Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

AW2 Ausarbeitung

Andreas Dubs

Entwicklung eines impliziten Aktionsplaners

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Motivation

1.1 Roboterorientiert

Eine roboterorientierte Programmierung zeichnet sich durch explizite Bewegungs- und Greiferbefehle aus. Dabei muss der Anwender einen Ablauf für ein gestelltes Problem erarbeiten und legt fest, ***wie*** eine Aufgabe gelöst werden soll. Für Industrielle- und spezielle Serviceroboter reicht es aus, weil die Aufgaben sich nicht mit der Zeit ändern, es können nur die Parameter unterschiedlich sein. Eine Neuplanung einer Aufgabe ist nicht erforderlich, der Ablauf bleibt der selbe. Es entstehen Anwendungen mit roboterspezifischen Datentypen, Funktionen und verbessern die Handhabung eines Roboters. So können die Parameter für bestehende Aufgaben und Bewegungsabläufe leichter eingegeben und verwaltet werden.

Für einen Assistenzroboter ist dieser Ansatz nicht mehr ausreichend. Die gestellten Aufgaben können sich in ihrem Lösungsablauf grundsätzlich unterscheiden. So wird eine Planungseinheit für die Suche eines Lösungsablaufs benötigt. Da es mehrere Abläufe zum Ziel führen könnten, soll nach Möglichkeit ein optimaler Weg gefunden werden.

1.2 Aufgabenorientiert

Das Ziel einer aufgabenorientierten Programmierung ist die Möglichkeit dem Roboter zu sagen, ***was*** getan werden soll, ohne dabei ins Detail zu gehen und explizit eine Lösung vorgeben. Also muss der Roboter eine höhere Abstraktionsebene enthalten und unter Einbeziehung von Sensoren, Weltmodellierung, Aktionsplanung und Optimierung einen möglichst besten Plan erzeugen und ausführen können.

Die Suche nach einem optimalen Aktionsplan kann Schwierigkeiten mit sich bringen, die gelöst werden müssen. Solche Schwierigkeiten können großer Speicherbedarf, enorme Rechenleistung, veraltetes oder unvollständiges Weltmodell aus fehlerhaften Sensordaten und Kollisionen sein, und dadurch zu misslungenen Plänen führen. Diese Aspekte müssen bei der Entwicklung der Planungs- und Steuereinheit berücksichtigt werden.

2 Architekturen

Wie der Aktionsplaner aussehen soll, hängt von der Architektur der Robotersteuerung ab. Nicht nur die Schnittstellen müssen stimmen, sondern auch die Aufbau des Aktionsplaners muss mit dem Konzept der Architektur im Einklang sein

2.1 Klassifikation

Die Architekturen weisen zwei Gegensatzpaare:

- funktionsorientiert (deliberativ) - verhaltensorientiert (reaktiv)
- hierarchisch - verteilt

