständig auszuführen, um seinem Anwender von Nutzen zu sein. Hierbei müssen zukünftige Systeme ihre Umgebung wahrnehmen und sich mit den gewonnen Informationen auch bei Abweichungen von erwarteten Ereignissen im Alltag bewähren. Das Ziel dieser Forschung ist ein System, welches nicht nur eine gegebene Aufgabe erfolgreich bewältigt, sondern auch die Intention der Aufgabe erkennt. Bisweilen fokussiert sich dieses Ziel in möglichst zuverlässigen und gleichermaßen intuitiven Benutzerschnittstellen, die einzelne Fähigkeiten des Menschen von separaten Sinneswahrnehmungen bis hin zu sozialer Kommunikation mittels Körpersprache modellieren.

Für einen Demonstrator werden mehrere Fähigkeiten in ein System integriert, um ein möglichst eigenständiges System zu erhalten. Ein beliebtes Szenario, in dem Fähigkeiten der klassischen Robotik, wie autonome Navigation, mit sozialkompetenten Interaktionsstrategien verknüpft werden, ist der sogenannte *Tour Guide*. In Ausstellungen dient ein Roboter dazu, Besuchern unterschiedliche Exponate zu zeigen. Dafür muss das System nicht nur erfolgreich in dem bekannten Areal navigieren ohne mit den sich bewegenden Besuchern zusammen zu stoßen, sondern auch auf die Reaktionen der von ihm geführten Gruppe eingehen. Diese Bedürfnisse können sowohl intuitiver Art, wie das Einhalten sozialer Abstände, als auch direkter Art, wie das Nachfragen detaillierterer Informationen bezüglich der Ausstellung, sein.

Bennewitz et al. [Ben05] entwickelten mit Alpha einen humanoiden Roboter, der auf Ausstellungsstücke in seiner Umgebung verweist und mit mehreren Besuchern simultan interagiert. Das 1,55 Meter große und aus Karbon-Verbundwerkstoff gefertigte System verfügt über 17 Freiheitsgrade in Armen, Beinen und Rumpf, sowie weitere 16 Freiheitsgrade am Kopf, von denen drei für die Kontrolle zweier, an Augenposition fixierter, Kameras dienen. Neben den Kameras verfügt Alpha über zwei Mikrophone. Während über die Kamerabilder Gesichter detektiert werden, dienen die Mikrofone sowohl der Spracherkennung, als auch der Richtungswahrnehmung des Sprechers. Um mit mehreren Anwesenden Personen interagieren zu können, wird jedem Interaktionspartner ein Aufmerksamkeitswert zugeordnet, abhängig von seiner Position zum Roboter und der letzten Wortäußerung. Alpha wendet sich der Person mit dem aktuell höchsten Aufmerksamkeitswert zu. Um die Aufmerksamkeit eines Besuchers auf ein Exponat zu lenken, richtet das System seinen Kopf auf das Objekt und führt anschließend eine Zeigegeste mit dem Arm aus. Abhängig von der jeweiligen Interaktionssituation generiert Alpha verschiedene Gesichtsausdrücke, basierend auf den Basisemotionen nach Ekman [Ekm72]. In mehreren Schritten wurde die Körpersprache des Systems von reinen Zeigegesten um komplexere Mimiken und sprachbegleitende Gesten auf eine intuitiv verständliche Rückmeldung [Ben07] von Systemzuständen erweitert. Dies führten zu der Entwicklung des aktuell verwendeten Demonstrators *Robotinho* (siehe Abb. 2.10(a)).

Ebenfalls mit dem Fokus auf soziale Interaktion gestaltet, ist der in Abbildung 2.10(b) gezeigte Roboter Maggie. Das Erscheinungsbild wurde auf eine dem Menschen möglichst sympathische Form abgestimmt [Sal06]. Neben einer Kamera, Ultraschall- sowie Infrarotsensoren und einem Laser-Range-Finder verfügt der Roboter über multiple Taktilsensoren an Kopf, Schultern, Armen und Rumpf. Die Detektion und das Tracking von Personen erfolgt in erster Linie visuell über die in Taillienhöhe montierte Kamera, während die Taktilsensoren es dem Roboter ermöglichen sollen, komplexere soziale Fähigkeiten, wie vom Menschen geführten Paartanz, auszuüben. Da Maggie nur über relativ wenige Freiheitsgrade für eine differenzierte Körpersprache verfügt, - jeweils einen pro Arm, zwei im Halsbereich und zwei im Kopf für bewegliche Augenlider - werden halbtransparente Teilbereiche des Roboters wie Mund und Rumpf rückseitig mit verschiedenfarbigen Lichtern illuminiert. Die Ansteuerung der Hardware sowie die Koordination von Interaktionsfähigkeiten ist in einer hybriden Software-Architektur realisiert, deren deliberativen Komponenten für komplexe Interaktionsaufgaben über ein gemeinsames Arbeitsgedächtnis mit den reaktiven Komponenten für Hardwareansteuerung und Perzeption verbunden sind. Die Koordinierung wird von einem emotionsbasierten zentralen Kontrollmodul [Mal04] erreicht, indem

(a) Robotinho (b) Maggie

Abbildung 2.10.: Die Forschungsroboter Robotinho und Maggie verfügen über autonome Verhaltensweisen und machen von Gestik und Mimik in der Interaktion mit Menschen Gebrauch.

die unterschiedlichen Aktionen und Zustände des Roboters Emotionen zugeordnet werden, wobei das System versucht eine möglichst hohen Grad an „Zufriedenheit“ zu erlangen.

Eine alternative Kontrollvariante, das *Procedural Reasoning System* (*PRS*) kommt bei dem Demonstrator Rackham (siehe Abb. 2.11(a)) zum Einsatz. Rackham wurde über 100 mal in einem mehr als zweieinhalb Jahre langen Zeitraum als Tour-Guide in einem Museum eingesetzt [Clo06]. Mittels Sprache oder Touch-Screen konnten die Besucher den Roboter um eine Tour zu unterschiedlichen Exponaten bitten. Dabei sind bezüglich der Navigation andere umherstehende Besucher zu berücksichtigen, ohne die geführten Gäste zu verlieren. Neben einer vorgegebene Karte der Ausstellung verwendet der 1,18 Meter große Roboter Daten einer Pan-Tilt-Videokamera, eines Laser-Range-Finders, Mikrofons und Gyroskops, um die Umgebung wahrzunehmen. Als Rückmeldung für die Benutzer dienen Touch-Screen-Display, Sprachausgabe, und wie bei Maggie, eine farbliche Illumination des Roboters. Die Personenaufmerksamkeit basiert auf Gesichtsdetektion anhand der Videodaten und Abweichungen der gemessenen Laser-Ranges von der vorgegebenen Karte. Die Interaktionsfähigkeiten von Rackham und dessen verwendeter Systemarchitektur [Ala98] sind als separate Module mit einheitlichen Schnittstellen angelegt. Das Grundgerüst eines solchen Moduls wird automatisch generiert und bettet den eigentlichen Algorithmus für einen Arbeitsablauf (*Procedure*) ein. Auf die einheitlichen Schnittstellen setzt ein Interpreter des eingangs erwähnten Procedural Reasoning Systems auf. Dieser Interpreter ruft je nach Situations- und Anforderungsanalyse, dem Reasoning, ein Modul und dessen Arbeitsablauf auf. Die für einen Ablauf notwendigen Daten werden über einen von allen Komponenten gemeinsam verwendeten Speicherbereich ausgetauscht, so dass im Idealfall keine direkten Modulverbindungen existieren.

Eine noch stärkere Modularisierung wird in dem *CoSy Architecture Shema* (*CAS*) [Haw08] propagiert. In CAS werden einzelne Module in semantischen Einheiten zusammengefasst, wie zum Beispiel Navigation, Lokalisation und Hindernisvermeidung. Ein Teil dieser Module kann durch einen Task-Manager koordiniert werden, während andere Module der Einheit autonom agieren. Der Datenaustausch wird ebenfalls über gemeinsam genutzten Speicher realisiert. Um komplexe Aufgaben mit den Demonstratoren zu erfüllen, werden mehrere semantische Einheiten miteinander verbunden, deren Datenaustausch durch Verknüpfung der jeweiligen Speicherbereiche geschieht. Die Aktivierung der Module innerhalb jeder Einheit geschieht automatisch bei Veränderung des jeweiligen Speicherinhaltes, indem diese Modifikation an alle beteiligten Module kommuniziert wird und diese darauf reagieren. Die Flexibilität dieses Ansatzes zeigt sich in dessen Domänenunabhängigkeit, welche auf den Robotern Minnie 2.11(b) und Playmate 2.11(c) in zwei unterschiedlichen Szenarien demonstriert wird.

Minnie [Moz07] basiert auf einer PeopleBot Plattform von MobileRobots Inc. und erlernt in Kooperation mit einem Anwender die Umgebung in der das System eingesetzt wird. Minnie verfügt über einen Laser-Range-Finder, mit dem sowohl Umrisse der Umgebung aufgezeichnet werden, als auch nach Beinpaaren für Personendetektion und -tracking gesucht wird. Mit einer zusätzlichen Videokamera werden unterschiedliche Einrichtungsobjekte erfasst. Sowohl Laser- als auch Bildinformationen werden für die Generierung einer Karte, beziehungsweise das Wiedererkennen von Umgebungen verwendet. Über Sprachanweisungen können Objekte oder Umgebungsbereiche benannt werden, während der Roboter dem Anwender folgt. Nach erfolgreichem Training ist das System in der Lage, autonom zwischen bekannten Zielen zu navigieren und beispielsweise Besucher herum zu führen.

(a) Rackham (b) Minnie (c) Playmate

Abbildung 2.11.: Die gezeigten Systeme verfügen über ein weniger soziales Erscheinungsbild, zeichnen sich aber durch eine hohe Autonomie in Navigation (a), Umgebungswahrnehmung (b) und Objektmanipulation (c) aus.

In dem Playmate-Szenario [Haw09] kennt der Demonstrator verschiedene einfache Spiele, wie zum Beispiel das Zuordnen von Farben oder Formen. Es ist die Aufgabe des Roboters, selbstständig anhand von Bewegungen und verwendeten Objekte zu erkennen, welches Spiel der Interaktionspartner spielt und, unter Berücksichtigung der Spielregeln, daran teilzunehmen. Der als Playmate bezeichnete Demonstrator besteht aus einer iRobot B21r Plattform, auf die eine Pan-Tilt-Einheit mit zwei Videokameras zur Betrachtung des Spielfeldes montiert ist. Für die Manipulation von Spielsteinen wird ein Arm der Katana-Baureihe von Neuronics verwendet, an dessen Endeffektorgelenk eine weitere Kamera befestigt ist. Diese dient der genauen Überwachung von Greifvorgängen. Die Interaktion mit dem Benutzer während eines Spiels basiert auf Spracherkennung und -synthese und beschränkt sich auf die Abfrage der aktuellen Spielsituation, beziehungsweise der Aufforderung einen Zug zu machen.

Auch wenn Benutzerinteraktion nicht im Fokus der von Santoro [San07] vorgestellten Arbeit steht, so weist die Architektur für einen Roboter der verschiedene Arten von Müll einsammeln und getrennt verstauen soll, einen interessanten Ansatz auf. Diese Aufgabe wurde von dem Demonstrator *Caesar* auf dem Eurobot Wettbewerb 2007 absolviert. Die verwendete Architektur, die Komponenten wie Bilderkennung zur Klassifikation des Mülls und Fahrplanung beinhaltet, basiert auf der in der Telekommunikation verbreiteten Skript-Sprache Erlang. In Erlang werden nicht nur die einzelnen Module verfasst, sondern auch ein Set von Regeln, welches zur Systemkoordinierung verwendet wird. Die Trennung von Modulen und Regeln ermöglicht eine deutlich höhere Variabilität in der Ausführung unterschiedlicher Aktionen und dem Hinzufügen neuer Verhaltensweisen durch das ändern, oder Hinzufügen von Software-Komponenten. Gleichzeitig wird durch die Wahl der gleichen Repräsentation der Aufwand für technische Ressourcen, wie Parser, und die Einarbeitungszeit der Entwickler in unterschiedliche Repräsentationsformen reduziert.

2.3. Zusammenfassung

Unterschiedliche Architekturen für interaktive Roboter zeigen ein breites Spektrum an erfolgreichen Strategien wie deliberative, reaktive, hybride und verhaltensbasierte Kontrolle. Jede Strategie besitzt spezifische Vor- und Nachteile. Es ist nicht möglich ein Verfahren generell einem anderen vorzuziehen. Ein automatisierter Staubsauger muss nicht über eine komplexe hybride und mehrschichtige Systemarchitektur verfügen, wenn das System mit zwei reaktiven Verhalten zur effizienten Abdeckung der zu reinigenden Fläche und dem Ausweichen von Hindernissen auskommt. Eine rein deliberative Architektur kann die sinnvollste Wahl darstellen, wenn es um die logische Ableitung von Wissen in Auskunftssystemen geht, sofern diese nicht zeitkritisch auf spontane Veränderungen in ihrer Datenbasis reagieren müssen. Der jeweilige Anwendungsbereich ist entscheidend für die Wahl einer erfolgreichen Integrationsstrategie und der zugrunde liegenden Architektur.

Gleiches gilt auch für die unterschiedlichen Hardware-Plattformen und die auf ihnen zum Einsatz gebrachte Software. Für Wartungs- und Reinigungsaufgaben bedarf es keiner aufwendig

dem Menschen nachempfundenen Demonstratoren, unabhängig ob es sich um Androide oder anthropomorphe Systeme mit menschenähnlichen Zügen handelt. Im Rahmen sozialer Kontakte, wie zum Beispiel der Pflege von Menschen, werden menschliche Maschinen mit menschlichen Attributen versehen, die intuitive Kommunikation zwischen Roboter und Anwender unterstützen und die Bedienung vereinfachen.

Betrachtet man die Entwicklung der nicht zur industriellen Produktion eingesetzten Roboter, so zeigt sich ein starkes Interesse an Systemen, die den Menschen in seinen alltäglichen Aufgaben unterstützen und in dessen privaten Umfeld agieren. Interaktive Roboter reagieren auf mehrere Eingabemodalitäten wie Sprache, Gestik, Mimik oder Berührung und sind teilweise in der Lage, diese zur Informationsausgabe zu verwenden. Während ein Teil der Entwicklung auf, für spezifische Einzelaufgaben konstruierte Systeme, wie Reinigung oder Überwachung ausgelegt ist, so werden für komplexe Aufgaben mehrere Fähigkeiten kombiniert. Wie bei den hier vorgestellten Systemen gezeigt, ist das Ziel, bei der Realisierung eine möglichst flexible und modulare Integrationsstrategie zu verfolgen. Diese ermöglicht es den Entwicklern, aus Experimenten neu gewonnene Erkenntnisse einfacher einzubeziehen. Dennoch zeigen die aktuellen Arbeiten immer noch einen hohen Grad an Domänenabhängigkeit bei der Architektur. Besonders bei Ansätzen mit hohem Planungsaufwand oder Subsumption-Modellen sind Veränderungen am Systemverhalten oder neue Verhaltensweisen mit großem Implementationsaufwand verbunden. Angestrebt wird ein einfach zu verstehender Aufbau, in dem alle Komponenten nach gleichem Verfahren auf gemeinsame Daten zugreifen und diese zur Verfügung stellen können, mit einer Systemkoordination, die dem gleichen Transparenzkriterium unterliegt, ohne Top-Down das gesamte System zu dominieren.

Mit dieser Arbeit wird eine Systemarchitektur vorgestellt, die nach diesem Prinzip *Glas-Box-Knowledge* einen höchst möglichen Grad an Flexibilität sowohl bei der Integration neuer Fähigkeiten, als auch bei der Domänenunabhängigkeit bietet. Sie wird bei zwei vollkommen unterschiedlichen Demonstratoren und in deren Einsatzszenarien angewendet. Ein modularer Aufbau und ein von den Modulen separiertes Regelwerk zur System- und Aufgabensynchronisation ermöglichen das Hinzufügen neuer Komponenten und die kontinuierliche Weiterentwicklung, genauso wie die Integration vorhandener Software, ohne diese zu modifizieren. Einen Einblick in diese als Grundlage für die vorliegende Arbeit verwendete Software und die Demonstratoren, auf der sie in den unterschiedlichen Szenarien eingesetzt wird, bietet das folgende Kapitel.

3. Entwicklungsbeispiele der Mensch-Roboter-Interaktion

Die Überlegungen im vorangegangenen Kapitel beschreiben zwar, welche Aspekte bei dem Design integrierter Systeme zu berücksichtigen sind, doch fokussieren die Implementierungen einzelne Aspekte einer umfangreicheren Mensch-Roboter- Interaktion. Der Verknüpfung dieser Aspekte und der kontinuierlichen Weiterentwicklung eines Robotersystems wird demgegenüber nur wenig Bedeutung beigemessen. Es ist die feste Überzeugung des Autors, das eben dieser Aspekt kontinuierlicher Erweiterbarkeit und der damit einhergehenden notwendigen Systemflexibilität ein wichtiger Schritt in Richtung alltagstauglicher Roboter ist.

Unter dem Gesichtspunkt kontinuierlicher Entwicklung und fortgesetzter Erweiterung der Systemfähigkeiten werden in diesem Kapitel zwei unterschiedliche Demonstratoren vorgestellt. In der Entwicklung der Demonstratoren wurden bestehende Szenarien stetig erweitert und neue Softwarekomponenten integriert. Dieses Kapitel stellt die Evolution der beiden Demonstratoren und die während dieser Evolution entwickelten Komponenten dar. Ziel ist es, einen Überblick bezüglich der Komplexität eines solchen Systems zu verschaffen und die Notwendigkeit eines geeigneten Integrationskonzeptes zu unterstreichen.

Durch die Wahl zweier unterschiedlicher Demonstratoren bietet sich die Möglichkeit, die Flexibilität des Integrationskonzeptes zu überprüfen. Dies gilt im Besonderen, da sich die Demonstratoren nicht nur im Aufbau ihrer Hardware unterscheiden, sondern auch in jeweils individuellen Interaktionsdomänen eingesetzt werden.

3.1. Szenariobezogene Systemintegration

Ist es möglich, eine Systemarchitektur so zu gestalten, dass sie vollkommen unabhängig von ihrem Anwendungsgebiet ist und dennoch ausreichend spezifisch, um den Entwicklern klare Richtlinien für ihre Arbeit aufzuweisen? Die Beantwortung dieser Frage fällt nicht leicht. Es lässt sich aber ein Ziel aus der Fragestellung ableiten: die Systemarchitektur so flexibel wie möglich zu halten und gleichzeitig klare Richtlinien für die Entwickler anzubieten. Der Vorteil liegt nicht nur in der ressourcenschonenden Wiederverwertbarkeit der Systemarchitektur, sondern auch in der Wiederverwertbarkeit einzelner Systemkomponenten, die unter den Richtlinien

der Architektur entwickelt wurden. Als Beispiel werden im Folgenden zwei Szenarien im Bereich der Mensch-Maschine-Interaktion (*engl. Human-Robot-Interaktion, HRI*) vorgestellt, in denen die beteiligten Demonstratoren auf der gleichen Systemarchitektur basieren und sich eine Vielzahl von Software-Komponenten teilen.

3.1.1. Ein mobiler Entdecker auf Home-Tour

Das *Home-Tour-Szenario* spielt in einer nahen Zukunft, in der Roboter bereits übliche Haushaltsgeräte sind und vergleichbar mit einer Waschmaschine bestellt und geliefert werden. Nach einer kurzen, maximal zehnminütigen Instruktion durch den Zusteller ist der neue Besitzer in der Lage, mit dem System umzugehen. Der Roboter stellt sich und seine Fähigkeiten leicht verständlich vor. Um nun Aufgaben erledigen zu können, bedarf das System einer Einführung in seine neue Arbeitsumgebung - einen Privathaushalt. Hierfür zeigt der Besitzer dem Roboter seine Wohnung. Die Art dieser Wohnungsbegehung geschieht in gleicher Weise, wie bei einem menschlichen Gegenüber: durch die Nutzung von Sprache, ohne an spezielle Kommandos gebunden zu sein, und Gesten, um auf Bereiche oder Objekte innerhalb der Wohnung zu verweisen und sie verbal zu benennen. Während der Home-Tour folgt der Roboter dem Benutzer und richtet seine Kamera auf Objekte aus oder sieht sich in neuen Räumen um. Dieser Lernvorgang ist essentiell, da es sinnlos wäre das System für Privatumgebungen vorzuprogrammieren. Selbst wenn zukünftige Besitzer eine Komplettansicht ihrer Wohnung mit der Bestellung an den Hersteller lieferten, was aus Gründen der Privatsphäre sicher nicht allen Anwendern recht wäre, so müsste bei jeder größeren Neugestaltung der Wohnung der Roboter durch einen Fachmann neu programmiert werden. Das interaktive Kennenlernen mit dem Benutzer umgeht dieses Problem, da dem Roboter Veränderungen auf möglichst menschenähnliche Art und Weise gezeigt werden können.

Ziel des beschriebenen Szenarios ist es, einen Roboter zu entwickeln, der Menschen im Alltag bei der Bewältigung ihrer Aufgaben unterstützt. Es ist offensichtlich, dass die Aufgabe einen Roboter mit den beschriebenen Fähigkeiten zu entwickeln, eine enorme Komplexität beinhaltet. In dem Fall des hier beschriebenen Systems liegt der Schwerpunkt auf der intuitiven und sozial geprägten Interaktion zwischen Benutzer und Roboter in einer natürlichen Umgebung. Im Rahmen des vorgestellten Systems und dieser Arbeit ist unter dem Begriff „intuitiv“ die einfache Benutzbarkeit des Roboters zu verstehen, so dass es nur eine kurze Einführung in das System gibt, vergleichbar mit der Gerätevorführung eines Verkäufers in einem Fachmarkt. Danach sollen Anwender und Roboter gemeinsam Aufgaben erfolgreich bewältigen. Der soziale Aspekt wird berücksichtigt, indem nicht nur Informationen auf sprachlicher Ebene ausgetauscht werden, sondern auch Aspekte wie Prosodie und Körpersprache mit Gestik und Mimik in die Interaktion einfließen. Auch wenn ein Roboter diese Modalitäten nur bedingt selbst vermitteln kann, so kann er dennoch auf sie reagieren.

Für eine möglichst natürliche Interaktion spielt die Umgebung eine entscheidende Rolle. Studien werden daher nicht in einem Labor durchgeführt, sondern in einem echten und zwischenzeitlich bewohnten Appartement (siehe Abb. 3.1), welches extra für die Entwicklung des Systems angemietet ist. Somit ergeben sich Situationen, die unter Laborbedingungen nicht auftreten, die aber

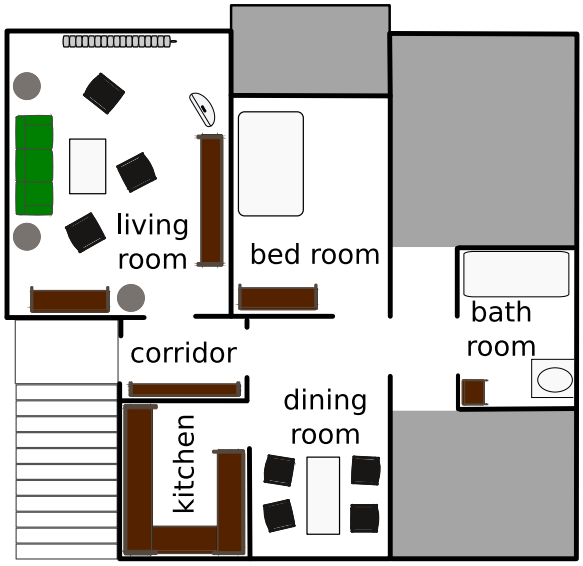


Abbildung 3.1.: Grundriss des Appartements, welches für die Home-Tour genutzt wird, mit den verwendeten Bezeichnungen für die Räume. Der Anwender führt den Roboter durch die verschiedenen Zimmer oder Bereiche der Wohnung. Er benennt die Zimmer und zeigt dem Roboter Einrichtungsgegenstände, die sich in den jeweiligen Zimmern befinden.

ein alltagstauglicher Roboter bewältigen muss. Seien es unterschiedliche Bodenbeläge, -höhen und Schlupf, verschiedene Licht- und Farbbedingungen oder die sich verändernde Akustik beziehungsweise Störgeräusche. Private Wohnungen sind weniger strukturiert und individuell nach ihren Bewohnern eingerichtet. Gegenstände können an unerwarteten Orten (insbesondere auf dem Boden) herumliegen und sind somit nur bedingt für den Roboter sichtbar. Die Unvorhersehbarkeit von Arbeitsumgebung und Benutzer verlangen ein Höchstmaß an Robustheit und Fehlertoleranz von dem System, welche zur Unterstützung der Entwickler bereits auf Architekturebene zu berücksichtigen sind. Unerwartete Fehlerquellen, die das Systemverhalten beeinflussen, sind optimiert über einfache Mechanismen in der Architektur anzupassen und nicht durch langwierige Modifikation von Algorithmen im Programm-Code. Beispiele hierfür sind unerwarteter Schlupf, der den Roboter Bewegungen nicht vollständig ausführen lässt oder Gegenlicht, welches die visuelle Wahrnehmung behindert. Recovery-Strategien auf Architekturebene erlauben in diesen Fällen ein Fortsetzen der aktuell durchgeführten Aktion, ohne dass alle Eventualitäten in Modulen berücksichtigt werden müssen. Ergeben sich überraschend neue Herausforderungen in der Arbeitsumgebung, unterstützt eine geeignete Architektur durch ihre Flexibilität die schnelle Anpassung des Systems.

3.1.2. Rezeptionist mit Körpersprache

Der Fokus bei diesem Szenario liegt auf dem Effekt, den ein humanoider Roboter in der Interaktion gegenüber Menschen bewirkt. Hierbei werden zum einen die Auswirkungen des rein äußerlichen Erscheinungsbildes untersucht und zum anderen der Einsatz non-verbaler Kommunikation in Form von Gesten und Mimiken. Im Vergleich zu Robotern mit einem funktionalerem Design (siehe Abschnitt 2.2.2) ist zu erwarten, dass dem Menschen nachempfundene Körpersprache das intuitive Verständnis verstärkt. Eine Grundlage für diese Annahme bildet die Beobachtung von Mensch zu Mensch Interaktion, in der Körpersprache in Abhängigkeit von dem erwarteten Verständnis des jeweiligen Gegenübers unterschiedlich stark ausgeprägt wird. Ein humanoider Roboter bietet die Möglichkeit, diese Erkenntnisse auf einen Demonstrator zu übertragen [Wre07].

Eine sehr einfach zu interpretierende Kategorie der Körpersprache sind Gesten, die auf den Versuch eine Gegenstand zu erreichen hinweisen [Sun05] oder als Zeigegesten auf Objekte referenzieren und somit die Aufmerksamkeit aller Beteiligten lenken [Bro06]. Unter diesem Gesichtspunkt wurde das Rezeptionistenszenario entwickelt. Der Roboter gibt als Empfangsmitarbeiter Auskunft über Orte und Objekte in seiner Umgebung. Dafür ist er, wie in Abbildung 3.2 dargestellt, hinter einem Tisch positioniert. Ein Interessent kann von der anderen Tischseite an den Roboter herantreten und ihn befragen. Dabei werden nicht nur natürliche Sprache sondern auch Gesten verwendet, um Richtungen anzugeben oder direkt auf Gegenstände in der Nähe zu zeigen.

Abbildung 3.2.: Interaktionsszenario mit einem humanoiden Roboter. Der Roboter ist als Empfangsmitarbeiter hinter einem Tisch positioniert, vergleichbar mit einem sitzenden Menschen und reagiert auf die Anfragen von ihm gegenüber befindlichen Personen.

So wird es dem menschlichen Gegenüber ermöglicht, die gleichen Kommunikationsmodalitäten zu wählen. Eine erfolgreiche Implementierung, in der ein Interaktionspartner nicht nur sprachlich, sondern auch durch Zeigegesten auf eine auf dem Tisch liegende Karte der Umgebung nach interessanten Bereichen fragen kann, ist in [Beu08] beschrieben.

Um die Interaktion flüssiger zu gestalten und die Akzeptanz zu steigern, selbst wenn es aus technischen Gründen zu Verzögerungen beim Antwortverhalten kommt, haben sich Mimiken als sehr vorteilhaft erwiesen. In einer ersten Studie [Heg06] zeigte sich, dass emotionale Mimiken, die auf der aktuellen Situation in einem von der Versuchsperson vorgetragenen Märchen beruhen, die Bewertung der Menschenähnlichkeit deutlich verbessert. Zur Durchführung wurde ein auf der Arbeit von Ekman et al. [Ekm72] basierendes Emotionsmodell entwickelt, welches sowohl die unterschiedlich starke Ausprägung von Emotionen, als auch den Übergang zwischen diesen realisiert. Die Implementierung des Modells wurde in das bestehende Robotersystem integriert und die vorhandenen grundlegenden Fähigkeiten zur Detektion von und Interaktion mit mehreren Personen entsprechend erweitert [Spe07b]. Die Grundlage für diese Arbeit wurde allerdings ursprünglich für den im Home-Tour-Szenario eingesetzten Demonstrator entwickelt.

3.2. Demonstratorbezogene Systemintegration

Die Flexibilität einer Systemarchitektur zeichnet sich nicht nur bezüglich des Anwendungsszenarien des betreffenden Systems aus, sondern im Bereich Mensch Roboter Interaktion auch gegenüber ihrer Anwendung auf unterschiedliche Demonstratoren. Obwohl es naheliegend erscheint, dass in unterschiedlichen Szenarien auch unterschiedliche Demonstratoren verwendet werden, ist dies keineswegs selbstverständlich. In den zuvor geschilderten Szenarien werden zwei unterschiedliche Demonstratoren eingesetzt. Obwohl sich ihre technischen Spezifikationen hinsichtlich Sensoren und Aktuatoren deutlich unterscheiden, ist es das Ziel, unter der gleichen Architektur möglichst viele Komponenten auf diesen und weiteren Demonstratoren, ohne umfangreiche Modifikationen, einsetzen zu können. Für einen Überblick bezüglich der technischen Unterschiede wird der Aufbau der beiden Demonstratoren im Folgenden beschrieben.

3.2.1. Der funktionale Assistenzroboter BIRON

BIRON, der **Bi**elefeld **Ro**bot Companio**n**, ist seit 2001 Gegenstand der Forschung für Mensch-Roboter-Kommunikation. Als Demonstrator im Home-Tour-Szenario liegt der Forschungsschwerpunkt auf der Schaffung eines Assistenzroboters, der in privaten Haushaltsumgebungen seine Anwender bei alltäglichen Aufgeben unterstützt, ohne dass die Benutzer über umfangreiches Wissen für die Bedienung des Systems verfügen.

BIRON basiert auf der *Pioneer PeopleBot*™ Plattform von *MobileRobots*, ehemals *ActiveMedia*. Diese wurde, wie in Abbildung 3.3 dargestellt, um Sensoren zur detaillierten Wahrnehmung von Personen und Umgebung ergänzt. Ein Differentialantrieb ermöglicht platzsparendes

(a) Roboter BIRON (b) Sensorausstattung

Abbildung 3.3.: Die Hardware des **Bi**elefeld **Ro**bot Companio**n** (3.3(a)) basiert auf dem *Pioneer PeopleBot*^{TM}und wurde für eine verbesserte Umgebungswahrnehmung um mehrere Sensoren (3.3(b)) ergänzt (von oben nach unten):*Sony EVI D31* Pan-Tilt-Videokamera, *AKG C 400 BL* Mikrofon, *Videre STH-MDCS-C* Stereo-Videokamera, *SICK LMS200* Laser-Range-Finder

Maneuvrieren im Raum. Ein drittes, nicht angetriebenes Stützrad an der Hinterseite des Basis dient zur Stabilisierung. Des Weiteren werden sowohl Wegaufnehmer an den Radachsen zur Positionsbestimmung über Odometrie, als auch Kontaktsensoren auf Bodenhöhe, die bei Kollision mit einem Hindernis auslösen, verwendet.

Auf die Basis aufgesetzt ist der turmförmige Hauptkörper des Systems. Die Ultraschallsensoren wurden deaktiviert und für die Tiefenwahrnehmung ein Laser-basiertes Distanzmessgerät (siehe Abb. 3.3(b), ganz unten), Typ *LMS200* der Firma *SICK* eingebaut. Dieses ist mittig auf der Rotationsachse der Basis montiert und zeichnet einen Bereich von 180° vor dem Roboter mit einer Entfernung von bis zu acht Metern auf. Hierzu werden mit einer Frequenz von vier Hertz jeweils 361 Abstandsmessungen mit einer Richtungsauflösung von 0,5° durchgeführt. Das Gerät liefert somit einen Querschnitt der Umgebung vor dem Roboter in der Montageebene von 30 Zentimetern über dem Boden. Im Home-Tour-Szenario wird dieser Querschnitt nicht nur für die Hinderniserkennung verwendet, sondern auch zur Suche nach Charakteristika für Personen, wie Beinpaare und für das Lernen der Umgebung durch Größenvermessung der verschiedenen Räume genutzt.

Oberhalb des Lasers in einer Höhe von circa 90 Zentimetern befinden sich zwei Lautsprecher, die für die Sprachsynthese während der Benutzerinteraktion verwendet werden. Eine Stereokamera für 3-D Zeigegestendetektion ist kurz unterhalb des zweiten Sonarkranzes angebracht. Es handelt sich um eine *Videre STH-MDCS-C* die genauer in Abbildung 3.3(b) (Mitte unten) zu

sehen ist. Wie bei dem menschlichen Sehsinn werden die Bildinformationen von 2 separaten Kameras mit konstantem Abstand aufgezeichnet und dann miteinander verrechnet, um die Tiefeninformation zu erhalten. Zur Richtungsdetektion von Schallquellen, wie einem sprechenden Menschen, sind auf der oberen Ebene des Hauptkörpers zwei *AKG C 400 BL* Grenzflächenmikrofone (siehe Abb. 3.3(b), Mitte oben) aufgesetzt. Zwischen den Mikrofonen befindet sich eine Halterung sowohl für eine weitere Kamera als auch für ein Touch-Screen-Display. Als Kamera findet eine *Sony EVI D31*, wie in Abbildung 3.3(b) (ganz oben) dargestellt, Verwendung. Die Kamera verfügt über eine integrierte Pan-Tilt-Einheit zum Neigen und Schwenken und wird für Detailaufnahmen der Umgebung zum Objektlernen sowie für Gesichtsdetektion und -identifikation verwendet. Der Touchscreen dient sowohl zur Anzeige der aktuellen Systemausgaben, als auch zur bequemen Parametereingabe an die Steuerungscomputer zur Laufzeit. Die Systemausgabe kann wahlweise in einer technischeren und für die Entwickler aussagekräftigeren Variante geschehen oder erfolgt über einen animierten Roboter (Mindy [Li07]), welcher den Systemzustand für Laien einfacher verständlich wiedergibt.

Die notwendige Rechenleistung stellen zwei Laptops zur Verfügung, die über eine einfach zu öffnende Halterung an der Rückseite des Roboters befestigt sind. Jedes Laptop ist mit einem Intel Core2Duo© Prozessor sowie zwei Gigabyte Arbeitsspeicher ausgerüstet. Im Vergleich zu fest installierten Onboard-Rechnern bietet die Laptoplösung die Möglichkeit, schnell und einfach Reparaturen oder Nachrüstungen vorzunehmen. Außerdem entlasten die Akkuzellen der Laptops den Stromverbrauch des Roboters, so dass sich dessen Betriebszeit verdoppelt. Verbunden sind die beiden Laptops per 100MBit Netzwerk über einen Router. Zur Überwachung des Systems während der Durchführung von Benutzerstudien kann über W-LAN ein weiteres Laptop zu dem Netzwerk hinzugefügt werden, so dass die Durchführung der Experimente nicht durch mehr anwesendes Personal gestört wird. Zusammen mit den Laptops ergibt sich für dass Gesamtsystem eine Breite von 44 Zentimeter, eine Tiefe von 54 Zentimeter und eine Höhe von 145 Zentimeter. Mit diesen Abmessungen kann BIRON mit Menschen in alltäglichen Umgebungen operieren, ohne Gefahr zu laufen, Korridore oder Türen nicht passieren zu können.

3.2.2. Der anthropomorphe Torso Roboter Barthoc

Barthoc, die Kurzform für **B**ielefeld **A**nthropomorphic **R**obo**t** for **H**uman **O**riented **C**ommunication, wurde gemeinsam mit der Firma *mabotic* entwickelt. Ziel ist es, neben akustischer Sprache auch Körpersprache als Kommunikationsmodalität in die Interaktion einfließen zu lassen. Das System wird kontinuierlich erweitert, so dass seitens der Hardware sowohl der Kopf, als auch die Sensoren modifiziert wurden. Die kontinuierlichen Modifikationen sind dem Ziel geschuldet, die sozialen Interaktionsfähigkeiten für den Einsatz als Rezeptionist zu verbessern. Gleichermaßen stellen diese Modifikationen Flexibilitätsanforderungen an die Software-Komponenten und die ihnen zugrunde liegenden Systemarchitektur. Der Aufwand derartiger Modifikationen muss bereits auf Architekturebene minimiert werden und sollte keinesfalls möglichen Fortschritt be- oder sogar verhindern.

Die Hardware von Barthoc (siehe Abb. 3.4) ist so konzipiert, dass das System in der Lage ist, sich wie ein sitzender Mensch zu bewegen. Es entspricht mit seiner Oberkörpergröße von 75 cm ab Taille einer ausgewachsenen Person. Das System ist auf einem 65 cm hohen Sockel montiert, in dem Stromversorgung, Steuerungsanschlüsse und ein Motor zur Drehung um die eigene Achse untergebracht sind. Ohne den Sockel beträgt das Gewicht in etwa 35 Kilogramm, wodurch der Transport vereinfacht wird.

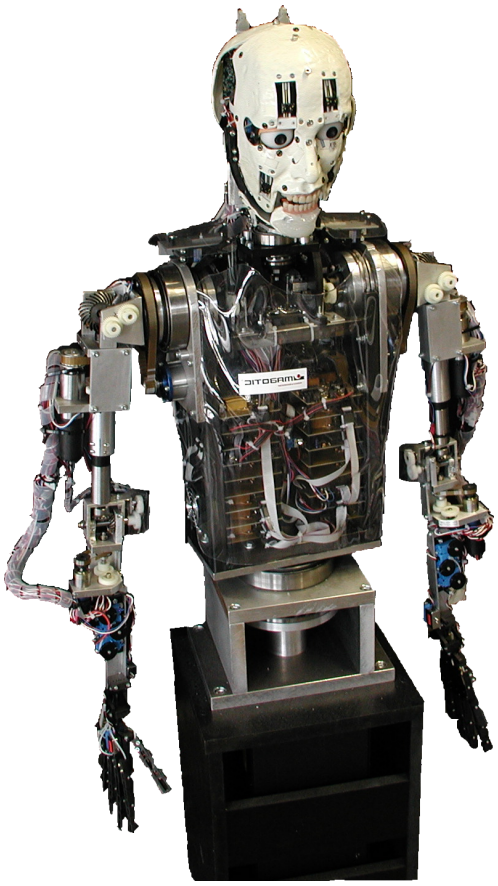
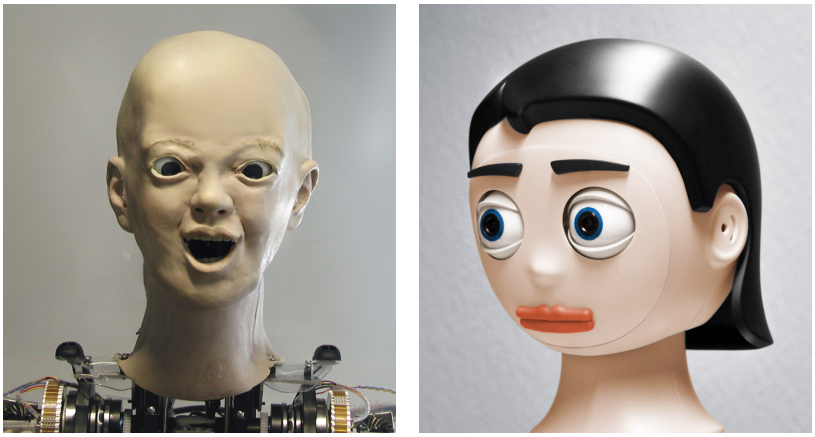


Abbildung 3.4.: Der anthropomorphe Roboter Barthoc verfügt über Größe und Beweglichkeit eines erwachsenen, sitzenden Menschen.

Der Torso des Roboters besteht aus einem Stahlrahmen, dessen Front mit einer transparenten Kunststoffschicht verkleidet ist. Innerhalb dieses „Körpers“ befinden sich die zur Bewegung notwendigen Steuerungsplatinen, die iModules. Diese werden über zwei USB Schnittstellen von einem Computer angesteuert. Eine USB-Verbindung übermittelt Daten zur Kopfsteuerung, während die zweite zur Steuerung aller Komponenten unterhalb des Nackens genutzt wird. Insgesamt werden somit 41 Aktuatoren kontrolliert. In der ersten Version konnten keine aktuellen Positionsinformationen abgerufen werden, sondern ausschließlich die zuletzt übermittelte Zielposition. Es zeigte sich schnell, dass dieses Verfahren für eine genaue und interaktive Steuerung ungenügend war. Da nicht alle Aktuatoren kritisch in ihrem zeitlichen Verhalten sind, verfügen nur die Hauptgelenke am Hals sowie an den Armen über Sensoren. Dementsprechend wurden 14 Potentiometer als Stellsensoren hinzugefügt und über eine Messkarte mit dem Steuerungscomputer verbunden.

Der Kopf des Roboters kann entsprechend seinem menschlichen Vorbild gedreht, seitlich und nach vorne geneigt sowie leicht vor und zurück bewegt werden. Um dem Menschen nachempfundene Mimiken zu generieren, werden sechs der 41 Aktuatoren im Gesichtsbereich zur Steuerung von Mundwinkeln, Augenbrauen und Augenlidern verwendet. In die Augen, die vertikal gekoppelt sind und horizontal autonom angesteuert werden können, sind zwei FireWire Farbkameras mit einem Auflösungsvermögen von 640x480 Pixeln integriert, die zur Wahrnehmung der Umgebung dienen. Die unabhängige, horizontale Kontrolle der Augen dient der Fixierung von Objekten und Personen, deren Distanz, basierend auf den Winkeleinstellungen der Augen, bestimmt werden kann.



(a) 1. Version des Kopfes mit Latexmaske

(b) Neu entwickelter Prototyp *Flobi*

Abbildung 3.5.: Die entwickelten Köpfe für Barthoc Junior. In der Ausgangsversion (a) wurde eine Latexmaske verwendet, mit dem Ziel, ein möglichst menschenähnliches Aussehen zu erreichen. Alternativ wurde ein zweiter Kopf mit einem humanoiden aber absichtlich artifiziellen Antlitz (b) entwickelt, um Auswirkungen in der Akzeptanz des Roboters durch Menschen zu untersuchen.

Neben einer ersten Kopfversion mit Latexmasken (siehe Abb. 3.5(a)) ist inzwischen eine weitere Version mit austauschbaren Kunststoffschalen (siehe Abb. 3.5(b)) für eine kindgroße Version, Barthoc Junior, entwickelt worden [Heg09]. Im Vergleich zu der Ausgangskonfiguration stößt diese auf eine höhere Akzeptanz bei Versuchspersonen. Die Anzahl der mimikgenerierenden Aktuatoren wurde im Mundbereich von zwei auf sechs erhöht. Leuchtdioden können die Wangen des Roboters erröten lassen. Die neuen Kontrollmodule für diese Hardware-Veränderung müssen sich schnell in das bestehende System integrieren lassen, ohne bereits existierende Komponenten negativ zu beeinflussen oder umfangreicherer Modifikationen am bestehenden System zu bedürfen.

An dem Torso sind zwei voll bewegliche Arme montiert, die sich, entsprechend dem menschlichen Vorbild, seitlich heben und parallel zum Torso um die Schulter rotieren lassen. Mittels Unterarmgelenken und zwei Fünf-Finger-Händen lassen sich sowohl Zeigegesten, als auch einfache Greifbewegungen realisieren. Die Finger jeder Hand sind dabei autonom mittels Servomotoren kontrollierbar. Außerdem sind zwei motorgetriebene Schulterplatten zur Simulation von „Achselzucken“ neben dem Hals angebracht, auf denen zwei Grenzflächenmikrofone für Richtungshören positioniert sind. Eine Montage am Kopf erwies sich wegen Störgeräusche der dort befindlichen Aktuatoren als problematisch.

3.3. Multimodale Umgebungswahrnehmung und Interaktionsfähigkeiten

Während die Demonstratoren und die Szenarien bereits erkennen lassen, wie wichtig Adaptivität und Robustheit für Systeme sind, verdeutlicht die Integration von Software diesen Aspekt. Hier wird die Software-Integration in zwei Aufgabenbereiche unterteilt: das Hinzufügen neu zu entwickelnder Komponenten und die Integration oder Erweiterung bereits bestehender Komponenten. Dieser Abschnitt gibt einen Überblick über die bereits existierenden Komponenten, die in den zuvor beschriebenen Szenarien und auf den zugehörigen Demonstratoren zum Einsatz kommen. Anhand der im Folgenden aufgezeigten Diversität und des Umfangs eingesetzter Komponenten werden die Anforderungen bezüglich der Systemarchitektur diskutiert.

3.3.1. Multimodales Personen-Tracking

Um mit einer Person unabhängig von der Home-Tour oder dem Rezeptionisten-Szenario interagieren zu können, muss diese durch den Roboter erfasst und kontinuierlich verfolgt werden. Hierfür wird auf dem als *Anchoring* bekannte Verfahren von Coradeschi und Saffiotti [Cor01] aufgebaut. Unter Anchoring ist die Zuordnung von zum Teil vorverarbeiteten Sensordaten, den sogenannten *Perzepten*, zu einer Repräsentation auf Software-Ebene, dem *Symbol*, zu verstehen. Für eine kontinuierliche Zuordnung (engl. *Tracking*) wird für jedes Perzept überprüft, ob es sowohl vom zeitlichen, als auch räumlichen Abstand einem vorhandenen Symbol oder einem neu erstellten Symbol zugeordnet wird.

Dieses Verfahren wurde in der Arbeit von Lang et al. [Lan03] auf ein multimodales Personen-Tracking mit Aufmerksamkeitssteuerung (*PTA*) adaptiert und vom Autor für den Einsatz auf unterschiedlichen Hardware-Plattformen erweitert [Spe06a]. Das PTA verwendet im Gegensatz zum ursprünglichen Ansatz mehrere Modalitäten, um eine Entität - hier eine Person - zu tracken. Für Personen werden Informationen über Beinpaar-, Torso- und Gesichtspositionen mit der Richtung von Schallquellen in Bezug gesetzt. Der Vorteil des multimodalen Anchorings liegt in dessen Robustheit. Bei unimodalem Anchoring wird ein Symbol gelöscht, wenn für eine gegebene Zeitspanne dem Symbol kein Perzept mehr zugeordnet wird. Das Personen-Symbol im PTA wird erst gelöscht, wenn für alle Bestandteile über eine gegebene Zeit keine Perzepte mehr zugeordnet werden (siehe Abb. 3.6).

Analog zu dem unimodalen Anchoring wird jede Modalität einem separaten, eingangs beschriebenen Anchoring-Prozess unterzogen. Um die einzelnen Personenbestandteile erfolgreich zu einem Personensymbol zusammen zu setzen, wird ein Personenmodell verwendet. Dieses berücksichtigt unter dem Begriff der *Komposition* die Anzahl der zuordnenbaren Bestandteile zu einer Person und deren maximale Abstände voneinander. Eine Person besitzt nach dem Modell nur ein Beinpaar und der Abstand zwischen Beinen und Kopf wird durch die Vorgaben der menschlichen Physis begrenzt. Zusätzlich zur Komposition wird die Bewegung abhängig von der Zeit, die zwischen zwei Perzepten vergangen ist, bewertet. Jedes Perzept beinhaltet einen Zeitstempel, der

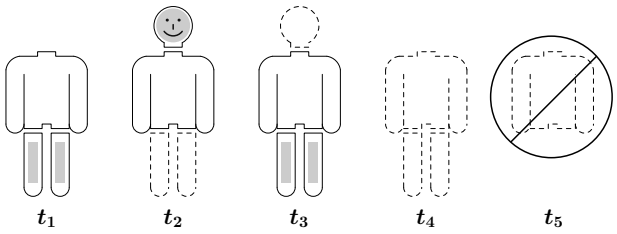


Abbildung 3.6.: Personen-Tracking: Ein Personensymbol (Torso) bleibt so lange vom System erfasst, wie noch Perzepte zu mindestens einem seiner Symbol-Bestandteile zugeordnet werden (eingegraut). Wenn kurzzeitig keine Perzepte zugeordnet werden (gestrichelt), bleibt das Symbol erhalten. Erst wenn alle Symbol-Bestandteile gelöscht sind, wird auch das Personensymbol frei gegeben.

Auskunft über den Zeitpunkt der Sensoraufnahme gibt. Das Bewegungsmodell vergleicht die zeitlichen Abstände mit den möglichen Positionsveränderungen, um zu bewerten, in wie weit es sich um die Bewegung der selben Person oder das Erscheinen einer neuen Person handeln kann. Anschließend werden die Informationen der Personenbestandteile für jede einzelne Person in eine gemeinsame Signatur zusammen geführt (*Fusion*) und chronologisch in eine Liste eingetragen. Diese Signaturliste bildet das Anchoring zwischen dem Personen-Symbol und den unterschiedlichen Perzepttypen. Abbildung 3.7 veranschaulicht das Verfahren.

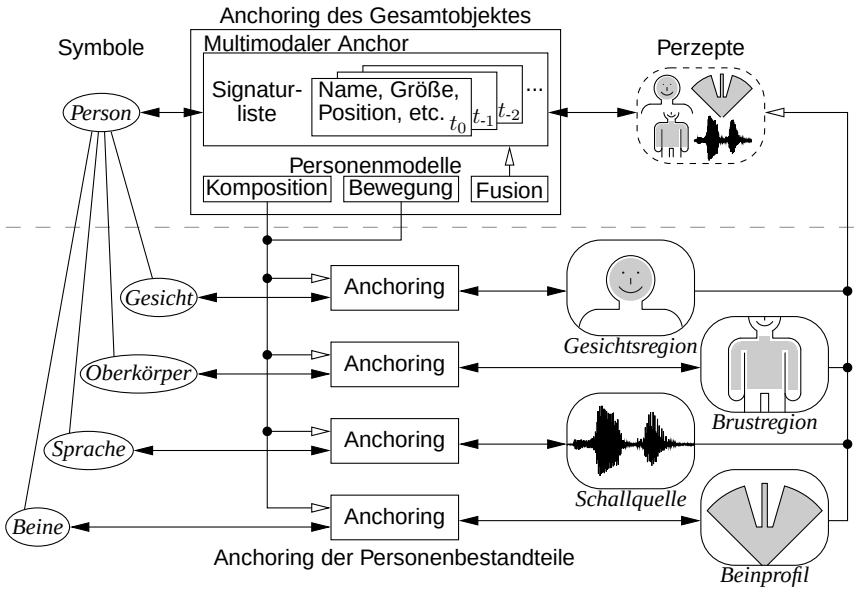


Abbildung 3.7.: Multimodales Anchoring nach [Lan03]: Die jeweiligen Personenbestandteile werden in einem eigenen Anchoring-Prozess nach [Cor01] den jeweiligen Symbolen zugeordnet und anschließend über ein Personenmodell zu einem multimodalen Anchor fusioniert.

Die Aufgabe des PTA liegt nicht nur in der kontinuierlichen Erfassen von Personen, sondern ist auch in der Lage, durch Generierung entsprechender Motorkommandos, der erfassten Person durch den Raum zu folgen. Da mit dem PTA mehrere Personen gleichzeitig erfasst werden, der Demonstrator sich jedoch immer nur einer Person aktiv zuwenden kann, muss eine Entscheidung getroffen werden, welcher Person der Roboter zuerst seine Aufmerksamkeit widmet, selbst wenn er durch die eigene Bewegung andere Personen aus dem Wahrnehmungsbereich verliert. Hierzu existiert innerhalb des PTA-Moduls eine Top-Down Heuristik, die Personen nach ihrem aktuellen Verhalten als unterschiedlich interessante Interaktionspartner bewertet. Am interessantesten sind Personen, die still stehen, in Richtung des Roboters blicken und etwas sagen. Am uninteressantesten sind Personen, die umhergehen und den Roboter nicht ansehen. Die perzeptuellen Informationen werden nicht nur für die Robustifizierung des Trackings verwendet, sondern auch für die Aufmerksamkeitssteuerung. Ein ähnlicher Ansatz für die Aufmerksamkeitssteuerung wird in [Tra08] präsentiert. Einem rein visuellen unimodalen Personen-Tracking werden Richtungsinformationen von Schallquellen zugeordnet, um während einer Konversation den aktuellen Sprecher zu ermitteln. Da dies erst nach dem Tracking geschieht, kann das Richtungshören nicht gleichzeitig zur Verbesserung des eigentlichen Trackings dienen, wie es im PTA-Ansatz geschieht.

Die umfangreiche Funktionalität des PTA stellt dabei ganz eigene Ansprüche an die Systemarchitektur. Das PTA ist als eine einzelne, monolithische Komponente realisiert, deren semantisch sinnvolle Einheiten, wie Perzeption, Tracking und Aufmerksamkeitssteuerung (siehe Abb. 3.8) sich nur unter großem Aufwand in voneinander entkoppelte Einheiten aufteilen lässt. Das PTA repräsentiert jedoch das Ergebnis eines umfangreichen Entwicklungsprozesses, auf das man wegen des Arbeitsaufwandes einer Neuimplementierung nicht verzichten möchte oder kann. Auf Grund seines Umfangs ist es aber nur noch schwer modifizierbar und gleich auf mehreren Gebieten, wie Aktuatorsteuerung und Verhaltenskontrolle, mit anderen Komponenten ungewünscht stark gekoppelt. Dementsprechend muss eine Architektur in der Lage sein, derartige und wegen

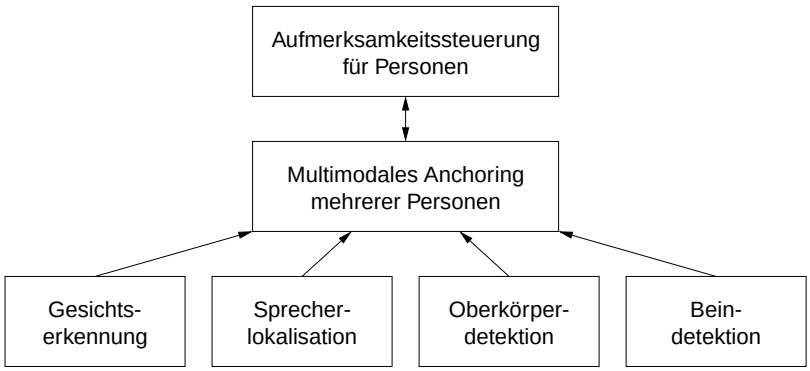


Abbildung 3.8.: Schichtenmodell des Personen-Tracking mit Aufmerksamkeitssteuerung: Beim Anchoring mehrerer Personen bestimmt eine feste Heuristik, welcher Person die Hauptaufmerksamkeit der Systems geschenkt wird. Dafür bedarf es der Daten Perzept generierender Komponenten.

eines zu hohen Reimplementationsaufwandes akzeptierte Kopplungen aufzufangen, so dass sie sich nicht nachhaltig negativ auf bestehende oder neu hinzuzufügende Komponenten auswirken. Dies gilt sowohl für die Arbitrierung konkurrierender Verhaltensweisen oder Aktuatoransteuerungen, als auch für die Adaption auf alternative perzeptgenerierende Module bei wechselnder Demonstrator-Hardware.

3.3.2. Perzeptgenerierende Module

Der Wechsel der Demonstratoren lässt sich am Beispiel von BIRON und Barthoc in Kombination mit dem perzeptgenerierenden Beinpaar-Detektor verdeutlichen. Der Beinpaar-Detektor ist die historisch erste verfügbare und in gleichem Maße simple, wie auch robuste perzeptgenerierende Komponente. Basierend auf den Messergebnissen des Laser-Range-Finders (siehe Abb. 3.9) werden zusammenhängende Segmente gesucht, die als Beinpaare klassifiziert werden. Die Klassifikation basiert auf einer simplen Heuristik. Zu Beginn werden zusammenhängende Segmente identifiziert, deren benachbarte Messpunkte um nicht mehr als jeweils 7,5 Zentimeter in der gemessenen Tiefe abweichen dürfen. Jedes Segment, dass eine reale Breite von fünf bis 25 Zentimeter repräsentiert, wird als potentielles Bein betrachtet, sofern in dem Segmentwerten eine Standardabweichung von 4 Zentimetern zur Durchschnittsdistanz des Segmentes nicht überschritten wird. Um die Spezifität gegenüber Möbelbeinen zu erhöhen, müssen die Beine paarweise auftreten, um einem Menschen zugeordnet zu werden. Als Paar werden Beine assoziiert, wenn ihr mittlerer Abstand nicht größer als 50 Zentimeter ist. Bei erfolgreicher Detektion stellt die Komponente sowohl Entfernung als auch den Winkel der gefundenen Beinpaare relativ zum Roboterzentrum zur Verfügung.

Jedes perzeptgenerierende Modul ist von einem oder mehreren Sensoren als Informationsquelle abhängig. So ist auch der Beinpaar-Detektor nur in Verbindung mit dem Laser-Range-Finder verfügbar. Bei variierenden Demonstratoren sind wegen der unterschiedlichen Sensorausstattung nicht immer alle perzeptgenerierende Module verfügbar. Der Demonstrator Barthoc besitzt

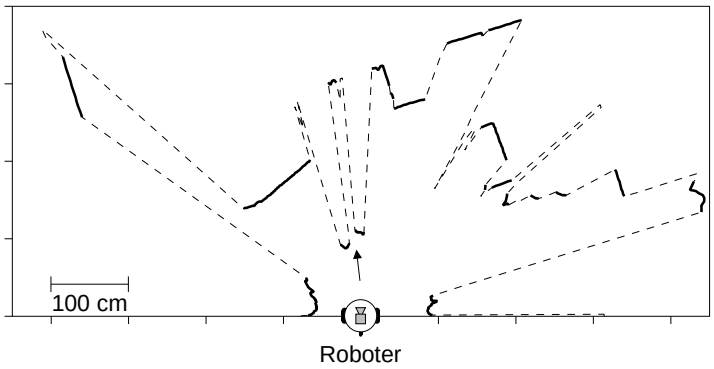


Abbildung 3.9.: Visualisierung eines Laser-Scans: Mit einer Auflösung von 0,5° werden 361 Distanzwerte in einem 180° umfassenden Bereich vor dem Roboter dargestellt. Der Pfeil verweist auf die charakteristische Signatur eines Beinpaares.

keinen Laser-Range-Finder oder einen anderen äquivalenten Distanzsensor, wodurch nicht nur ein Perzept zur Unterstützung des Anchorings fehlt, sondern, durch die monolithische Struktur des PTA, auch Daten die Seitens des PTA vorausgesetzt wurden. Die daraus resultierenden umfangreichen Modifikationen des PTA weisen deutlich die Nachteile einer derartig starken Informationsverknüpfung auf, deren Vermeidung bei der im Folgenden beschriebenen Weiterentwicklung der Systemarchitektur Berücksichtigung findet. Andere perzeptgenerierende Module wurden dementsprechend als rein optionale Komponenten integriert, so dass sie die Funktionen des PTA erweitern, ihr Fehlen jedoch nicht zum Ausfall des PTA oder anderer Systembestandteile führt. Beispiele hierfür sind die Sprecherlokalisation und die Gesichtsdetektion.

Die Sprecherlokalisation erkennt die Richtungen in der sich Schallquellen befinden. Im PTA mit den Positionsdaten des Beinpaardetektors kombiniert, lassen sich so Aussagen treffen, ob eine Person zu einem gegebenen Zeitpunkt spricht. Die Sprecherlokalisation wird vergleichbar mit dem menschlichen Gehör über zwei Mikrofone realisiert und ermöglicht eine Richtungsbestimmung in der Ebene der Mikrofone. Wie in Abbildung 3.10 gezeigt, ergeben sich abhängig von der Position einer Schallquelle (s) unterschiedliche Distanzen zu den beiden Mikrofonen (m_1, m_2). Bei konstanter Schallgeschwindigkeit, erreicht das Signal folglich die Mikrofone mit einem zeitlichen Versatz. Unter Anwendung einer *Cross-Powerspectrum-Phase* Analyse (*CSP*) auf die separat aufgezeichneten Signale der beiden Mikrofone wird dieser Zeitunterschied berechnet und liefert unter Berücksichtigung des Mikrofonabstandes b die Richtung der Schallquelle. Bei der hier verwendeten Implementation [Hoh03] wird nicht nur die stärkste Schallquelle ausgewählt. Es werden neben dem absoluten Maximum der CSP auch alle lokalen Maxima, die einen gegebenen Schwellwert übersteigen, berücksichtigt. Zwar nimmt die Genauigkeit ab, wenn die Schallquelle über oder unter der Mikrofonebene liegt, jedoch sind die Ergebnisse für Interaktionsszenarien ausreichend zuverlässig.

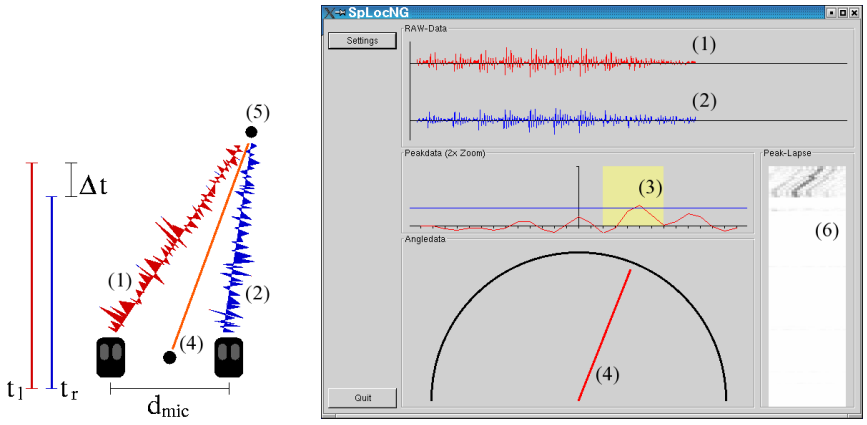


Abbildung 3.10.: Nach Aufnahme des linken (1) und rechten (2) Audiosignals über Mikrofone, verwendet das Sprecher-Lokalisations-Modul (*SpLoc*) die Ergebnisse der CSP, zur Berechnung des Unterschiedes Δ_t in der Signallaufzeit t_1 und t_2. Zusammen mit dem Mikrofonabstand d_{mic} ergibt sich die Richtung (3, 4) der Schallquelle (5) und deren zeitlicher Verlauf (6).

Der als dritte Modalität verwendete Gesichtsdetektor basiert auf einem von Viola & Jones ursprünglich zur Objektdetektion eingesetzten Verfahren [Vio01]. Es findet eine Klassifikation von Bildausschnitten unterschiedlicher Größe und Position statt, bis das gesamte Ausgangsbild erfasst wurde. Um die Effizienz zu steigern, wird eine Klassifikationspyramide nach [Zha02] verwendet, deren einzelnen Klassifikatoren nach dem *AdaBoost* Algorithmus [Sch01] trainiert wurden. Durch die Effizienz dieses Verfahren zur Laufzeit ist es möglich, die horizontale Ausrichtung des Kopfes (siehe Abb. 3.11) zu bestimmen [Pet06]. Hierzu werden vier weitere Klassifikationspyramiden verwendet, deren Trainingsmengen aus seitlich gedrehten Gesichtern mit einer jeweiligen Rotation um 20°, 40°, 60° und 80° bestehen. Für Drehungen nach links und rechts wird das Bild an der Mittelsenkrechten gespiegelt und die gleichen vier Klassifikatoren erneut angewendet.

Abbildung 3.11.: Auf unterschiedlich ausgerichtete Gesichter trainierte Klassifikatoren ermöglichen Gesichts- und Blickrichtungsdetektion.

Sowohl die Gesichts-, als auch die Sprecherdetektion haben den Nachteil, dass sie nur Daten liefern, wenn sich eine Person dem Roboter weitestgehend zuwendet und kontinuierlich spricht. Geht eine Person vor dem Roboter her, müsste sie sich folglich rückwärts bewegen oder dürfte nie still sein, sonst kann ein Mensch nur noch über die Beinpaardetektion wahrgenommen werden. Befindet sich beispielsweise ein Papierkorb zwischen Roboter und Mensch, so dass der Laser den entsprechenden Bereich nicht mehr erfassen kann, geht die Person verloren. Folglich wurde ein weiteres perzeptgenerierendes Modul hinzugefügt, welches den Oberkörper erfasst und anhand der Farben und Textur der getragenen Bekleidung trackt [Hel09]. Für die farbliche Erfassung wird eine Histogrammanalyse durchgeführt, die auf den Farbwerten eines fest vorgegeben Bereiches unterhalb eines detektierten Gesichtes initialisiert wird. Anschließend wird das Bild nach Bereichen durchsucht, die ein möglichst ähnliches Farbhistogramm aufweisen. Um die Beleuchtungsabhängigkeit zu verringern, wird in dem vom Histogramm erfassten Bereich zusätzlich nach grauwertabhängigen Texturen gesucht. Die Texturen werden auf Grund der Intensitätsunterschiede benachbarter Pixel mittels Local Binary Pattern [Mäe00] klassifiziert.

3.4. Integration von Interaktion

Während die zuverlässige Detektion von potentiellen Interaktionspartnern ein gemeinsames Fundament in der Mensch-Roboter-Interaktion darstellt, sind die darauf aufbauenden Interaktionsfähigkeiten in der Regel stark vom Szenario abhängig, in dem der Roboter agieren soll. Architekturen sind darauf ausgelegt, das festgelegte Szenario optimal zu unterstützen. Der hier verfolgte Ansatz geht einen Schritt weiter. Existierende Szenarien werden nicht durch neue ersetzt, sondern kontinuierlich erweitert. Die im Folgenden vorgestellten Komponenten realisieren unterschiedliche Fähigkeiten, die in mehrere einzelne Szenarien zusammengefasst werden könnten, stattdessen aber ein bestehendes System erweitern und selbst wieder als Basis für neue Erweiterungen fungieren. Der Anspruch an die Systemarchitektur wächst dementsprechend, da sie flexibel den jeweiligen Entwicklungsschritt unterstützen muss und gleichzeitig die Integration durch klar strukturierte Vorgaben koordinieren soll. Es ist die feste Überzeugung des Autors, dass diese *Systemevolution* die Entwicklung von Robotern hin zu Alltagshelfern deutlich unterstützt.

Eine grundlegende Erweiterung des Systems stellte ein Dialogsystem (*DLG*) zur sprachbasierten Kommunikation dar. In der Implementierung nach [Li07] verwendet das DLG-Modul die Ergebnisse einer Spracherkennung für eine vereinfachte Bedienbarkeit über Spracheingabe und Sprachsynthese. Hierzu wird ein als *Grounding* bekanntes Prinzip (siehe Abb. 3.12) angewendet. Wird eine Äußerung von einem Dialogpartner, sei es Mensch oder Roboter, getätigt, auf den eine Antwort zu erwarten ist, wird diese Äußerung auf einem Stack abgelegt. Erkennt der Dialog eine Antwort zu dem zuletzt auf dem Stack eingefügten Eintrag, so wird dies als erfolgreicher Informationsaustausch (engl. *Exchange*) erkannt und der Eintrag vom Stack entfernt. Anschließend wird der vorangehende Eintrag aufgerufen. So ist es möglich, auch klärende Zwischenfragen [Rod04, Pur06] seitens System oder Anwender zu berücksichtigen, zwischenzeitlich den Kontext zu wechseln oder Aufgaben durchzuführen und nach Beendigung des Exkurses wieder an den vorangegangenen Inhalt anzuknüpfen. Eingesetzt wird dieses Verfahren zum Beispiel während des Umgebungslernens in der Home-Tour, wo der Roboter zwischenzeitlich Aktionen, wie das Umherfahren zur Raumerkennung, ausführt und anschließend wieder den Dialog fortsetzt. Für diesen Aufgabenwechsel greift die hier beschriebene DLG-Implementierung jedoch auf Systemwissen zu und ist sogar mit der Systemkontrolle verwoben. Die Funktion, als sprachbasierte Schnittstelle zwischen Mensch und System zu operieren, wird dabei deutlich überschritten und hatte zur Folge, dass bei Erweiterung des Systems, neben der Anpassung der Dialogführung, aufwändige Modifikationen, im zum Dialogsystem gehörenden Bereich der Systemkontrolle, erfolgen mussten. Eine klare Kapselung der semantischen Einheiten wurde damit zu einem deutlichen Ziel der Architekturüberarbeitung.

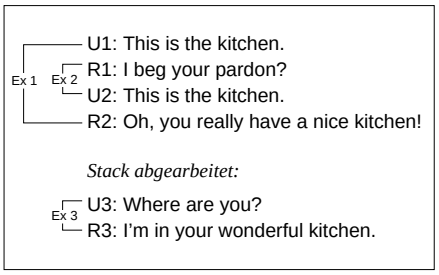
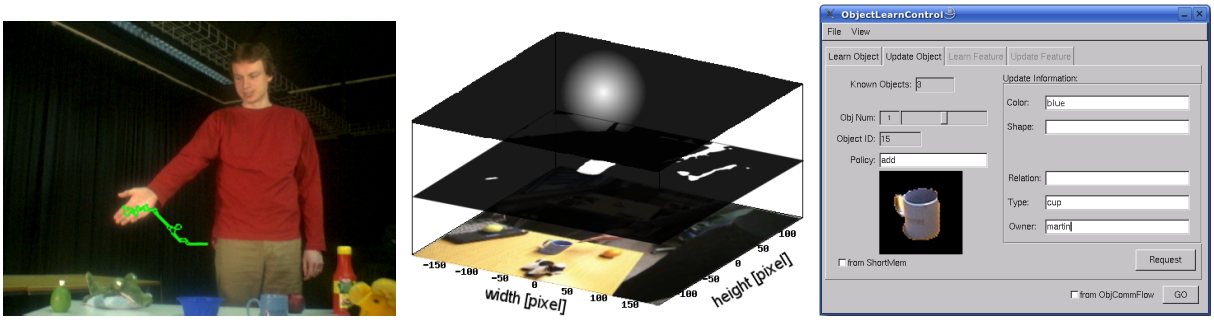


Abbildung 3.12.: Dialog Grounding

Nicht nur verbale, sondern auch auf non-verbale Kommunikation wurde in der nächsten Systemerweiterung berücksichtigt. Dies geschieht durch die Verwendung von unterschiedlichen

Gesten [Hof07] - insbesondere Zeigegesten als Referenzierung auf ein verbal benanntes Objekt, dessen Erscheinung zusammen mit einer verbalen Beschreibung erfasst werden soll. Für die Erkennung einer Geste wird, wie in Abbildung 3.13(a) zu sehen, die Hand des Interaktionspartners auf Basis von Hautfarbendetektion getrackt und die daraus resultierende Trajektorie klassifiziert. Wurde durch eine Zeigegeste der Bereich eines neu zu erlernenden Objektes definiert, werden beschreibende Attribute aus der Spracheingabe zur Bezeichnung des Objektes, wie *Größe*, *Form* oder *Farbe* verwendet, um über Filtermechanismen der Bildverarbeitung auch bei mehreren dicht beieinanderstehenden Objekten das Relevante zu identifizieren 3.13(b)). Dieses Objekt-Aufmerksamkeits-System (*OAS*) [Haa05] selektiert anschließend im Kamerabild den entsprechenden Objektbereich, schneidet ihn aus und speichert ihn zusammen mit der Bezeichnung (siehe Abb. 3.13(c)). Erweitert werden die Bezeichnungen um den semantischen Bezug, der sprachbasiert aus dem Thema einer Konversation erkannt wird [Maa06]. Somit wird auch bei gleichnamigen Objekten, wie zum Beispiel einer Bürste (zur Geschirr- oder Zahnreinigung), eine korrekte Zuordnung getroffen. Auch wenn die soeben vorgestellten Komponenten bereits stärker von der Systemkontrolle entkoppelt waren als das Dialogsystem, so zeichnete sich vor allem die Integration des OAS durch eine hohe Anzahl direkter Datenverbindungen und einer festen Verarbeitungsreihenfolge innerhalb dieser Verbindungen aus. Bei einer unerwartet verzögerten Bearbeitung oder einer fehlerhaften Datenübermittlung fielen alle nachfolgenden Prozesse aus, auch wenn die bereits verfügbaren Informationen zur erfolgreichen Bearbeitung genügt hätten.

Während der Vervollständigung des Home-Tour-Szenarios durch OAS und Gestenerkennung wurde die Hardware des Demonstrators Barthoc fertig gestellt. Um diesen schnellstmöglich in Betrieb zu nehmen und ressourceneffektiv zu arbeiten, sollte die bestehende Interaktions-Software weitestgehend von BIRON auf Barthoc übertragen werden. Trotz eines bereits modular gehaltenen Ansatzes, erwies sich die Übertragung arbeitsaufwendiger als erwartet. Wie bereits am Beispiel des PTA und Dialoges erläutert, waren Informationen über das Gesamtsystem in einzelne Komponenten eingeflossen und lösten unerwartete Seiteneffekte mit den durch die unterschiedliche Hardware bedingten neuen Software-Module aus. Dies betraf neben der diskutierten,



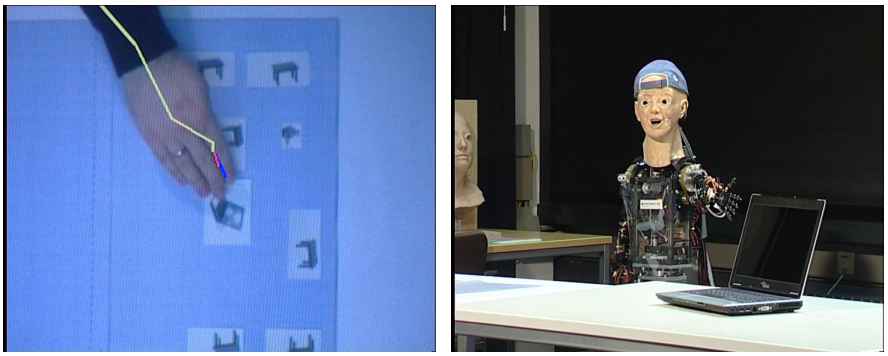
(a) Gestendetektion (b) Gesten- und Farbfilter (c) Objektbeschreibung

Abbildung 3.13.: Nachdem eine Zeigegeste des Anwenders erkannt wurde (a) wird in dem verwiesenen Bereich nach Objekten gesucht (b), die der verbalen Beschreibung entsprechen und diese gespeichert (c).

unterschiedlichen Sensorausstattung auch die veränderte Aktuatorik und die damit verbundenen Annahmen über die Systembeweglichkeit. Es gelang dem Autor, die betreffenden Informationen zu entfernen und die Struktur der Programme durch stärkere Kapselung generischer zu gestalten, um die Flexibilität gegenüber Hardware-Modifikationen zu erhöhen. In diesem Rahmen entstand das erste Interaktionsszenario mit Barthoc, welches PTA, Dialog und Gestendetektion verband. Mehrere Personen können mit dem Roboter interagieren und per Zeigegeste auf Objekte im Sichtbereich verweisen, zu denen Barthoc sich hindreht.

Diese Funktionalität deckte sich mit den bereits auf BIRON realisierten Fähigkeiten. Um die Auswirkung nonverbaler Kommunikation mittels Barthoc näher zu untersuchen, wurde die Fähigkeit zur Generierung von Gesichtsausdrücken entsprechend der Basisemotionen nach Ekman [Ekm72] integriert, genauso wie ein Prosodieklassifikator, der den aktuellen emotionalen Zustand des Interaktionspartners auf Grund seiner Stimme detektiert. Diese beiden für die sogenannte *Rotkäppchen-Studie* [Heg06] verwendeten Komponenten, konnten aufgrund der erfolgreichen Modulrestrukturierung für das erste Interaktionsszenario mit Barthoc ohne Anpassung des existierenden Systems verwendet werden.

Ziel der zuvor genannten Fähigkeiten ist nicht nur, dem System Wissen zu vermitteln, welches wegen seiner Variationsmöglichkeit nicht im Vorhinein festzustellen ist, sondern das trainierte System zur Unterstützung von Menschen nutzbringend einzusetzen. Mit dem *Environment Information System* (*EIS*) [Beu08] wurde die Systemevolution in einem parallelen Zweig auf dem Roboter Barthoc fortgesetzt. EIS verbindet die zuvor genannte Gestendetektion mit der visuellen Erfassung einer Karte. Diese Karte liegt auf einem Tisch zwischen Roboter und Interaktionspartner und repräsentiert die sichtbare Umgebung. Der Benutzer kann gezielt nach Bereichen fragen, die auf der Karte abgebildet und dem System bekannt sind. Die Referenzierung der Bereiche geschieht entweder über die verbale Benennung oder durch eine Zeigegeste des Benutzers auf den entsprechenden Kartenbereich (siehe Abb. 3.14(a)). Der Roboter entgegnet mit einer kurzen, sprachlichen Bestätigung und weist dem Interessenten durch eine Zeigegeste mit einem seiner



(a) Kartenreferenzierung (b) Robotergeste

Abbildung 3.14.: Der Roboter Barthoc ist in der Lage, Zeigegesten auf einer Karte (a) mit seiner Umgebung zu assoziieren und selber Gesten zur Referenzierung von erfragten Positionen einzusetzen (b).

beiden Arme die Richtung (siehe Abb. 3.14(b)). Alternativ können auch zusätzliche zu den Bereichen gespeicherte Informationen durch Nachfragen abgerufen werden. Die Entwicklung dieser Komponente und deren Integration geschah bereits auf Grundlage der zuvor gewonnenen und hier bislang präsentierten Erkenntnisse. Auf längere Aktionssequenzen wurde verzichtet und der Datenaustausch verlief bereits über einen zentralen Informationsspeicher, der Daten persistent vorhielt, um sie zeitunkritisch bei Bedarf zur Verfügung zu stellen.

Wie wichtig es ist, parallele Ansätze flexibel integrieren zu können, zeigte sich beim sogenannten Umgebungslernen. Neben den Bezeichnungen von Objekten bedarf es der Information wo, beziehungsweise in welchem Raum sich die Objekte befinden. Entsprechend dem Home-Tour-Szenario wird auch diese Information nicht vorprogrammiert, sondern mit einer Systemerweiterung innerhalb der Benutzerinteraktion vom Roboter gelernt. Durch die Kooperation mit mehreren Partnern ergab sich die Gelegenheit gleich drei alternative Ansätze zu verwenden:

1. Eine auf Laser-Daten basierende, topologische [Top08] Klassifikation des aktuellen Aufenthaltsortes, die die Umgebung hierarchisch in mehrere Räume mit jeweils mehreren Bereichen und den Bereichen zugeordneten Objekten unterteilt [Spe07a, Pel09]. Da der mobile Roboter BIRON nur über einen nach vorne gerichteten Laser verfügt, muss der Roboter sich hierfür zwischenzeitlich um seinen eigene Achse drehen.

2. Ein visuelles Verfahren, welches SIFT-Features [Low04] auf die Bilder einer omnidirektionalen Kamera anwendet, um Positionen im Raum zu bestimmen und anschließend ähnlich klassifizierte Bereiche als Cluster in einem Graphen repräsentiert. [Spe06b, Boo08]

3. Die Verwendung einer aktiv gesteuerten Pan-Tilt-Kamera [Sch07] zur Erfassung möglichst vieler Bilddaten und anschließende Bestimmung von für die Raumunterscheidung signifikanten Features [Vio01] analog zur beschriebenen Gesichtsdetektion.

Um die verschiedenen Ansätze einfach gegeneinander auszutauschen, bedarf es natürlich einer guten Planung der jeweiligen Schnittstellen. Die jeweils unterschiedlichen Verfahrensweisen und Steuerung unterschiedlicher Aktuatoren für den jeweiligen Lokalisationsansatz machten es jedoch unmöglich, identische Schnittstellen zum System zu kreieren. Es zeigte sich erneut die Notwendigkeit einer flexiblen Architektur, die es ermöglicht, mit möglichst geringen oder gar keinen Auswirkungen bezüglich des bestehenden Systems, auf die spezifischen Anforderungen der hier zur Auswahl stehenden Module einzugehen.

Um in einer unbekannten Umgebung die jeweils neu zu erlernenden Bereiche zu erreichen, folgt der Roboter dem Menschen. Ein erster Ansatz im PTA ließ den Roboter stets die kürzeste Route zwischen seiner aktuellen Position und der Personenposition fahren. Dies erwies sich jedoch als wenig geschickt, da sich auf dem direkten Weg Hindernisse befinden können, die der Mensch absichtlich umgangen hat. Außerdem versagte das Folgen wenn der Mensch um eine Ecke ging und somit seine Position nicht weiter bestimmt werden konnte. Der Roboter blieb dann an seiner gegenwärtigen Position stehen. Als Weiterentwicklung wurde ein *Path-Following* [Yua08] entwickelt, welches die Route des Menschen als Kette aus einzelnen Wegpunkten nachfährt. Gerät die verfolgte Person aus dem Blickfeld des Roboters, folgt der Roboter den Wegpunkten bis zur letzten bekannten Personenposition, um dort erneut nach seinem Interaktionspartner zu suchen.

Dieses Verfahren bietet eine relative Sicherheit gegenüber statischen Hindernissen, eignet sich jedoch nicht in dynamischen Umgebungen mit mehreren Menschen, die den Weg des Roboter kreuzen. Hierfür wird eine Navigationskomponente [Phi06] verwendet, die sowohl für die Fahrplanung zwischen den einzelnen Wegpunkten beim Personenfolgen Verwendung findet, als auch bei der autonomen Navigation zu zuvor gelernten Bereichen, um zum Beispiel von dort Objekte für den Benutzer zu holen. In Echtzeit werden über den Laser Hindernisse erfasst, alternative Routen zum Erreichen einer Position berechnet und aktualisierte Bewegungskommandos an die Motoren übermittelt. Die Integration dieser beiden Module barg erneut die Herausforderung, dass sie mit dem Folgeverhalten des bestehenden Systems, genauer des PTA, kollidierte. Statt der aufwendigen Modifikation des PTA, wie es noch bei der Verwendung der Ausgangsarchitektur notwendig gewesen wäre, konnte das im weiteren Verlauf dieser Arbeit dargelegte Architekturkonzept die neuen Fähigkeiten mit dem bestehenden System erfolgreich koordinieren und so dem System einen weiteren Evolutionsschritt ermöglichen.

Die vorangegangenen Beschreibung der kontinuierlichen Weiterentwicklung vorhandener Module zu Szenarien und die Erweiterung dieser Szenarien zu neuen umfangreicheren Szenarien wird in Abbildung 3.15 zusammenfassend dargestellt. Die Abbildung zeigt, wie sich durch das Hinzufügen neuer Komponenten und den daraus resultierenden Fähigkeiten neue und komplexere Anwendungsgebiete erschließen. Dabei ist die Erweiterung nicht nur auf die Software-Ebene begrenzt, sondern auch die Hardware unterliegt einer kontinuierlichen Entwicklung. Dies zeigt sich sowohl in der Verwendung neuer Demonstratoren, wie Barthoc für Körpersprache als neuen Interaktionsaspekt, als auch in der Modifikation bestehender Hardware, in die die Erfahrungen aus zuvor gemeisterten Herausforderungen einfließen. Die Entwicklung reicht von der Anschaffung der ersten Plattform im Jahr 2001 bis hin zur erfolgreichen Teilnahme mit dem aktuellsten Demonstrator bei der RoboCup@Home Challange 2009, wobei durch das Design der Systemarchitektur Neuimplementationen vermieden und stabile Komponenten weiter verwendet werden.