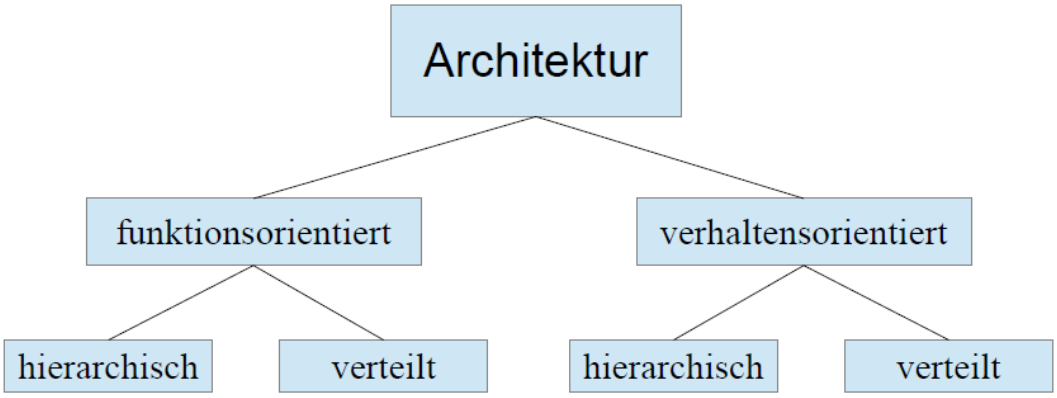

Abbildung 2.1: Arten der Architektur

2.2 Funktionsorientierte (deliberative) Architektur

Für eine funktionsorientierte Architektur ist ein Weltmodell erforderlich. Die Planung von Aktionen findet im Weltmodell statt. Eine vorausschauende Planung erfordert ein genaues Welt-

modell und Kenntnis über alle kausale Konsequenzen einer Aktionen. Die Sensoreingaben führen zu Aktualisierungen im Weltmodell, dadurch kann das Modell während der Ausführung eines Plans veraltet sein und der Plan unter Umständen misslingen. Eine Überprüfung und evtl. Neuplanung während Ausführung der geplanten Aktionen ist daher notwendig.

2.2.1 Hierarchische

Klassische Architekturen sind hierarchische funktionsorientierte Architekturen. Die Hierarchie entsteht durch mehrere Abstraktionsebenen (Abbildung 2.2). Solche Architektur ist für Real-time Robotersysteme gut geeignet.

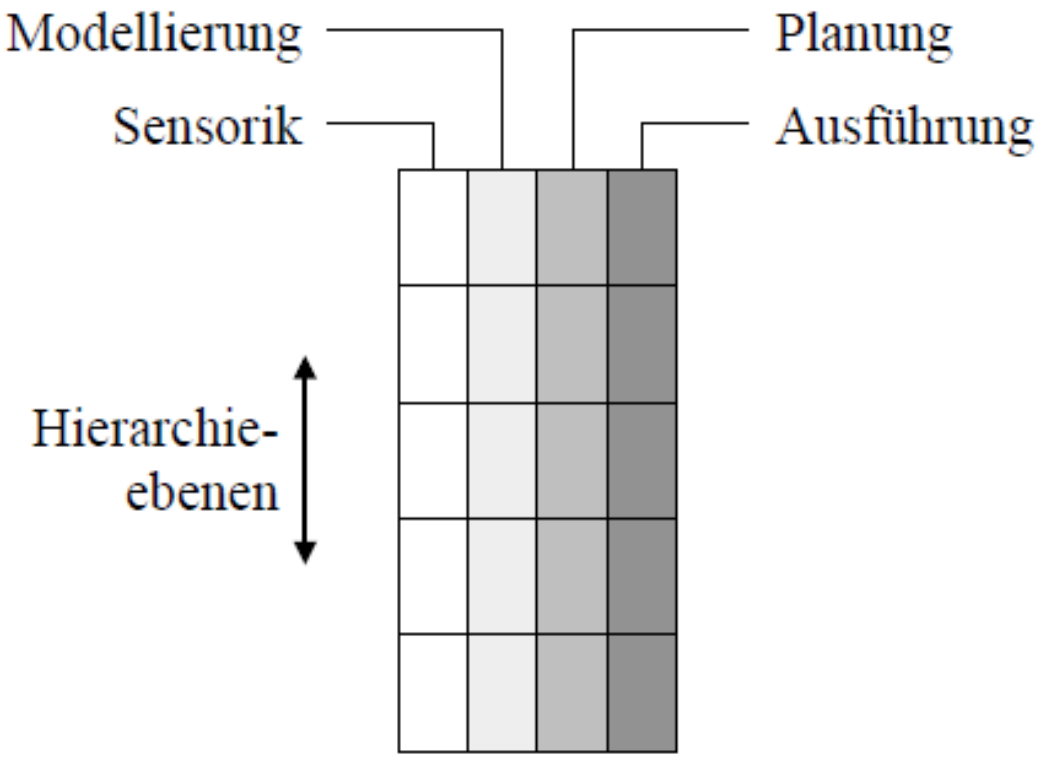

Abbildung 2.2: hierarchische funktionsorientierte Architektur [Zweigle]