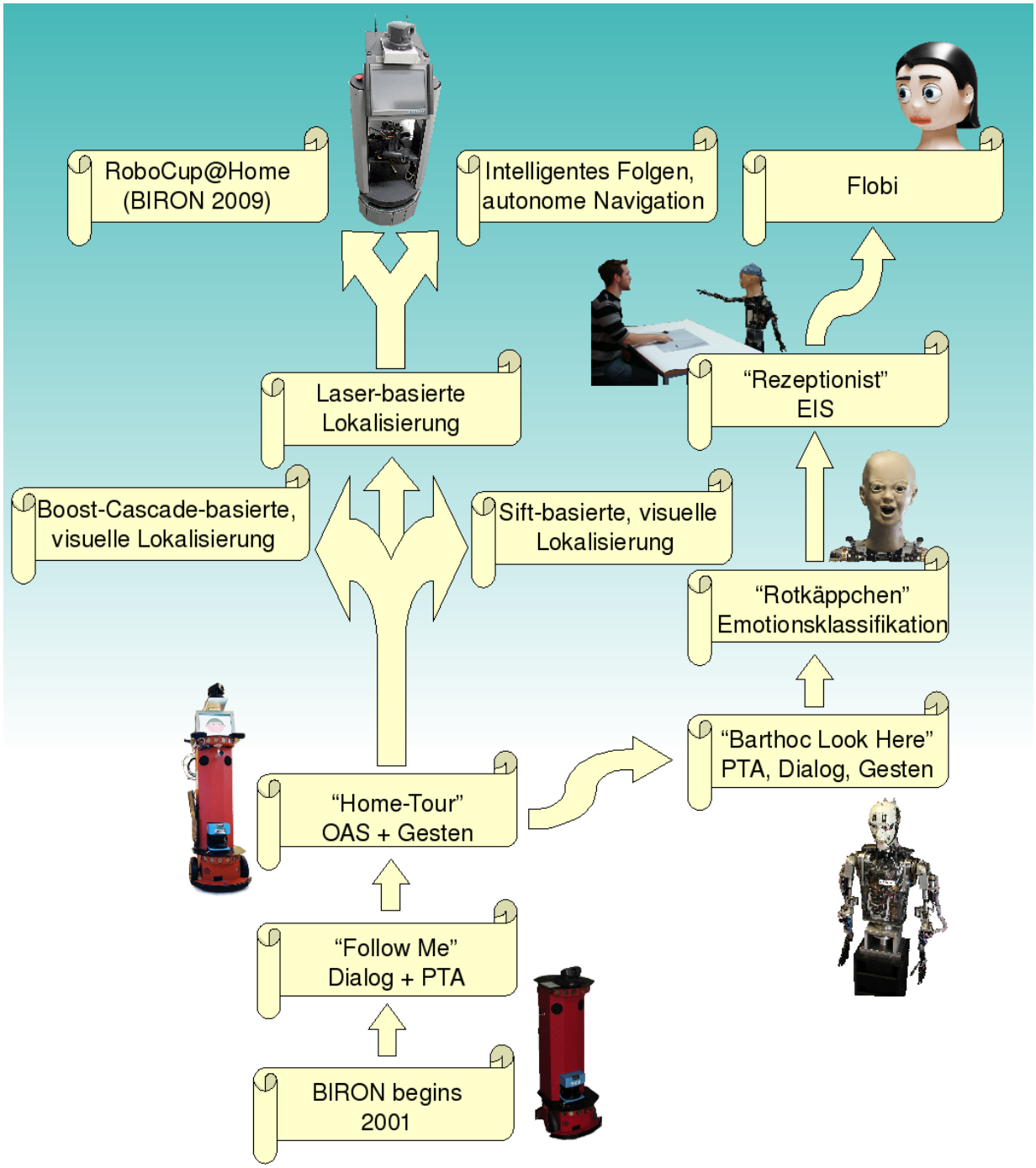


Abbildung 3.15.: Anstatt nach erfolgreicher Erforschung eines Interaktionsszenarios neu mit dem nächsten zu beginnen, werden bestehende Systeme kontinuierlich erweitert und, wenn nötig, auf neue Plattformen portiert, um differenzierten Fragestellungen nachgehen zu können. Der farblich hinterlegte Bereich zeigt die Entwicklung unter der evolutionären Systemintegration seit 2007.

3.5. Zusammenfassung

Die in diesem Kapitel beschriebenen Szenarien auf den Plattformen BIRON und Barthoc zeigen, wie komplex und vielfältig derartige Systeme sind, um auch nur in klar definierten Teilbereichen des Alltags eingesetzt werden zu können. In diesem Zusammenhang sind neben natürlicher, gesprochener Sprache auch Kommunikationsmodalitäten, wie Gestik und Mimik, zu berücksichtigen. Diese muss ein Roboter sowohl erkennen, als auch selbst sinnvoll einsetzen können. Um nach erfolgreicher Interaktion Aufgaben absolvieren zu können, die dem Benutzer den Alltag erleichtern, muss der Roboter sich in seinem Umfeld zurecht finden, obwohl dieses auf menschliche Bedürfnisse zugeschnitten ist. Besonders hoch ist diese Individualisierung in Privathaushalten, weshalb diese Umgebung zur Evaluation in der Home-Tour verwendet wird.

Die Beschreibung der einzelnen Softwarekomponenten, die der Interaktion mit Menschen und zur Wahrnehmung der Umgebung dienen, zeigt die Komplexität der Thematik und die Ansprüche, die sich daraus für eine geeignete Systemarchitektur ergeben. Gleichzeitig ergibt sich durch die kontinuierliche Evolution bestehender Systeme und Szenarien ein größeres Entwicklungspotential für die interaktive Robotik, wie sich an den, in diesem Kapitel beschriebenen, fortgesetzten Erweiterungen der Systeme zeigt. Dementsprechend wichtig ist die Forschung im Bereich Systemintegration und -architektur. Sowohl die steigende Anzahl der eingesetzten Komponenten, als auch deren Verknüpfung untereinander erschweren bei einer ungeeigneten Integrationsstrategie kontinuierlich die Wart- und Erweiterbarkeit mit jeder neuen Fähigkeit, die einem Roboter verliehen wird. Dennoch bedarf ein alltagstauglicher Roboter all dieser und noch weiterer zukünftig zu entwickelnder Fähigkeiten. Dabei werden die verschiedenen Fähigkeiten längst nicht mehr von einer einzelnen Person oder einer Forschungseinrichtung entwickelt, sondern verteilen sich auf mehrere, in der Regel internationale Institute, die innerhalb eines Projektes gemeinsam an der Entwicklung eines Roboters arbeiten. Unter Verwendung hart implementierter Abhängigkeiten, die auf den ersten Blick eine vermeintlich schnelle Lösung für Datenaustausch und Koordinierung bieten, ist nur all zu schnell eine Grenze erreicht. Diese macht eine flexible Anpassung des Systems an neue Erkenntnisse unter vertretbarem Aufwand unmöglich, insbesondere, wenn die jeweiligen Entwickler nicht mehr erreichbar sind oder neuen Aufgaben zugeteilt wurden.

Aus diesen Überlegungen resultiert der Bedarf nach einer flexiblen Koordination als Bestandteil einer Systemarchitektur, die auf die Spezifikationen der zu integrierenden Komponenten eingeht und sie gleichzeitig voneinander entkoppelt. Gemeinsam mit einem geeigneten Kommunikations-Framework ermöglicht ein derartiges Vorgehen die Weiterentwicklung, sowohl der hier beschriebenen Systeme im Speziellen, als auch der Mensch-Roboter-Interaktion im Allgemeinen, um letztendlich Roboter zu entwickeln, die alle notwendigen Fähigkeiten besitzen, um autonom in einem von Menschen geprägten Alltag agieren zu können.

4. Entwicklungsgrundlagen für interaktive Roboter

Zur Realisierung, der zuvor präsentierten Interaktionsszenarien, bedarf es eines hohen Maßes an Systemflexibilität, um ein System schnell neuen Umweltbedingungen, wie verändertem Aktionsbereich oder Anwendern, anpassen zu können. In diesem Kapitel wird diskutiert, welche Voraussetzungen seitens der Systemintegration berücksichtigt werden müssen, um den gewünschten Grad an Flexibilität zu erreichen und welche Vorzüge sich daraus in einem interaktiven Robotikszenario, beziehungsweise bei dessen Entwicklung ergeben. Dabei ist zu bedenken, ob und welche architektonischen Elemente aus dem vorhandenen System übernommen, beziehungsweise angepasst werden können. Aufwand und Gewinn einer Neuentwicklung sind den gleichen Werten bei einer Anpassung existierender Ressourcen gegenüberzustellen. Datenaustausch und Systemkoordination werden unter diesen Gesichtspunkten betrachtet und Rückschlüsse über deren Verwendung in der hier beschriebenen evolutionären Systemintegration gezogen.

4.1. Evolutionäre Systementwicklung statt Neubeginn

In welchem Rahmen Systemintegration durchgeführt wird, hängt stark von dem gewählten Fokus des zu entwickelnden Systems ab. Diese Arbeit richtet sich nach den Anforderungen, die im Forschungsbereich der Mensch-Roboter-Interaktion aufkommen. Es ist das erklärte Ziel dieser Arbeit, den Entwicklungsprozess von interaktiven Robotern zu beschleunigen, indem sie die Entwickler einzelner Komponenten, genauso wie die Betreuer von Benutzerstudien in ihrer Arbeit unterstützt. Für die Durchführung von Benutzerstudien bedarf es nicht nur eines stabil funktionierenden Systems. Immer wieder zeigt sich in der Praxis, dass Erwartungshaltung der Experimentatoren und die Interaktion von Anwender und Demonstrator nicht übereinstimmen. In der Grundlagenforschung existieren häufig nur grobe Annahmen darüber, wie zum Beispiel ein Roboter mit einem Menschen zusammen eine enge Stelle passieren soll [Pet09], oder wann das System initiativ Hilfe bei seiner Bedienung anbietet [Spe08a], ohne den Anwender durch ständiges Nachfragen zu irritieren. Erkenntnisse aus derartigen Studien wirken wie ein Selektionsdruck auf das Systemverhalten des Roboters. Je geringer die Adaptionsfähigkeit des Systems ist, auf derartige Erkenntnisse eingestellt zu werden, um so größer ist die Wahrscheinlichkeit, dass es durch ein neues System ersetzt wird. Der Aufbau eines neuen Demonstrators ist jedoch mit

einem entsprechenden Aufwand versehen. Viel sinnvoller erscheint es daher, bereits bei der Entwicklung eines Demonstrators darauf zu achten, dass das Systemverhalten schnell und flexibel adaptiert werden kann.

Diese Überlegung gilt nicht nur für die Entwicklung eines Demonstrators und dessen Anwendungsszenario, sondern lässt sich auf unterschiedliche Demonstratoren und Szenarien erweitern. Bei einem ausreichend flexiblen Ansatz kann sich so aus einer Forschungslinie eine weitere Linie parallel entwickeln, die beide einen gemeinsamen Entwicklungspunkt als Vorfahren aufweisen. Bestandteil dieser Entwicklungen ist nicht nur das Systemverhalten mit der Veränderung von einzelnen Parametern wie Wartezeiten oder Distanzen zu verstehen. Vielmehr berücksichtigt diese Entwicklung auch die Veränderung von Systemfähigkeiten. So sollen die Aktionsmöglichkeiten eines Systems in der Regel stetig erweitert werden, um immer komplexeren Aufgaben nachzukommen. Natürlich muss auch diese Erweiterung in der Modellierung von flexiblem Verhalten berücksichtigt werden.

Den potentiellen Erweiterungen darf aber nicht nur auf Ebene des Verhaltendesigns Rechnung getragen werden. Bei der Erweiterung von bestehenden Systemen treten häufig viel profanere Probleme auf. Dies beginnt in der Regel schon mit der Analyse der zur Verfügung stehenden Schnittstellen. Besonders bei der Kooperation in einrichtungsübergreifenden Forschungsprojekten besteht die Gefahr, dass eine interessante Integration auf Grund zu hohen Integrationsaufwandes verschoben oder letztendlich ganz ausgesetzt wird. Aber auch innerhalb einer Forschungsgruppe steigt die Gefahr von Inkompatibilitäten mit der Betriebszeit eines Demonstrators. Über mehrere Jahre hinweg ändern sich Standards, wie vorherrschende Programmiersprachen oder die Art des Datenaustausches. Eine Vereinfachung des Interfacings muss bei der Entwicklung folglich ebenso berücksichtigt werden.

Schnittstellen machen jedoch nur einen Aspekt bei der Erweiterung eines Systems aus. Für das schnelle Einbinden und Austauschen von neuen Systemkomponenten und den mit ihnen verknüpften Demonstratorfähigkeiten ist die Kopplung beziehungsweise Entkopplung der Komponenten ebenso entscheidend. Der Bereich der Kopplung wird in dieser Arbeit unter zwei Aspekten betrachtet, die im Folgenden als *syntaktische* und *semantische* Kopplung bezeichnet werden. Unter der *syntaktischen Kopplung* ist die technische Implementierung des Datenaustausches zu verstehen. Für einen Datenstrom gilt zum Beispiel, dass der Sender vor dem Empfänger verfügbar sein muss, damit sich der Empfänger auf den entsprechenden Datenstrom registriert. Ein kontinuierliches Wiederverbinden des Empfängers auf einen Sender wäre ein alternativer Ansatz gewesen, jedoch löst dies nicht das Problem der *semantischen Kopplung*. Die semantische Kopplung bezieht sich auf den Inhalt der ausgetauschten Informationen und deren Auswirkung auf das System. Ein Beispiel hierfür ist, dass eine Komponenten eine „Erwartungshaltung“ gegenüber dem Wert des nächsten zu empfangenen Datums, beziehungsweise gegenüber der datengenerierenden Komponente besitzt und nur mit dem Wert dieses Datums weiterarbeiten kann. Wird aufgrund eines Fehlers der Neustart einer derartigen Komponenten notwendig oder bricht die Datenkommunikation während der Übertragung ab, so gehen die gegenseitigen Erwartungshaltungen oder die erwarteten Informationen verloren und das System gerät in einen inkonsistenten, asynchronen Zustand.

Alle Komponenten sollten derartig voneinander entkoppelt sein, dass sowohl der beabsichtigte Austausch als auch ein fehlerbedingter Ausfall keinen Domino-Effekt auf andere Systemkomponenten nach sich zieht. Ansonsten bedingt auch eine geplante Systemmodifikation eine umfangreiche Modifikation von bestehenden Komponenten, welche sich im ungünstigsten Fall über das ganze System fort propagiert. Der damit verbundene Aufwand stünde dann kaum mehr in einem Verhältnis mit der Adaption des Systems und dies führt häufig zu einem Entwicklungsstopp auf dem besagten Demonstrator mit den bereits aufgewiesenen Folgen.

Aus den hier angestellten Überlegungen lassen sich folgenden Grundkonzepte ableiten, die eine evolutionär geprägte Systemintegration zu erfüllen hat, um die kontinuierliche Weiterentwicklung eines interaktiven Roboters zu ermöglichen:

- Schnelle und einfache Modifikation des Systemverhaltens und der darunter liegenden Arbitrierungs- und Kontrollstrategien bis hin zur Adaption auf neue Anwendungsszenarien und zugehörige Demonstratoren
- Flexibler Umgang mit Schnittstellen hinsichtlich Datenformate und Art des Datenaustausches durch verschiedene Kommunikationsstrukturen wie Datenströme oder externer Funktionsaufruf
- Entkopplung von Komponenten, so dass sie flexibel ausgetauscht oder das existierende System erweitern können, wobei bestehende Komponenten so wenig wie möglich angepasst werden müssen

Das Ziel, welches durch die Umsetzung dieser Anforderungen erreicht werden soll, ist, den Entwicklungsaufwand für Erweiterungen bestehender Systeme deutlich zu senken. Dadurch können häufig praktizierte und nur für ein Projekt bezogene Neuentwicklungen vermieden werden, die all zu oft als teure Ausstellungsstücke hinter Glas oder als Staubfänger in Lagerräumen ihrer jeweiligen Forschungseinrichtungen enden.

4.2. Diskussion des Kommunikations-Frameworks

Grundlage für eine erfolgreiche fortgesetzte Systementwicklung bildet in diesem Ansatz die Modulentkopplung, auf die die Koordination des Systemverhaltens aufsetzt. Die Entkopplung kann maßgeblich durch ein geeignetes Kommunikations-Framework unterstützt werden, wenn dieses einen Datenaustausch ohne direkte Verknüpfung von Komponenten und eine persistente Datenhaltung für fehlgeschlagene Datenübermittlung unterstützt. Jedoch ist Kopplung nicht das einzige Kriterium für die Wahl des Kommunikations-Frameworks. Weitere Aspekte sind eine möglichst einfacher Anwendbarkeit für die an Integrationsprozessen beteiligten Entwickler, ein schneller und übersichtlicher Datenaustausch zwischen asynchronen und in einem Netzwerk auf mehreren Computern verteilten eigenständigen Modulen sowie eine von Menschen lesbare Datenrepräsentation für eine vereinfachte Fehleranalyse. Diese Kriterien legen den Grundstein für eine integrierte Forschungs- und Entwicklungsumgebung nach Ceravola et. al. [Cer06], wie sie für eine erfolgreiche, langfristige Systementwicklung notwendig ist. Basierend auf

diesen Anforderungen wird im Folgenden das bereits auf dem BIRON-Demonstrator verwendete Kommunikations-Framework [Wre09] betrachtet. Dieses wurde unter der Leitung von Sebastian Wrede parallel zu dieser Arbeit zu einem *Event-basierten* Kommunikationsprozess weiterentwickelt. In seiner aktuellen Version verfügt es außerdem über eine persistente Datenverwaltung, wodurch es hinsichtlich syntaktischer und semantischer Kopplung auf der Datenkommunikationsebene für die Erfüllung der beschriebenen Anforderungen in Frage kommt.

4.2.1. Datenaustausch via *XCF*

Das für den direkten Datenaustausch zwischen Software-Modulen bereits auf BIRON verwendete *XML based Communication Framework* (*XCF*) realisierte zu Beginn dieser Arbeit bereits einen Teil der erwähnten Auswahlkriterien [Fri07]. Dazu gehören einfache Anwendbarkeit, asynchroner und in einem Netzwerk verteilter Datenaustausch sowie eine von Menschen lesbare Datenrepräsentation. Es stellt zwei Kommunikationsmodelle zur Verfügung, zum einen Datenströme für 1:n Verbindungen mit einem Sender (*Publisher*) und mehreren Empfängern (*Subsribers*) und zum anderen 1:1 Verbindungen über *Remote-Method-Invocation* (*RMI*). Datenströme bieten sich an, wenn Informationen in hoher Frequenz generiert werden und ungerichtet systemweit zur Verfügung gestellt werden, ohne dass eine Rückmeldung über den Erhalt der gesendeten Information benötigt wird. In den hier vorgestellten Systemen finden Datenströme bei der Übertragung von unverarbeiteten Sensordaten wie Odometrie oder Distanzmessungen Verwendung.

Für Funktionen, deren Ausführung durch eine Rückmeldung gewährleistet werden muss, wie beispielsweise die erfolgreiche Arbitrierung der Hardware, werden RMIs eingesetzt. Für RMI stellt ein Modul einen Funktionen-Server zur Verfügung, dessen Funktionen zu einem gegebenen Zeitpunkt von einem weiteren Modul über einen Remote-Server aufgerufen werden kann. Der aufgerufenen Funktion kann ein Argument übergeben werden, genauso wie die Funktion ein Ergebnis zurückliefern kann.

Während bei der Genom-Architektur [Ala98] der Datenaustausch über gemeinsam genutzten Arbeitsspeicher eines Rechners realisiert wird, verwendet XCF ein eigenständiges *Dispatcher*-Modul (siehe Abb. 4.1) bei dem sich alle beteiligten Publisher, Subscriber, Funktionen-Server und Remote-Server anmelden, so dass die Daten auch rechnerübergreifend zugestellt werden. Zwar verringert sich damit die Übertragungsgeschwindigkeit im Vergleich zu gemeinsam genutzten Speicher erheblich, jedoch sind die Ressourcenansprüche bezüglich des Datenaustausches in der hier vorgestellten Domäne relativ zu der benötigten Rechenkapazität deutlich geringer. Während die Rechenkapazität in verteilten Systemen durch Erweiterung des Netzwerkes mit weiteren Computern einfacher erhöht werden kann, ist dies in einem geschlossenen System mit gemeinsamen Arbeitsspeicher aufwendiger.

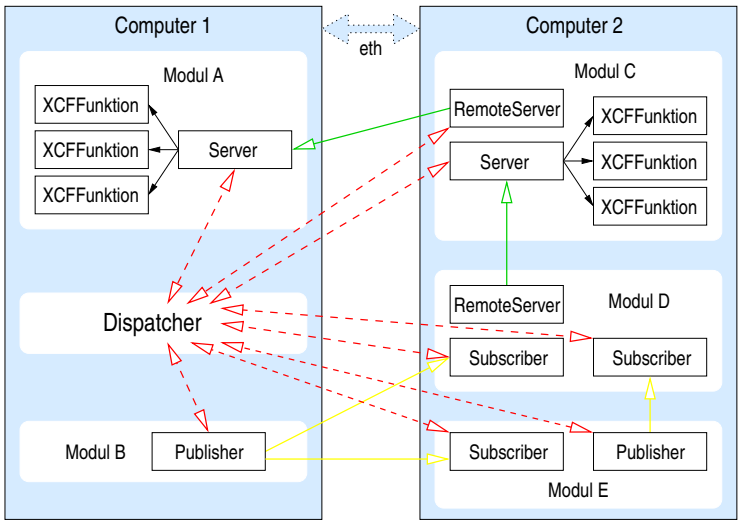


Abbildung 4.1.: Beispiel für die Kommunikation mehrerer Module über XCF. Ein Modul kann Publisher, Subscriber, Funktionen-Server und Remote-Server verwenden, die sich zum Datenaustausch in einem aus mehreren Computern bestehenden Netzwerk an einem Dispatcher Modul anmelden.

4.2.2. Flexible Schnittstellendefinition mit *XMLTIO*

Wie die Bezeichnung „XML based Communication Framework“ vermuten lässt, wird als generelle Datenrepräsentation für die Kommunikation XML verwendet. Es bietet den Vorteil einer weiten Verbreitung und Akzeptanz seitens der Modulentwickler. Da XML im Gegensatz zu Binärdaten als Text lesbar ist, können die übertragenen Daten direkt von den Entwicklern gelesen und Fehler einfacher gefunden werden. Ferner existieren eine Großzahl an Werkzeugen, um XML-Dokumente zu erstellen, ihren Inhalt syntaktisch und semantisch zu prüfen und zu spezifischen Abschnitten zu navigieren, falls nur Teilinformationen benötigt werden. In dem hier beschriebenen Ansatz wird auf die *Schema-Validierung* und auf *XPath-Expressions* des *World Wide Web Consortium* (*W3C*) zurückgegriffen.

Um die Anbindung ihrer Module an XCF und XML als Datenrepräsentation zu erleichtern, wird eine von Ingo Lütkebohle entwickelte Bibliothek, das *XML Template I/O* oder kurz *XMLTIO* verwendet. Dieses ermöglicht unter Verwendung von XPath einen direkten Lese- oder Schreib-Zugriff auf beliebige Bereiche des XML-Dokumentes. Ferner können mit XMLTIO Templates definiert werden, die es dem Entwickler ermöglichen, komplexe Datenstrukturen von der Repräsentation einer Programmiersprache direkt in ein XML-Dokument zu überführen und umgekehrt. Bezüglich der Systemintegration kann dem Modulentwickler somit eine fertige Funktion zur Konvertierung seiner modulspezifischen Datenstruktur in eine generische XML-Repräsentation zur Verfügung gestellt werden. Dies erhöht die Akzeptanz des Frameworks, da der Entwickler weniger Aufwand mit Anpassungen seiner Komponente hat, und erleichtert in Kombination mit einer bei der Umwandlung automatisch vollziehbaren Schemavalidierung die Fehlersuche.

4.2.3. Datenspeicherung im *Active Memory*

Obwohl XCF in Kombination mit XMLTIO eine gute Möglichkeit bietet, um schnell und einfach unterschiedliche Module miteinander zu verbinden, so birgt die intensive Nutzung von Datenströmen und RMI die Gefahr der eingangs erwähnten Kopplung. Selbst wenn die Module semantisch entkoppelt seien mögen, so ergeben sich durch Datenströme syntaktische Kopplungen der Subscriber von den jeweiligen Publishern. Bricht das sendende Modul ab, so fällt auch der Empfänger aus. Daraus ergibt sich weiterhin eine feste Startreihenfolge, bestehend aus Ketten von Modulen, bei denen ein sendendes Modul immer vor seinem Empfänger gestartet werden muss. Gleiches gilt auch bei der RMI-Kommunikation. Nur wenn ein Funktionen-Server aktiv ist, können die zugehörigen Remote-Server-Module ausgeführt werden. Ein Neustart einzelner Module, oder sogar Austausch zur Laufzeit ohne das restliche System in Mitleidenschaft zu ziehen, ist damit kaum möglich.

Des Weiteren bleibt die Frage offen, wie mit semantischen Kopplungen im Falle eines Datenverlustes durch Kommunikationsstörung umgegangen werden kann. Während bei RMI eine entsprechen Rückmeldestrategie zumindest auf die Situation hinweist, gehen Informationen bei einem unerwarteten Abbruch von Datenströmen unbemerkt verloren. Eine persistente Datenspeicherung, die gleichzeitig eine Entkoppelung von Modulen auf der Kommunikationsebene realisiert, bietet die Erweiterung von XCF um das *Active Memory*. Das Active Memory (AM) realisiert einen Touple Space für Informationen wie von [Cab99] vorgeschlagen, basierend auf einem Datenbank-Managementsystem [Lüt04]. Die Bezeichnung *Active* bezieht sich auf den integrierten, Event-basierten Datenaustausch. Statt einer direkten Kommunikation zwischen zwei Modulen, registriert sich ein Modul bei der Datenbank auf die benötigten Daten, die mittels XPath spezifiziert werden. Zusätzlich wird bei der Registrierung ein Datenbank-Event wie das Einfügen, Verändern oder Löschen dieser Informationen definiert. Tritt ein solches Event in Zusammenhang mit den über XPath spezifizierten Daten auf, werden die betreffenden Daten dem registrierten Modul übermittelt (siehe Abb 4.2).

Genauso wie sich jedes Modul auf beliebige Events registrieren kann, kann auch jedes Modul Daten in das AM einfügen oder verändern. Es wird bei den an einem Datenaustausch beteiligten Modulen zwischen *Event-Sources* als Informationsquellen, die Daten in das AM einfügen oder verändern, und *Event-Sinks* als registrierte Module, die Informationen verwenden, unterschieden. Dabei ist zu beachten, dass ein Modul für eine Datentyp als Event-Source dienen kann und gleichzeitig Event-Sink eines anderen Datentyps darstellt. Die erste Anwendung dieses Verfahrens fand im Vampire Projekt [Han06, Wre06] statt und wurde dort als eigenständiges Architekturmodell verwendet. Im Rahmen der hier präsentierten Arbeit zeigte sich jedoch die Bedeutung von Aspekten wie Schnittstellenadaption, Ressourcen Arbitrierung und Sequenzierung mit Komponentenkoordination, die nicht Bestandteil von XCF sind. Der hier beschriebene Ansatz verwendet XCF und erweitert es in der resultierenden Architektur um eben diese noch offenen Punkte. Dadurch wird die einfache Generierung komplexer Systemverhalten für die in Abschnitt 3.1 beschriebenen Szenarien auf der Architekturebene ermöglicht und von den Modulen entkoppelt.

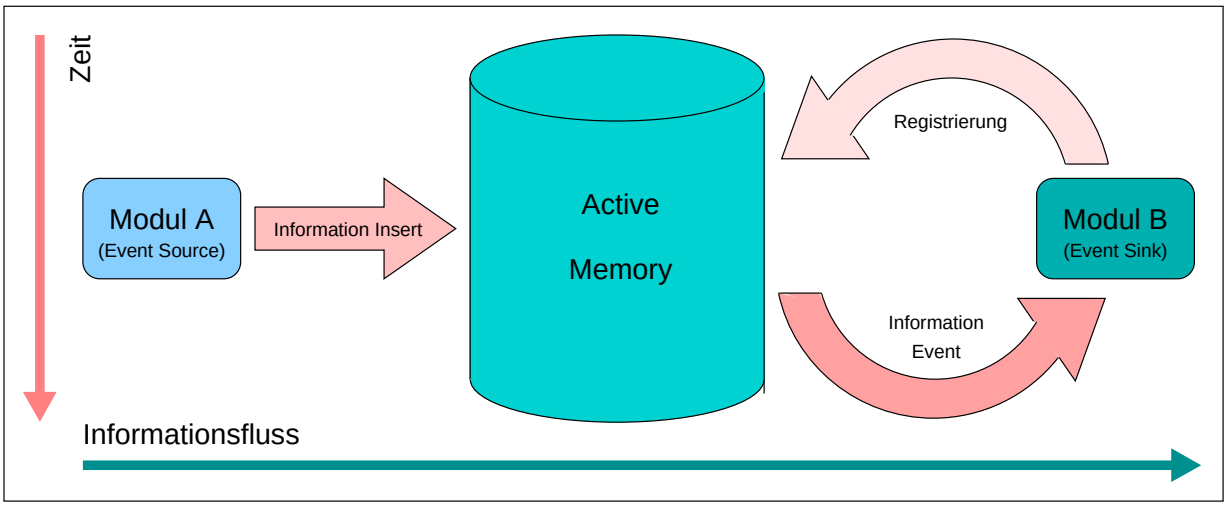


Abbildung 4.2.: Modul A stellt Daten als Event-Source zur Verfügung. Nachdem sich Modul B einmalig auf einen spezifizierten Datentyp registriert hat, wird es vom AM automatisch benachrichtigt. Die Module A und B haben keine direkte Verbindung zueinander.

4.3. Remodellierung einer zentralen Kontrolle

Auch vor der Anwendung des evolutionären Ansatzes (siehe Abb. 3.15, Seite 49) wurde der Strategie gefolgt, Arbitrierung und Komponentenkoordination durch die Verwendung eines zentralen Kontrollmoduls, des sogenannten *Execution Supervisor*s (*ESV*), zu realisieren. Die Aufgabe des ESV war es, den hier vorgestellten und in den beschriebenen Szenarien verwendeten Komponenten Zugriff auf dieselben Ressourcen, wie Sensoren und Aktuatoren zu ermöglichen. Während das Personen-Tracking die Kamera auf Gesichter auszurichten versucht, benötigt ein Ansatz des Umgebungslernens die Kamerakontrolle zur Raumbeobachtung. Eine alternative Lokalisierung lässt den Roboter rotieren, obwohl für das Personenfolgen Wegpunkte angefahren werden sollen. Eine Arbitrierung (siehe Abschnitt 2.1.4) der asynchron operierenden Module ist folglich unumgänglich. Hierzu verlief der gesamte Informationsaustausch aller Module über den als endlichen Automaten realisierten ESV.

Ohne auf die Details dieser Arbeit [Kle05] einzugehen, wurden alle Informationen über, in den Komponenten namentlich kodierte, feste XCF-Datenströme an den ESV geschickt. Die übermittelten Daten bewirkten als *Eingabe* eine Transition zwischen den modellierten Systemzuständen. Als Resultat einer Transition wurden *Ausgaben* an weitere Module via XCF-Datenstrom verschickt. Trotz der aus Abbildung 4.3 ersichtlichen Komplexität war es notwendig, auch in Einzelkomponenten, wie dem Dialog und dem Personen-Tracking, Informationen über Systemzustände und Transitionen zu hinterlegen. Dies bewirkte eine enge semantische Kopplung von Komponenten untereinander und einen entsprechenden Aufwand bei Erweiterungen des Systems, da Änderungen in einer Komponente auch Modifikationen in den angeschlossenen Mo-

dulen mit sich führten. Begleitet wurde dieser Effekt durch eine hohe syntaktische Kopplung in Form einer hohen Anzahl von direkten Verbindungen der Komponenten untereinander, die als Datenströme mit Sender und Empfänger realisiert waren. Wies ein Sender eine Fehlfunktion auf, wurden alle nachfolgenden Empfänger und deren Empfänger in Mitleidenschaft gezogen. Weiterhin bedurfte die Abhängigkeit eines Empfängers von einem Sender eine feste Startreihenfolge bei der Initialisierung der Module, so dass sich hier zusätzliche Kopplungen auf syntaktischer und semantischer Ebene ergaben.

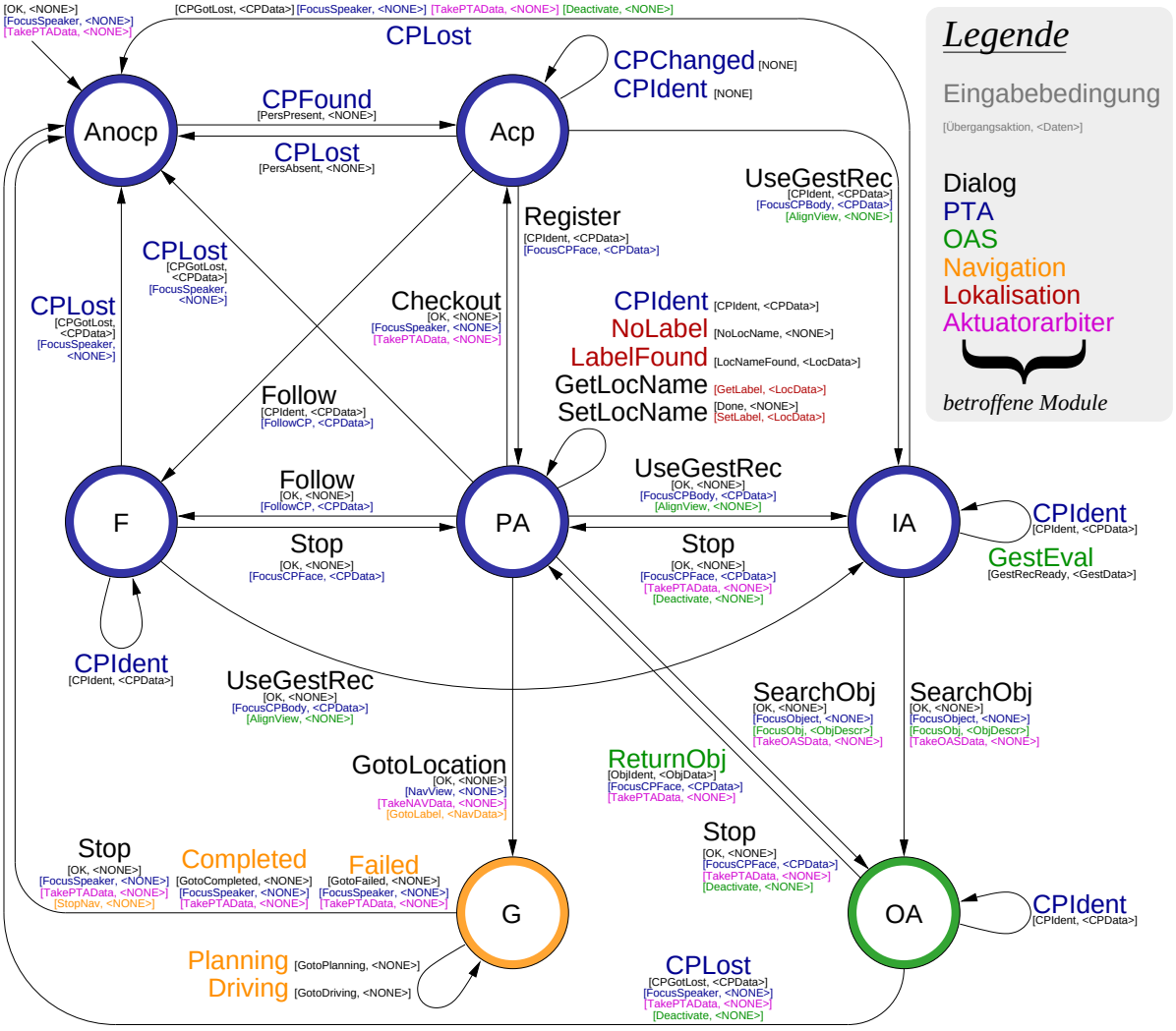


Abbildung 4.3.: Modellierung des endlichen Automatens für den situationsabhängigen Nachrichtenaustausch der einzelnen Komponenten. Die statisch implementierten Datenverbindungen zu den angeschlossenen Modulen und ihre gegenseitige Abhängigkeit von den Prozessabläufen anderer Komponenten erschwerten Systemmodifikationen.

Diese Initialisierungsreihenfolge ist nicht nur seitens der Sender-Empfänger-Beziehung kritisch. Vielmehr erschwert das fehlende Einbeziehen persistenter Informationsspeicherung die Systemkoordination im Fehlerfall. Fällt eine Komponente aus, so müssen gegebenenfalls alle Module, die mit der Komponente direkt oder indirekt über andere Komponenten Daten austauschen, neu gestartet werden, um erneut ein synchrones Gesamtsystem zu gewährleisten. Der aktuelle Systemzustand wird nicht bereitgestellt und neu gestartete oder zur Laufzeit flexibel hinzugefügte Komponenten werden erst durch entsprechende Transitionen innerhalb des ESV mit Daten versorgt. Der ESV verfügte aber über keine Möglichkeit, bei zwischenzeitlichem Komponentenstart ausreichend flexibel zu reagieren und diese aktiv mit den zwischenzeitlich angefallenen Informationen zu versorgen. Auf Basis dieser Analyse werden für die Integration neuer Komponenten in die hier vorgestellte Systemarchitektur folgenden Paradigmen angewendet:

1. Einzelne Komponenten müssen so wenige Informationen wie möglich über die Funktion oder Existenz anderer Komponenten beinhalten oder für ihre eigene Ausführung verwenden, um die Komponenten semantisch so wenig wie möglich miteinander zu koppeln. Es wird dem Ansatz von Sosio und Tisato [Sos99] in soweit gefolgt, dass architekturrelevante Bestandteile explizit in eigenständige Komponenten auszulagern und die übrigen Funktionen in separaten Modulen zu realisieren sind.

2. Die Verwendung einer festgelegten Klasse als Prototyp für alle Module ist ungeeignet für das Zusammenführen von Modulen aus anderen Institutionen, da diese sowohl in ihren Schnittstellen, als auch Prozessroutinen dem Prototypen folgen und neu implementiert werden müssten. Verhaltensweisen, die ein koordiniertes Zusammenspiel mehrere Einzelkomponenten erfordern, werden zentral außerhalb der Komponenten verwaltet, ohne jedoch bei den Komponenten auf zuvor definierte Prototypen zurückgreifen, wodurch der vorhandenen Code weitestgehend unberührt bleiben kann.

3. Die Schnittstellen einer neuen Komponente müssen der Datenrepräsentation des Gesamtsystems (hier zum Beispiel XML) entsprechen und gegebenenfalls innerhalb der Komponente angepasst werden. Die Architektur muss jedoch Maßnahmen zur Verfügung stellen, Schnittstellenabweichungen innerhalb der gleichen Repräsentation zu kompensieren.

4. Der Informationsaustausch zwischen Komponenten findet über einen zentralen, persistenten Datenspeicher statt. Bei Teilausfall oder dynamischem Komponentenstart zur Laufzeit sind aktuelle Systeminformationen sowie Zwischenergebnisse verfügbar und können von den Komponenten jederzeit abgerufen werden, um die Systemsynchronisation wieder herzustellen. Dies gilt ebenso für die zentrale Koordination, deren Ausfall in der Ursprungsarchitektur einen kompletten Systemausfall bedingt hätte.

5. Potentielle Schnittstellenabweichungen von externen Komponenten werden durch Datenmanipulation direkt auf den Informationen im Datenspeicher aufgelöst. Dadurch bedarf dieser Zwischenschritt keiner Veränderung in der Kommunikationsstruktur, wie sie im Falle von Datenströmen nötig wäre.

Als Resultat dieser Überlegungen wurde in dieser Arbeit ein Koordinationsmodul entwickelt, welches Aspekte der Systemarchitektur an einem zentralen Punkt vereint, anstatt diese implizit in den beteiligten Komponenten zu realisieren. Dieses Modell bietet vier Integrationsstrategien für Schnittstellendefinition, Arbitrierung, Sequenzierung und die Komposition des gewünschten Systemverhaltens. Es ist als eigenständige Komponente, das *Active Control Memory Interface* (*ACMI*) implementiert. ACMI realisiert keinen Controller im herkömmlichen Sinne, der andere Komponente direkt ansteuert und sich an der Spitze einer Hierarchie befindet. Vielmehr ist ACMI ein dem Prozessablauf nach allen anderen Komponenten gegenüber gleich berechtigtes Modul, welches Informationen im ActiveMemory des Systems einfügt, löscht oder verändert. Es unterscheidet sich vielmehr darin, dass es die, den anderen Modulen vorenthaltenen Systeminformationen verwendet, um Memory-Inhalte derlei zu modifizieren, dass das gewünschte Systemverhalten entsteht. Somit ist es möglich, neue Komponenten in das bestehende System einzufügen sowie die Anzahl notwendiger Modifikationen am bestehenden System zu minimieren und zu zentralisieren [Spe08b]. Der Aufbau der jeweiligen Integrationsstrategie wird in den folgenden Kapiteln veranschaulicht.

4.4. Zusammenfassung

Welche Aspekte gilt es bei der Systemintegration besonders im Bereich Mensch-Roboter-Interaktion zu beachten, um eine ausreichend flexible Systemarchitektur zu entwickeln, die es ermöglicht, auf unvorhergesehene Erkenntnisse schnell einzugehen und in ihrer Anwendung dennoch Praktikabel bleibt? Das vorangegangene Kapitel zeigt auf, wie wichtig es ist, bereits bei der Konzeption eines Systems auf die Erweiterbarkeit zu achten. Ergebnisse aus zum Beispiel Benutzerstudien müssen schnell und flexibel auf das System übertragbar sein. Erweiterte Roboterfähigkeiten in Form von neuen Software-Modulen sollen nicht auf Grund zu hohen Integrationsaufwandes zurückgestellt werden. Ein modular aufgebautes System erfüllt diese Anforderungen, wenn es durch ein geeignetes Framework für den Datenaustausch unterstützt wird. Das hier vorgestellte Framework bietet sowohl klassische Kommunikationsmöglichkeiten mit Publish/Subscribe- und RMI-Verwendung, als auch Event-basierte Kommunikation über als Memory verwendete Datenbanken. Besonders der Einsatz der Event-basierten Kommunikation ermöglicht eine möglichst lose Kopplung von Modulen, da diese nicht wie bei Publish/Subscribe und RMI direkt miteinander verbunden sind und so keinerlei wissen über andere im System aktive Komponenten besitzen müssen. Des Weiteren sollen die einzelnen Komponente keine architekturbezogenen Informationen beinhalten, die sie semantisch miteinander verbinden, dies vereinfacht das Hinzufügen oder Austauschen einzelner Komponenten. Dennoch müssen die Informationen über verwendete Schnittstellen, Arbitrierung begrenzter Ressourcen, Sequenzierung bei Beteiligung mehrerer Komponenten und das angestrebte Systemverhalten verfügbar sein. Diese werden zentral im ACMI-Modul verwaltet, dessen Aufgabe es ist, über Modifikation der Memory-Inhalte genau diese vier Punkte zu realisieren, ohne dass der Modifikationsvorgang an sich für das restliche System sichtbar ist.

5. Evolutionäre Systemintegration

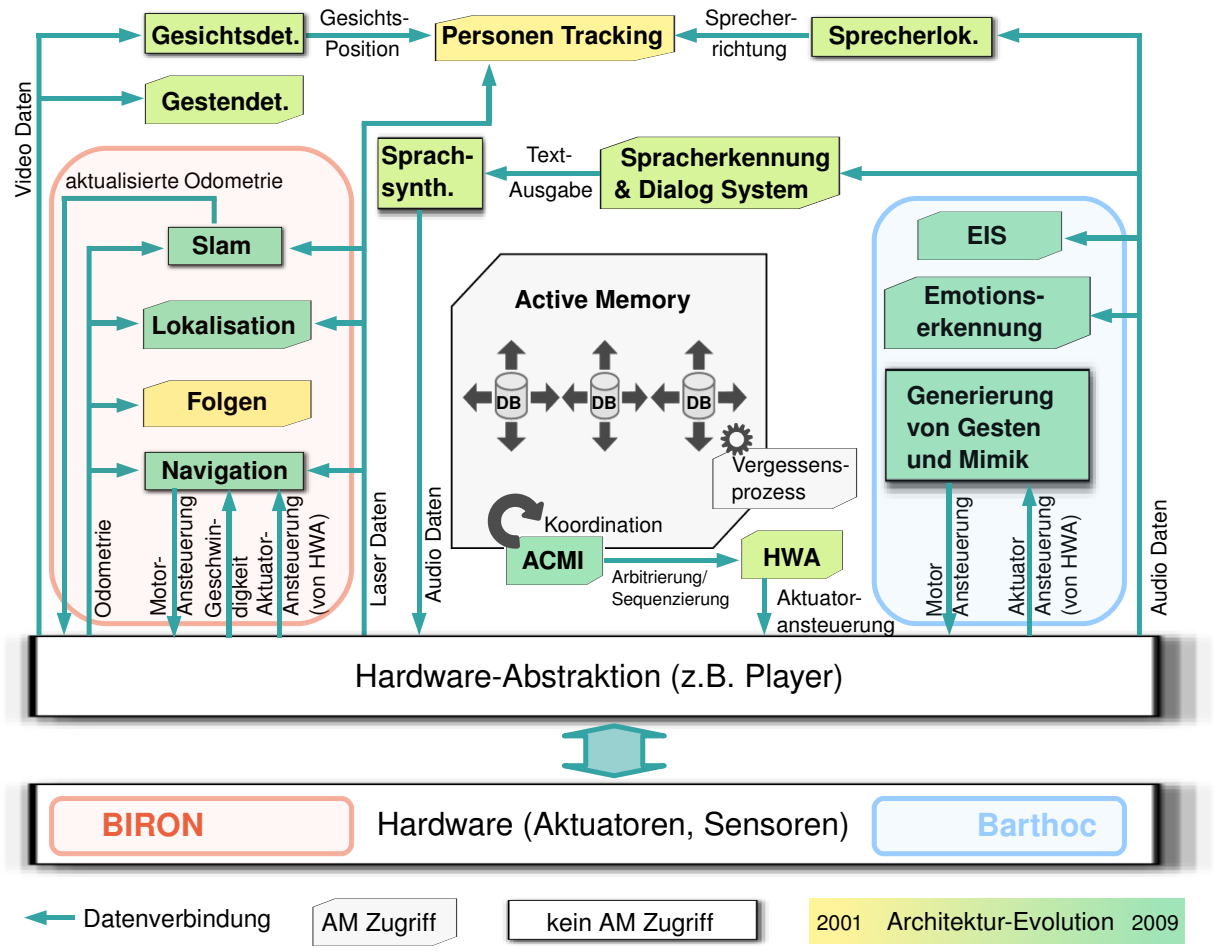


Abbildung 5.1.: Architektur-Evolution vom Entwicklungsbeginn 2001 bis 2009. Es werden sowohl historische Komponenten ohne Memory-Zugriff, als auch neue Komponenten erfolgreich und strukturiert miteinander verknüpft. Die Evolutionäre Architektur zur Systemintegration (EASI) ist nicht nur flexibel gegenüber der zeitlichen Entwicklung, sondern auch gegenüber den verwendeten Demonstratoren. Sie wird auf den Systemen BIRON und Barthoc erfolgreich eingesetzt und vereint demonstratorspezifische Komponenten mit Fähigkeiten, die gemeinsam für unterschiedlichen Szenarien verwendet werden.

Die Überlegungen des vorangegangenen Kapitels bezüglich Systemarchitektur, Kommunikations-Framework, und zentraler Systemkoordination prägen die Umsetzung der in diesem Kapitel beschriebenen *evolutionären Architektur zur Systemintegration* (*EASI*). Obwohl auch der EASI-Ansatz eine zentrale Koordination in Form eines einzelnen Moduls verwendet, beschreibt diese Arbeit keine starre hierarchische Kommandostruktur. Vielmehr muss eine moderne Architektur den sich besonders in der Wissenschaft kontinuierlich verändernden Umständen Rechnung tragen und eine Evolution bezüglich integrierter Komponenten und Schnittstellen unterstützen. Die Architektur wird geprägt von den Anforderungen, die durch neu hinzukommende Komponenten gestellt werden (siehe Abb. 5.1). Sie restringiert diese nicht, sondern stellt vielmehr einen Leitfaden für eine möglichst reibungslose Integration zur Verfügung. Die Befolgung des Leitfadens ist erstrebenswert, aber keine Grundvoraussetzung für die erfolgreiche Integration. Vielmehr muss der Leitfaden ausreichend flexibel sein, um sich neuen Entwicklungen und Erkenntnissen anzupassen, ohne jedoch in dem „Gewirr“ eines gordischen Knotens zu enden.

Dieses Kapitel beschreibt die Umsetzung der zuvor angestellten Überlegungen bei der Gestaltung einer evolutionären Architektur zur Systemintegration. Ein besonderes Merkmal besteht hierbei in dem Erhalt vorhandener oder der Integration fremder Software-Komponenten, welche entsprechend ihres Alters oder ihrer Herkunft nicht dem aktuellen Leitfaden der gegebenen Architektur folgen. Das nun vorgestellte Modell einer zentralen Verwaltung der Architekturaspekte Schnittstellen, Arbitrierung, Sequenzierung und des hieraus resultierenden Systemverhaltens greift die Idee dieses flexiblen Leitfadens auf. Statt einer starren Kontrolle, wird die Idee einer flexibleren Koordination verfolgt, die auf feste Schnittstellen zu den koordinierten Komponenten verzichtet.

5.1. Das Active Control Memory Interface

Die Aufgabe des *Active Control Memory Interface* (*ACMI*) ist die zentrale Koordination des Datenaustausches, für dessen Bearbeitung modulübergreifende Informationen notwendig sind. Diese Informationen sollen nicht über das System gestreut werden, indem sie Bestandteile der Modulprogrammierung sind. So wird eine einfachere Modifikation des Systems ohne ungewollte Seiteneffekte ermöglicht. Im Gegensatz zu Ansätzen mit einer zentralen Verwaltung wird durch ACMI keine direkte Steuerung von anderen Komponenten favorisiert, vielmehr wird in dieser Arbeit die These *Koordination statt Kontrolle* als erfolgreiche Integrationsstrategie aufgestellt und verfolgt. Komponenten, die für Ihre Aufgabe keine modulübergreifenden Informationen verwenden, bedürfen auch keiner Kontrolle. Nicht für alle Operationen einer Komponente, die gelegentlich modulübergreifenden Informationen verwendet, ist eine Kontrolle notwendig. Wo es aus oben genannten Gründen möglich ist, wird die Idee einer Selbstorganisation ähnlich zu [Gri08] verfolgt, mit dem Unterschied, dass dies nicht auf der Ebene unterschiedlicher Roboter, sondern bei Software-Komponenten geschieht, deren Verknüpfung untereinander flexibel zur Laufzeit reorganisiert werden kann. Die Vermeidung fester Kontrollstrukturen ermöglicht den Verzicht auf entsprechende Schnittstellen der verwendeten Komponenten zu einem Controller und verringert die Voraussetzungen, die ein Entwickler bei der Integration seiner Software zu erfüllen hat.

Maßgeblich unterstützt wird diese Art der Koordination durch die Verwendung des Active Memorys. Da sich der Informationsaustausch an diesem Punkt bündelt, wird implementatorisch auf direkte Verbindungen zwischen ACMI und den einzelnen Modulen verzichtet. Stattdessen existieren Schnittstellen zwischen den Datenbanken des Memorys und ACMI. ACMI manipuliert die sich im Memory befindlichen Daten so, dass ein gewünschtes Verhalten der Einzelmodule und folglich des Gesamtsystems entsteht. Durch die Datenmanipulation im Memory bleibt ACMI für die über das Memory angeschlossenen Komponenten unsichtbar im Hintergrund.

Als sicherheitsrelevante Ausnahme von dieser Vorgehensweise hat sich die direkte Verbindung der Hardware-Kontrolle des jeweiligen Demonstrators mit der Koordinationskomponente über RMI herausgestellt. Die als Hardware-Kontrolle bezeichnete Komponente übernimmt die Ansteuerung und Arbitrierung des jeweiligen Demonstrators und ist individuell für ein System konzipiert. Einmal entwickelt und bei konstanter Hardware wird sie in der Regel nicht mehr verändert. Sie unterliegt damit deutlich weniger dem Flexibilitätsanspruch komplexerer HRI-Komponenten zur Interaktion, was in diesem Fall den Nachteil einer festen Verbindung zum ACMI gegenüber ihrem Vorteil zurücktreten lässt. Dieser Vorteil besteht aus der garantierten Rückmeldung, die die Koordination über den aktuellen Zustand der Hardware erhält, wenn zum Beispiel die Arbitrierung für die Generierung eines konsistenten Systemverhaltens (siehe Kapitel 6.1) durchgeführt wird. Trotz dieser sicherheitsrelevanten Erweiterung entspricht das vorgestellte Modell einer Systemkoordination, die sich so weit wie möglich zurück hält und nur im Bedarfsfall aktiv wird.

Wann dieser Bedarfsfall eintritt, lässt sich auf unterschiedliche Weise definieren. In der hier vorliegenden Implementierung (siehe Abb. 5.2) wird eine Regelbasis aufgestellt, die definiert, welche Aktionen wann auszuführen sind. Die Regeln entsprechen dem von Goldin et al. in [Gol04] vorgestellten *Event-Condition-Action-Modell* (*ECA*). Demnach wird eine Aktion ausgeführt, wenn ein zu spezifizierendes Ereignis (engl. Event) stattfindet und eine oder mehrere Voraussetzungen (engl. Condition) zum Zeitpunkt des Ereignisses erfüllt sind. Als Ereignisse im Sinne der Regelauswertung finden die Datenbank-Events des Active Memorys Anwendung. ACMI registriert sich auf diejenigen Datenbank-Events, die Bestandteil eines modulübergreifenden Informationsaustausches sind. Als Condition wird der aktuelle Systemzustand verwendet. Diese Vermischung von Ereignissen und Zuständen erweitert die wenn-dann-Syntax klassischer Regelproduktionssysteme um kontextabhängige Auswertung. Im Vergleich zu alternativen Kontrollmodellen, wie endlichen Automaten oder Petri-Netzen, werden Zustände nicht künstlich in das Modell übertragen oder innerhalb des Controllers mitverwaltet. Der Systemzustand lässt sich einfach durch den aktuellen Inhalt des Memorys auswerten, auf den bei der Regelauswertung jederzeit zugegriffen werden kann.

Sowohl bei der Event-Registrierung, als auch bei jedem anderen Memory-Zugriff wird seitens ACMI analog zu allen anderen an das Memory angeschlossenen Komponenten vorgegangen, so dass ACMI aus syntaktischer Sicht keine Sonderposition einnimmt. Es nutzt vielmehr die vorhandene Infrastruktur in gleicher Weise, wie alle anderen Komponenten, was, abstrahiert von der konkreten Implementation, ein klarer Vorteil des Modells bei Anwendung auf andere existierende Systeme und deren Kommunikationsinfrastruktur ist.

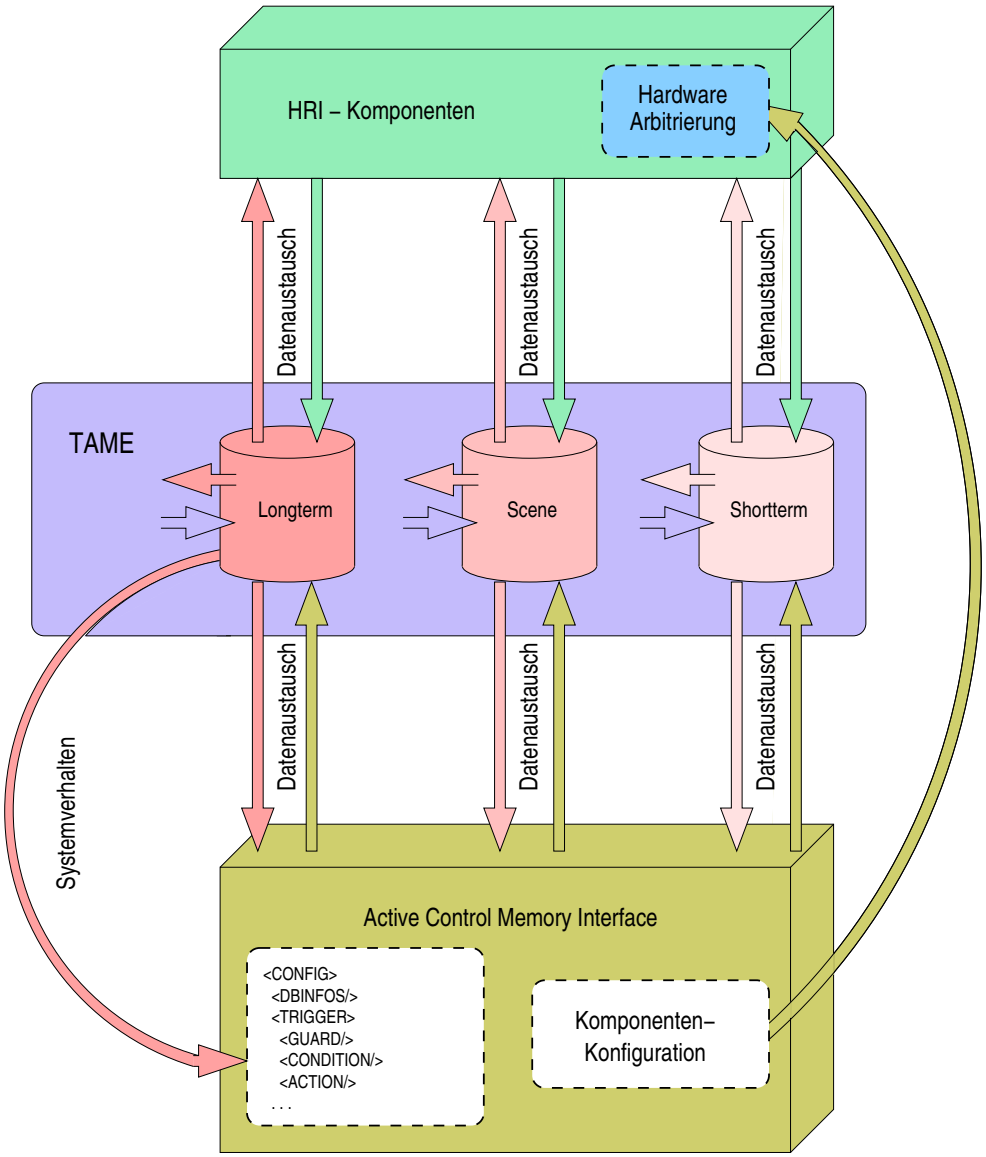


Abbildung 5.2.: Funktionsprinzip des Active Control Memory Interface (ACMI): Die integrierten HRI-Komponenten tauschen Daten über die Datenbanken des Memorys aus, ohne direkt mit ACMI in Verbindung zu stehen. ACMI hat Zugriff auf alle Daten des Memorys und kann diese bei Bedarf verändern. Welche Veränderungen durch ACMI wann durchgeführt werden, ist in einer XML-Konfiguration abgelegt, die selbst Bestandteil des Memorys ist. Somit kann die Konfiguration genau so wie alle anderen Elemente des Memorys mit einem Editor (TAME) zur Laufzeit von außen modifiziert werden.

Die Wiederverwendung bereits existierender Mechanismen ermöglicht ein schlankes Modell mit wenigen unterschiedlichen Repräsentationsformen. So wird auch die Regelbasis selbst in XML-Notation als Element im Memory verwaltet. Mit den gleichen Methoden, die den Modulen Zugriff auf ihre Daten ermöglichen, kann jederzeit - auch zur Laufzeit - die Regelbasis modifiziert werden. Für den vereinfachten Zugriff auf Memory-Daten im Allgemeinen und die Bearbeitung der Regelbasis im Besonderen wurde im Rahmen dieser Arbeit ein Editor (*TAME - Trigger, Action, and Memory Editor*) entwickelt, dessen Funktionen in Kapitel 6.3 genauer beschrieben werden. Da sich ACMI genau so auf seine Konfiguration, wie auf die Moduldaten registriert, werden Modifikationen an der Konfiguration sofort übernommen, ohne dass ein Neustart notwendig ist. Dies unterstützt den Integrations- und Entwicklungsprozess, da Modifikationen und Fehlerbehebung auf Gesamtsystemebene zur Laufzeit durchgeführt werden können.

Entwicklung Modulübergreifenden Verhaltens

Bei dem Entwurf der Regelbasis wurde besonders auf eine übersichtliche und schnell zu erlernende Syntax geachtet und der Aussage *„It is the interaction, processing, and results of simple algorithms which produce complex, intelligent behavior“* [Kel06] gefolgt. Als Resultat wurde ein möglichst kleiner Satz unterschiedlicher Regeltypen verwendet, um ein komplexes Verhalten zu generieren, anstatt eine breite Basis an Regeltypen für die unterschiedlichsten Situationen zur Verfügung zu stellen. Dies ermöglicht es dem Anwender, sich schnell in die Verwendung der Regeln einzuarbeiten und Veränderungen an der Regelbasis vorzunehmen. Zwar erhöht sich bei weniger komplexen Regeltypen häufig die Anzahl der Regeln in einer Regelbasis, durch die im weiteren Verlauf dieses Abschnittes erklärten Maßnahmen wird jedoch vermieden, dass es zu größeren Abhängigkeiten der Regeln in einer Regelbasis kommt. Der Großteil der Regeln ist so angelegt, dass eine Regel für sich alleine steht. Bei ihrer Veränderung werden keine Seiteneffekte auf andere Regeln innerhalb der Regelbasis ausgeübt. Dadurch ist auch eine größere Anzahl von Regeln einfach zu warten. Im Fall der Implementierung für BIRON (siehe Anhang A.1) werden insgesamt 37 Regeln verwendet, um dessen Verhalten in der Home-Tour zu Koordinieren.

Alle Regeln sind, wie in Abb. 5.3 dargestellt, gleich aufgebaut und realisieren auf Seite der Implementation einen Datenbank-Trigger. Eine Regel besteht aus einer `CONDITION` und einer `ACTION`, optional können eine beliebige Anzahl von `GUARD`- Elementen und ein `AFFIRM`-Element hinzugefügt werden. Die `CONDITION` beschreibt das Datenbank-Event, welches stattfinden muss, damit die zugehörige `ACTION` ausgeführt wird. Parametrisiert werden `CONDITION` und `ACTION` durch die Datenbankbezeichnug innerhalb des Memorys, für die diese Regel gültig ist, deren mittels XPath spezifizierten Datenbankelemente und die mit diesen Datenbankelementen durchgeführte Datenbankoperation (`action`). Um die Regeln aus oben genannten Gründen disjunkt zu halten, wird eine genaue Spezifizierung der `CONDITION` vorgenommen. Das Hinzufügen von `GUARD`-Elementen ermöglicht die Verknüpfung von Ereignissen mit Systemzuständen, welche durch die aktuellen Zustände aller im Memory befindlichen Datenbanken repräsentiert werden. Durch diese kontextabhängige Regelauswertung werden unbeabsichtigte

```
1 <TRIGGER>
2  <GUARD database="Scene" xpath="/WORLD/EMPTY" exists="yes" >
3  <CONDITION database="ShortTerm" xpath="/HELLO" action="INSERT"/>
4  <ACTION name="REPLACE">
5   <SOURCE database="ShortTerm" xpath="/HELLO" node="/HELLO/PERSON"/>
6   <TARGET database="Scene" xpath="/WORLD" node="/WORLD/EMPTY"/>
7  </ACTION>
8  <AFFIRM value="completed"/>
9 </TRIGGER>
```

Abbildung 5.3.: Regel für das Ersetzen von Daten innerhalb des Memorys. Der Unterknoten `EMPTY` des `WORLD`-Dokumentes wird nach Begrüßung einer Person durch deren Daten ersetzt.

Seiteneffekte zu anderen Regeln vermieden. Nur wenn zu dem Zeitpunkt eines spezifizierten Ereignisses ein gegebener Systemzustand vorliegt, führt dies zur Regelanwendung. Durch einen `GUARD` wird ein Datenbankelement definiert und geprüft, ob dieses zum Zeitpunkt der Regelauswertung Bestandteil des Memorys sein muss oder nicht sein darf. Mehrere `GUARD`s sind durch den logischen $\wedge$-Operator miteinander verknüpft und erhöhen somit die Spezifität der Regel. Das optionale `AFFIRM`-Elementen ermöglicht eine Bestätigung an das unter `CONDITION` spezifizierte Datenbankelement, ob eine Regel erfolgreich durchgeführt werden konnte oder nicht. Dies geschieht indem dem Datenbankelement ein `STATUS`-Element mit der entsprechenden Information hinzugefügt wird.

Syntaktisch unterscheiden sich die Regeln nur durch die Art der von ihnen ausgelösten Aktionen. Das in dieser Arbeit entwickelte minimale Set von Aktionen für die Systemkontrolle besteht aus den folgenden fünf Operationen:

REMOVE entfernt alle spezifizierten XML-Dokumente aus der genannten Datenbank.

COPY kopiert ganze XML-Dokumente oder durch `node` spezifizierte Zweige von Dokumenten. Die Kopie kann unabhängig von dem Art des Originals als eigenständiges Dokument oder als Zweig in ein existierendes Dokument in eine Datenbank eingefügt werden.

REPLACE agiert analog zu `copy`, ersetzt jedoch eventuell vorhandene Informationen am Ziel, indem es vor dem Einfügen der Kopie alte Dokumente oder Zweige entfernt. Die Operation wird innerhalb der Koordination atomar ausgeführt und unterscheidet sich darin zu einer Kombination aus `remove` und `copy`.

CONFIGURE sendet ein XML-Dokument aus dem Memory über RMI an eine Systemkomponente. Die Rückmeldung wird in Form eines XML-Dokumentes im Memory abgelegt.

ASSERT greift die Funktion des `AFFIRM`-Elementes auf und setzt Informationen innerhalb des `STATUS`-Elements, ohne dass eine andere Aktion ausgeführt wurde. Hierdurch können zeitweilig deaktivierte Module nicht sichtbar für das System überbrückt oder Sequenzen von Regeln (siehe Abschnitt 6.2) realisiert werden.

Bei der Überbrückung oder Ergänzung von Informationen während der Entwicklung des Systems zeigt sich ein weiterer Vorteil der einheitlichen Repräsentation von Regeln, Daten und dem sich daraus ergebenden Systemzustand. Fallback-Ergebnisse oder Modulkonfigurationen werden als vordefinierte fertige Dokumente innerhalb des Memorys gespeichert und bei Bedarf an die entsprechende Position des Memorys kopiert oder per RMI verschickt. Durch die Verwendung von XPaths innerhalb der Regeln kann auf unterschiedlichste Dokumente innerhalb des Memorys verwiesen werden. Analog zu verbreiteten Programmiersprachen wird nicht mehr das Datenbank-Element selber Gegenstand des Datenaustausches, sondern dessen XPath als Positionsbestimmung im Memory. Dieses, im Folgenden als *Memory Pointing* bezeichnete Modell verringert die Gefahr von Inkonsistenzen durch multiple Instanzen eines XML-Dokumentes innerhalb des Memorys und unterstützt somit den modifizierenden Zugriff auf die Standardwerte. Es ist keinesfalls beabsichtigt, diese Informationen unberührt im Memory zu belassen, vielmehr sollen diese Werte von einer Startkonfiguration aus weiterentwickelt werden und bieten so Anpassungsmöglichkeiten, die von der aktuellen, entwicklergetriebenen Parameteradaption bis zu einer angestrebten Selbstkonfiguration des Systems nach [Lun07] reichen.

5.2. Schnittstellenanalyse und Adaption

Ein erster Ansatzpunkt für die Adaptionsfähigkeit eines Systems sind die während eines Integrationsprozesses potentiell auftretenden Design-Abweichungen bei den Schnittstellen. Unabhängig davon, welchem Basiskonzept (siehe Kapitel 2.1) eine Systemarchitektur folgt, bilden Schnittstellen die Grundlage für ein erfolgreiches Zusammenspiel der unterschiedlichen Systemkomponenten. Der Begriff „Schnittstelle“ bezieht sich in dieser Arbeit auf den Austausch von Daten zwischen zwei oder mehr Komponenten eines integrierten Systems, wie den bereits beschriebenen Demonstratoren BIRON und Barthoc. Schnittstellen definieren, wie dieser Datenaustausch vollzogen wird. Bestandteil dieser Definition ist einerseits die Art des Datenaustausches, zum Beispiel Datenströme, RMI oder Eventnotification, wie im vorangegangenen Kapitel beschrieben. Andererseits gehört zur Schnittstellendefinition die gewählt Repräsentation, wie Text in XML Notation und Absprache der Einheiten, beispielsweise eine Entfernung in [mm].

Als Grundlage für das gewählte Modell zum Datenaustausch konnte auf Erfahrungen aus dem bereits bestehenden System zurückgegriffen werden. Gleichzeitig beschränkt die Existenz bestehender Komponenten die Entwicklungsmöglichkeiten. Ziel des Schnittstellendesigns ist es, ein tragendes Konzept für zukünftige Integrationen zu erstellen und die notwendigen Aktualisierungen im bestehenden System zu minimieren, ohne diese aufgrund des Arbeitsaufwandes gänzlich zu vernachlässigen. Außerdem musste ein schrittweises Vorgehen für eben diese notwendigen Modifikationen gefunden werden, damit ein System auch in der Umstellungszeit weiterhin funktionsfähig bleibt. Letzteres erweist sich als überaus wichtig. Wird ein System zu lange wegen Umstellung oder Erweiterung deaktiviert, beziehungsweise verbleibt ungenutzt und somit ungetestet, potenzieren sich kleinere Fehler und Ungenauigkeiten der in Bearbeitung befindlichen

Komponenten. Bei erneuter Aktivierung nach mehreren Monaten bedarf es einer umfangreichen Fehlersuche. Außerdem verzögert sich die geplante Reaktivierung häufig, da die beteiligten Entwickler kontinuierlich Erweiterungen an ihren Komponenten vornehmen und so feste Releases nur schwer zu realisieren sind. Natürlich ist diesem Problem theoretisch mit festen Zwischenfristen entgegen zu wirken, doch der Forschungsalltag bei längerfristigen Projekten mit einer Großzahl beteiligter Projektpartner entspricht nicht immer der Theorie.

5.2.1. Design modulübergreifenden Datenaustausches

Im ersten Schritt ist daher zu prüfen, welche Schnittstellen weiterhin sinnvoll verwendet werden können, welche Schnittstellen perspektivisch modifiziert werden sollten und welche Schnittstellen möglichst zeitnah umgestellt werden müssen, da sie nicht weiter unterstützt werden können. In dem von Kleinehagenbrock [Kle05] vorgestellten System wurden für den Datenaustausch vornehmlich Streams verwendet. Sie bieten den Vorteil einfacher, unidirektionaler Kommunikation von einem Sender zu einer beliebigen Anzahl von Empfängern unter geringem Einsatz von Systemressourcen. Weitere Empfänger können einfach hinzugefügt werden ohne Veränderungen am Sender oder den anderen Empfängern vorzunehmen. Durch die unidirektionale Kommunikation ist jedoch nicht sicherzustellen, ob die Informationen den Empfänger erreicht haben, und bei dem Neustart einer Komponenten existiert keine Garantie für Informationskonsistenz aller Module. Dennoch bieten Streams Vorteile, wo Daten mit hoher Frequenz an potenziell mehrere Empfänger verschickt werden und das Verlorengehen eines Datensatzes keine schwerwiegenden Folgen für das System hat. Bei dem hier vorgestellten Datenaustausch trifft dies auf die perzeptgenerierenden Module zu. Die kaum vorverarbeiteten Daten der Sensoren, wie Distanzwerte, Bildinformationen oder akustische Signale, können problemlos als Datenströme mit hoher Frequenz übertragen werden. Natürlich ist ein Ausfall dieser Daten nicht wünschenswert, dennoch hat dies im Einzelfall kaum Auswirkungen auf das System. Da ein Sensor nicht kontinuierlich Werte liefert, wird das versehentliche Fehlen von Daten so gewertet, als ob aktuell keine Messergebnisse vom Sensor vorliegen, womit jedes sensorbasierte System umzugehen hat.

Im weiteren Verlauf werden aus den Sensordaten höherwertige Informationen abgeleitet, die wiederum Aktionen des Systems veranlassen. Um eine derartige Aktion oder Aufgabe zu bewältigen, sind in der Regel mehrere Module beteiligt, die sich untereinander synchronisieren müssen. Gehen in diesem Prozess Informationen verloren, sei es durch einen Fehler in dem Kommunikations-Framework oder durch den Neustart einer beteiligten Komponente, kann die Aufgabe nicht mehr absolviert werden. Um einen konsistenten Systemzustand zu erreichen, muss das gesamte System neu gestartet oder zumindest auf einen Initialwert zurückgesetzt werden. Alle zwischenzeitlich gesammelten Informationen gehen dabei verloren. Es ist einzusehen, dass Streams keine geeignete Kommunikationsart für diese Anforderung darstellen. Daher wurden in [Kle05] RMIs als Alternative eingesetzt, um den Erfolg eines Datenaustausches sicherzustellen. Durch eine Rückmeldung ist es dem Sender möglich, zu erkennen, ob die Daten erfolgreich beim Empfänger angekommen sind. Im Falle eines Fehlers kann der Sender entsprechend

reagieren. Um angemessen zu reagieren, müssen dem Sender jedoch Informationen über das System und die Bedeutung der Empfänger zur Verfügung stehen, welches Sender und Empfänger semantisch enger miteinander koppelt und den Austausch von Komponenten sowie die Erweiterung unter Vermeidung von ungewünschten Seiteneffekten erschwert.

In der hier vorgestellten EASI-Architektur wurde bezüglich der Kommunikationsebene weitestgehend auf die Verwendung von RMI aus oben genannten Gründen verzichtet. Sie findet ausschließlich bei der im folgenden Kapitel beschriebenen Arbitrierung Anwendung und wird daher an dieser Stelle nicht weiter erläutert. Statt dessen findet neben den eingangs erwähnten Streams für wenig vorverarbeitete Sensordaten hauptsächlich das Konzept des Active Memories (AM) (siehe Kap. 4.2.3) Anwendung. Statt direkt Daten zwischen zwei Modulen auszutauschen, werden die Daten vom Sender in das AM gestellt. Der Empfänger registriert sich zuvor einmalig auf den gewünschten Datentyp und bekommt danach vom AM die jeweils eingestellten Daten übermittelt. Dadurch werden Sender und Empfänger syntaktisch voneinander entkoppelt. Beide können nicht einmal feststellen, ob ihr Pendant aktuell gestartet ist oder nicht. Dies erleichtert den Neustart oder Austausch von Komponenten - sogar zur Laufzeit - erheblich. Eine direkte Bestätigung für den Erhalt der Daten beim Empfänger existiert aber auch hier nicht. Wo liegt also der Vorteil gegenüber der Verwendung von Streams? Bei der Kommunikation über das AM wird der Sender darüber informiert, ob das Einstellen der Informationen erfolgreich war, genauso wie der Empfänger informiert wird, wenn ein Fehler bei der Zustellung der Daten vorgekommen ist. Unabhängig von den beteiligten Sendern oder Empfängern hat der Modulentwickler nur zu beachten, was im Falle einer fehlgeschlagenen Kommunikation mit dem AM, zu geschehen hat. In der Regel ist dies für den Sender das erneute Einstellen der Informationen und für den Empfänger eine aktive Anfrage an das AM. Die Anfrage an das AM ist möglich, weil dieses im Vergleich zu Streams die eingestellten Informationen speichert. Diese Speicherung ermöglicht auch den Konsistenzerhalt bei der Synchronisation mehrere Module. Wird ein Modul neu gestartet, kann es aktiv die Systeminformationen an einer zentralen Stelle abfragen und muss sich nicht mit allen anderen beteiligten Komponenten verbinden. Es benötigt folglich nicht die Informationen, welche anderen Komponenten im System aktiv sind. Dies entspricht der ersten aufgestellten Forderung aus Kapitel 4.3.

Datenrepräsentation

Neben der Art des Datenaustausches spielt die Repräsentation der Daten eine gleichwertige Rolle, wenn es um die Wart- und Erweiterbarkeit eines integrierten Systems geht. Besonders bei der Entwicklung von Prototypen, wie den hier verwendeten Demonstratoren mit ihrer Software, ist die Verwendung einer für Menschen einfach lesbaren Darstellung vorteilhaft, da die ausgetauschten Daten für eine potentielle Fehlersuche leicht überprüft werden können. Sicherlich stellt eine binäre Repräsentation eine effizienten Datenaustausch dar, jedoch sind die Übertragungsraten und -mengen in der hier vorliegenden Domäne der Mensch-Roboter-Interaktion auch in Textform vollkommen ausreichend, so dass man auf deren beschriebene Vorzüge nicht verzichten muss. Die Verwendung von XML als eine mögliche Variante dieser Repräsentation

begründet sich auf der sehr erfolgreichen Verwendung in XCF und im AM. Wie auch andere Auszeichnungssprachen bietet XML den Vorteil der Erweiterbarkeit. So können in bestehende XML-Strukturen weitere Elemente eingefügt werden, die von neuen Systemkomponenten verwendet werden, während ältere Komponenten nur die bisherige Struktur auslesen. XML im besonderen genießt inzwischen einen hohen Bekanntheitsgrad und findet breite Unterstützung in gängiger kommerzieller sowie kostenfreier Software. Dies erhöht die Wahrscheinlichkeit, dass XML bereits bei Projektpartnern verwendet wird oder vergrößert zumindest deren Akzeptanz, diese Repräsentation bei der Software-Integration zu verwenden.

Nachdem XML als Repräsentationsform gewählt wurde, blieb die Frage nach dem Aufbau der unterschiedlichen XML-Dokumente für die jeweiligen Datentypen. Generell ergeben sich zwei Ansätze: entweder wird ein möglichst generisches Dokument für alle Datentypen mit möglichst vielen Gemeinsamkeiten verwendet, oder für jeden Typ wird ein möglichst spezifisches Dokument erzeugt. Im Verlauf dieser Arbeit wurden beide Ansätze verfolgt. Das Ausgangssystem verwendete ein generisches XML-Dokument, wie es in Abbildung 5.4 dargestellt ist. Alle Dokumente weisen den gleichen Aufbau auf und unterscheiden sich nur durch das <DATA>-Tag, welches die jeweils transportierte Information beinhaltet. Durch die Verwendung generischer Dokumente reduziert sich der Aufwand der damals in den Modulen integrierten Parser. Außerdem ist die Struktur für bereits mit dem Dokument vertraute Entwickler leicht für weitere Komponenten zu übernehmen. Es zeigte sich jedoch, dass mit steigender Komplexität des Systems nicht mehr alle Informationen sinnvoll unter <DATA> abzulegen waren. Mit Erweiterung des restlichen Dokumentes nahm auch die Dokumentgröße für alle anderen Schnittstellen zu. Besonders bei der Integration mit Projektpartnern erwiesen sich die Inhalte diese historisch gewachsenen Dokumente als schwer zu vermitteln.

```
 1 <MSG xmlns:xs="http://www.w3.org/2001/XMLSchema−instance" xs:type="status">
 2  <GENERATOR>ESV</GENERATOR>
 3  <TIMESTAMP>1120831962990</TIMESTAMP>
 4  <ID>
 5   <ORIGIN mod="DLG">122</ORIGIN>
 6  </ID>
 7  <NAME>SetLocName</NAME>
 8  <STATE>PersonAlertness</STATE>
 9  <DATA>
10   <LOCDATA>
11    <LOCATION label="kitchen" corrected="no">
12   </LOCDATA>
13  </DATA>
14 </MSG>
```

Abbildung 5.4.: Generisches XML-Dokument des Ausgangssystems. Die spezifischen Informationen wurden unter <DATA> eingehangen. Der Rest des Dokumentes blieb weitestgehend unverändert.

Mit der Verwendung von XMLTIO (siehe Abschnitt 4.2.2) reduziert sich des Weiteren der Parsing-Aufwand erheblich, so dass die Verwendung von generischen Dokumenten zugunsten schnittstellenspezifischer Dokumente aufgegeben wurde. Diese sind deutlich übersichtlicher und geben die Art der transportierten Information bereits über den Namen des Wurzel-Tags preis. Ohne von einem Extrem in das andere zu verfallen, beinhalten die neuen Dokumente (siehe Abb. 5.5) einen generischen Header mit Werten für den Zeitstempel und Absender, die sich über den Entwicklungsverlauf für alle Komponenten als sinnvoll ergeben haben. Durch die leichte Erweiterbarkeit des XML-Dokumentes ist bei der inhaltlichen Schnittstellendefinition darauf zu achten, das Dokument möglichst schlank zu halten. All zu oft wird aus dem Wunsch heraus, zukünftige Entwicklungen hervorzusehen, ein enormer Overhead an Datenfeldern verwendet, welche letztendlich gar nicht oder nur unvollständig genutzt werden. Die Erweiterbarkeit eines Dokumentes unter Erhalt der Kompatibilität zu älteren Software-Modulen wirkt dieser Verfahrensweise entgegen. Stattdessen steht die Übersichtlichkeit des Dokumentes im Vordergrund. Dies unterstützt nicht nur die Entwickler, sondern steigert auch letzten Endes die Effizienz des Datenaustausches. Optionale Daten werden dabei tiefer im Dokument abgelegt als obligatorische Informationen, welche sich am Wurzelknoten des Dokumentes befinden, da das Dokument beim Parsen rekursiv durchlaufen wird.

```
1 <LOCATION label="kitchen" corrected="no">
2  <GENERATOR>DLG</GENERATOR>
3  <TIMESTAMP>
4    <INSERTED value="1120831962990" />
5    <UPDATED value="1120831962990" />
6  </TIMESTAMP>
7  <STATUS value="initiated"/>
8 </LOCATION>
```

Abbildung 5.5.: Datentypspezifisches XML-Dokument des aktuellen Systems. Für den Betrachter ist der Inhalt schneller zu erkennen und der Overhead des Dokumentes nimmt im Vergleich zu der vorherigen Variante (vgl. Größe des Rahmens) deutlich ab. Ein Header beinhaltet Elemente, die bei allen ausgetauschten XML-Dokumenten vorhanden sind.

5.2.2. Bridging the Gap: Verknüpfung von Schnittstellen

Die bisherigen Überlegungen befassten sich mit dem Design neuer oder der Modifikation existierender Schnittstellen, basierend auf den gesammelten Erfahrungen. Komplexe integrierte Systeme, wie die in dieser Arbeite vorgestellten Demonstratoren, verfügen nicht nur über Komponenten, die sich aktuell in der Entwicklung befinden, sondern auch über Module, deren Entwicklung bereits abgeschlossen ist und deren Entwickler/In bereits neuen Aufgaben nachgeht. Natürlich besteht die Möglichkeit, sich in die Funktion einer fremden Komponente einzuarbeiten. Dies ist jedoch zeitaufwendig und bringt in der Regel wenig verwertbare wissenschaftliche Erkenntnisse

mit sich. Die Beteiligung von externen Kooperationspartnern ist außerdem ein maßgeblicher Bestandteil moderner Systementwicklung. Der Austausch von Wissen und den darauf basierenden Entwicklungen ermöglicht erst die Gestaltung von Robotern, deren Fähigkeiten für aufgabenorientierte Interaktion mit Menschen ausreichen. Die Integration von Modulen der Kooperationspartner wird deutlich vereinfacht, wenn der Integrationsaufwand so gering wie möglich gehalten wird. Ziel ist es, fertige Komponenten auszutauschen, ohne diese für den neuen Demonstrator modifizieren zu müssen oder zumindest den Integrationsaufwand für den Projektpartner so gering wie möglich zu halten, um das Interesse an weiterführender Zusammenarbeit zu verstärken.

Die Möglichkeit, die vorhandene Kommunikationsstruktur beizubehalten, erweist sich als besonders vorteilhaft, da mit dieser auch der Programmablauf der jeweiligen Komponente korreliert. In den hier beschriebenen Systemen wurden frühe Komponenten über Streams miteinander verbunden. Eine Konsequenz dieser Kommunikationsmethode war, dass für die kontinuierliche Abfrage, ob neue Daten über den Stream verfügbar sind, Programmschleifen als Kontrollfluss der Module verwendet wurden. Bei dem inzwischen verwendeten Event-basierten Datenaustausch wird die Verarbeitung in einer Komponente mit dem Eintreffen der Daten als selbstständiger

```
 1 <CONFIG>
 2  <ACMI>
 3   <Memory>
 4    <XCFPREFIX name="xcf:"/>
 5    <DATABASE name="Scene">
 6     <INPUTSTREAM name="LOCATIONinfo"/>
 7    </DATABASE>
 8   </MEMORY>
 9   <TRIGGERS>
10    <TRIGGER>
11     <CONDITION database="Scene" xpath="/LOCATION" action="INSERT"/>
12     <ACTION name="CONFIGURE">
13      <SOURCE database="Scene" xpath="/LOCATION"/>
14      <TARGET module="LOC" database="Scene"/>
15     </ACTION>
16    </TRIGGER>
17   </TRIGGERS>
18  </ACMI>
19 </CONFIG>
```

Abbildung 5.6.: Konfiguration für eine Adaption unterschiedlicher Kommunikationsmethoden: Die Informationen des Datenstroms `LOCATIONinfo` werden in die Datenbank `Scene` eingefügt (Zeilen 5-7), wodurch ein entsprechendes `INSERT`-Event ausgelöst wird. Anlässlich dieses Events (Zeile 11) werden Informationen per RMI weitergegeben (Zeilen 12-14). In diesem Beispiel: die zuvor erhaltenen `LOCATION`-Daten der Datenbank `Scene` an ein Modul mit Methodenserver `LOC`. Eine potentielle Rückantwort wird in der Datenbank `Scene` abgelegt.