2.2.2 Verteilte

Verteilte Architektur zeichnet die Kooperation spezialisierter und unabhängiger Teilsysteme aus (Abbildung 2.3). Für den Datenaustausch zwischen Teilsystemen ist ein Kommunikati-

onsmechanismus notwendig. Damit die Teilsysteme unabhängig sind, wird meist eine asynchrone Kommunikation eingesetzt. Dabei ist oft Real-time Einsatz nicht mehr möglich. Große Anzahl der aktuellen Projekten weisen eine verteilte funktionsorientierte Architektur auf.

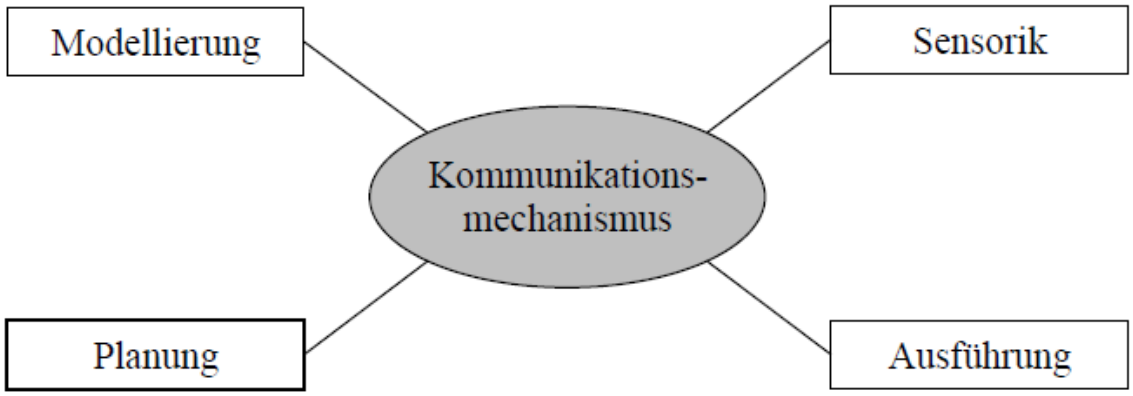

Abbildung 2.3: verteilte funktionsorientierte Architektur [Zweigle]

2.3 Verhaltensorientierte (reaktive) Architektur

Im Gegensatz zu einer funktionsorientierten Architektur braucht die verhaltensorientierte Architektur kein internes Weltmodell. Der Verhalten des Roboters ist durch Regeln spezifiziert. Die Sensoreingaben werden über alle Regeln, die diese brauchen, iteriert und daraus resultiert eine Reaktion des Roboters. Der Hauptvorteil der verhaltensorientierten Architektur ist eine gute Reaktionszeit des System auf äußere Einflüsse. Der Nachteil dieser Architektur besteht in der fehlender Lernfähigkeit und keiner Möglichkeit Langstrategien zu planen.

2.3.1 Hierarchische

Ein klassischer Beispiel für eine hierarchische verhaltensorientierte Architektur ist die Subsumptionsarchitektur von R. A. Brooks. Sie wurde an Massachusetts Institute of Technology entwickelt und die Regeln sind in verschiedenen Verhaltensebenen angeordnet. Weil die Sensoreingaben von verschiedenen Regeln verarbeitet werden, können Konflikte zwischen Reaktionsausgaben entstehen. Die höhere Kompetenzebenen überschreiben die Ausgaben von niedrigeren Ebenen (Abbildung 2.4) und lösen somit die Konflikte. Die Subsumptionsarchitektur eignet sich hervorragend für reflexive Systeme, die Aktionsplanung ist dagegen sehr schwer oder gar nicht realisierbar.