Prozess realisiert. Der Aufwand, alle Komponenten in ihrem Kontrollfluss zu überarbeiten, bei einem System, welches im Falle des Demonstrators BIRON seit über acht Jahren kontinuierlich weiterentwickelt wird, wäre enorm gewesen. Deswegen auf die Vorteile Event-basierter Kommunikation und vor allem auf eine Weiterentwicklung der Systemarchitektur, mit dem damit einhergehenden Erkenntnisgewinn zu verzichten, stellt jedoch im Forschungsbereich keine Alternative dar.

Zur Lösung dieser Herausforderung wird als ein Ergebnis dieser Arbeit ein Adaptermodell vorgestellt. Dieses Modell berücksichtigt nicht nur Unterschiede in den Definitionen der ausgetauschten Daten, sondern koordiniert auch die unterschiedlichen Kommunikationsmethoden, namentlich Event-Notification, AM-Zugriff, Streams oder RMI (siehe Abb. 5.6). Dies erleichtert nicht nur die schrittweise Umstellung von in Entwicklung befindlichen Komponenten, sondern ermöglicht auch die Verwendung von Modulen, deren Datenaustausch aus oben genannten Gründen nur bedingt modifiziert werden kann. Voraussetzung für die Anwendbarkeit des Adapters ist die Einbindung der XCF- oder AM-Kommunikationsbibliotheken in den jeweiligen Modulen und die Repräsentation der auszutauschenden Daten in XML. Die Einbindung der Bibliotheken und deren Anwendung in den Modulen ist durch wenige Zeilen Quelltext realisierbar, um die bisherige Schnittstelle zu ersetzen. Diese Modifikation lässt sich verhältnismäßig schnell durchführen, da an dem Kontrollfluss des Moduls nichts verändert wird. Die

```
 1 <CONFIG>
 2  <ACMI>
 3   ...
 4    <TRIGGER>
 5     <CONDITION database="Scene" xpath="/MSG/DATA/LOCDATA"
 6                        action="INSERT"/>
 7     <ACTION name="COPY">
 8      <SOURCE database="Scene" xpath="/MSG/DATA/LOCDATA"
 9                    node="/MSG/DATA/LOCDATA/LOCATION"/>
10      <TARGET database="Scene" xpath="/" node="/"/>
11     </ACTION>
12    </TRIGGER>
13   ...
14  </ACMI>
15 </CONFIG>
```

Abbildung 5.7.: Konfiguration für die Adaption von Datenformaten: Eine ältere XML-Repräsentation (siehe Abb. 5.4) soll in die aktuelle Form (siehe Abb. 5.5) überführt werden. Wird ein entsprechendes Dokument eingefügt (Zeilen 5/6) wird der unter `node` spezifizierte Teilbereich in eine andere Datenbank und ggf. in ein darin befindliches Dokument kopiert (Zeilen 8-10). Anschließend Können weitere Transformationen an dem neu erstellten Dokument vollzogen werden, bis das gewünschte Resultat erzielt wurde.

ursprüngliche Kommunikationsmodalität wird beibehalten. Durch die Verbreitung von XML ist es häufig nicht notwendig, die Datenrepräsentation zu verändern. Die bereits vorhandene XML-Struktur wird unverändert übernommen. Strukturunterschiede im XML-Dokument werden durch die Adapterfunktionen angepasst (siehe Abb. 5.7)). Sollte in der zu integrierenden Komponente ursprünglich keine XML-Repräsentation verwendet worden sein, so kann fallabhängig zwischen den folgenden Vorgehensweisen gewählt werden:

1. Es liegt ursprünglich eine textbasierten Repräsentation entweder in Form einer Auszeichnungssprache, oder einer durch Trennzeichen separierte Sequenz von Werten vor. Der Quelltextbereich der Komponente, in dem die Daten in die Ursprungsrepräsentation umgewandelt wurden, wird entsprechend modifiziert, so dass der nun entstehende Text der XML-Notation entspricht. Für den Export nach und Import von XML existieren eine große Anzahl kommerzieller oder frei verfügbarer Bibliotheken. Unter Verwendung von, zum Beispiel XMLTIO (siehe 4.2.2), bereitet der System-Architekt eine Transformationsfunktion vor, welche sowohl einzelne Werte, als auch ganze Strukturen direkt in XML-Notation umwandelt und auf Empfängerseite entsprechend ausliest. Der Modulentwickler übergibt der Funktion nur noch seine Daten und erhält automatisch das XML-Dokument. Analog werden Daten aus einem XML-Dokument ausgelesen. Die Struktur des zu übertragenden XML-Dokumentes ist in Form eines XML-Templates Bestandteil der Transformationsfunktion. Das Template weist die XML-Struktur des verwendeten Dokumentes auf, ohne mit Werten gefüllt zu sein. Die Template-Struktur wird zuvor gemeinsam mit dem Modulentwickler vereinbart.

2. Für die Übertragung binärer Datenrepräsentationen, wie zum Beispiel Bilder, eignet sich das oben genannte Verfahren nicht. Stattdessen wird von der Möglichkeit Gebrauch gemacht, Daten unverändert als *Attachment* (engl. Anhang) an ein XML-Dokument anzufügen. Das XML-Dokument wird anschließend übertragen und das Attachment dem Empfänger zur Verfügung gestellt. Die Möglichkeit, Attachments zu verwenden, ist unabhängig von der Kommunikationsmethode und benötigt nur ein minimales XML, welches zum Beispiel nur aus einem Wurzelknoten besteht. Natürlich ist auch die Kombination von Attachments mit komplexeren XML-Dokumenten möglich.

Durch die Anwendung einer der beiden Vorgehensweisen wird schnell und einfach die Grundvoraussetzung für die Integration in die hier beschriebene EASI-Architektur erfüllt, ohne dass die zu integrierende Komponente in ihrer Funktion an das System angepasst werden muss. Auch wenn diese Zielsetzung bereits in der vorangegangenen Systemarchitektur Berücksichtigung fand, so war die Arbeitsweise des Execution Supervisors als zentrales Kontrollmodul bindend für die beteiligten Komponenten. Dies galt von der Verwendung starrer Datenrepräsentationen als uniforme Dokumente mit entsprechendem Overhead, über die Vorgabe der Kommunikationsmodalität in Form von Datenströmen, bis hin zum als *Polling* vorausgesetzten Prozess Flow. Das hier gewählte Verfahren ersetzt nicht etwa ein Kontrollmodul durch ein anderes. Vielmehr ist die Komponente ACMI als Koordinator im Hintergrund angelegt. Weder müssen alle Informationen zuerst durch ACMI geprüft, noch einzelne Komponenten aktiviert werden. Können Komponenten ohne systemrelevante Informationen Daten austauschen, bleibt ACMI vollkommen

unbeteiligt. Dadurch bleiben die integrierten Komponenten nicht nur möglichst unabhängig voneinander, sondern auch von der Systemkoordination. Nur im Bedarfsfall wird die Koordination aktiv und modifiziert im Memory Daten, ohne jedoch direkt Anweisungen an Module zu erteilen, die nicht zu den Grundfunktionen des Systems gehören. Dieser Bedarfsfall betrifft die Arbitrierung begrenzt zur Verfügung stehender Ressourcen, wie zum Beispiel die Hardwareansteuerung, und wird im nachfolgenden Kapitel beschrieben.

5.3. Zusammenfassung

Das Grundverständnis des hier geprägten Begriffes einer evolutionären Architektur zur Systemintegration (EASI) liegt in der Anpassungsfähigkeit des Systems gegenüber neuen Anforderungen begründet. Dennoch wird ein Leitfaden für die Integration zur Verfügung gestellt, der ein unkontrolliertes Ausufern von Schnittstellen verhindert. Gewährleistet wird die Adaptivität durch eine Verwaltungskomponente, die modulübergreifende Informationen organisiert, ohne für die betroffenen Komponenten sichtbar zu sein. Realisiert ist diese Verwaltungskomponente durch das Active Control Memory Interface - ACMI, dessen Funktion der Zielsetzung *Koordination statt Kontrolle* folgt. Hierdurch wird eine zu feste Kopplung mit den koordinierten Modulen vermieden und dennoch ein modulübergreifender Datenaustausch realisiert. Die Konfiguration dieses Systemverhaltens kann zur Laufzeit vorgenommen werde und ist mittels Referenzierung auf modifizierbare Memory-Inhalte, dem Memory-Pointing, hoch flexibel. Dabei basiert der Konfigurationsprozess auf der gleichen Repräsentation und den entsprechenden Zugriffsmöglichkeiten wie Moduldaten im Memory des Systems. Folglich müssen für den Integrationsprozess und die Verwendung der Verwaltungskomponente keine weiteren Repräsentationen durch die Entwickler erlernt werden.

Eine erste und einfache, aber grundlegende Anwendung dieses Verfahrens findet bei der Integration von Komponenten statt, deren Schnittstellen ursprünglich nicht dem in dieser Arbeit entwickelten Design entsprechen. So können sowohl Komponenten, deren Entwicklung erfolgreich abgeschlossen wurde, als auch Komponenten kooperierender Einrichtungen, die anderen Architekturkonzepten folgen, mit geringerem Personalaufwand integriert werden. Das langfristige Ziel eine evolutionären Systemarchitektur muss es sein, nicht nur den Integrationsprozess durch eine hier vorgestellte Adaptivität zu unterstützen, sondern modulübergreifendes Lernen wie in [Gep08] beschrieben zu ermöglichen. Ein erster Schritt in diese Richtung sind flexiblere Arbitrierungs- und Sequenzierungs-Mechanismen, wie sie von von der hier vorgestellten Koordination realisiert und im folgenden ausführlich beschrieben werden.

6. Generierung und Adaption des Systemverhaltens

Mit der steigenden Anzahl neuer Fähigkeiten eines Roboters steigt auch der Aktionsumfang des Systemverhaltens und somit die Komplexität, alle einzelnen Fähigkeiten zu dem gewünschten Systemverhalten zusammenzufügen. Eine Top-Down-Modellierung, die alle auftretenden Eventualitäten behandelt, ist in dem hier betrachteten HRI-Szenario kaum, wenn überhaupt, zu realisieren. Zu umfangreich sind die Variationen im Umgang mit dem Roboter, die sich aus verschiedenen Einsatzumgebungen und Benutzern ergeben. Der hier beschriebene Weg mit dieser Herausforderung umzugehen, fokussiert stattdessen ein möglichst flexibles System, welches schnell auf die aktuelle Situation adaptiert werden kann. Das Systemverhalten „ergibt“ sich aus der Koordination der beteiligten Komponenten durch das in Abschnitt 5.1 beschriebene ACMI. Nicht alle Aufgaben innerhalb des Home-Tour-Szenarios (siehe Abschnitt 3.1.1) werden von Seiten des Demonstrators über dieses Modul koordiniert, sondern nur diejenigen, die sich systemweit auswirken. Die Arbitrierung begrenzter Ressourcen und der auf diesen Ressourcen operierenden Aktionen ist eine derartige Aufgabe, genau wie die Sequenzierung oder Synchronisation der, aus Gründen der Modularität, unabhängig voneinander operierenden HRI-Komponenten. Sie stellen zusammen mit der zuvor genannten Schnittstellenverwaltung die drei Kernaspekte der hier präsentierten Systemarchitektur dar, für die das hier vorgestellte Koordinationsmodell eine geschlossene Lösungsstrategie bietet.

6.1. Arbitrierung konkurrierender Aktionen

Die Arbitrierung konkurrierender Aktionen lässt sich leicht in Zusammenhang mit begrenzten Hardware-Ressourcen veranschaulichen. Um für den Anwender sichtbare Aktionen auszuführen, erhalten die Aktuatoren des Roboters unterschiedliche Befehle. Ein Aktuator kann immer nur einen Befehl zu einer Zeit ausführen. Konkreter bedeutet dies, der Roboter kann nicht gleichzeitig nach rechts und links fahren. Um dieser Anforderung gerecht zu werden, existieren zwei Alternativen: Der Aktuator erhält mehrere Befehle unterschiedlicher Quellen und es wird nach einer zu bestimmenden Heuristik eine Quelle ausgewählt, deren Anweisung zur Ausführung kommt. Alternativ ist allen Quellen bekannt, wann sie über die jeweiligen Aktuatoren verfügen können und nur die berechtige Quelle sendet Befehle an den Aktuator.

In dem Home-Tour-Szenario ist eine solche Quelle zum Beispiel das Personen-Tracking (siehe Abschnitt 3.3.1). Dieses sendet Befehle an die Roboterbasis, um möglichst viele potentielle Interaktionspartner in dem Wahrnehmungsbereich der Sensoren zu halten. Gleichzeitig ist es Ziel der Lokalisation, Informationen über die Umgebung zu sammeln, wozu es notwendig ist, den Roboter um seine Achse drehen zu lassen. Beide Komponenten erzeugen Bewegungsbefehle, die sich widersprechen können. Da es sich bei dem hier gezeigten Beispiel um Aktionen handelt, deren Ausführung sich widersprechende Bewegungsbefehle erzeugen, setzt die Arbitrierung nicht primär bei den resultierenden Befehlen an, sondern idealerweise bei den konkurrierenden Aktionen (siehe Abb. 6.1, rechts). Generalisiert lässt sich die folgende Arbitrierungstrategie formulieren: Eine Aktion sollte nur dann durchgeführt werden, wenn deren Resultate verarbeitbar sind. Dieser im weiteren Verlauf als *präaktive Arbitrierung* bezeichnete Ansatz spart Systemressourcen. Es lässt sich mit einem Event-basierten Kommunikations-Framework, wie es die hier vorgestellten Arbeit verwendet, einfach realisieren, indem das aktionsauslösende Ereignis gezielt manipuliert wird.

Allerdings ist die präaktive Arbitrierung nicht immer anwendbar, vor allem wenn es sich um „historisch gewachsene“ Systeme oder zu integrierender Komponenten externer Abteilungen handelt. Rufen diese Komponenten Daten durch eine sich ständig wiederholende Programmschleife ab, muss deren Prozessablauf angepasst werden. Das ist in der Regel mit entsprechendem Aufwand verbunden. Außerdem ist es bei Komponenten, deren Resultate regelmäßig verwendet

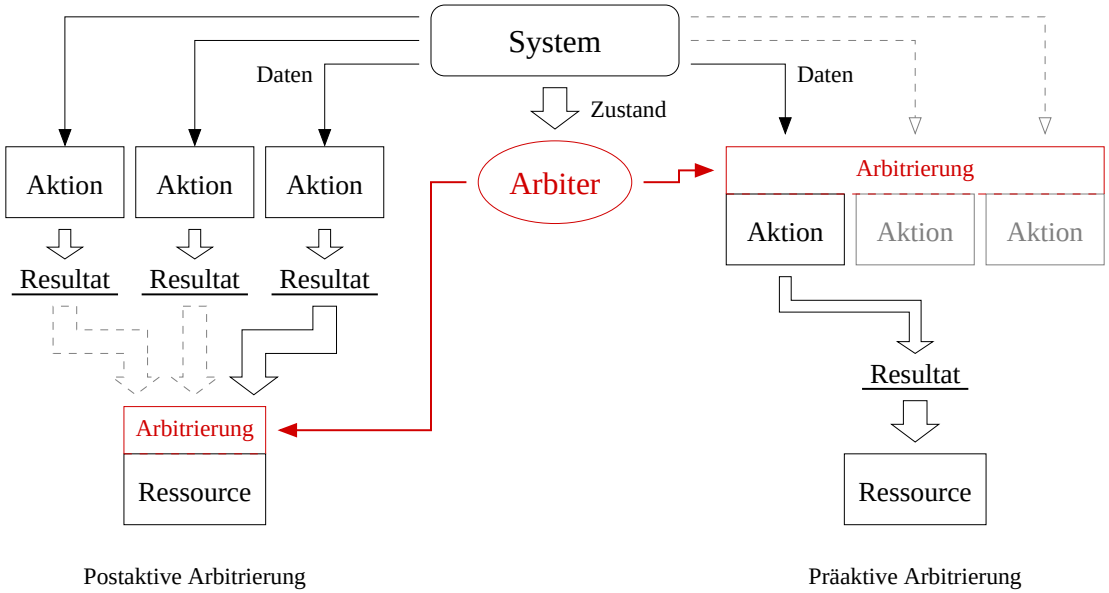


Abbildung 6.1.: Abhängig von der Analyse des Systemzustandes wird bei der *postaktiven Arbitrierung* entschieden, welches der sich potentiell widersprechenden Aktionsresultate von einer begrenzten Ressource, wie zum Beispiel Hardware, verarbeitet wird. Bei der *präaktiven Arbitrierung* wird nur die Aktion durchgeführt, deren Resultate anschließend Verwendung finden.

werden und die nur in wenigen Situationen in Konflikt mit anderen Komponenten treten, nicht sinnvoll für jede einzelne Berechnung einen Aufruf durchzuführen. Im Falle dieser beiden Überlegungen bietet sich die *postaktive Arbitrierung* an, welche auf Seite der zu arbitrierenden Ressource entscheidet, welche Daten zur Verarbeitung verwendet werden (siehe Abb. 6.1, links).

Umgesetzt werden diese beiden Arbitrierungsstrategien erneut durch das Konzept einer zentralen Koordination. Aus Gründen der Modularität enthalten an einer Arbitrierung beteiligte Komponenten nicht selber das Wissen darüber, in welchem globalen Systemzustand die von ihnen erstellten Ergebnisse umgesetzt werden können und in welchem dies nicht möglich ist. In der Ausgangsarchitektur, die dieser Arbeit zugrunde liegt, wurde die Arbitrierung durch das Dialogsystem [Li06] verwaltet. Da der Roboter selbst keine Aktionen initiierte, die einer Arbitrierung bedurften, wurde bei der Verarbeitung eines Benutzerkommandos seitens des Dialogs geprüft, ob die gewünschte Aktion zum jeweiligen Zeitpunkt ausführbar war. Entsprechend wurde das Ergebnis dieser Überprüfung dem Benutzer mitgeteilt. Für die Überprüfung war das Dialogsystem eng mit dem Execution Supervisor (siehe Abschnitt 4.3 und Abb. 4.3) verknüpft. Bei einer Veränderung einer, an der Arbitrierung beteiligten Systemkomponenten, musste nicht nur der endliche Automat des Execution Supervisors, sondern auch die im Quelltext des Dialoges beinhaltete Heuristik überarbeitet werden. Ferner bedingt dieses Verfahren, dass die Initiative für eine Aktion von Seiten des Anwenders oder des Dialogsystem ergriffen wird und nicht von einer anderen Komponente. Eine solche Komponente müsste die initiierte Aktion zuerst bei dem Dialogsystem anmelden, was eine noch engere Kopplung des Gesamtsystems zur Folge hätte.

Prä- und postaktive Arbitrierung mittels ACMI

Als Konsequenz dieser Überlegungen wurde der Dialog auf seine eigentliche Aufgabe zurückgeführt, als reine Kommunikationsschnittstelle zwischen Anwender und Roboter zu fungieren. Er nimmt eine gleichwertige Position innerhalb der HRI-Komponenten ein und teilt sich nicht mehr die gesamten architekturbezogene Informationen mit der Kontrollkomponente, beziehungsweise mit der aktuellen Koordination. Die Arbitrierung wird als memorybasierte Koordination durch ACMI realisiert. In den Systemen BIRON und Barthoc wird über ACMI sowohl die prä- und postaktive Arbitrierung realisiert. Hierbei ist deutlich hervorzuheben, dass die präaktive Arbitrierung unter den zuvor genannten Gründen in der Regel vorzuziehen ist. Jedoch handelt es sich bei dem hier vorgestellten System um ein Modell zur flexiblen Integration von ursprünglich nicht eingeplanten Komponenten. Ist deren Anpassung an die präaktive Arbitrierung zu aufwendig und soll auf den, mit der Integration einhergehenden Gewinn neuer Fähigkeiten nicht verzichtet werden, bedarf es auch der Unterstützung postaktiver Arbitrierung.

Für die postaktive Arbitrierung werden die Resultate mehrere Aktionen, beziehungsweise der für diese Aktionen verantwortlichen Komponenten, in das Memory gestellt. Ein Beispiel hierfür ist das Personen-Tracking. Es stellt kontinuierlich Zielpositionen für die Ausrichtung der Roboter Basis und der Pan-/Tilt-Kamera bei BIRON, sowie für den Torso und den Kopf bei Barthoc, in das Memory. Analog werden Zielpositionen ausgehend von anderen Modulen, wie der Lokalisation und dem Folgeverhalten auf der Plattform BIRON, beziehungsweise dem

Environment Information System (EIS) des Demonstrators Barthoc in das Memory eingefügt. Die Zielpositionen werden von der demonstratorspezifischen Hardware-Kontrolle (HWC) in Aktuatorbefehle umgesetzt. Für die Arbitrierung enthält jede Zielposition ein `GENERATOR`-Element (siehe Abb. 5.5) mit der Bezeichnung der Absenderkomponente. Diese Absenderbezeichnung muss seitens der Entwickler für jede Komponente unikal, bezogen auf das gesamte System, gewählt werden. ACMI verwendet die in Abschnitt 5.1 auf Seite 66 beschriebene RMI-Kommunikation, um die Hardware-Kontrolle so zu rekonfigurieren, dass sie jeweils nur die Zielpositionen mit einem bestimmten `GENERATOR`-Element verarbeitet.

Die Information, welcher Generator zu welchem Zeitpunkt Verwendung findet, ist nicht fester Bestandteil des ACMI-Programm-Codes, sondern wird flexibel innerhalb des Memorys verwaltet. Hierfür greifen zwei Konzepte ineinander: Als erstes stellt eine zu arbitrierenden Ressource, wie zum Beispiel die Hardware-Kontrolle, ihre eigene Schnittstellenkonfiguration als eine Art „Bedienungsanleitung“ über das Memory zur Verfügung. Die Schnittstellenkonfiguration besteht aus einem XML-Dokument, welches die unterschiedlichen Parameter für einen RMI-Aufruf dieser Ressource definiert. Innerhalb der Regelbasis von ACMI ist festgelegt, in welchem Systemzustand ein bestimmter RMI-Parameter Anwendung findet. ACMI verwendet diese bei neu hinzugefügten Ressourcen, ohne dass eine Veränderung des Quelltextes der Koordination notwendig ist. Diese Aufteilung in zwei Konzepte ermöglicht eine noch stärkere Kapselung der einzelnen Module und eine größere Unabhängigkeit bei der getrennten Entwicklung bis zur Integration. Bei der Implementierung über die ACMI-Regelbasis finden das als *Memory-Pointing* eingeführte Verfahren während der Auswertung einer `CONFIGURE`-Regel Anwendung. In der Definition der Regel wird auf den zur aktuellen Situation passenden Bereich der „Bedienungsanleitung“ verwiesen und die dort definierten Parameter an beispielsweise die Hardware-Kontrolle per RMI übermittelt. Die für eine RMI-Kommunikation typische Rückmeldung wird von ACMI entgegengenommen und entsprechend der Definition der `CONFIGURE`-Regel in das Memory eingefügt. Die Antwort gibt Aufschluss über den Erfolg der Rekonfiguration und steht anderen Modulen als Information zur Verfügung. Das Verfahren ist in Abbildung 6.2 dargestellt.

Während das Personen-Tracking aufgrund seiner historischen Entwicklung kontinuierlich Zielpositionen generiert und somit postaktive Arbitrierung voraussetzt, bietet sich für neue, rechenintensive Komponenten die präaktive Arbitrierung an. Ein Beispiel hierfür ist die Lokalisation des BIRON-Systems. Bei Betreten eines unbekannten Bereiches ergreift die Lokalisation die Initiative und versucht für diesen Bereich eine Repräsentation mit dem Laser-Range-Finder zu erstellen [Pel09]. Hierfür muss sich der Roboter um seine eigene Achse drehen. Im Gegensatz zum Personen-Tracking berechnet die Lokalisation nicht kontinuierlich Zielpositionen, um diese in das Memory einzufügen. Die Berechnung findet nur statt, wenn die aus den Ergebnissen resultierende Roboterrotation in der gegenwärtigen Systemsituation zulässig ist. Versucht eine Komponente die Initiative zu ergreifen, so meldet sie dies dem System, indem sie, der von ihr in das Memory eingefügten Information, ein `STATUS`-Element anhängt (siehe Abb. 5.5) und auf eine Rückmeldung der Koordinationskomponente wartet. Erst wenn eine positive Rückmeldung erfolgt, indem der Status aktualisiert wird, setzt die Komponente ihre Funktion fort. Die Verwendung unterschiedlicher Status unterstützt nicht nur die hier beschriebene Arbitrierung, sondern wird in Form fester *Interaction Patter* für die Sequenzierung des Informationsflusses verwendet.

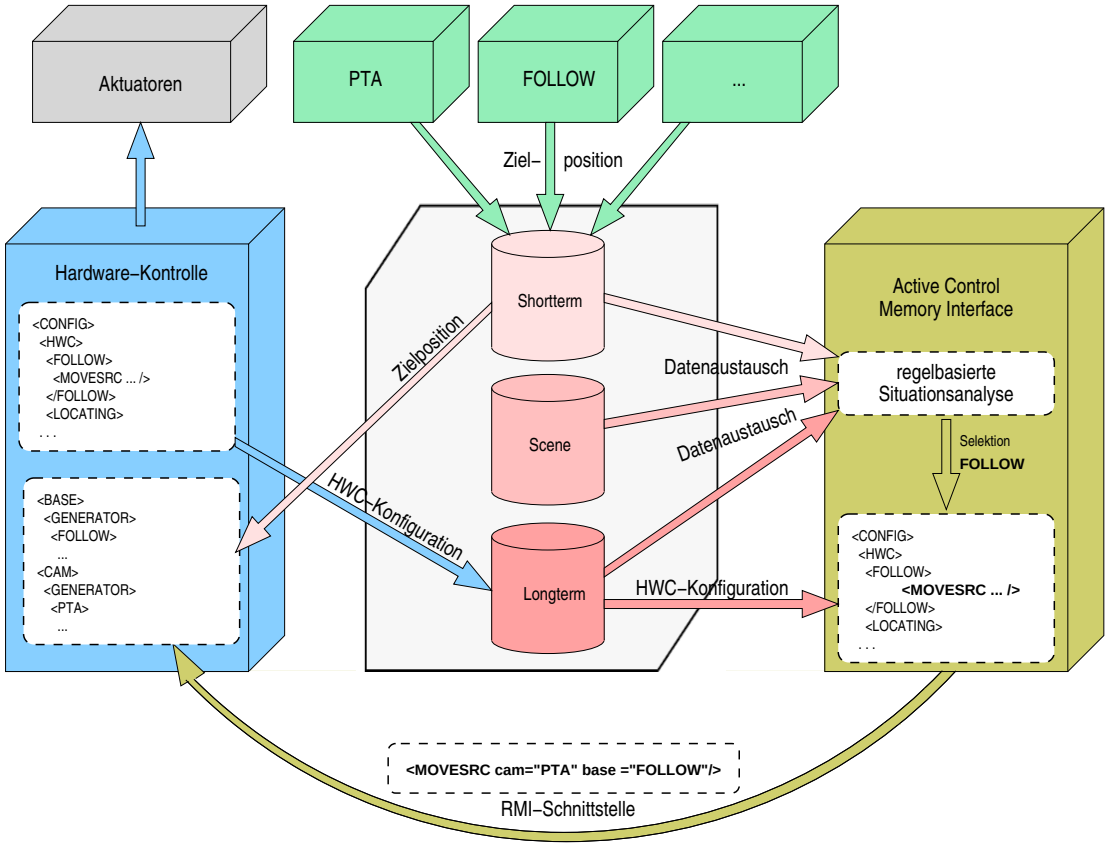


Abbildung 6.2.: Postaktive Arbitrierung mittels RMI am Beispiel der Hardware-Kontrolle. Bei Initialisierung wird die HWC-Konfiguration in das Memory geladen und an das ACMI weitergeleitet. Abhängig von dem Systemzustand wird der betreffende Teil der HWC-Konfiguration über eine ACMI-Regel selektiert und per RMI an die Hardware-Kontrolle übermittelt. Diese registriert sich auf den per RMI-Parameter-Übertragung spezifizierten Zielpositionen und liest sie vom Memory.

6.2. Sequenzmodell zur Koordination asynchroner Prozesse

Das Bestreben nach voneinander möglichst entkoppelten Modulen bedingt auch, dass die mit diesen Modulen einhergehenden Aktionen asynchron realisiert sind und, wie bereits bei der Arbitrierung verdeutlicht, im Idealfall keine gegenseitige Kenntnis übereinander besitzen. Jedes Modul arbeitet als selbstständige Komponente vergleichbar mit einem Software-Agenten. In der Domäne möglichst natürlicher Mensch-Roboter-Interaktion sind, neben einfacheren Aktionen, wie dem Ausweichen von Hindernissen, auch komplexere Aufgaben, beispielsweise das eingangs erwähnte Fetch & Carry Szenario, zu realisieren. Derartige komplexere Aktionen sollten nicht als atomare Einheiten, sondern vielmehr als Zusammenspiel koordinierter Subaktionen realisiert werden. So können Subaktionen wie Navigation oder Objekterkennung in einem anderen

Kontext wiederverwendet werden, ohne dass die ihnen zugrundeliegenden Module angepasst werden. Um diese Überlegung zu realisieren, werden in dieser Arbeit Aktionssequenzen verwendet, die die Aktivierungsreihenfolge der beteiligten Module je nach der übergeordneten Aufgabe flexibel definieren. Dabei stellte sich heraus, dass auch komplexe Aktionen auf das gleiche, hier entwickelte Sequenzmodell zurückzuführen sind.

Anwendung von Aktionssequenzen im System

Die Notwendigkeit, zeitliche Abläufe bei Event-basierten Systemen definieren zu können, wird bereits in [Qia07] motiviert. Qiao et al. verwenden ebenfalls ECA-Regeln, deren Bedingungen durch zeitliche Aspekte wie *innerhalb einer Minute* oder *nach 15 Minuten* erweitert werden. Diese Vorgehensweise lässt sich gleichermaßen über geeignete XPath-Ausdrücke in der hier vorgestellten Regelbasis realisieren, indem Zeitstempel ausgewertet werden, die jedem Datenbankelement automatisch bei dessen Aktualisierung zugeordneten werden. Das Vormerken von Ereignissen, die erst nach dem Verstreichen einer Zeitspanne auszuführen sind, wird bislang nicht benötigt, lässt sich aber durch Erweiterung der Koordination in das bestehende System integrieren, ohne anderweitige Systemkomponenten zu verändern. Die genaue Spezifikation zeitlicher Abfolgen in festen Intervallen birgt die Gefahr, ein großes Maß an Einzelsituationen vordefinieren zu müssen und weniger flexibel auf Abweichungen von der Erwartungshaltung der Entwickler reagieren zu können.

Anstatt zeitlicher Vorgaben oder eine feste Definition von Aktionsabläufen innerhalb einer Regel zu definieren, ergeben sich Sequenzen von Aktionen über die Veränderungen, die eine Regel am Zustand des Memorys durchführt. Eine derartige Veränderung liefert das Ereignis, welches zur Anwendung der nächsten Regel führt. Zwar kann dieses Vorgehen auf jede Veränderung des Memorys angewendet werden, um jedoch die Gefahr von unbeabsichtigten Seiteneffekten zu minimieren, wurde dieses Verfahren auf die Modifikation des bereits zur Arbitrierung verwendeten `STATUS`-Tags in den jeweiligen Memory-Elementen eingeschränkt. Der Status kann auf beliebige Werte gesetzt werden, die wiederum als Trigger für weitere Regeln fungieren. Im Falle der zuvor beschriebenen präaktiven Arbitrierung wird über diesen Mechanismus gewährleistet, dass zuerst die begrenzte Ressource bereit ist, Daten entgegen zu nehmen, bevor die selektierte Aktion ausgeführt wird und somit alle Daten dieser Aktion auch von der Ressource verarbeitet werden. Erst wenn diese Aktion erfolgreich durchgeführt und der Status der betreffenden Memory-Elemente erneut modifiziert wurde, signalisiert dies die Bereitschaft zu einer erneuten Arbitrierung. Durch die Allgemeinheit dieses Konzeptes ist es auf Aktionen unterschiedlichster Art anwendbar. So wird es nicht nur für die zuvor beschriebenen Lokalisation eingesetzt, sondern findet auch für das Objektlernen und das Folgeverhalten auf dem Demonstrator BIRON sowie für die koordinierte Sprachausgabe und Gestengenerierung auf Barthoc Verwendung.

Die hier beschriebene Sequenzierung und Kontrolle des Systemablaufes anhand von Regelattributen wird in [Ish95] mit einer Architektur verglichen, die auf derartige Attribute verzichtet und ausschließlich die Veränderung auf Memory-Ebene analysiert. Als Motivation dieser Arbeit liegt eine möglichst einfache Regelgestaltung mit einer geringeren Anzahl zu berücksichtigender

Attribute zu Grunde. Der geringere Regelumfang wird mit einem komplexeren Interpreter kompensiert, der deutlich mehr Wissen über die Systemabläufe beinhaltet und so schwieriger auf alternative Szenarien oder Demonstratoren zu portieren ist. Das ACMI-Modell verwendet einen szenarien- und demonstratorunabhängigen Interpreter, der keine Systeminformationen beinhaltet und ermöglicht dennoch die Koordination einzelner Aktionen zu einer Gesamtaufgabe unter Verwendung nur eines weiteren Regelattributes für die Systemsynchronisation, basierend auf Arbitrierung und Sequenzierung.

Synchronisierung von Aktionen durch ACMI

Obwohl jede Zeichenkombination als Status verwendet werden kann und so beliebig lange kausale Ketten von Regeln geschaffen werden können, hat sich in der Praxis gezeigt, dass nur eine begrenzte Anzahl von Elementen in der kausalen Kette und damit eine entsprechend begrenzte Anzahl von Status notwendig sind, um das bisherige System zu realisieren. Es haben sich fünf verschiedene Status etabliert. Diese werden zur Bildung von kausalen Ketten mit bis zu drei Gliedern eingesetzt. Sie realisieren immer wieder das gleiche Interaktionsmuster zwischen den beteiligten Modulen: *Anforderung von Ressourcen*, *Bestätigung* und *Verarbeitung der Daten*. Abbildung 6.3 zeigt die verwendeten Status-Bezeichnungen innerhalb einer kausalen Kette, wie sie auch bei der Roboterlokalisierung zum Erlernen neuer Bereiche bei BIRON Anwendung findet. Dabei können die fünf semantisch fixierten Status-Typen für Zwischenschritte beliebig erweitert werden. Die Anwendung der kausalen Ketten ist nicht nur auf eine lineare Auswertung begrenzt. Wie bei einem Baum können von der Modifikation eines Status-Wertes auch mehrere Abzweigungen erfolgen, um synchronisiert mehrere Aktionen durchzuführen, die nicht miteinander in Konflikt stehen. So wird auf dem Demonstrator Barthoc parallel zur Objektreferenz mit dem Roboterarm die Sprachausgabe mit Informationen zum Objekt ausgelöst. Um eine solche aufgefächerte Kette erfolgreich abzuschließen, können - müssen jedoch nicht - mehrere Regelergebnisse schrittweise über Zwischen-Status oder *Guards* zusammengefasst werden.

Wann eine Ressource, wie in dem vorangegangenen Beispiel die Hardware-Kontrolle, verfügbar ist und wann nicht, wird situationsabhängig entschieden. Der aktuelle Systemzustand wird in einem eigenen XML-Dokument als *Situation* zusammengefasst und durch Regelanwendung aktualisiert. Gleichzeitig wird die Situation in den *Guards* der Regeln ausgewertet, so dass nur bei dem Vorliegen einer bestimmten Situation oder eben außerhalb einer Situation die jeweilige Regel aktiviert werden kann, selbst wenn ein zutreffendes Event vorliegt. Bei dem zuvor genannten Beispiel findet die Rekonfiguration nur statt, wenn die aktuelle Situation es zulässt. Das Setzen der Situationen lässt sich mit einem Mutex vergleichen, welcher Zugriffskonflikte vermeidet. Um ein Steckenbleiben des Systems (*Deadlock*) wegen einer Modul-Fehlfunktion innerhalb einer kausalen Kette zu vermeiden, kann ein separater Memory-Prozess die kausale Kette als fehlgeschlagen abschließen, sobald eine zu spezifizierende Zeit seit dem letzten Status-Wechsel vergangen ist und somit die Kette abschließen. Welche Situation wann eintritt und welche Auswirkungen sie auf die unterschiedlichen Aktionen hat, ist ebenfalls in der Regelbasis des ACMI definiert und kann damit schnell neuen Anforderungen angepasst werden.

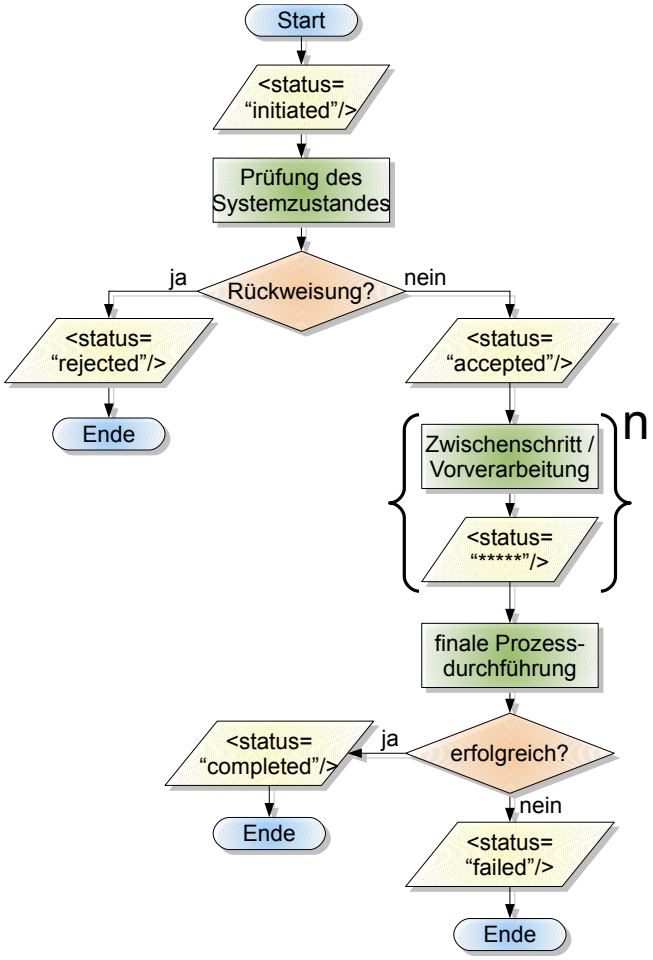


Abbildung 6.3.: Für die Sequenzierung werden den XML-Dokumenten unterschiedliche Status zugewiesen, deren Modifikation wiederum den nächsten Verarbeitungsschritt einleiten. Für längere Sequenzen können neben den, sich als Standard etablierten, Status-Bezeichnungen weitere beliebige Bezeichnungen für Zwischenschritte verwendet werden.

6.3. Dynamische Verhaltensanpassung

In den vorangehenden Abschnitten wurde beschrieben, wie sich ein Gesamtsystem flexibel aus einzelnen Komponenten aufbauen lässt und wie diese Module zu einem gewünschten Systemverhalten koordiniert werden können. Der Gewinn des hier beschriebenen Ansatzes setzt sich in Form einer dynamischen Verhaltensanpassung des Roboters zur Laufzeit fort. Die Regelbasis im Memory kann jederzeit modifiziert und das Verhalten des Roboters sogar innerhalb einer Interaktion verändert werden. Um dieses Vorgehen für den Entwickler / Verhaltensdesigner zu erleichtern, wurde eine grafische Repräsentation *TAME* (*Trigger & Active Memory Editing*) der Regelbasis und ihrer Anwendung hinzugefügt. Teile von TAME stammen aus dem Open Source

Projekt *jEdit* [Pes07] und wurden im Rahmen der GPL 2.0 weiterentwickelt. TAME verbindet sich auf das Active Memory, liest dessen Inhalte bei jeder Veränderung aus und zeigt sie an (siehe Abb. 6.4). Da die Regelbasis genauso Bestandteil des Memorys ist wie die ausgetauschten Moduldaten, kann analog zu der Arbeit von Cabri et al. [Cab99] mit demselben Editor auf die unterschiedlichen Inhalte gleichermaßen zugegriffen werden. Neben der Anzeige der im Memory befindlichen Daten und Regeln können, diese jederzeit modifiziert werden und damit zur Laufzeit Veränderungen im Systemverhalten bewirken.

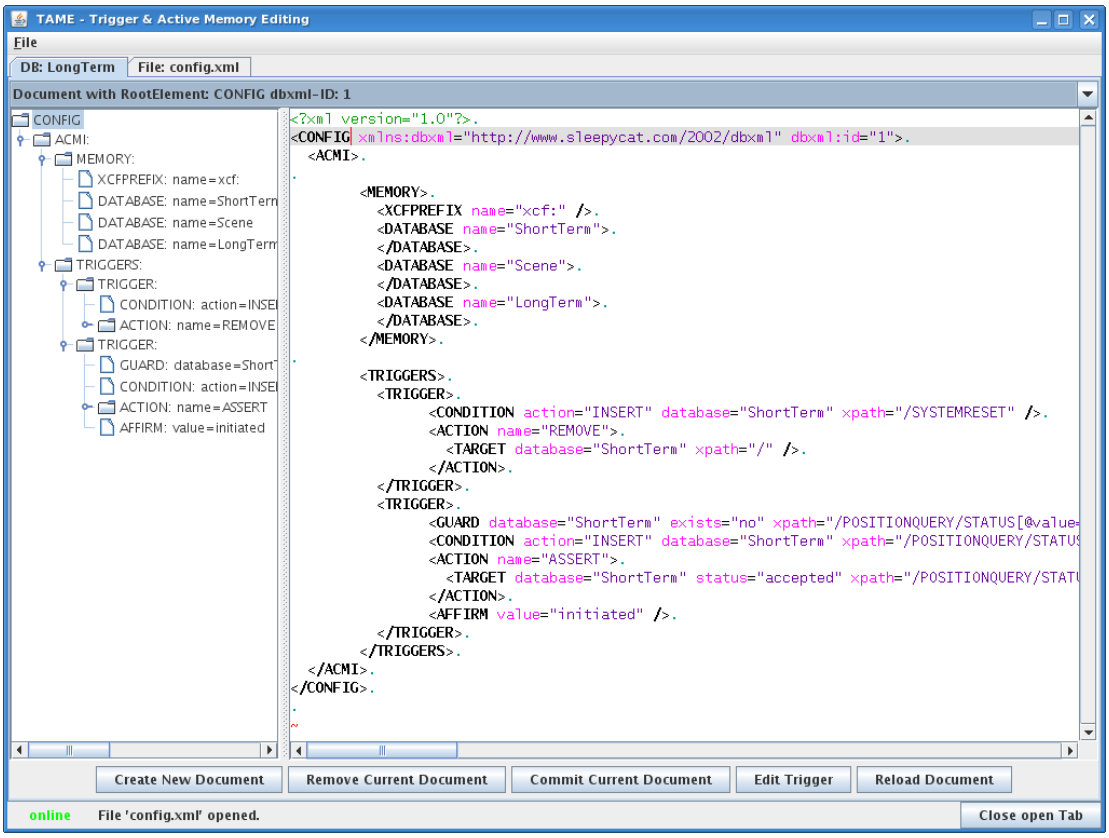


Abbildung 6.4.: Die Hauptansicht des Editors TAME bietet die Möglichkeit den aktuellen Inhalt des Memorys einzusehen und zu ändern, dies gilt sowohl für die Moduldaten als auch für die Regelbasis von ACMI.

TAME unterstützt den Entwickler explizit bei der Erstellung und Verwaltung der Regelbasis. Hierfür existiert eine eigene Ansicht (siehe Abb. 6.5) in der der Systementwickler die Bestandteile jeweils einer Regel durch Auswahl aus Menüs zusammenstellen kann. Abhängig von dieser Wahl werden kontextsensitiv weitere Schaltflächen im Hauptfenster aktiviert oder deaktiviert. So werden für den Anwender nur zulässige Auswahlmöglichkeiten verfügbar gemacht und syntaktisch korrekte Regeln erzeugt, ohne ihn mit zu vielen und aktuell nicht verwendbaren Informationen zu überfordern [End01]. Gleichzeitig bietet TAME dem Anwender eine Zusammenfassung aller aktuell anwendbaren Regeln in Verbindung mit dem auslösenden Ereignis. Jede Regel kann selektiert und in ihren Attributen bearbeitet werden.

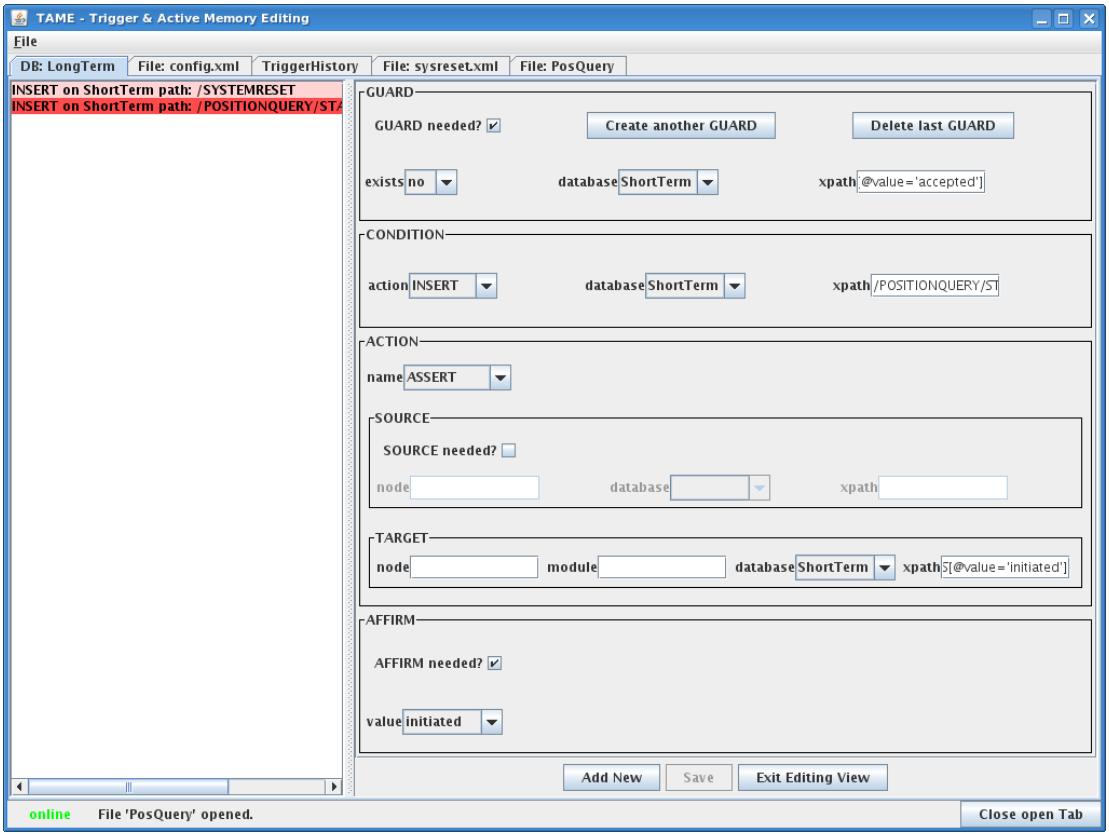


Abbildung 6.5.: Menüführung von TAME zur Erstellung von Regeln. Im rechten Hauptfenster werden die notwendigen Einstellungen vorgenommen und anschließend als Regel im Memory abgelegt. Eine Liste auf der linken Seite erlaubt den schnellen Zugriff auf alle bereits existierenden Regeln und zeigt durch farbliche Hervorhebung den Zeitpunkt ihrer Aktivierung an.

Um den Zusammenhang zwischen Ereignis und Regel für den Entwickler leichter erfassbar zu machen, unterstützt TAME die Systemüberwachung durch Visualisierung des Regelaufrufes. Die aktuell angewendete Regel wird farblich hervorgehoben. Über den Farbverlauf ist die zeitliche Abfolge für den Betrachter einfach zu erkennen. Parallel wird die Aufrufreihenfolge für eventuelles Debugging aufgezeichnet. Mit dem Debugging in Verbindung steht die Verwendung von TAME als rudimentäre Simulationsumgebung. Mit dem Zugriff auf alle Memory-Inhalte - und nicht nur auf die Regelbasis, die ja an sich nur einen weiteren Memory-Inhalt darstellt - können die Ausgaben anderer Komponenten in das Memory simuliert werden. Als Konsequenz können sowohl die Funktion von Regeln, als auch einzelnen Komponenten unabhängig vom Gesamtsystem genutzt werden. Die vielfältige Einsetzbarkeit von TAME basiert dabei auf der *technischen Gleichbehandlung von Verhalten und Daten* als Konzept der hier vorgestellten Architektur. TAME unterstützt die Systemflexibilität für Erweiterungen und eventuelle Modifikationen der Regeltypen durch Verwendung einer Spezifikationsdatei wie sie in Anhang A.5 dargestellt ist. Die Spezifikation beinhaltet sowohl Standardwerte für Regelparameter, als auch die Bezeichnungen der Wahlelemente in den unterschiedlichen Menüs.

6.4. Zusammenfassung

Die Integrationsstrategie *Koordination statt Kontrolle* setzt sich ausgehend von der abstrakteren Systemarchitektur hin zu der Realisierung von Arbitrierung, Sequenzierung und Synchronisation fort. Ein generelles Kommunikationsmodell wurde entwickelt, um begrenzte Systemressourcen situationsabhängig zuzuweisen ohne in konfliktfreie Aktionen einzugreifen. Aktionen, die sich gegenseitig ungewünscht beeinflussen, werden durch die präaktive Arbitrierung bereits vor der Ausführung zurückgewiesen, um Rechenzeit zu sparen. Dennoch wird aus Kompatibilitätsgründen auch die postaktive Arbitrierung berücksichtigt und beide Arbitrierungsvorgehen mit der aus Abschnitt 5.1 bekannten Regelbasis umgesetzt.

Unter Verwendung eines zusätzlichen `STATUS`-Attributes in den ausgetauschten Moduldaten wird mit dem gleichen Koordinationsmodell auch die Sequenzierung von Aktionen realisiert. Anstatt eine Sequenz fest in einer Regel zu definieren, werden mehrere Regeln miteinander verknüpft, indem sie jeweils auf die Systemveränderungen und Modifikation des Status der Ausführung einer vorangegangenen Regel reagieren. Das Ableiten von Situationen aus dem gesamten Informationsgehalt des Memorys und die Anwendung von situationsbezogenen *Guards* verhindern unbeabsichtigte Seiteneffekte zwischen den, in der Synchronisation parallel ausgeführten Aktionen, ohne die Übersichtlichkeit der Regelbasis zu stark einzuschränken.

Ergänzend wurde ein Editor entwickelt, der es den Systementwicklern durch entsprechende Menüführung erleichtert, neue Regeln zu entwickeln oder bereits bestehende Regeln zu verändern sowie beliebige Informationen zur Simulation in das Memory einzufügen. Das Auslösen von Regeln wird dabei in Echtzeit visualisiert und aufgezeichnet, um eine vereinfachte Fehlerkorrektur seitens der Regeldefinitionen selbst im laufenden System zu ermöglichen und die Auswirkungen sofort betrachten zu können. Die Anwendung der selben Mechanismen auf die unterschiedlichen Anwendungsgebiete im Bereich der Systemarchitektur und -integration untermauern die Flexibilität und den Nutzen des hier vorgestellten Modells und seiner Regelbasis.

7. Analyse und Diskussion

Der Nutzen der hier vorgestellten Systemarchitektur und der damit verknüpften Integrationsstrategie lässt sich in verschiedener Hinsicht betrachten. Zum einen analytisch, indem die Architekturveränderungen und die daraus resultierenden Vorteile für die Integration neuer Komponenten, genauso wie die Wartung des bestehenden Systems diskutiert werden. Zum anderen ist das übergeordnete Ziel, in dessen Rahmen diese Arbeit fällt, die Entwicklung alltagstauglicher Roboter in wenig strukturierten Umgebungen, wie zum Beispiel Privathaushalten. Hierfür sind Überlegungen aus dem praktischen Einsatz für eine Bewertung unumgänglich.

Die Analyse der Architektur wird unter dem Gesichtspunkt eines möglichst langlebigen und entwicklungsfähigen Systems betrieben. Sowohl der syntaktischen und semantischen Kopplung von Modulen, als auch dem Design der zentralen Koordination werden hier besondere Aufmerksamkeit zuteil. Die Verbesserung dieser Merkmale führt zu einer evolutionsartigen Systementwicklung, bei der sich ein Ausgangssystem neuen Erkenntnissen einfacher anpasst und parallel in unterschiedliche Forschungsrichtungen weiterentwickelt. In dem hier vorgestellten Ansatz zeigt sich dies anhand der Architekturentwicklung von einem gemeinsamen Ausgangspunkt zu den verschiedenen Systemen BIRON und Barthoc. Außerdem bietet der Ansatz die Möglichkeit, den selben Demonstrator mit wenig Aufwand und wiederverwendeten Modulen für unterschiedliche Szenarien einzusetzen. Die Anpassungsfähigkeit ermöglicht Szenarien außerhalb der Laborumgebung mit entsprechend positiver Wahrnehmung seitens externer Wissenschaftler sowie interessierter Laien.

7.1. Software-Integration und Systemwartbarkeit

Es wurde bereits dargestellt, in wie weit ein modularer und möglichst stark entkoppelter Aufbau die Systemintegration unterstützt und welche Maßnahmen zu treffen sind, um mit den entkoppelten Komponenten ein koordiniertes Systemverhalten zu erzielen. Nun werden die Auswirkungen der für die Integration neu entwickelten Architektur im Vergleich zu der Ausgangssituation diskutiert. Obwohl dies anhand konkreter Beispiele geschieht, steht diese Systemimplementation exemplarisch für den hier beschriebenen Ansatz. Die Ausgangsarchitektur (*SIRCLE - System Infrastructure for Robot Companion Learning and Evolution*) verfolgte bereits einen modularen Aufbau, basierte aber grundsätzlich auf direkten Kommunikationsverbindungen zwischen den einzelnen Komponenten und akzeptierte ein größeres gegenseitiges Wissen über andere Systemkomponenten in den jeweiligen Modulen. In Abbildung 7.1 wird die syntaktische Kopplung

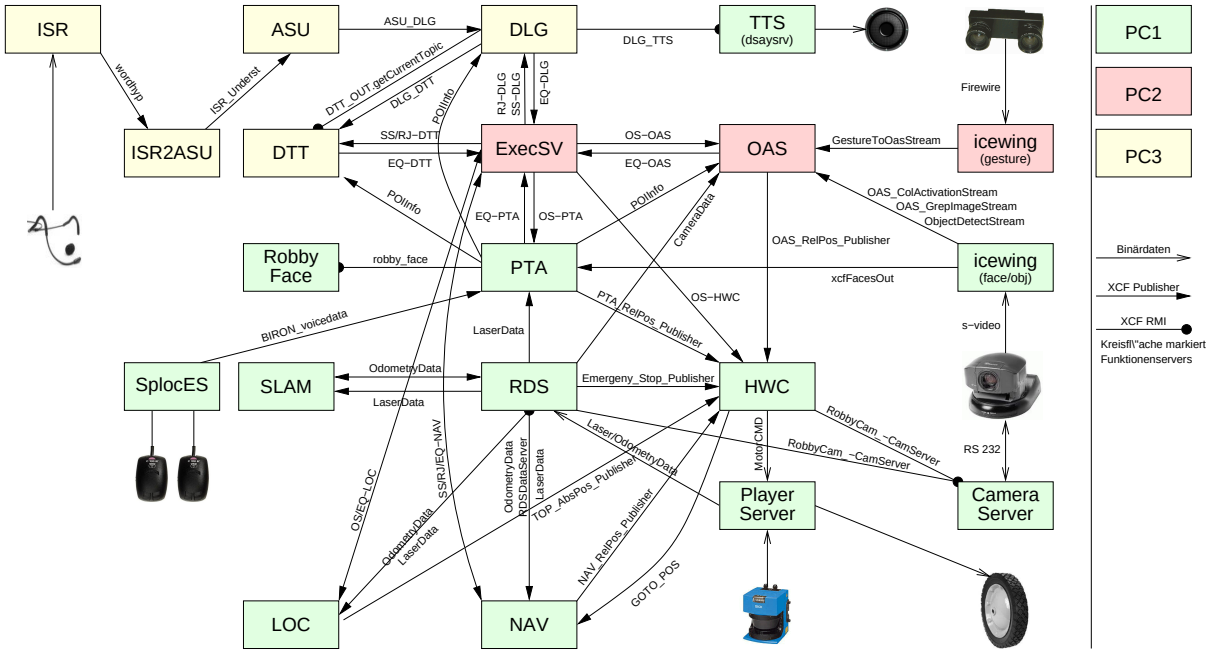


Abbildung 7.1.: Die ursprüngliche Architektur favorisierte bereits ein modularisiertes System. Die Komponenten waren jedoch sehr stark durch 52 direkte Datenverbindungen (*XCF Publisher / RMI*) miteinander verbunden. Bei einem Ausfall einer Komponente bedeutete dies meist einen systemweiten „Domino-Effekt“.

des Kommunikations-Frameworks beispielhaft für den Demonstrator BIRON dargestellt. Jede Bezeichnung an einer Pfeilverbindung repräsentiert einen unidirektionalen Datenstrom. Verbindungen mit einem Kreisende stehen für RMI, wobei das Kreisende jeweils die Komponente markiert, die den Methoden-Server bereit stellt. Viele Komponenten empfangen Daten, genauso wie sie selber Daten versenden. Es ist klar ersichtlich, dass die Fehlfunktion einer Komponente einen Domino-Effekt auslösen kann, der innerhalb des gesamten Systems fort propagiert wird und zu einem Totalausfall führt.

Die in dieser Arbeit realisierte evolutionäre Architektur zur Systemintegration (EASI) ermöglicht unter Verwendung eines gemeinsamen Memorys und der zentralen Koordination die syntaktische Entkopplung der Komponenten, wie in Abbildung 7.2 dargestellt. Direkte Datenverbindungen, seien es Datenströme oder RMIs, wurden weitestgehend durch Schnittstellen mit dem Memory ersetzt. Ausgenommen hiervon sind Module, die Rohdaten der Sensoren, wie Abstandsmessungen des Laser-Range-Finders oder Odometriedaten zur Verfügung stellen. Die entsprechenden Daten werden mit einer ausreichend hohen Frequenz zur Verfügung gestellt, so dass der Verlust eines einzelnen Datensatzes kaum Auswirkungen auf das System hat. Diese Komponenten sind des Weiteren demonstratorspezifisch und unterliegen somit einer deutlich geringeren Modifikationsrate wie die weiterverarbeitenden Module, deren Algorithmen häufiger dem aktuellen Forschungsstand angepasst werden. Dementsprechend haben die demonstratorspezifischen Komponenten einen höheren Grad an Robustheit erreicht.

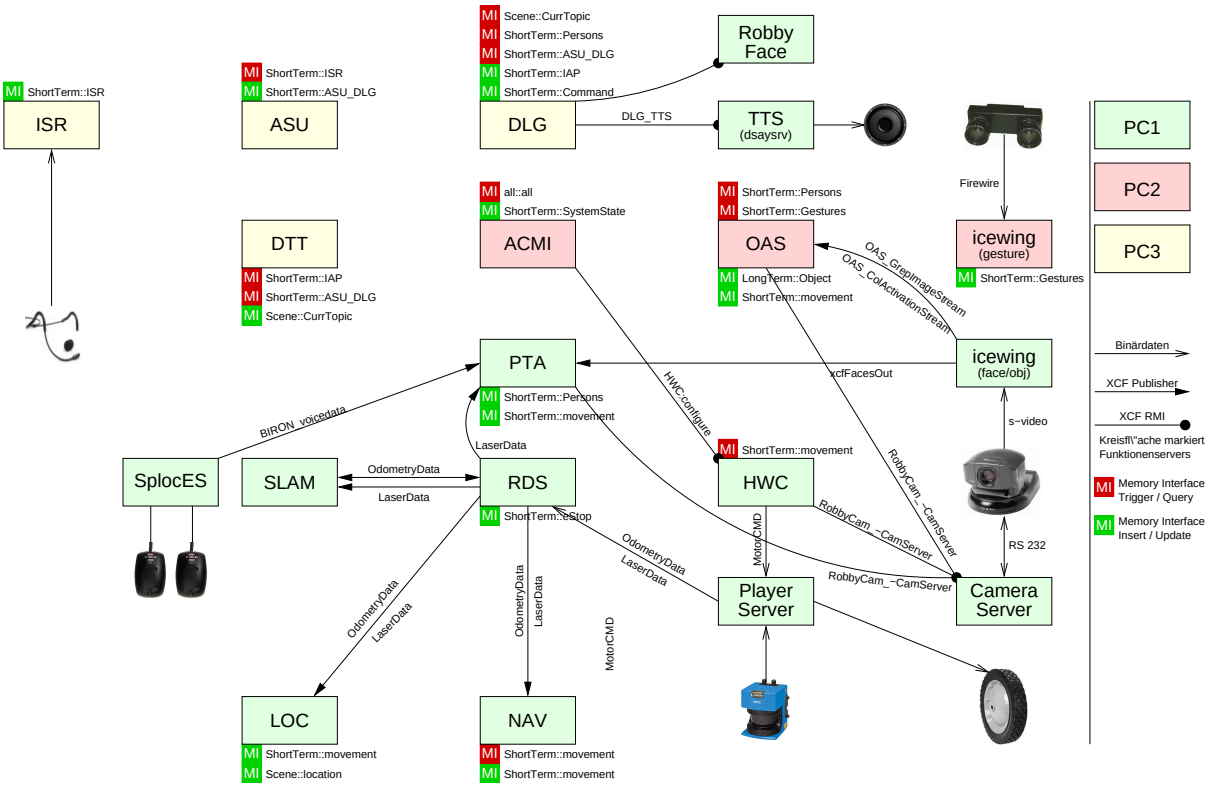


Abbildung 7.2.: Die Kommunikationsstruktur der hier vorgestellten EASI-Architektur greift nur noch auf 21 verbliebene Datenverbindungen (*XCF Publisher / RMI*) zurück. Dies erhöht die Austauschbarkeit erheblich und verringert im Fehlerfall die Auswirkungen auf weitere Systemkomponenten.

In wie weit die hier präsentierte EASI-Architektur die syntaktische Entkopplung der Komponenten unterstützt, lässt sich durch den Vergleich der direkten Datenverbindungen beider Ansätze gut veranschaulichen. Der vorangegangenen Implementation mit ihren 52 Verbindungen stehen bei gleicher Modul- und Schnittstellenanzahl nur noch 21 Verbindungen des EASI-Ansatzes gegenüber. Dies entspricht einer Verringerung der direkten Kommunikationsverbindungen um 59,6%. Elf der verbliebenen Datenverbindungen bestehen aus Datenströmen, die mit hoher Frequenz vorverarbeitete Sensordaten übermitteln und entsprechend der Argumentation aus Abschnitt 5.2.1 als solche erhalten bleiben können. Die restlichen Verbindungen basieren entweder auf RMI zur sicheren Hardware-Kontrolle oder sind im Rahmen der kontinuierlichen Systementwicklung zur Umstellung vorgesehen.

Wie in Abschnitt 4.1 erwähnt, wurde in Betracht gezogen, der syntaktischen Kopplung durch eine entsprechende Ausnahmebehandlung entgegenzuwirken, jedoch löst dies nicht das Problem der semantischen Kopplung in Bezug auf verlorene Daten beim Datenaustausch. Hierfür werden synchronisationssensible Daten im Memory des Systems vorgehalten, so dass sie jederzeit von Modulen abgefragt werden können. Für Komponenten, die historisch oder kooperationsbedingt in ihrer Prozessdurchführung nicht auf eine derartige Memory-Nutzung angepasst werden

können, übernimmt ACMI die Resynchronisation. Es analysiert auf Basis der zugrundeliegenden Regeln den Memory-Inhalt und modifiziert die Informationen, um die Systemkonsistenz wieder herzustellen. Im Extremfall geschieht dies durch Ausführen eines `RESET`s, welches die aktuellen Daten auf Standardwerte setzt.

Die erzielte Verringerung von syntaktischer und semantischer Kopplung spiegelt sich in der direkten Modulabhängigkeit sowohl bei Start, als auch partiellem Neustart, dem *Recovery* von Komponenten bei Fehlfunktion, wieder. Die Abbildungen 7.3 und 7.4 zeigen die Abhängigkeiten der Module des BIRON-Demonstrators. Es werden beide Kopplungsarten für die Bewertung der Abhängigkeit in Betracht gezogen. Die direkten Abhängigkeiten der einzelnen Komponenten untereinander werden durch die Pfeilverbindungen repräsentiert, wobei die Pfeilspitze auf die Komponente verweist, die zur Ausführung der Komponente am Pfeilende notwendig ist. Um die maximale Tiefe dieses Abhängigkeitsbaumes abzuschätzen, wurden unterschiedliche *Run-Level* definiert. Komponenten, die die gleichen Abhängigkeiten aufweisen, untereinander aber unabhängig sind, werden zu einem Run-Level zusammengefasst. Natürlich müssen bei Ausfall einer Komponenten in einem Run-Level nicht alle Komponenten aller darüber liegenden Run-Level neu gestartet werden, sondern lediglich die Komponenten, die eine direkte oder indirekte

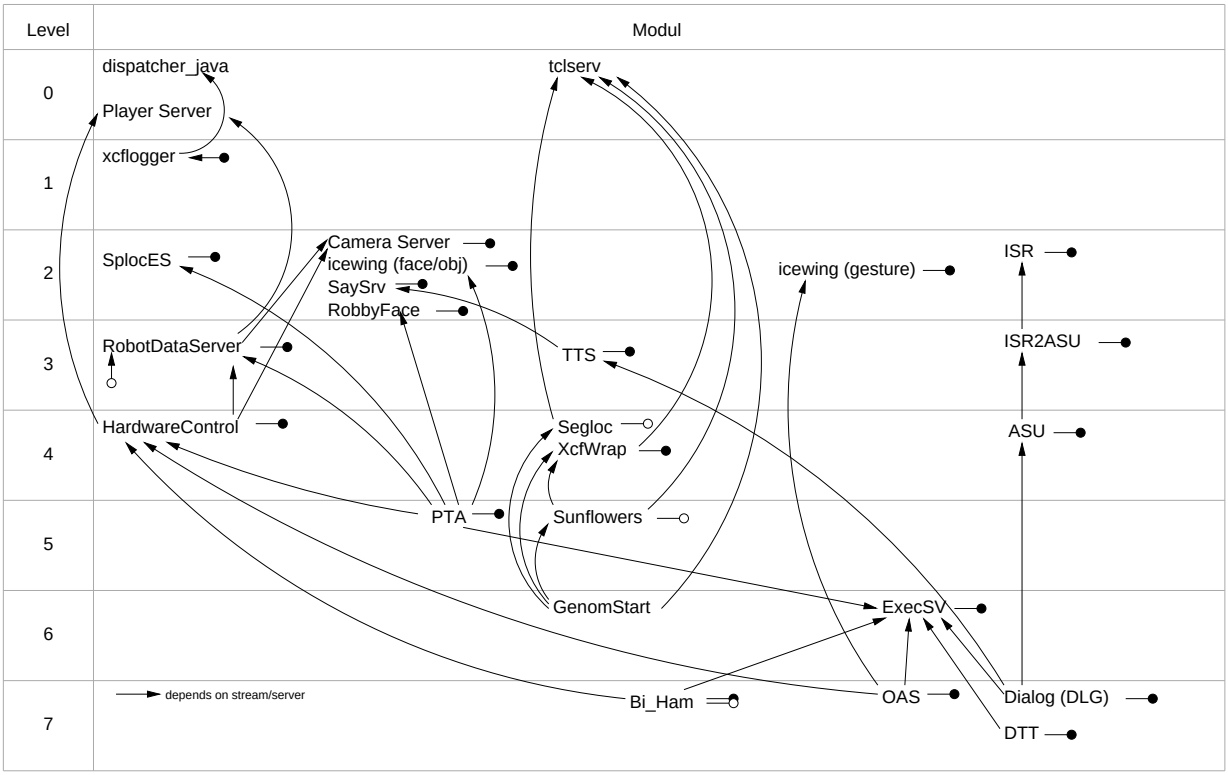


Abbildung 7.3.: Modulabhängigkeiten des Ausgangssystems: Komponenten, die die gleichen Abhängigkeiten aufweisen, untereinander aber unabhängig sind, werden zu einem *Run-Level* zusammengefasst. Ohne eine zentrale Koordination und ein Memory für den Datenaustausch existierten acht Level mit 55 Abhängigkeiten.

Abhängigkeit zu dieser Komponenten aufweisen. Die ersten beiden Run-Level umfassen die Hardware-Treiber und das Kommunikations-Framework, während in den darauf folgenden Leveln die perzeptgenerierenden und -verarbeitenden Komponenten folgen.

Unter Verwendung des hier beschriebenen Architekturkonzeptes ließen sich sowohl die Abhängigkeiten von 55 auf 35 um 36%, als auch die daraus resultierenden Run-Level um mehr als ein Drittel von acht auf fünf reduzieren. Das heißt, dass zuvor in 55 Fällen der beabsichtigte Austausch oder ein notwendiges Recovery einer Komponente zur Laufzeit die Funktion von mindestens einer weiteren Komponente störte. Fiel diese Komponente als Konsequenz der Störung selbst aus, konnte dies weitere Komponenten beeinträchtigen. Dieser Domino-Effekt konnte sich von dem aktuellen Run-Level bis zum obersten Run-Level fortsetzen. Entsprechend umfangreich waren eventuelle Neustarts im Ausgangssystem, so dass häufig das komplette System neu gestartet wurde, da dies einfacher war, als alle Abhängigkeiten aufzulösen. Die zwischenzeitlich gesammelten Daten gingen verloren. Daher ist die Reduktion der Run-Level und der Abhängigkeiten entsprechend wichtig, um mit weniger betroffenen Komponenten einen teilweisen Neustart mit möglichst geringem Datenverlust durchzuführen. Betrachtet man nur die Abhängigkeiten der perzeptgenerierenden und -verarbeitenden Komponenten ohne die für alle Komponenten obligatorischen Verbindungen mit dem Kommunikations-Framework und ohne die Hardware-Treiber aus den Leveln eins und zwei, so wird der Erfolg der Umstellung noch deutlicher. Die Abhängigkeiten reduzieren sich um mehr als 60% von 28 auf 11 und die Anzahl benötigter Level wird von sechs auf drei halbiert. Wird das volle Potential der hier vorgestellten EASI-Architektur genutzt, sind abgesehen von Kommunikations-Framework und Hardware-Treiber nur noch zwei Level notwendig. In diesen existiert ausschließlich die, der Memory-Performance

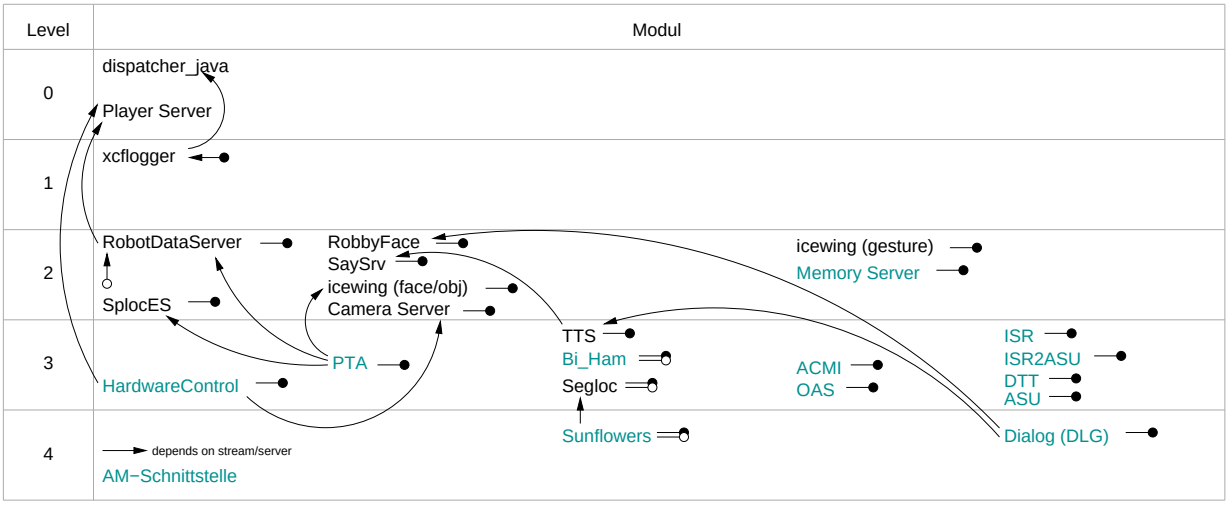


Abbildung 7.4.: Nach der Architekturumstellung reduzieren sich die Abhängigkeiten auf nur noch 35 Verknüpfungen. Durch die zentrale Koordination konnten einzelne Subkoordinationen der Lokalisierung und Navigation entfernt werden. Die Komponenten lassen sich in fünf Run-Leveln verwalten, was den zeitlichen Umfang teilweiser oder ganzer Systemneustarts deutlich verringert.

geschuldete, datenstrombasierte Abhängigkeit zwischen perzeptgenerierenden Komponenten in Level drei und perzeptverarbeitenden Komponenten in Level vier. Der Austausch höherwertiger, aus den Perzepten abgeleiteter Informationen geschieht entkoppelt über das Memory. Das modulübergreifende, semantisch koppelnde Systemwissen wird durch ACMI verwaltet, welches durch Modifikation des Memory-Inhaltes die einzelnen Module koordiniert.

Für die Bestimmung der Kopplung von Modulen innerhalb eines Systems ist die Anzahl der direkten Abhängigkeiten eine gute Schätzung, doch Faktoren, wie die inhaltliche Unterscheidung des Informationsaustausches in Daten- und Kontrollfluss, müssen ebenso berücksichtigt werden, wie die eventuelle Nutzung globaler Variablen. Ein geeignetes Bewertungsverfahren wird von Pressmann in [Pre82] als *Kopplungsmetrik* vorgestellt. Die Kopplung C eines Moduls lässt sich demnach einfach nach Formel 7.1 berechnen. d_i und c_i stehen hierbei für die Anzahl unterschiedlicher Daten- und Kontrollparameter, die ein Modul als Eingabeparameter benötigt, d_o und c_o stehen entsprechend für die Ausgabe. Die Anzahl globaler Daten- und Kontrollvariablen sind in g_d und g_c gegeben. Die permanenten Kommunikationsverbindungen finden sich als schreibende w und lesende r Modulschnittstellen wieder. Da Kontrollparameter in der Regel einen stärkeren Einfluss auf das System haben, kommt ihnen bei der Entkopplung eines Systems eine besonders hohe Bedeutung zu, weshalb sie bei diesem Modell doppelt gewichtet werden.

$$C = 1 - \frac{1}{d_i + 2c_i + d_o + 2c_o + g_d + 2g_c + w + r} \tag{7.1}$$

Als Beispiel, wie sich die Werte aus den vorliegenden Modulen ableiten lassen, wird das Dialogsystem (DLG) näher betrachtet. In der SIRCLE-Architektur besaß der Dialog vier unterschiedliche Eingabeparameter auf Dateneben ($d_i = 4$): Ergebnis des Sprachverstehens, aktuelles Gesprächsthema, personenbezogenen Informationen und Dateneingaben, die über den Execution Supervisor (ESV) weitergeleitet wurden. Weiterhin war der Informationseingang des Dialogs durch zwei unterschiedliche Kontrollparameter ($c_i = 2$) mit dem Execution Supervisor verbunden. Nach der erfolgten Verarbeitung wurden als Ausgabeparameter ($d_o = 3$) Informationen für die Sprachsynthese, die Themenerkennung und über den ESV weiter zu leitenden Informationen generiert. Der ESV erhielt zusätzlich Informationen über den aktuellen Dialogstatus ($c_o = 1$). Auf globale Variablen wurde sowohl in der ursprünglichen Implementierung, als auch in der aktuellen Architektur für alle Komponenten verzichtet. Die einzige Ausnahme bildet hier das ACMI mit seiner global zugreifbaren Regelbasis als Kontrollparameter ($g_c = 1$). Die Anzahl der permanenten Kommunikationsverbindungen des Dialogs ($w = 3$ und $r = 5$) finden sich in Abbildung 7.1 wieder. Der daraus resultierende Kopplungswert C beträgt 0,95. Nach der Architekturumstellung ist C auf 0.89 gesunken. Jetzt entspricht $d_i = 3$ den unterschiedlichen Dokumenttypen, auf denen sich der Dialog im Memory registriert und $d_o = 4$ der Summe unterschiedlicher Dokumenttypen, die der Dialog in das Memory schreibt, beziehungsweise über verbliebene RMI-Verbindungen versendet (siehe Abb. 7.1). Durch den Verzicht auf direkte Kontrollstrukturen ergibt sich für $c_i = c_o = 0$. Da der Event-basierte Datenaustausch zwischen zwei Komponenten keine permanente Kommunikationsverbindung zwischen eben diesen Komponenten erfordert entfällt dieser aus der Bewertung. Es ergeben sich für $w = 2$ und $r = 0$, basierend auf den Kommunikationsverbindungen zur Sprachsynthese und der Darstellung des Systemzustandes in Form einer Roboteranimation.

Tabelle 7.1 führt die Berechnungen und die ihnen zugrundeliegenden Parameter für die verwendeten Softwarekomponenten bezüglich der Ausgangsarchitektur und dem in dieser Arbeit entwickelten Ansatz auf. Ein Kopplungswert nahe 1 steht dementsprechend für eine hohe Kopplung, während ein Wert von 0,66 für eine niedrige Kopplung steht. Die durchschnittliche Kopplung aller Komponenten des SIRCLE-Systems mit einem Wert von 0.79 weist bereits auf eine deutlich modular gehaltenen Architektur hin. Durch Anwendung einer zentralen Koordination auf Basis des Event-basierten Datenaustausches ließ sich dieser Wert auf 0.69 für den EASI-Ansatz reduzieren. Dabei konnten vor allem die Kopplung bei Modulen verringert werden, die zuvor stark an die Kontrolle des ESV gebunden waren. Beispiele hierfür sind die Gesprächsthemenerkennung (*Dynamic Topic Tracking*, *DTT*) sowie Lokalisation (*LOC*) und Navigation (*NAV*), deren Kopplungswerte durch die Architekturumstellung von über 0,9 auf 0,67 beziehungsweise 0,83 gesunken sind. Die Verringerung dieses Wertes lässt sich bei allen Komponenten bis auf den *Camera Server* feststellen. Diese Komponente ermöglicht den Zugriff auf die Pan-/Tilt-Kamera von BIRON und deren aktuelle Ausrichtung. Während die Ausrichtungsinformationen der Kamera ursprünglich nur durch einen Zwischenschritt über den *Robot Data Server* (*RDS*) verfügbar waren, erfolgt der lesender Zugriff nun effizienter direkt. Durch den direkten Zugriff mehrerer Komponenten, der für Hardware-nahe Module per RMI erfolgt, ist folglich die Kopplung hier leicht von 0,75 auf 0,8 gestiegen.

Modul	SIRCLE									EASI								
	d_i	c_i	d_o	c_o	g_d	g_c	w	r	C	d_i	c_i	d_o	c_o	g_d	g_c	w	r	C
ISR	0	0	1	0	0	0	1	0	0,50	0	0	1	0	0	0	0	0	0,00
ASU	1	0	1	0	0	0	1	1	0,75	1	0	1	0	0	0	0	0	0,50
DLG	4	2	3	1	0	0	3	5	0,95	3	0	4	0	0	0	2	0	0,89
TTS	1	0	0	0	0	0	0	1	0,50	1	0	0	0	0	0	0	1	0,50
DTT	3	2	2	1	0	0	2	4	0,94	2	0	1	0	0	0	0	0	0,67
ESV	6	6	6	9	0	0	10	6	0,98	-	-	-	-	-	-	-	-	-
ACMI	-	-	-	-	-	-	-	-	-	8	0	3	1	0	1	1	0	0,94
OAS	6	1	2	1	0	0	2	7	0,95	5	0	2	0	0	0	0	4	0,91
gesture	0	0	1	0	0	0	1	0	0,50	0	0	1	0	0	0	0	0	0,00
RobbyFace	1	0	0	0	0	0	0	1	0,50	1	0	0	0	0	0	0	1	0,50
PTA	4	1	4	1	0	0	6	4	0,95	4	0	2	0	0	0	0	4	0,90
face/obj	0	0	4	0	0	0	4	0	0,88	0	0	4	0	0	0	4	0	0,88
SplocES	0	0	1	0	0	0	1	0	0,50	0	0	1	0	0	0	1	0	0,50
SLAM	2	0	1	0	0	0	1	2	0,83	2	0	1	0	0	0	1	2	0,83
RDS	3	0	3	0	0	0	10	4	0,95	2	0	3	0	0	0	7	2	0,93
HWC	5	1	2	0	0	0	3	6	0,94	1	1	2	0	0	0	2	1	0,88
PlayerServer	1	0	2	0	0	0	2	1	0,83	1	0	2	0	0	0	1	1	0,80
CamServer	1	0	1	0	0	0	1	1	0,75	1	0	1	0	0	0	2	1	0,80
LOC	3	1	2	1	0	0	2	3	0,93	2	0	2	0	0	0	0	2	0,83
NAV	5	2	2	1	0	0	2	6	0,95	3	0	1	0	0	0	0	2	0,83
Summe	46	16	38	15	0	0	52	52	./.	37	1	32	1	0	1	21	21	./.
Durchschnitt	2,42	0,84	2,00	0,79	0	0	2,74	2,74	0,79	1,95	0,05	1,68	0,05	0	0,05	1,11	1,11	0,69

Tabelle 7.1.: Die Berechnung der Modulkopplung für die Module des BIRON-Demonstrators nach Formel 7.1 zeigt eine stärkere Entkopplung für den EASI-Ansatz, verglichen mit SIRCLE.

Die gezeigte Entkopplung ist nicht nur Grundlage für die bislang angeführte Modifikation und Erweiterung eines Systems, sondern auch für die Übertragung von Komponenten in neue Systeme für alternative Demonstratoren und beziehungsweise oder alternative Szenarien. Für den Demonstrator Barthoc wurden aus den 19 Modulen (siehe Abb 7.1) 11 Module aus den Bereichen Sprachverarbeitung sowie Personen- und Objekterkennung von BIRON übernommen. Durch die geringere Kopplung mussten algorithmisch nur an der Personenerkennung Modifikationen hinsichtlich der Hardware-Unterschiede vorgenommen werden. Nachdem die entsprechenden Modulfunktionalitäten des BIRON-Demonstrators auf Barthoc hergestellt worden waren, wurde darauf aufbauend das EIS-Szenario (siehe Abschnitt 3.1.2) realisiert. Für die Entwicklung hin zu diesem Szenario wurden Module zur Verwaltung der Umgebungsinformationen und zur körpersprachlichen Referenzierung hinzugefügt. Bis auf die Modifikation der Sprachinteraktion zur Verarbeitung neuer szenariotypischer Anfragen, blieben die restlichen Komponenten unverändert, da konkurrierende Hardware-Zugriffe für Personendetektion und Körpersprache über ACMI neu koordiniert wurden.

Koordinationskomplexität

Eine Entkopplung von Komponenten durch zentralisierte Koordination der Systeminformationen birgt nur dann einen Gesamtvorteil in sich, wenn der Verwaltungsaufwand dieser zentralen Koordination geringer ausfällt, als der Verwaltungsaufwand stärker gekoppelter Module mit geringerer zentraler Koordination. Daher muss für eine Bewertung dieses Ansatzes die Struktur der ACMI-Regelbasis in Umfang und Komplexität mit einem alternativen Modell unter möglichst gleichen Rahmenbedingungen verglichen werden. Endliche Automaten sind nicht nur eine weit verbreitete Alternative, sie wurden sowohl zur Steuerung von Einzelkomponenten wie dem Personen-Tracking und dem Dialogsystem auf BIRON eingesetzt, als auch für die Synchronisation des Gesamtsystems (siehe Abbildung 4.3, Seite 58).

Betrachtet man die Verknüpfungen der ursprünglichen Systemsynchronisation näher, so ergeben sich 36 Transitionen mit ihren jeweiligen Datenein- und -ausgaben zwischen den verschiedenen Systemzuständen. Die Abbildung 4.3 visualisiert auch die starke Abhängigkeit der jeweiligen Transition von den vorausgegangenen Transitionen. Bei einer Systemerweiterung spielt dies hinsichtlich unbeabsichtigter Seiteneffekte genauso eine Rolle, wie bei dem Erhalt eines konsistenten Systemzustandes nach Austausch oder Neustart einer Software-Komponente. Durch eine hohe Verknüpfung von Transitionen müssen bei einer Software-Integration an einer größeren Anzahl von Transitionen Anpassungen vorgenommen werden, als bei linearen oder sogar disjunkten Transitionen, beziehungsweise Regeln. Selbiges gilt auch für den Neustart von Einzelkomponenten im laufenden System. Es sollten möglichst wenig Transitionen oder Regeln durchlaufen werden müssen, um wieder in einen synchronen Zustand aller Komponenten zu gelangen.