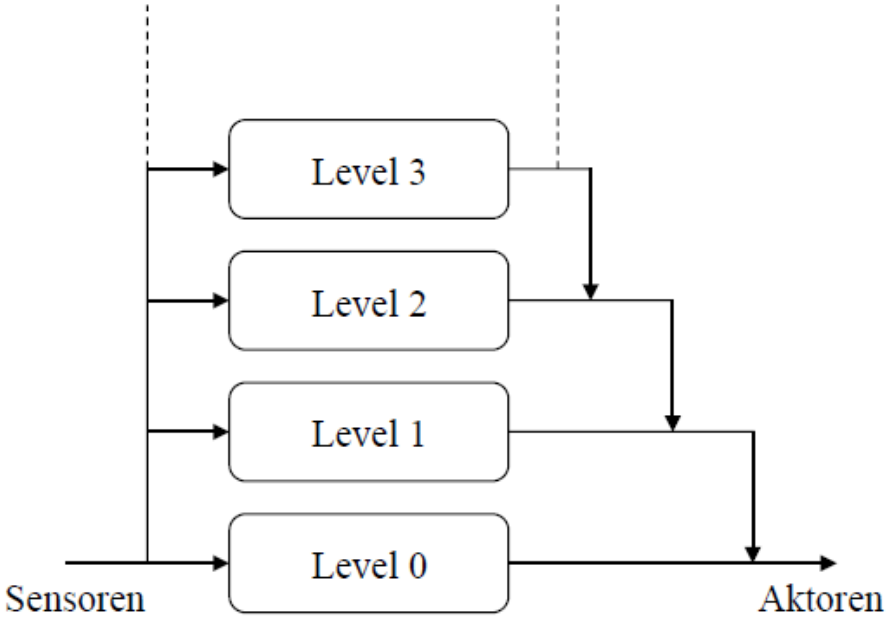

Abbildung 2.4: Subsumptionsarchitektur [Zweigle]

2.3.2 Verteilte

Die verteilte verhaltensorientierte Architektur arbeitet als eine Menge von unabhängiger Teilsysteme mit identischen Verhaltensmustern, die miteinander konkurrieren. Es muss eine zentrale Arbitrierungseinheit vorhanden sein, die entscheidet, welches Teilsystem die Ausgaben erzeugen darf. Diese Architektur eignet sich für die kooperative Roboter (z.B. Fußballroboter).

Ein Beispiel der Architektur stellen die Multi-Agenten-Systeme (MAS). Die Eigenschaften von MAS sind: Selbstorganisation, Verhandlung, assoziative Speicherung von Information und Erkennung von bestimmten Mustern. MAS-Architektur von Roboter Fußball Club (RFC) Stuttgart ist ein Repräsentant dieser Klasse.

3 verwandte Arbeiten

In diesem Kapitel möchte ich zwei Projekte vorstellen. Beide Projekte liegen zwar länger zurück, können aber als Grundlage für viele aktuelle Projekten angesehen werden.

3.1 BDI-Konzept

Der Ursprung des Konzepts liegt im *Rational Agency Project* (RAP) am Stanford Research Institute in den 1980er Jahren basierend auf der Arbeit von MICHAEL E. BRATMAN, Professor in Stanford für Philosophie, und soll menschliche Entscheidungsfindung (*practical reasoning*) nachbilden. Eine Weiterführung ist *procedural reasoning system* (PRS) von Mike Georgeff.

3.1.1 Datenstrukturen

Die Aufbau besteht aus Datenstrukturen (*beliefs*, *desires*, *intentions*, Plan-Bibliothek) und einem Interpreter, der mit diesen arbeitet (Abbildung 3.1).

Beliefs (Meinungen) repräsentieren die Informationen über die Umgebung, Menge von ausführbaren Aktionen und möglichen Meinungen über unsicheres Wissen mit ihren Eigenschaften und Initialisierungen. *Beliefs* werden kontinuierlich aktualisiert, beeinflusst von der Wahrnehmung (Sensorik) und internen Schlussfolgerungen (*Intentions*).

Desires (Wünsche) stellen eine Menge von Zielen und Ereignissen, auf die der Roboter reagieren kann, mit ihren Initialisierungen dar. Die Ziele verändern sich normalerweise nicht und können widersprüchlich zu einander sein.