Für die Erweiterung des Demonstrators BIRON um die Lokalisation mit Navigation, welche Analog zu der Architekturentwicklung stattgefunden hatte, war bei der letzten Anwendung der FSM-Synchronisation die Anzahl der Transitionen von 36 um 17 auf 53 angestiegen. Die Anzahl

unterschiedlich definierter Aktionen, die bei den Transitionen ausgeführt wurden, steigerte sich von 18 um 12 auf 30. Im Vergleich hierzu werden bei dem aktuellen Ansatz kontinuierlich fünf unterschiedliche Aktionstypen benötigt, die von insgesamt 35 Regeln aus aufgerufen werden. Unter den 35 Regeln beträgt die Anzahl von Regeln, die für die Lokalisationserweiterung notwendig sind acht. Mit diesen acht Regeln werden jedoch parallel zwei alternative Lokalisationsansätze unterstützt. Würde der letzte Evolutionsschritt der FSM-Synchronisation auf der ACMI-Regelbasis nachvollzogen, so erweiterte sich diese von ursprünglich 27 auf 39 Regeln. Dies entspricht sowohl in der jeweiligen Anzahl als auch im Wachstum nur zwei Dritteln der FSM-Synchronisation.

Zusätzlich zu der reinen Regelanzahl wird auch die oben motivierte, möglichst geringe Verknüpfung von Regeln berücksichtigt. Die Grafik 7.5 symbolisiert den Aufbau der verwendeten Regelbasis. Jede horizontale Linie steht dabei für eine Regel. 24 dieser Regeln und somit mehr als zwei drittel der Regelbasis sind voneinander disjunkt. Das heißt, ihre Ausführung hat keine direkte Auswirkungen auf andere Regeln. Dies wird durch das abschließende Quadrat symbolisiert. Nur 11 Regeln stehen so miteinander in Beziehung, dass die Ausführung einer Regel eine weitere nach sich zieht. Die jeweiligen Regeln sind mit den assoziierten Informationen dargestellt und durch Pfeilspitzen markiert. Bei diesen direkten Abhängigkeiten wurde explizit auf die Vermeidbarkeit von sich auffächernden Baumstrukturen oder Rekursionen geachtet. Zwar können beide Strukturen mit dem hier vorgestellten Ansatz einfach realisiert werden, aber die Wartbarkeit vermindert sich durch die damit einhergehenden Abhängigkeiten. Dies gilt besonders für baumartige Verzweigungen in denen eine Regelausführung mehrere weitere Regeln auslöst. Bei der Modifikation der Ausgangsregel müssen die Auswirkungen auf alle Folgeregeln beachtet werden. Baumartige Verzweigungen zeigen sich bei dem BIRON-System nur als „Verjüngung“ in Bezug auf die Lokalisation und Objektdetektion. Jeweils zwei Regeln mit dem Status `init` aber unterschiedlichen Voraussetzungen (`GUARD`s) bewirken bei ihrer Anwendung den Aufruf einer gemeinsamen Folgeregel mit dem Status `accepted`.

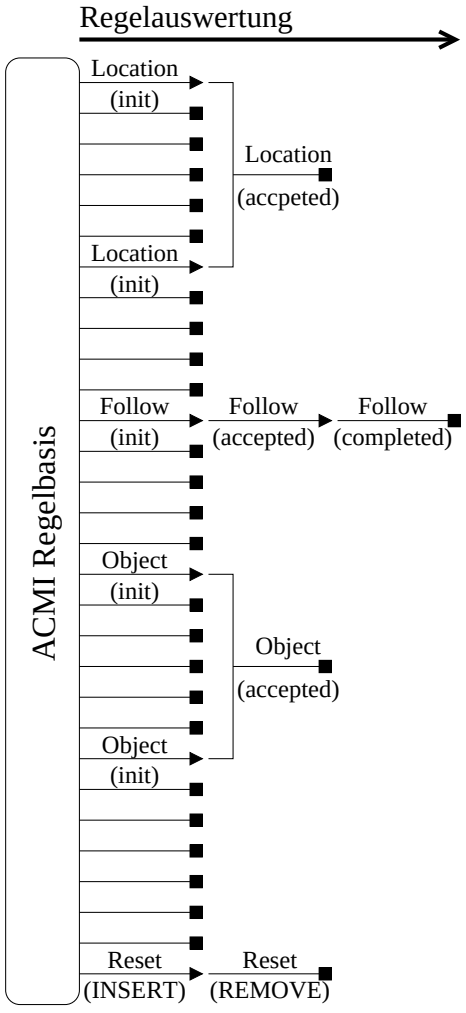


Abbildung 7.5.: Nur elf Regeln besitzen direkte Abhängigkeiten. Rekusionen können der Übersichtlichkeit halber vermieden werden.

Durch die Vermeidung von Aufspaltungen bleibt nicht nur ein einfacherer Koordinationsüberblick gewahrt, sondern einzelne Regeln können auch bei der Anwendung auf neue Demonstratoren oder der Weiterentwicklung von Szenarien besser portiert, erweitert oder entfernt werden. Für das EIS-Szenario mit Barthoc konnte die Regelbasis auf nur zwei Regeln für das Zurücksetzen der Systeminformationen und die Hardware-Zugriffe reduziert werden (siehe Anhang A.2). Da keine generelle Kontrolle auf die Systemkomponenten ausgeübt wird, kann auch ein sehr komplexes System entwickelt werden, in dem auf Koordinationsinformationen in den einzelnen Komponenten verzichtet wird und bei dem die Regelbasis dennoch überschaubar bleibt. Dies stellt für schnelle Modifikationen im praxisnahen Einsatz der hier vorgestellten Systeme eine deutliche Arbeitserleichterung dar.

7.2. Praktische Anwendung der Systemarchitektur

Für den praxisnahen Einsatz ist ein möglichst stabiles und überschaubares Systemverhalten auch bei steigender Anzahl von Komponenten unumgänglich. Syntaktische und semantische Fehler potenzieren sich schnell mit steigender Komponentenanzahl. Dies gilt für die technische Umsetzung im Sinne von Einfachheit bei der Systemverwaltung durch technisches Personal, für die Fehlerbehandlung bei Integration und Interaktion sowie in Bezug auf Robustheit in der Anwendung durch ungeübte Benutzer. Hierzu wird auf die Ergebnisse mit dem Demonstrator BIRON während zweier Benutzerstudien eingegangen. Außerdem werden Erfahrungen geschildert, die mit Barthoc während und nach der Portierung der Architektur auf diesen Demonstrator gemacht wurden und die diesen Ansatz für eine flexibel einsetzbare Architektur unterstützen.

Software-Integration

Die Anwendbarkeit einer Systemarchitektur zeigt sich deutlich bei der Integration einzelner Komponenten. Zeit spielt hierbei eine wichtige Rolle, sei es in der Wissenschaft oder in der Industrie. Zeit ist ein repräsentatives Maß für die Komplexität einer Aufgabe, sofern Anzahl und Kompetenz, der mit dieser Aufgabe betreuten Personen, konstant sind. Die folgende Zusammenfassung beschreibt das Vorgehen bei drei Integrationsarbeiten, die gemeinsam mit Kollegen von unterschiedlichen Forschungseinrichtungen durchgeführt wurden, ohne dass deren Software ursprünglich für den verwendeten Demonstrator BIRON entwickelt wurde. Bei den zu integrierenden Komponenten handelte es sich um zwei alternative Lokalisationsansätze und ein Modul zur autonomen Navigation. Dabei wurde folgender Entwicklungsprozess, der sich als erfolgreiche Strategie erwiesen hat, durchlaufen.

1. Fernmündliche Absprache der Schnittstellen
2. Unabhängige Vorbereitung der Schnittstellen im jeweiligen Heimatinstitut
3. Upload der Software in die Software-Umgebung des Demonstrators und Kompilation durch die Entwickler

4. Treffen für die Integration am Standort des Demonstrators mit abschließender Anpassung der Schnittstellen.

Während die Punkte 2. und 3. eine individuellere Arbeitsweise ermöglichen und sich an dem Vorgehen der Entwickler orientieren, ist für die erste Absprache und die finale Integration eine gute Koordination aller Beteiligen unabdingbar. Natürlich lassen sich durch eine gute Vorbereitung besondere Herausforderungen in der Integration früher identifizieren und somit leichter meistern. Realistisch betrachtet ergeben sich jedoch, durch Missverständnisse oder geringe menschliche Unachtsamkeiten, in der finalen Integration immer wieder Hindernisse, die es zu bewältigen gilt.

An diesem Punkt zeigt sich bereits der Vorteil des hier beschriebenen Ansatzes. Nicht nur die in Abschnitt 5.2.2 erläuterten Adaptionsfähigkeiten unterstützen die abschließende Anpassung der Schnittstellen, sondern auch die zuvor beschriebene Entkopplung der Einzelkomponenten. Bei der Fehleranalyse während der Integration werden zeitsparend nur relevanten Teile des Systems verwendet und gegebenenfalls neu gestartet. Der Integrationszyklus, bestehend aus Design, Test, Fehleranalyse und Redesign, verkürzt sich im Bereich des Test deutlich. Mit der Verwendung von Schema-Validierungen für die ausgetauschten Informationen (hier XML) werden sowohl syntaktische, als auch semantische Fehler bereits beim ersten Versenden der Informationen lokalisiert. Dies erspart besonders bei logischen Fehlern, die sich erst an einer ganz anderen Stelle im System zeigen, eine umfangreiche Suche nach dem Auslöser. Auch wenn dies kein wissenschaftliches Maß darstellt, so soll hier kurz ein Erfahrungswert für die bei der Integration aufgewendete Zeit gegeben werden. Nach der Bereitstellung der noch individuell erstellten Software in Schritt 3 des Entwicklungsprozesses, bedurfte es in keinem der drei in [Spe09] beschriebenen Integrationsvorhaben mehr als drei Tage. Innerhalb dieser Zeit gelang es jedes mal in Kooperation mit einem Entwickler der Fremd-Software, diese Software vor Ort in das Gesamtsystem zu integrieren und erfolgreich im laufenden System einzusetzen.

Home-Tour Studie

Dieses laufende System ist letztendlich das Ziel der Systemintegration und wurde dementsprechend getestet. Die besondere Herausforderung bestand darin, dass diese Tests unter möglichst realen Bedingungen stattfanden. Dementsprechend wurde die in Abschnitt 3.1.1 genannte Home-Tour in dem beschriebenen Apartment (siehe Abb. 3.1, Seite 31) durchgeführt. Als Interaktionspartner wurden Personen ausgewählt, die noch nie zuvor mit BIRON interagiert hatten. Dazu bekamen sie eine kurze Einführung von durchschnittlich ca. acht Minuten Länge, in der die Versuchspersonen auch Gelegenheit für Rückfragen hatten. Anschließend bestand die Aufgabe darin, den Roboter durch die Wohnung zu führen und ihm drei Räume mit sich darin befindliche Einrichtungsgegenstände zu zeigen. Die Abbildung 7.6 zeigt einige Versuchspersonen während des Experimentes in den unterschiedlichen Wohnungsbereichen. Der Versuch wurde aus zeitlichen Gründen in zwei Blöcken zu 10 und 14 Testpersonen mit drei Monaten Abstand durchgeführt. Unter den 24 Versuchspersonen waren je 12 Frauen und Männer mit einem Durchschnittsalter von 37,2 Jahren.

Wohnzimmer Flur Esszimmer

Abbildung 7.6.: Benutzerstudie mit BIRON: Die Versuchspersonen führen den Roboter durch verschiedene Räume der Wohnung (siehe Abb. 3.1, Seite 31) und zeigen ihm Einrichtungsgegenstände.

Aus der Sichtweise der Systemintegration ist ein besonderes Augenmerk auf die Robustheit des Gesamtsystems während der Benutzerinteraktion zu legen. Im Gegensatz zu den Entwicklern mit ihren sich wiederholenden gleichen Testabläufen, zeigen unerfahrene Anwender häufig ein unerwartetes Verhalten, welches jedoch nicht zum Versagen des Gesamtsystems führen darf. Auch wenn durch sogenannte „Stress-Tests“ einzelne Systemfähigkeiten sicherlich genauer geprüft werden können, stellt der Einsatz unter realen Bedingungen mit naiven Anwendern eine weitaus höhere Anforderung an die Flexibilität des System. Diese Anforderung bekommt besonderes Gewicht, sofern man berücksichtigt, dass zum Zeitpunkt dieser Studie die Umstellung des Demonstrators auf die hier beschriebene Architektur durchgeführt wurde. Während ein Teil der Komponenten noch über den endlichen Automaten synchronisiert wurde, waren andere Module bereits an das zentrale Memory angeschlossen und wurden von ACMI koordiniert.

Die hier beschriebene Architektur war erfolgreich in der Lage, einen Ausfall des Demonstrators wegen der Systemanpassungen zu verhindern und so ihrer Hauptaufgabe, der Unterstützung einer kontinuierlichen Weiterentwicklung, nachzukommen. Tatsächlich fielen während der insgesamt 7,2 Stunden des Experiments acht mal verschiedene Einzelkomponenten aus: je zweimal das Personen-Tracking und die Objektaufmerksamkeit sowie je einmal die Gesichtsdetektion, die Gestendetektion, die Lokalisation und die Sprachverarbeitung. Nur der Ausfall der Sprachverarbeitung, welche zu diesem Zeitpunkt noch nicht auf die neue Architektur angepasst war, bedurfte eines Systemneustarts, um das Experiment fortzusetzen. In den anderen Fällen genügte

ein Neustart der jeweiligen Komponente für eine erfolgreiche Systemerholung. Die Resynchronisation der ausgefallenen Komponente mit dem System fand über das Memory statt und wurde über die verbale Anweisung „Reset“ vom Anwender initiiert.

Über die „Reset“Anweisung hat der Anwender ebenfalls die Möglichkeit, aus jeder Systemaktion des Roboters an einen, dem Anwender bekannten, Ausgangspunkt zu Beginn der Konversation zurückzukehren. Dieses für den Anwender simple Verfahren wird durch die zentrale Datenverwaltung im Memory direkt unterstützt. Alle notwendigen Modifikationen werden am gemeinsamen Speicherort der beteiligten Komponenten durchgeführt und nicht erst über mehrere Transitionen eines Automaten ohne Rückmeldung übermittelt. Entsprechend hoch war die Akzeptanz dieses Verfahrens [Spe08a], welches nicht ausschließlich durch den Anwender ausgelöst werden kann. Erkennt das Dialogsystem wiederholt eine Anweisung, die aufgrund der verfügbaren Informationen nicht umsetzbar ist, schlägt es dem Anwender vor, lieber erneut zu beginnen und setzt das System nach dessen Zustimmung zurück. So werden Sackgassen, die zum Beispiel auf unerwartetem Benutzerverhalten basieren, umgangen, bis das System entsprechend der neuen Erkenntnisse angepasst wird.

Rezeptionist Szenario

Abbildung 7.7.: Öffentliche Präsentation von Barthoc in der Haupthalle der Universität Bielefeld anlässlich des jährlichen Tages der offenen Tür. Barthoc erteilte über Sprache und Zeigegesten Auskunft über die verschiedenen Räumlichkeiten der Universität.

Parallel zu den Studien mit dem BIRON-Demonstrator wurde damit begonnen, die Architektur auf den Demonstrator Barthoc (siehe Abschnitt 3.2.2) zu übertragen und für den *Sonderforschungsbereich 673 - Alignment in Communication*[1] der Deutschen Forschungsgesellschaft nutzbar zu machen. Obwohl der Demonstrator sowohl von seinem Aufbau, als auch von dem Einsatzgebiet stark von der ursprünglichen Entwicklungsplattform BIRON abweicht, konnten - wie im vorigen Abschnitt beschrieben - viele Komponenten übernommen werden. Szenarien- und demonstratorspezifische Module konnten dank der flexiblen Systemarchitektur, wie in Abbildung 5.1 auf Seite 61 gezeigt, einfach ausgetauscht oder ergänzt werden. Hierfür musste nur

[1] http://www.sfb673.org

die neue Regelbasis für das ACMI (siehe Anhang A.2) verwendet werden. Als Ergebnis wurde eine interaktive Demonstration im Rahmen des erfolgreichen Wettbewerbes *„Deutschland - Land der Ideen“ (2007)* gegeben. Gegenstand dieser Demonstration war ein Auskunftssystem (*Environment Information System - EIS*) für die Räumlichkeiten der Universität Bielefeld (siehe Abschnitt 3.1.2). Dieses System wurde im Rahmen der Geniale 2008 erneut der Öffentlichkeit präsentiert, wobei nun auch Besucher die Möglichkeit hatten, direkt mit dem Roboter zu interagieren. Die Abbildung 7.7 zeigt den Aufbau und das rege Interesse der Besucher. Wegen des hohen öffentlichen Interesses und datenschutzrechtlicher Bestimmung bezüglich Videoaufzeichnungen war eine quantitative Evaluation der Besucherinteraktionen leider nicht möglich. Qualitativ ließ sich festhalten, dass aufgrund der Geräuschkulisse die Spracherkennung gelegentlich an ihre Grenzen stieß, sich das System jedoch insgesamt als robust zeigte.

RoboCup@Home

Abbildung 7.8.: Das *Team of Bielefeld* (*ToBI*) präsentierte sich erfolgreich auf dem RoboCup 2009. Zwischen den offiziellen Wettbewerben in einer nachgestellten Privatwohnung herrschte ebenfalls reges Besucherinteresse an dem System.

Eine weitere Möglichkeit für eine öffentlichkeitsnahe Evaluation von Robotersystemen besteht in der Teilnahme an Wettbewerben. Ein Wettbewerb, der sich mit den Zielen der Home-Tour – einen flexiblen Roboter für den Privatanwender zu schaffen – sehr gut vereinbaren lässt, ist der *RoboCup@Home*[2]. Als eigenständiger Teil des auf Fußball fokussierten RoboCups, müssen im RoboCup@Home unterschiedliche Szenarien gemeistert werden, die in Zusammenhang mit Service-Robotern in alltäglichen Umgebungen stehen. Dementsprechend wird eine möglichst natürliche Umgebung, wie eine kleine Wohnung nachgebildet, in welcher verschiedene Aufgaben zu bewältigen sind. Die Aufgaben verändern sich von Jahr zu Jahr, um bei wachsenden technischen Möglichkeiten eine ständige Herausforderung zu gewährleisten. Die nachgebaute Testumgebung ist bis zur Ankunft der Teams unbekannt. Somit werden beide Aspekte der Architektur, nämlich kontinuierliche Weiterentwicklung der Fähigkeiten und schnelle Adaption an unbekannte Einsatzgebiete gefordert. Eine besondere Herausforderung bei den unterschiedlichen Szenarien ist die sogenannte *Open Challange*, eine Kür, in der jedes Team die Möglichkeiten seines Demonstrators frei darstellen kann. Das inzwischen vollständig auf ACMI umgestellte

[2] http://www.robocupathome.org

und auch seitens der Hardware aktualisierte BIRON-System [Wac09] erhielt bei seiner ersten RoboCup-Weltmeisterschaft mit der Home-Tour auf Anhieb den 4. Platz in der Open Challenge in einer absolut fehlerfreien Demonstration. Damit wurde trotz mangelnder Manipulationsfähigkeit insgesamt ein guter 8. Platz unter den 18 internationalen Mitbewerbern erzielt.

EU-Projekt Bewertung

Eine deutlich stärker wissenschaftlich geprägte, aber gleichermaßen internationale Beurteilung, wurde dem BIRON-System aus dem EU-Projekt *Cogniron – The Cognitive Robot Companion*[3] zuteil. An diesem Projekt beteiligten sich neun Forschungseinrichtungen aus sechs verschiedenen Ländern, um gemeinsam neue Ansätze für den Bau intelligenter Roboter im alltäglichen Einsatz zu entwickeln. Das Ziel dieses Projektes war die Integration der unterschiedlichen Ansätze auf eine limitierte und festen Szenarien zugewiesene Anzahl von Demonstratoren. Durch die hohe Anzahl von Projektpartnern und die große räumliche Trennung kam der Integration und Gestaltung einer geeigneten Systemarchitektur eine entsprechend hohe Bedeutung zu und wurde dementsprechend als eigenständiges Arbeitspaket definiert. Die Beurteilung der Reviewer bezüglich des Home-Tour-Szenarios (*Key Experiment 1 - KE1*) im Abschlussbericht [Coh08] liest sich diesbezüglich wie folgt:

> *„KE1 represents an impressive feat of integration. [...] KE1 also involves the integration of components from seven different partners. Although this is in some sense 'mere' software engineering, it is nonetheless a difficult step which is too often skipped and the partners are to be commended for it. This kind of real-world use of components outside the labs that built them also often helps mature those components, making them more likely to be useful to other groups after the termination of the project.“*

Natürlich ist bei den Systembewertungen zu beachten, dass die Entwicklung nur durch die gute und langjährige Zusammenarbeit vieler Projektmitglieder erreicht werden konnte. Die Kommentare beziehen sich damit nicht ausschließlich auf die hier vorgestellte Architektur. Die Architektur ist jedoch ein entscheidender Bestandteil dieses Systems und wurde zusammen mit dem Active Memory auch so wahrgenommen.

> *„[...] were pleased to note the developments in the overall Cognitive Architecture, [...] facilitating integration and code sharing. This is a useful advance which has the potential to live on beyond the scope of this project. The development of the active memory aspect of the architecture also represents a novel contribution, interesting new contribution, allowing distributed decision making in a good software engineering fashion.“*

[3] http://www.cogniron.org; FP6-IST-002020

Abgesehen von der Verwendung der gleichen Programmiersprache, existierten kaum Gemeinsamkeiten was die Implementation der jeweiligen Software-Komponenten anging. Dies galt besonders für den Datenaustausch und dessen Schnittstellendefinitionen. Dass die jeweiligen Komponenten an den entsprechenden Instituten jeweils wieder in eigenen Systemen eingesetzt wurden oder aber auf bereits vorhandener Software aufbauten, erschwerte zwar die Integrationsaufgabe, wurde aber entsprechend anerkannt.

> *„We would also like to note the considerable resources spent on the integration of complementary detection and tracking algorithms on the demonstration platforms, which was certainly not an easy task since the components were constructed in different software infrastructures.“*

Letztendlich wurden verschiedene Architekturansätze abhängig von den Demonstratoren in ihren jeweiligen Szenarien umgesetzt, die erfolgreich Komponenten von mehreren Partnern berücksichtigten. Auf dem Demonstrator BIRON kamen mit Komponenten von insgesamt sieben Projektpartnern die mit Abstand umfangreichste Integration zum Einsatz. Im Vergleich zu alternativen Architekturen operierten die Module von jeweils sechs Projektpartnern gleichzeitig zur Laufzeit in einem gemeinsamen System, während abwechselnd zwei alternative Lokalisationsverfahren flexibel ausgetauscht werden konnten. Nicht zuletzt durch die Unterstützung der Partner und der flexiblen Integrationsmöglichkeit wurde die Home-Tour sehr erfolgreich aufgenommen.

> *„[...] this key experiment represented the best aspect of integration across the consortium, making good use of components from a wide range of partners [...]“*

Augmented Reality based Interception Interface

Auch wenn die hier vorgestellte Systemarchitektur mit ihrer Koordinationskomponente ACMI ursprünglich im Hinblick auf die umfangreichen Software-Systeme der Demonstratoren BIRON und Barthoc entwickelt wurde, kann sie auch bei geringeren Aufgaben ohne großen architektonischen Overhead eingesetzt werden. So wird die Einzelkomponente ACMI als Adapter für die Struktur von Datenelementen und deren Kommunikationsmodalität eingesetzt. ACMI wird im ARbInI-System (*Augmented Reality based Interception Interface*) [Die09] dem Nachfolgeprojekt von VAMPIRE[4] mit den entsprechenden Regeln (siehe Anhang A.3) als Adapter zum Einlesen mehrerer Datenströmen in ein Memory verwendet. In der Kenntnis, dass das Active-Memory ursprünglich im Rahmen des VAMPIRE Projektes entwickelt wurde, schließt sich der Entwicklungskreislauf und zeigt, wie gut die unterschiedlich fokussierten Ansätze letztendlich wieder ineinander greifen.

[4] http://www.vampire-project.org/

7.3. Zusammenfassung

Die Analyse einer Systemarchitektur und der aus ihr hervorgehenden Integration von Software-Komponenten birgt andere Anforderungen als die Bewertung neuer Algorithmen oder Klassifikatoren. Letztere können einfacher hinsichtlich ihrer Effizienz und Zuverlässigkeit bewertet werden. Um den in dieser Arbeit vorgestellten EASI-Ansatz hinsichtlich seiner Anwendbarkeit zu evaluieren, wurden zuerst stärker theoretisch geprägte Überlegungen bezüglich der Systemkomplexität und Wartbarkeit angestellt. Es wurde gezeigt, dass ein zentral koordiniertes System, dessen Elemente jedoch nicht zwingend kontrolliert werden, ein hohes Maß an Modularität und Flexibilität in sich birgt. In Vergleichen mit der vorherigen und bereits modular ausgerichteten SIRCLE-Architektur konnten gegenseitige Abhängigkeiten der Komponenten weiter reduziert werden. Auch die Restrukturierung der ursprünglichen und stark miteinander verknüpften Transitionen eines endlichen Automaten in eine möglichst disjunkte Regelbasis ohne Rekursionen erleichtert die kontinuierliche Modifikation eines Forschungssystems erheblich.

Letztendlich muss sich eine Systemarchitektur aber immer in ihrer eigentlichen Bestimmung beweisen, der Funktionalität des integrierten Systems im Einsatz. Hierzu wurden sowohl das praktische Vorgehen bei der Integration neuer Komponenten in das bestehende System geschildert, als auch die Robustheit in Anwenderstudien. Die Systemzuverlässigkeit blieb bestehen, selbst wenn sich einzelne Komponenten noch in einem weniger stabilen Zustand befanden. Dies ermöglicht somit die parallele Weiterentwicklung von Systemfähigkeiten ohne längere Ausfallzeiten des gesamten Forschungsdemonstrators.

Diese Ergebnisse und die externen positiven Beurteilungen ermutigten zur Verwendung des EASI-Ansatzes in weiteren Forschungsprojekten und betonen die Bedeutung der hier vorgestellten Integrationsarbeit für das jeweilige Gesamtsystem. Somit wird die Architektur auch weiterhin auf unterschiedlichen Systemen und in deren Anwendungsszenarien erfolgreich zum Einsatz kommen und ihre Anpassungsfähigkeit überzeugend demonstrieren.

8. Zusammenfassung und Ausblick

„Es gibt menschliche Funktionen, die nicht durch Computer ersetzt werden sollten.“

(Joseph Weizenbaum)

Zusammenfassung

Auch bei der Entwicklung von Service-Robotern sollte die oben genannte Aussage nie außer Beachtung gelassen werden. Dennoch besteht in einer alternden Gesellschaft das Bedürfnis nach Strategien, um die Lebensqualität hinsichtlich einer selbstbestimmten Unabhängigkeit im Alltag lange zu erhalten. Einen Weg zur Realisierung bieten Serviceroboter, die Menschen bei Aufgaben unterstützen, die ein Mensch auf Grund körperlicher oder geistiger Einschränkungen selbst nur noch bedingt ausführen kann. Als ersten Schritt in diese Zukunft müssen aktuelle Roboter in einer auf den Menschen angepassten Umwelt agieren können. Lernen und Adaptivität zur Laufzeit sind hierfür unumgänglich, da ein Vorprogrammieren aller eventuell eintretender Ereignisse unmöglich ist. Domänenspezifisches Wissen, wie Objektbezeichnungen oder Aktuatoransteuerung, lassen sich bislang deutlich einfacher über die Interaktion mit dem Anwender vermitteln, als das Zusammenspiel einzelner Fähigkeiten. Diese Art der Systemerweiterung wird in der Regel noch durch menschliche Entwickler im Rahmen der Systemintegration realisiert. Doch bereits dieser Prozess birgt Herausforderungen, welchen auf unterschiedliche Weise begegnet werden kann.

Es ist die feste Überzeugung des Autors, dass es nicht die allgemeingültige, ultimative Lösung für die Integration von Fähigkeiten in Systemen, konkreter von Softwarekomponenten in einen Roboter, gibt. Vielmehr existieren vielfältige Ansätze für eine entsprechend hohe Anzahl konkreter Projekte. Nach einem Vergleich dieser Ansätze zeigte sich jedoch, dass sie sich nur unzulänglich nach den Ansprüchen einer sich in kontinuierlicher Entwicklung befindlichen Roboterplattform richten. Wiederverwendbarkeit und Modularisierung werden häufig als Eigenschaften oder zumindest Ziele benannt und auf Komponentenebene durchaus realisiert. Doch die Flexibilität der Architektur selbst wird hinsichtlich des gewünschten Systemverhaltens, alternativer Kommunikationsmodalitäten und des Prozessablaufes der zu integrierenden Komponenten vernachlässigt. Besonders bei der Kooperation unterschiedlicher Institute und gegenseitigem Austausch existierender Komponenten können derartige Eigenschaften bei einer starren Architektur nur schwer auf einen gemeinsamen Nenner gebracht werden.

Die Flexibilitätsansprüche gelten auch für den Bereich aktiver Forschung innerhalb eines Institutes. Ergebnisse können von den Erwartungen abweichen oder sich sogar in das Gegenteil umkehren. Wie soll in so einem Fall mit der ursprünglichen Systemarchitektur und deren Integration verfahren werden? Den alten Plan verfolgend die neuen Ergebnisse „relativieren“ oder alles bisherige zeitaufwendig überarbeiten? Weitaus sinnvoller ist es, derartige Möglichkeiten bereits bei der Entwickelung der Systemarchitektur zu berücksichtigen und die Integrationsverfahren gegenüber unerwarteten Modifikationen zu öffnen, ohne hilfreiche Integrationsstrukturen aufzulösen und in einem unübersichtlichen Software-Gewebe zu enden. Genau diesen Aspekten wurde, im Gegensatz zu vielen alternativen Ansätzen, gleich von Beginn dieser Arbeit an Rechnung getragen.

Als Beispiel für eine evolutionäre Systemintegration hinsichtlich der Mensch-Roboter-Interaktion wurde in dieser Arbeit der Bielefeld Robot Companion BIRON genannt. Der Roboter wird seit dem Jahr 2001 kontinuierlich an der Universität Bielefeld weiterentwickelt und seit diesem Zeitpunkt für unterschiedliche Projekte eingesetzt. Dabei ist es das Ziel, dessen Fähigkeiten kontinuierlich zu verbessern und von einem Projekt in das nächste zu übernehmen, statt mit jedem Projekt nahezu neu zu beginnen, wie dies wegen mangelnder Systemflexibilität leider häufig der Fall ist. Die Interaktion in Alltagsumgebungen, wie einer echten Wohnung, haben sich gleichermaßen sowohl als Herausforderung, aber auch als Quelle neuer Erkenntnisse erwiesen, die auf Grund der hier vorgestellten und sich weiterentwickelnden Systemarchitektur zeitnah zu integrieren waren. Diese Entwicklungsfähigkeit begünstigte auch die einfache Portierung auf die in Gestalt und Anwendungsszenario stark abweichende Plattform Barthoc, so dass innerhalb einer Architekturumgebung für beide Systeme entwickelt wird und Software-Ressourcen von beiden Demonstratoren gemeinsam genutzt werden.

Grundlage dieser gemeinsamen evolutionären Architektur zur Systemintegration ist die Kombination eines bereits existierenden Event-basierten Datenaustausches über das ActiveMemory mit dem, in dieser Arbeit entwickelten ACMI als zentrale Koordination. Die Zielsetzung war eine möglichst einfach anwendbare Systemintegration zu realisieren, die gleichermaßen die Wartbarkeit und Langlebigkeit des Ausgangssystems unterstützt. Hierzu wurde die bereits erfolgreiche Kapselung der Einzelkomponenten in ein modularisiertes System weiter vorangetrieben. Verbindungen semantischer oder syntaktischer Art zwischen einzelnen Modulen sind so weit wie möglich durch den Informationsaustausch mit dem ActiveMemory ersetzt worden. Die einzelnen Systemkomponenten sind für einander nicht länger gegenseitig sichtbar, was den Austausch oder die Modifikation selbst zur Laufzeit erheblich erleichtert. Notwendige Schnittstellenanpassungen werden über das ACMI realisiert. Dies gilt gleichermaßen sowohl für die Art des Datenaustausches, als auch für die Datenstruktur an sich.

Die Kapselung der Komponenten wird auch auf Ebene des Systemverhaltens konsequent weiterverfolgt. Zwar wird für die Bearbeitung komplexer Aufgaben das Zusammenspiel mehrere Komponenten benötigt, doch findet die Koordination mittels ACMI über das ActiveMemory und unbemerkt von den beteiligten Komponenten statt. Diese Koordination, die sich in Arbitrierung und Sequenzierung widerspiegelt, wird flexibel über Regeln definiert, die selber als Daten innerhalb des Memorys repräsentiert sind. Für den Entwickler wird die Arbeit mit den Regeln

zur Systemkoordination durch einen Editor erleichtert, der gleichzeitig zur Systemüberwachung verwendet wird. Außerdem wurde bereits beim Design der Regelstruktur darauf geachtet, dass Rekursionen oder Nebenläufigkeiten vermieden werden, sofern es der Entwickler nicht explizit wünscht.

Wie in der anschließenden Analyse und Diskussion der Ergebnisse gezeigt, bleibt die Regelbasis hierdurch übersichtlicher, verglichen mit der ursprünglichen Implementation. Obwohl bereits die SIRCLE-Architektur neue Maßstäbe in Bezug auf Modularität und Systemerweiterbarkeit setzte, wurden die Abhängigkeiten durch den hier gezeigten EASI-Ansatz in deutlichem Maße gesenkt und die Flexibilität des Systems hinsichtlich unerwarteter Modifikationen oder Erweiterungen signifikant gesteigert. Letztendlich muss sich jedoch jedes System im Einsatz beweisen. Dies wurde sehr erfolgreich auf zwei Ebenen umgesetzt: zum einen hinsichtlich der Integration fremder Software von Projektpartnern alternativer Forschungseinrichtungen und zum anderen durch Interaktionsstudien in realen Anwendungsumgebungen mit den Demonstratoren BIRON und Barthoc. Die positiven Beurteilungen im Rahmen des Integrierten EU-Projektes COGNIRON zeigen die auch von externer, wissenschaftlicher Seite bestätigte hohe Relevanz der hier geleisteten Arbeit als Teil einer internationalen Kooperation. Die Reviewer betonen die Bedeutung von Systemarchitektur und implementierter Integration nicht nur als Ingenieursleistung, sondern auch als wissenschaftlichen Beitrag.

Ausblick

Als Instrument einer kontinuierlichen Systementwicklung im Forschungsumfeld stellt natürlich auch der hier präsentierte Ansatz den Ausgangspunkt weiterer Entwicklungen dar. Es ist das Ziel und die Hoffnung des Autors mit den hier präsentierten Ergebnissen eine Grundlage für weitere Forschungsfragen zu legen. Auch wenn das hier präsentierte System bereits interaktiv Informationen speichern kann, so ist das Lernen von Verhaltensweisen oder sogar neuen Fähigkeiten weiterhin ein ungelöstes Problem. Dieses muss für autonome Roboter, die sich als wirkliche Unterstützung in unserem Alltag beweisen sollen, jedoch noch gelöst werden.

Als ersten Schritt in diese Richtung können die Verhaltensregeln der präsentierten Architektur bereits jetzt von beliebiger Seite im Memory zur Laufzeit modifiziert werden. Die Zusammenfassung in Hierarchien von Regelsystemen nach [Ish90], die kontextsensitiv dynamisch zur Laufzeit geladen werden, weist einen ersten Schritt in diese Richtung. Mit Bezug auf biologische Lernbeispiele ist sicherlich auch eine stärkere Einbeziehung der zeitlichen Korrelation in die Verhaltensregeln zu untersuchen. So ließen sich auch Einzelinformationen, wie aktueller Aufenthaltsort und kürzlich detektierte Objekte, automatisch miteinander assoziieren und deduktiv Informationen über den Objektort ableiten. Dieses komponentenübergreifende Lernen bildet die Voraussetzung für eine wirklich kognitive Architektur nach biologischem Vorbild [Goe07]. Die Repräsentation von Regeln als bewusster, prozeduraler Wissensspeicher im Vergleich zu beispielbasiertem Wissensgewinn über Support-Vector-Machines oder neuronale Netze mag dabei nur eine alternative Implementation sein, wird jedoch durch Ergebnisse aus der

Verhaltensforschung gestützt [Fum03]. Die Entwicklung eines geeignetes Lernverfahrens, um diese Regeln auf Memory-Inhalten basierend aufzubauen, oder um eine gegebene Regelbasis zu modifizieren, wird eine der nächste Herausforderungen sein, die es zu meistern gilt.

A. Systemimplementierung und -konfiguration

A.1. ACMI Regelbasis - BIRON

```
<CONFIG>
  <ACMI>
    <MODULES>
      <MODULE name="HWC" com="MI"/>
    </MODULES>
    <MEMORY>
      <XCFPREFIX name="xcf:"/>
      <DATABASE name="ShortTerm"/>
      <DATABASE name="Scene"/>
      <DATABASE name="LongTerm"/>
    </MEMORY>
    <TRIGGERS>
      <TRIGGER>
        <GUARD database="Scene" xpath="/SITUATION[@value='follow']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
        <CONDITION database="Scene" xpath="/IAP[GENERATOR='DLG']" action="INSERT"/>
        <ACTION name="REPLACE">
          <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
                  node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
          <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
        </ACTION>
      </TRIGGER>
      <TRIGGER>
        <GUARD database="Scene" xpath="/SITUATION[@value='follow']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
        <CONDITION database="Scene" xpath="/IAP[GENERATOR='DLG']" action="INSERT"/>
        <ACTION name="CONFIGURE">
          <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
          <TARGET module="HWC" database="ShortTerm"/>
        </ACTION>
      </TRIGGER>
      <TRIGGER>
        <GUARD database="Scene" xpath="/SITUATION[@value='follow']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
        <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
        <CONDITION database="Scene" xpath="/IAP" action="REMOVE"/>
        <ACTION name="REPLACE">
          <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
                  node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
          <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
        </ACTION>
      </TRIGGER>
      <TRIGGER>
        <GUARD database="Scene" xpath="/SITUATION[@value='follow']" exists="no" />
```

```
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="Scene" xpath="/IAP" action="REMOVE"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/ALERTNESS/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='mutexed']" action="REPLACE"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/FOLLOW/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
    <AFFIRM value="accepted"/>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='initiated']" action="INSERT"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='follow']"
              node="/CONFIG/SYSTEM/SITUATION[@value='follow']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
    <AFFIRM value="mutexed"/>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='completed']" action="REPLACE"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='completed']" action="REPLACE"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
              node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/IAP" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='failed']" action="REPLACE"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/ALERTNESS/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/IAP" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='failed']" action="REPLACE"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
```

```
            node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/IAP" exists="yes" />
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='failed']" action="REPLACE"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/IAP" exists="yes" />
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/FOLLOW/STATUS[@value='failed']" action="REPLACE"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
            node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/STOPCURRENTACTION" action="INSERT"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
            node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="ShortTerm" xpath="/STOPCURRENTACTION" action="INSERT"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
    <AFFIRM value="completed"/>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']" action="INSERT"/>
    <ACTION name="CONFIGURE">
      <SOURCE database="LongTerm" xpath="/CONFIG/HWC/LOCATING/MOVESRC"/>
      <TARGET module="HWC" database="ShortTerm"/>
    </ACTION>
    <AFFIRM value="accepted"/>
  </TRIGGER>
  <TRIGGER>
    <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
    <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
    <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']/STATUS[@value='accepted']"
          action="REPLACE"/>
    <ACTION name="REPLACE">
      <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='localize']"
            node="/CONFIG/SYSTEM/SITUATION[@value='localize']"/>
      <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
    </ACTION>
```

```
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
  <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']/STATUS[@value='initiated']"
             action="REPLACE"/>
  <ACTION name="CONFIGURE">
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/LOCATING/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
  <AFFIRM value="accepted"/>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
  <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
  <CONDITION database="Scene" xpath="/OBJECT[GENERATOR='DLG']/STATUS[@value='initiated']"
             action="INSERT"/>
  <ACTION name="CONFIGURE">
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/OBJECT/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
  <AFFIRM value="accepted"/>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
  <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
  <CONDITION database="Scene" xpath="/OBJECT[GENERATOR='DLG']/STATUS[@value='accepted']"
             action="REPLACE"/>
  <ACTION name="REPLACE">
    <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='object']"
            node="/CONFIG/SYSTEM/SITUATION[@value='object']"/>
    <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="no" />
  <GUARD database="Scene" xpath="/SITUATION[@value='localize']" exists="no" />
  <CONDITION database="Scene" xpath="/OBJECT[GENERATOR='DLG']/STATUS[@value='initiated']"
             action="REPLACE"/>
  <ACTION name="CONFIGURE">
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/OBJECT/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
  <AFFIRM value="accepted"/>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/REGION/STATUS[@value='completed']" exists="no" />
  <GUARD database="Scene" xpath="/REGION/STATUS[@value='failed']" exists="no" />
  <CONDITION database="Scene" xpath="/REGION[GENERATOR='DLG']" action="INSERT"/>
  <ACTION name="ASSERT">
    <TARGET database="Scene" xpath="/REGION[GENERATOR='DLG']" status="completed"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/REGION/STATUS[@value='completed']" exists="no" />
  <GUARD database="Scene" xpath="/REGION/STATUS[@value='failed']" exists="no" />
  <CONDITION database="Scene" xpath="/REGION[GENERATOR='DLG']" action="REPLACE"/>
  <ACTION name="ASSERT">
    <TARGET database="Scene" xpath="/REGION[GENERATOR='DLG']" status="completed" />
  </ACTION>
</TRIGGER>
<TRIGGER>
  <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']/STATUS[@value='completed']"
             action="REPLACE"/>
  <ACTION name="CONFIGURE">
```

```
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/ALERTNESS/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/IAP" exists="no" />
  <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']/STATUS[@value='completed']"
             action="REPLACE"/>
  <ACTION name="REPLACE">
    <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
            node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
    <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/IAP" exists="yes" />
  <CONDITION database="Scene" xpath="/LOCATION[GENERATOR='DLG']/STATUS[@value='completed']"
             action="REPLACE"/>
  <ACTION name="REPLACE">
    <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
            node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
    <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
  <GUARD database="Scene" xpath="/IAP" exists="no" />
  <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='failed']" action="REPLACE"/>
  <ACTION name="CONFIGURE">
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/ALERTNESS/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
  <GUARD database="Scene" xpath="/IAP" exists="no" />
  <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='failed']" action="REPLACE"/>
  <ACTION name="REPLACE">
    <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
            node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
    <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
  <GUARD database="Scene" xpath="/IAP" exists="yes" />
  <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='failed']" action="REPLACE"/>
  <ACTION name="CONFIGURE">
    <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
    <TARGET module="HWC" database="ShortTerm"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
  <GUARD database="Scene" xpath="/IAP" exists="yes" />
  <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='failed']" action="REPLACE"/>
  <ACTION name="REPLACE">
    <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
            node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
    <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
  </ACTION>
</TRIGGER>
<TRIGGER>
  <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
  <GUARD database="Scene" xpath="/IAP" exists="no" />
```

```
      <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='completed']" action="REPLACE"/>
      <ACTION name="CONFIGURE">
        <SOURCE database="LongTerm" xpath="/CONFIG/HWC/ALERTNESS/MOVESRC"/>
        <TARGET module="HWC" database="ShortTerm"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
      <GUARD database="Scene" xpath="/IAP" exists="no" />
      <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='completed']" action="REPLACE"/>
      <ACTION name="REPLACE">
        <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
                node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
        <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
      <GUARD database="Scene" xpath="/IAP" exists="yes" />
      <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='completed']" action="REPLACE"/>
      <ACTION name="CONFIGURE">
        <SOURCE database="LongTerm" xpath="/CONFIG/HWC/PERSON/MOVESRC"/>
        <TARGET module="HWC" database="ShortTerm"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <GUARD database="Scene" xpath="/SITUATION[@value='object']" exists="yes" />
      <GUARD database="Scene" xpath="/IAP" exists="yes" />
      <CONDITION database="Scene" xpath="/OBJECT/STATUS[@value='completed']" action="REPLACE"/>
      <ACTION name="REPLACE">
        <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='person']"
                node="/CONFIG/SYSTEM/SITUATION[@value='person']"/>
        <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <CONDITION database="ShortTerm" xpath="/SYSTEMRESET" action="INSERT"/>
      <ACTION name="REMOVE">
        <TARGET database="Scene" xpath="/"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <CONDITION database="ShortTerm" xpath="/SYSTEMRESET" action="INSERT"/>
      <ACTION name="REMOVE">
        <TARGET database="ShortTerm" xpath="/"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <CONDITION database="ShortTerm" xpath="/SYSTEMRESET" action="REMOVE"/>
      <ACTION name="REPLACE">
        <SOURCE database="LongTerm" xpath="/CONFIG/SYSTEM/SITUATION[@value='alertness']"
                node="/CONFIG/SYSTEM/SITUATION[@value='alertness']"/>
        <TARGET database="Scene" xpath="/SITUATION" node="/SITUATION"/>
      </ACTION>
    </TRIGGER>
    <TRIGGER>
      <CONDITION database="ShortTerm" xpath="/SYSTEMRESET" action="INSERT"/>
      <ACTION name="CONFIGURE">
        <SOURCE database="LongTerm" xpath="/CONFIG/HWC/SYSTEMRESET/MOVESRC"/>
        <TARGET module="HWC" database="ShortTerm"/>
      </ACTION>
    </TRIGGER>
  </TRIGGERS>
</ACMI>
<HWC>
```

```
    <FOLLOW>
      <MOVESRC cam="PTA" base="Following"/>
    </FOLLOW>
    <SYSTEMRESET>
      <MOVESRC cam="PTA" base="PTA"/>
    </SYSTEMRESET>
    <PERSON>
      <MOVESRC cam="PTA" base="PTA"/>
    </PERSON>
    <ALERTNESS>
      <MOVESRC cam="PTA" base="PTA"/>
    </ALERTNESS>
    <GOTO>
      <MOVESRC cam="PTA" base="NAV"/>
    </GOTO>
    <LOCATING>
      <MOVESRC cam="PTA" base="TOP"/>
    </LOCATING>
    <OBJECT>
      <MOVESRC cam="OAS" base="OAS"/>
    </OBJECT>
  </HWC>
  <SYSTEM>
    <SITUATION value="person">
      <GENERATOR>ACMI</GENERATOR>
      <TIMESTAMP>
        <INSERTED value="" />
        <UPDATED value="" />
      </TIMESTAMP>
    </SITUATION>
    <SITUATION value="follow">
      <GENERATOR>ACMI</GENERATOR>
      <TIMESTAMP>
        <INSERTED value="" />
        <UPDATED value="" />
      </TIMESTAMP>
    </SITUATION>
    <SITUATION value="alertness">
      <GENERATOR>ACMI</GENERATOR>
      <TIMESTAMP>
        <INSERTED value="" />
        <UPDATED value="" />
      </TIMESTAMP>
    </SITUATION>
    <SITUATION value="localize">
      <GENERATOR>ACMI</GENERATOR>
      <TIMESTAMP>
        <INSERTED value="" />
        <UPDATED value="" />
      </TIMESTAMP>
    </SITUATION>
    <SITUATION value="object">
      <GENERATOR>ACMI</GENERATOR>
      <TIMESTAMP>
        <INSERTED value="" />
        <UPDATED value="" />
      </TIMESTAMP>
    </SITUATION>
  </SYSTEM>
</CONFIG>
```

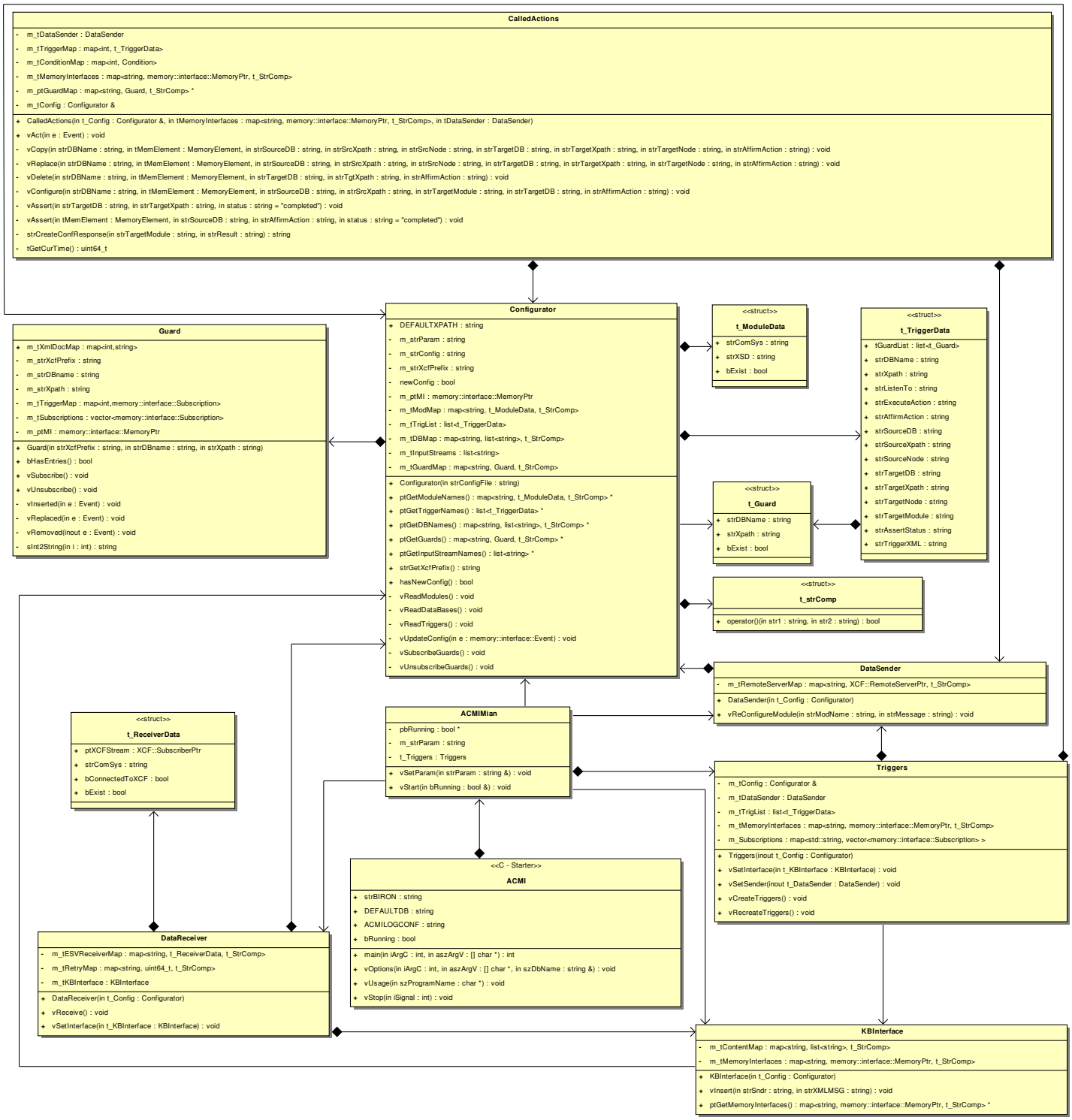
A.2. ACMI Regelbasis - Barthoc

```
<CONFIG>
  <ACMI>
    <MEMORY>
      <XCFPREFIX name="xcf:" />
      <DATABASE name="ShortTerm" />
      <DATABASE name="Scene" />
      <DATABASE name="LongTerm" />
    </MEMORY>
    <TRIGGERS>
      <TRIGGER>
        <CONDITION action="INSERT" database="ShortTerm" xpath="/SYSTEMRESET" />
        <ACTION name="REMOVE">
          <TARGET database="ShortTerm" xpath="/" />
        </ACTION>
      </TRIGGER>
      <TRIGGER>
        <GUARD database="ShortTerm" exists="no"
               xpath="/POSITIONQUERY/STATUS[@value='accepted']" />
        <CONDITION action="INSERT" database="ShortTerm"
                   xpath="/POSITIONQUERY/STATUS[@value='initiated']" />
        <ACTION name="ASSERT">
          <TARGET database="ShortTerm" status="accepted"
                  xpath="/POSITIONQUERY/STATUS[@value='initiated']" />
        </ACTION>
      </TRIGGER>
    </TRIGGERS>
  </ACMI>
</CONFIG>
```

A.3. ACMI Regelbasis - ARbInI

```
<CONFIG>
  <ACMI>
    <MEMORY>
      <XCFPREFIX name="xcf:"/>
      <DATABASE name="newTest">
        <INPUTSTREAM name="LAFORGE1_PICARD"/>
        <INPUTSTREAM name="LAFORGE2_PICARD"/>
        <INPUTSTREAM name="ARTCOORDS"/>
        <INPUTSTREAM name="ARTCOORDS_1"/>
        <INPUTSTREAM name="ARTCOORDS_2"/>
        <INPUTSTREAM name="MT9Sensor"/>
        <INPUTSTREAM name="MT9Sensor1"/>
        <INPUTSTREAM name="MT9Sensor2"/>
        <INPUTSTREAM name="SCOTTY"/>
        <INPUTSTREAM name="viconRT"/>
        <INPUTSTREAM name="Nexus"/>
        <INPUTSTREAM name="WiiData1"/>
        <INPUTSTREAM name="WiiData2"/>
      </DATABASE>
    </MEMORY>
  </ACMI>
</CONFIG>
```

A.4. ACMI Klassendiagramm



A.5. TAME Spezifikation

```
<TRIGGER location="/CONFIG/ACMI/TRIGGERS">
  <GUARD tmt_count="0" database="Scene,LongTerm,ShortTerm" exists="no,yes" xpath=""/>
  <CONDITION tmt_count="1" action="INSERT,REMOVE,REPLACE,QUERY,ALL"
             database="Scene,LongTerm,ShortTerm" xpath=""/>
  <ACTION tmt_count="1" name="ASSERT,CONFIGURE,REMOVE,REPLACE,COPY">
    <SOURCE tmt_count="-1" database="Scene,LongTerm,ShortTerm" xpath="" node=""/>
    <TARGET tmt_count="1" database="Scene,LongTerm,ShortTerm" module="" node="" xpath=""/>
  </ACTION>
  <AFFIRM tmt_count="-1" value="initiated,accepted,rejected,completed,failed" />
</TRIGGER>
```

Literaturverzeichnis

[Ala98] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand: *An Architecture for Autonomy*, *The International Journal of Robotics Research*, Bd. 17, Nr. 4, 1998, S. 315–337.

[And93] J. R. Anderson: *Rules of the Mind*, Lawrence Erlbaum Associates, Inc., 1993.

[Ark98] R. C. Arkin: *Behavior-Based Robotics*, MIT Press, Cambridge, MA, 1998.

[Bas03] L. Bass, P. Clements, R. Kazman: *Software Architecture in Practice*, Bd. 2, Addison-Wesley Professional, April 2003.

[Ben04] D. P. Benjamin, D. Lyons, D. Lonsdale: *ADAPT: A cognitive architecture for robotics*, in *Proceedings of the International Conference of Cognitive Modeling*, 2004.

[Ben05] M. Bennewitz, F. Faber, D. Joho, M. Schreiber, S. Behnke: *Towards a Humanoid Museum Guide Robot that Interacts with Multiple Persons*, in *Proceedings of 2005 5th IEEE-RAS International Conference on Humanoid Robots*, 2005.

[Ben07] M. Bennewitz, F. Faber, D. Joho, S. Behnke: *Fritz - A Humanoid Communication Robot*, in *The 16th IEEE International Symposium on Robot and Human interactive Communication, 2007. RO-MAN 2007.*, 2007, S. 1072–1077.

[Beu08] N. Beuter, T. Spexard, I. Lütkebohle, J. Peltason, F. Kummert: *Where is this? - Gesture Based Multimodal Interaction With An Anthropomorphic Robot*, in *International Conference on Humanoid Robots*, IEEE-RAS, IEEE-RAS, Daejeon, Korea, 2008.

[Boo08] O. Booij, B. Kröse, J. Peltason, ThorstenSpexard, M. Hanheide: *Moving from augmented to interactive mapping*, in *Proceedings of the Robotics: Science and Systems Conference*, Zurich, 2008.

[Bre05] C. Brelsford: *A Machine with a Personal Touch*, *password - Philips Research technology magazine*, Bd. 23, May 2005, S. 14–15.

[Bro86] R. Brooks: *A robust layered control system for a mobile robot*, *Robotics and Automation*, Bd. 2, Nr. 1, March 1986, S. 14–23.

[Bro90] R. A. Brooks: *Elephants Don t Play Chess*, *Elephants Don´t Play Chess*, Bd. 6, Nr. 1-2, June 1990, S. 3–5.

[Bro06] A. G. Brooks, C. Breazeal: *Working with robots and objects: revisiting deictic reference for achieving spatial common ground*, in *Proceedings of the 1st ACM SIGCHI/-SIGART Conference on Human-Robot Interaction, HRI*, Salt Lake City, Utah, USA, March 2006, S. 297–304.

[Bry00] J. Bryson: *Cross-Paradigm Analysis of Autonomous Agent Architecture*, *Journal of Experimental and Theoretical Artificial Intelligence*, Bd. 12, Nr. 2, 2000, S. 165–190.

[Bul05] P. Bull, R. Limb, R. Payne: *Intelligent spaces: the application of pervasive ICT*, Kap. Pervasive Home environments, Birkhäuser, 2005, S. 81–92.

[Bur98] W. Burgard, A. Cremers, D. Fox, D. Hänel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun: *The Interactive Museum Tour-Guide Robot*, in *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[Cab99] G. Cabri, L. Leonardi, G. Reggiani, F. Zambonelli: *Design and implementation of a programmable coordinationarchitecture for mobile agents*, in *Proceedings of Technology of Object-Oriented Languages and Systems*, Nancy, France, 1999, S. 10–19.

[Cañ05] J. M. Cañas, V. M. Olivera: *Integrating behaviors for mobile robots: an ethological approach*, in V. Kordic (Hrsg.): *Cutting Edge Robotics*, Pro Literature Verlag / ARS, 2005, S. 311–330.

[Cañ07] J. M. Cañas, J. Ruíz-Ayúcar, C. Agüero, F. Martín: *Jde-neoc: Component Oriented Software Architecture for Robotics*, *Journal of Physical Agents*, Bd. 1, Nr. 1, September 2007, S. 1–6.

[Cer06] A. Ceravola, F. Joublin, M. Dunn, J. Eggert, C. Goerick: *Integrated Research and Development Environment for Real-Time Distributed Embodied Intelligent Systems*, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE Press, Bejing, 2006.

[Cla99] A. Clark, R. Grush: *Towards a cognitive robotics*, *Adapt. Behav.*, Bd. 7, Nr. 1, 1999, S. 5–16.

[Clo06] A. Clodic, S. Fleury, R. Alami, R. Chatila, G. Bailly, M. Brethes, L.and Cottret, P. Danes, X. Dollat, F. Elisei, I. Ferrane, M. Herrb, G. Infantes, C. Lemaire, F. Lerasle, J. Manhes, P. Marcoul, P. Menezes, V. Montreuil: *Rackham: An Interactive Robot-Guide*, in *In Proc. of the 15th IEEE International Symposium on Robot and Human Interactive Communication, ROMAN*, Hatfield, September 2006, S. 502–509.

[Coh08] A. Cohn, J. L. Crowley, I. Horswill, M. Beetz: *COGNIRON 4th Year Final Review Report*, Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS), Toulouse, France, April 2008.

[Cor01] S. Coradeschi, A. Saffiotti: *Perceptual anchoring of symbols for action*, in *Proc. of the 17th IJCAI Conference*, Seattle, Washington, 2001, S. 407–412.

[Cro03] J. L. Crowley: *Context driven observation of human activity*, in *Lecture notes in computer science; EUSAI 2003 : European symposium on ambient intelligence*, 2003, S. 101–118.

[Die09] A. Dierker, T. Bovermann, M. Hanheide, T. Hermann, G. Sagerer: *A Multimodal Augmented Reality System for Alignment Research*, in *International Conference on Human-Computer Interaction*, San Diego, USA, 18/07/2009 2009, S. 422–426.

[Ekm72] P. Ekman, W. V. Friesen, P. Ellsworth: *Emotion in the human face: guidelines for research and an integration of findings*, Pergamon Press, New York, 1972.

[End01] M. R. Endsley: *Designing for Situation Awareness in Complex System*, in *Proceedings of the Second intenational workshop on symbiosis of humans, artifacts and environment, K*, Kyoto, Japan, 2001.

[Fab09] F. Faber, M. Bennewitz, C. Eppner, A. Goeroeg, C. Gonsior, D. Joho, M. Schreiber, S. Behnke: *The Humanoid Museum Tour Guide Robotinho*, in *In Proc. of 18th IEEE International Symposium on Robot and Human Interactive Communication (ROMAN)*, Toyama, Japan, September 2009.

[Fou08] E. Foulk: *Robots turn off senior citizens in aging Japan*, reuters.com, September 2008.

[Fri07] J. Fritsch, S. Wrede: *An Integration Framework for Developing Interactive Robots*, Bd. 30 von *Springer Tracts in Advanced Robotics*, Springer, Berlin, 2007.

[Fum03] D. Fum, A. Stocco: *Instance vs. rule based learning in controlling a dynamic system*, in *Proc. of the Fifth International Conference on Cognitive Modelling*, 2003.

[Gat92] E. Gat: *Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots*, in *Proceedings of the National Conference on Artificial Intelligence*, 1992, S. 809–815.

[Gat98] E. Gat, R. P. Bonnasso, R. Murphy, A. Press: *On three-layer architectures*, in *Artificial Intelligence and Mobile Robots*, AAAI Press, Cambridge, 1998, S. 195–210.

[Gep08] A. Gepperth, J. Fritsch, C. Goerick: *Cross-module learning as a first step towards a cognitive system concept*, in *Proceedings of the First International Conference on Cognitive Systems*, 2008.

[Goe07] C. Goerick, E. Körner: *Elements of Cognitive Robotics*, *it - Information Technology*, Bd. 49, 2007, S. 213–217.

[Gol04] D. Goldin, S. Srinivasa, V. Srikanti: *Active Databases as Information Systems*, in *Proc of the International Database Engineering and Applications Symposium*, IEEE, July 2004, S. 123–130.

[Gri08] M. Gritti, A. Saffiotti: *A Self-Configuration Mechanism for Software Components Distributed in an Ecology of Robots*, in *Proceedings of the 10th International Conference on Intelligent Autonomous Systems*, 2008.

[Haa05] A. Haasch, N. Hofemann, J. Fritsch, G. Sagerer: *A Multi-Modal Object Attention System for a Mobile Robot*, in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE, IEEE, Edmonton, Alberta, Canada, August 2005, S. 1499–1504.

[Haa07] A. Haasch: *Attention-controlled Acquisition of a Qualitative Scene Model for Mobile Robots*, Dissertation, Universität Bielefeld, Technische Fakultät, 2007.

[Han06] M. Hanheide: *A Cognitive Ego-Vision System for Interactive Assistance*, phdthesis, Technische Fakultät – Universität Bielefeld, dec 2006.

[Haw08] N. Hawes, J. Wyatt, A. Sloman: *Exploring Design Space For An Integrated Intelligent System*, in *Proceedings of the Twenty-eighth SGAI International Conference on Artificial Intelligence (AI-2008)*, Cambridge, England, 2008.

[Haw09] N. Hawes, J. Wyatt, M. Sridharan, M. Kopicki, S. Hongeng, I. Calvert, A. Sloman, G.-J. M. Kruijff, H. Jacobsson, M. Brenner, D. Skočaj, . A. Vreˇko, N. Majer, M. Zillich: *The PlayMate System*, in H. I. Christensen, A. Sloman, G.-J. M. Kruijff, J. Wyatt (Hrsg.): *Cognitive Systems*, Springer-Verlag, 2009.

[Heg06] F. Hegel, T. Spexard, T. Vogt, G. Horstmann, B. Wrede: *Playing a different imitation game: Interaction with an Empathic Android Robot*, in *Proc. 2006 IEEE-RAS International Conference on Humanoid Robots (HumanoidsÂ´06)*, IEEE, IEEE, December 2006, S. 56–61.

[Heg09] F. Hegel: *Gestalterisch konstruktiver Entwurf eines sozialen Roboters*, Dissertation, Bielefeld University, 2009.

[Hel09] A. Helwart: *Erkennung von Personen anhand ihrer Kleidung und die Anwendung auf dem Roboter BIRON*, Applied Informatics, Bielefeld University, 2009.

[Hof07] N. Hofemann: *Videobasierte Handlungserkennung für die natürliche Mensch-Maschine-Interaktion*, Phd thesis, Bielefeld University, Bielefeld, 2007.

[Hoh03] S. Hohenner, S. Lang, M. Kleinehagenbrock, G. A. Fink, F. Kummert: *Multimodale Sprecherlokalisation für Mensch-Roboter-Interaktionen in einer Multi-Personen-Umgebung*, in K. Kroschel (Hrsg.): *Studientexte zur Sprachkommunikation*, Bd. 28, Karlsruhe, 2003, S. 162–169.

[Ish90] T. Ishida, Y. Makoto, G. Les: *An organizational approach to adaptive production systems*, in *Proc. of the Association for the Advancement of Artificial Intelligence*, Bd. 1, Boston, MA, USA, July 1990, S. 52–58.

[Ish95] T. Ishida, Y. Sasaki, K. Nakata, Y. Fukuhara: *A Meta-Level Control Architecture for Production Systems*, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, Bd. 7, Nr. 1, 1995, S. 4452.

[Ish07] H. Ishiguro: *Scientific Issues Concerning Androids*, *Int. J. Rob. Res.*, Bd. 26, Nr. 1, 2007, S. 105–117.

[Kel06] T. D. Kelley: *Developing a Psychologically Inspired Cognitive Architecture for Robotic Control: The Symbolic and Subsymbolic Robotic Intelligence Control System (SS-RICS)*, *International Journal of Advanced Robotic Systems*, Bd. 3, 2006, S. 219–222.

[Kie95] D. E. Kieras, D. E. Meyer: *Predicting Human Performance in Dual-Task Tracking and Decision Making with Computational Models using the EPIC Architecture*, in *Proceedings of the 1995 International Symposium on Command and Control Research and Technology*, 1995.

[Kle05] M. Kleinehagenbrock: *Interaktive Verhaltenssteuerung für Robot Companions*, Dissertation, Universität Bielefeld, Technische Fakultät, http://bieson.ub.uni-bielefeld.de/volltexte/2005/684/, 2005.

[Lan03] S. Lang, M. Kleinehagenbrock, S. Hohenner, J. Fritsch, G. A. Fink, G. Sagerer: *Providing the Basis for Human-Robot-Interaction: A Multi-Modal Attention System for a Mobile Robot*, in *Proc. Int. Conf. on Multimodal Interfaces*, ACM, ACM, Vancouver, Canada, November 2003, S. 28–35.

[Lan05] S. Lang: *Multimodale Aufmerksamkeitssteuerung fÃ¼r einen mobilen Roboter*, Dissertation, Universität Bielefeld, Technische Fakultät, 2005.

[Leh98] J. F. Lehman, J. E. Laird, P. S. Rosenbloom: *An Invitation to Cognitive Science*, Bd. 4, MIT Press, 1998.

[Li06] S. Li, B. Wrede, G. Sagerer: *A dialog system for comparative user studies on robot verbal behavior*, in *Proc. 15th Int. Symposium on Robot and Human Interactive Communication*, IEEE Press, IEEE Press, Hatfield, United Kingdom, September 2006, S. 129–134.

[Li07] S. Li: *Multi-modal Interaction Management for a Robot Companion*, Phd, Bielefeld University, Bielefeld, 2007.

[Lóp07] R. López, O. Javier: *Multilayered Evolutionary Architecture for Behaviour Arbitration in Cognitive Agents*, *Engineering Letters*, Bd. 15, 2007, S. 24–33.

[Low04] D. G. Lowe: *Distinctive Image Features from Scale-Invariant Keypoints*, *International Journal of Computer Vision*, Bd. 60, Nr. 2, 2004, S. 91–110.

[Lun07] R. Lundh, L. Karlsson, A. Saffiotti: *Dynamic Self-Configuration of an Ecology of Robots*, in *Proc of the IEEE/RSJ Int. Conf on Intelligent Robots and Systems (IROS)*, San Diego, CA, 2007, S. 3403–3409.

[Lüt04] I. Lütkebohle, S. Wrede: *Catwalk. Einsatz der XML-Datenbank Berkeley DB XML*, *iX, Heise*, Bd. 9, 2004, S. 68–73.

[Maa06] J. F. Maas, T. Spexard, J. Fritsch, B. Wrede, G. Sagerer: *BIRON, what's the topic? - A Multi-Modal Topic Tracker for improved Human-Robot Interaction*, in *Proc. IEEE Int. Workshop on Robot and Human Interactive Communication (ROMAN)*, IEEE, IEEE, September 2006, S. 26–32.

[Mäe00] T. Mäenpää, T. Ojala, M. Pietikäinen, S. Maricor: *Robust Texture Classification by Subsets of Local Binary Patterns*, in *Proc. of the 15th International Conference on Pattern Recognition*, 2000, S. 947–950.

[Mal04] M. Malfaz, M. A. Salichs: *A New Architecture for Autonomous Robots based on Emotions*, in *Fifth IFAC Symposium on Intelligent Autonomous Vehicles*, Portugal, Jule 2004.

[Mit06] N. Mitsunaga, T. Miyashita, H. Ishiguro, K. Kogure, N. Hagita: *Robovie-IV: A Communication Robot Interacting with People Daily in an Office*, in *Proceedings of the*

International Conference on Intelligent Robots and Systems, IEEE, Beijing, October 2006, S. 5066–5072.

[Moz07] O. M. Mozos, P. Jensfelt, H. Zender, G. Kruijff, W. Burgard: *From labels to semantics: An integrated system for conceptual spatial representations of indoor environments for mobile robots*, in *In Proceedings of the ICRA-07 Workshop on Semantic Information in Robotics*, Rome, Italy, April 2007.

[Nil82] N. J. Nilsson: *Principles of Artificial Intelligence*, Springer, 1982.

[Nis07] S. Nishio, H. Ishiguro, N. Hagita: *Humanoid Robots: New Developments*, Kap. Geminoid: Teleoperated Android of an Existing Person, I-Tech Education and Publishing, Vienna, Austria, 2007, S. 343 – 352.

[Oh06] J.-H. Oh, D. Hanson, W.-S. Kim, I. Y. Han, J.-Y. Kim, I.-W. Park: *Design of Android type Humanoid Robot Albert HUBO*, in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2006, S. 1428–1433.

[Par08] C. Parlitz, M. H"agele, P. Klein, J. Seifert, K. Dautenhahn: *Care-o-bot 3 - rationale for human-robot interaction design*, in *Proceedings of the International Symposium on Robotics (ISR)*, Seoul, Korea, 2008, S. 275–280.

[Pel09] J. Peltason, F. H. Siepmann, T. P. Spexard, B. Wrede, M. Hanheide, E. A. Topp: *Mixed-Initiative in Human Augmented Mapping*, in *Proceedings of the Int. Conf. on Robotics and Automation*, 2009, to appear.

[Pes07] S. Pestov: *jEdit 4.3pre 12*, Dezember 2007, http://www.jedit.org.

[Pet06] V. Peters: *Effizientes Training ansichtsbasierter Gesichtsdetektoren*, Diplomabeit, Universität Bielefeld, 2006.

[Pet09] A. Peters, T. P. Spexard, P. Weiss, M. Hanheide: *Make room for me - A spatial and situational movement concept in HRI*, in *Workshop on Behavior Monitoring and Interpretation at th 32nd Annual Conf. on Artificial Intelligence*, Paderborn, Germany, 2009.

[Phi06] R. Philippsen, B. Jensen, R. Siegwart: *Autonomous Navigation in Dynamic Environments*, Kap. Towards Real-Time Sensor-Based Path Planning in Highly Dynamic Environments, Springer Tracts on Advanced Robotics, 2006, S. 135–148.

[Pre82] R. Pressman: *Software Engineering: A Practitioner's Approach*, McGraw-Hill Science/Engineering/Math, 1982.

[Pro07] M. Proetzsch, T. Luksch, K. Berns: *The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems*, in *Lecture Notes in Computer Science, KI 2007: Advances in Artificial Intelligence*, 2007.

[Pur06] M. Purver: *CLARIE: Handling Clarification Requests in a Dialogue System*, *Research on Language and Computation*, Bd. 4, Nr. 2-3, October 2006, S. 259–288.

[Qia07] Y. Qiao, K. Zhong, H. Wang, X. Li: *Developing event-condition-action rules in real-time active database*, in *Proc. of the ACM symposium on Applied computing*, ACM, Seoul, Korea, 2007, S. 511 – 516.

[Rod04] K. Rodriguez, D. Schlangen: *Form, Intonation and Function of Clarification Requests in German Task-Oriented Spoken Dialogues*, in *Proceedings of the 8th Workshop on the Semantics and Pragmatics of Dialogue (Catalog)*, Barcelona, 2004, S. 101–108.

[Saf07] A. Saffiotti, M. Broxvall, B. Seo, Y. Cho: *The PEIS-Ecology Project: a progress report*, in *Proc. of the ICRA-07 Workshop on Network Robot Systems*, Rome, Italy, 2007, S. 16–22.

[Sah04] A. Sahara, M. Imai, Y. Anzai: *CAHRA: Collision Avoidance System for Humanoid Robot Arms with Potential Field*, in *International Conference on Systems, Man and Cybernetics*, Bd. 3, October 2004, S. 2889 – 2895.

[Sal06] M. Salichs, R. Barber, M. Khamis, A.M.and Malfaz, J. Gorostiza, R. Pacheco, R. Rivas, A. Corrales, E. Delgado, D. Garcia: *Maggie: A Robotic Platform for Human-Robot Social Interaction*, in *Robotics, Automation and Mechatronics, 2006 IEEE Conference on*, Bangkok, December 2006, S. 1–7.

[San07] C. Santoro: *An erlang framework for autonomous mobile robots*, in *Proc. of the SIGPLAN workshop on ERLANG Workshop*, ACM, Freiburg, Germany, 2007, S. 85 – 92.

[Sch01] R. Schapire: *The boosting approach to machine learning: An overview*, in *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 2001.

[Sch07] F. Schubert, T. Spexard, M. Hanheide, S. Wachsmuth: *Active Vision-based Localization For Robots In A Home-Tour Scenario*, in *Proc. of International Conference on Machine Vision Applications ("International Conference on Computer Vision Systems" 2007)*, Bielefeld, Germany, March 2007, DOI: 10.2390/biecoll-icvs2007-101.

[Shi06] S. Shiotani, T. Tomonaka, K. Kemmotsu, S. Asano, K. Oonishi, R. Hiura: *World's First Full-fledg ed Communication Robot "wakamaru" Capable of Living with Family and Supporting Persons*, 1, Mitsubishi Heavy Industries, Ltd., January 2006.

[Som00] I. Sommerville: *Software Engineering*, Addison Wesley, 6. Ausg., August 2000.

[Sos99] A. Sosio, F. Tisato: *A Design Model for Object Oriented Systems with Explicit Architecture*, in *Technology of Object-Oriented Languages and Systems*, 1999, S. 46 – 55.

[Spe06a] T. Spexard, A. Haasch, J. Fritsch, G. Sagerer: *Human-like Person Tracking with an Anthropomorphic Robot*, in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, IEEE, Orlando, Florida, May 2006, S. 1286–1292.

[Spe06b] T. Spexard, S. Li, B. Wrede, J. Fritsch, G. Sagerer, O. Booij, Z. Zivkovic, B. Terwijn, B. Krï¿½se: *BIRON, where are you? - Enabling a robot to learn new places in a real home environment by integrating spoken dialog and visual localization*, in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE, IEEE, Beijing, P.R. China, October 2006, S. 934–940.

[Spe07a] T. P. Spexard, S. Li, B. Wrede, M. Hanheide, E. A. Topp, H. Hüttenrauch: *Interaction Awareness for Joint Environment Exploration*, in *Proceedings of the Special Session*

on Situation Awareness in Social Robots at the International Symposium on Robot and Human Interactive Communication (RO-MAN), IEEE, IEEE, Jeju Island, Korea, August 2007, S. 546–551.

[Spe07b] T. Spexard, M. Hanheide, G. Sagerer: *Human-Oriented Interaction With an Anthropomorphic Robot*, *IEEE Transactions on Robotics*, Bd. 23, 2007, S. 852–862.

[Spe08a] T. P. Spexard, M. Hanheide, S. Li, B. Wrede: *Oops, Something Is Wrong - Error Detection and Recovery for Advanced Human-Robot-Interaction*, in *Proc. of the Workshop on Social Interaction with Intelligent Indoor Robots at the Int. Conf. on Robotics and Automation*, Pasadena, California, 19/05/2008 2008.

[Spe08b] T. P. Spexard, F. H. K. Siepmann, G. Sagerer: *A Memory-based Software Integration for Development in Autonomous Robotics*, in *International Conference on Intelligent Autonomous Systems*, Baden-Baden, Germany, 23/07/2008 2008, to appear.

[Spe09] T. P. Spexard, M. Hanheide: *System Integration Supporting Evolutionary Development and Design*, in *Proc. of the 3rd int. Workshop on Human Centered Robotic Systems (HCRS)*, Springer, Springer, Bielefeld, Germany, 19/11/2009 2009, to Appear.

[Sun05] G. Sun, B. Scassellati: *A Fast and Efficient Model for Learning to Reach*, *International Journal of Humanoid Robotics*, Bd. 2, 2005, S. 391–413.

[Thr91] S. Thrun, K. Möller, A. Linden: *Planning with an adaptive world model*, in D. Touretzky, R. Lippmann (Hrsg.): *Advances in Neural Information Processing Systems (NIPS)*, Bd. 3, Morgan Kaufmann, San Mateo, CA, 1991, S. 450–456.

[Top08] E. A. Topp: *Human-Robot Interaction and Mapping with a Service Robot: Human Augmented Mapping*, Dissertation, KTH Computer Science and Communication, 2008.

[Tra08] J. G. Trafton, M. D. Bugajska, B. R. Fransen, R. M. Ratwani: *Integrating vision and audition within a cognitive architecture to track conversations*, in *HRI '08: Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, ACM, New York, NY, USA, 2008, S. 201–208.

[Tri07] B. Tribelhorn, Z. Dodds: *Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education*, in *IEEE International Conference on Robotics and Automation*, Roma, Italy, April 2007, S. 1393–1399.

[vB04] A. van Breemen: *Animation engine for believable interactive user-interface robots*, in *Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*, Bd. 3, IEEE, October 2004, S. 2873–2878.

[Vio01] P. Viola, M. Jones: *Rapid object detection using a boosted cascade of simple features*, in *Proc. Conference on Computer Vision and Pattern Recognition*, Kauai, Hawaii, 2001, S. 511–518.

[Wac09] S. Wachsmuth, M. Hanheide, F. Siepmann, T. P. Spexard: *ToBI - Team of Bielefeld: The Human-Robot Interaction System for RoboCup@Home 2009*, Applied Informatics, Bielefeld University, 06/2009 2009.

[Wre06] S. Wrede, M. Hanheide, S. Wachsmuth, G. Sagerer: *Integration and Coordination in a Cognitive Vision System*, in *Proc. of International Conference on Computer Vision Systems*, IEEE, IEEE, St. Johns University, Manhattan, New York City, USA, 2006.

[Wre07] B. Wrede, K. J. Rohlfing, T. P. Spexard, J. Fritsch: *Towards tutoring an interactive robot*, incollection 31, ARS, 2007, S. 601–612.

[Wre09] S. Wrede: *An Information-Driven Architecture for Cognitive Systems Research*, Dissertation, Bielefeld University, 2009.

[Yua08] F. Yuan, M. Hanheide, G. Sagerer: *Spatial Context-Aware Person-Following for a Domestic Robot*, in *Proc. of the International Workshop on Cognition for Technical Systems*, Munich, Germany, 2008.

[Zha02] Z. Zhang, L. Zhu, S. Z. Li, H. Zhang: *Real-Time Multi-View Face Detection*, in *Proc. of the IEEE, Int. Conf. on Automatic Face and Gesture Recognition*, Washington, D.C., 2002, S. 149–154.

[Zie96] C. Zielinski: *Reactive Robot Control Applied to Acquiring Moving Objects*, in *Proc. 3rd International Symposium on Methods and Models in Automation and Robotics*, Bd. 3, Miedzyzdroje, Poland, September 1996, S. 893–898.

Abbildungsverzeichnis

1.1. Entwicklung in der Service-Robotik - Fotos: (a) AP / Topfoto, (b) Fran Miller . . 2
1.2. Care-O-Bot 3 - Foto: Bernd Müller 2

2.1. Deliberative Systemarchitektur - aus [Kle05] 10
2.2. Reaktive Kontrolle - aus [Bro86] 12
2.3. Subsumption-Ansatz - aus [Bro86] 12
2.4. Deliberative Systemarchitektur Hybride Schichtenarchitektur - aus [Kle05] . . . 13
2.5. Verhaltensbasierte Kontrolle - aus [Kle05] 15
2.6. Verhaltenskoordination - aus [Kle05] 16
2.7. Kommerzielle Haushaltshilfen - Fotos: (a) iRobot®, (b, c) Philips 19
2.8. Kommerzielle Multifunktionsplattformen - Fotos:
(a) Mitsubishi Heavy Industries, Ltd. , (b) ATR-Robotics, (c) Fraunhofer IPA . . 20
2.9. Androide Forschungsroboter - Fotos: (a) KAIST HUBO Lab,
(b,c) ATR Intelligent Robotics and Communication Laboratories, (d) AIST Japan 22
2.10. Humanoide Forschungsroboter - aus (a) [Fab09], (b) [Sal06] 25
2.11. Adaptionsfähige Forschungsroboter - aus (a) [Clo06], (b) [Moz07],
(c) Cognitive Systems for Cognitive Assistants, The PlayMate 26

3.1. Grundriss des Home-Tour-Appartements 31
3.2. Interaktionsszenario mit einem humanoiden Roboter - Grafik: Frank Hegel 32
3.3. BIRON Hardware 34
3.4. Barthoc Hardware 36
3.5. Köpfe für Barthoc Junior - Photos: Frank Hegel 37
3.6. Personen-Tracking - Grafik basierend auf [Lan05] 39
3.7. Multimodales Anchoring - Grafik basierend auf [Lan05] 39
3.8. Schichtenmodell des Personen-Tracking - aus [Kle05] 40
3.9. Visualisierung eines Laser-Scans - aus [Lan05] 41
3.10. Akustische Richtungsbestimmung 42
3.11. Gesichts- und Blickrichtungsdetektion 43
3.12. Dialog Grounding 44
3.13. Deiktische Referenzierung des Menschen - (a) aus [Hof07], (b,c) aus [Haa07] . . 45
3.14. Deiktische Referenzierung des Roboters 46
3.15. Evolution der Forschungsszenarien 49

4.1. XCF Kommunikationsbeispiel 55

4.2. Memory-basierter Datenaustausch 57
4.3. FSM zur ursprünglichen Systemkoordination - Grafik: Marcus Kleinehagenbrock 58

5.1. Evolutionäre Architektur zur Systemintegration 61
5.2. Funktionsprinzip des Active Control Memory Interface 64
5.3. Beispiel einer Regel zur Systemkoordination 66
5.4. Beispiel einer Ortsrepräsentation im Ausgangssystem 70
5.5. Beispiel einer aktuellen Ortsrepräsentation 71
5.6. Konfiguration zur Adaption von Kommunikationsmodalitäten 72
5.7. Konfiguration zur Adaption von Datenformaten 73

6.1. Post- und Präaktive Arbitrierung 78
6.2. Beispiel einer postaktiven Arbitrierung 81
6.3. Statusmodell zur Aktionssequenzierung 84
6.4. Trigger & Active Memory Editing (TAME) 85
6.5. TAME Wizzard zur Regeldefinition 86

7.1. Ursprüngliche Kommunikationsstruktur - Grafik: Marcus Kleinehagenbrock . . . 90
7.2. Aufgelockerte Kommunikationsstruktur 91
7.3. Modulabhängigkeiten des Ausgangssystems - Grafik: Marcus Kleinehagenbrock 92
7.4. Reduzierte Modulabhängigkeiten 93
7.5. Aufbau der Regelbasis von BIRON 97
7.6. Benutzerstudie mit BIRON 100
7.7. Öffentliche Interaktion mit Barthoc - Photos: (a-c) Ingo Lütkebohle 101
7.8. RoboCup@Home - Photos: Ingo Lütkebohle 102