Intentions (Absichten) sind die erreichbare Ziele, auf die Ausführung welcher der Roboter sich verpflichtet hat. Aus diesen Zielen werden Pläne abgeleitet. *Intentions* können im unterschied zu *Desires* keine Widersprüche haben und können die *Beliefs* beeinflussen, z.B. auf der Annahme, dass das Ziel erreicht wird.

Die Plan-Bibliothek beinhaltet Abarbeitungsvorschriften, mit denen sich ein bestimmter Teilzustand der Welt in einen anderen überführen lässt [Schönmann (2003)]. Jeder Plan besteht

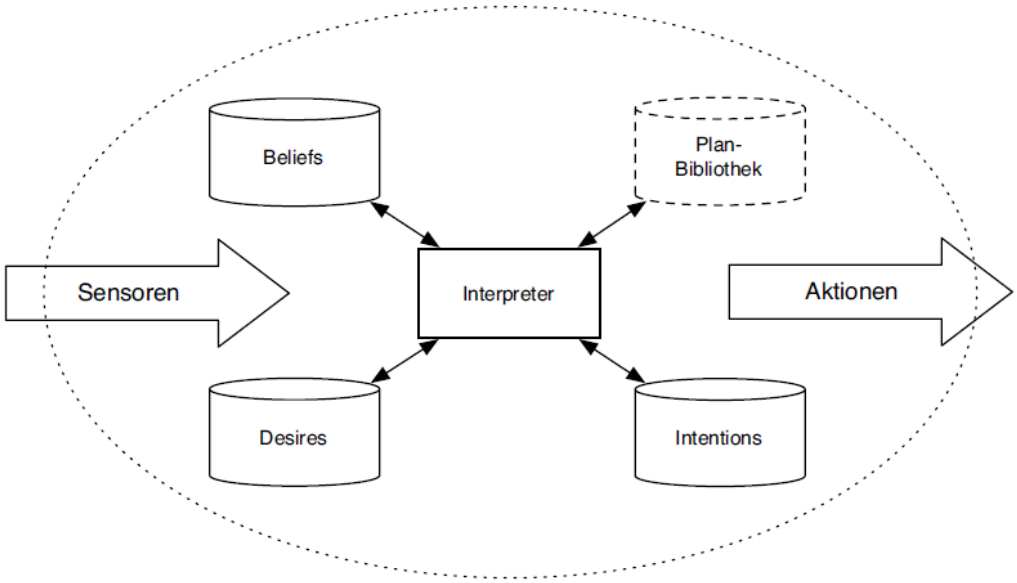

Abbildung 3.1: Grundlegende Aufbau [Schönmann (2003)]

aus Vorbedienungen, Nachtbedienungen und möglicherweise Kontextbedienungen, die während der Ausführung gelten müssen. Außerdem besitzt ein Plan einen Rumpf mit direkten ausführbaren Aktionen und Teilzielen. Durch Teilziele ist die Möglichkeit gegeben, Pläne zu staffeln.

3.1.2 Ablauf

Die Wahrnehmung erzeugt Events. Die Events bewirken Aktualisierung von *Beliefs* und lösen *commitment* (Festlegung auf ein Ziel) aus (Abbildung 3.2). Dabei werden mit *Beliefs* erfüllbare Ziele ausgewählt und dazu passende Pläne in der Plan-Bibliothek gesucht und parametrisiert. Weil Ziele Widersprüche aufweisen können, wird mit Hilfe von *Intentions* nur für ein Ziel entschieden, damit keine Konflikte mit aktuellen *Intentions* entstehen können. *Intentions* sind in mehreren Stacks aufgeteilt, jeder Stack entspricht einem verfolgten Ziel und bildet eine Absicht. Es wird eine günstige Absicht ausgewählt und der Plan abgearbeitet. Wenn eine direkte Aktion vorliegt, wird sie ausgeführt, wenn es um einen Teilziel handelt, wird ein interner Event erzeugt, der wiederum nächsten *commitment* auslöst.

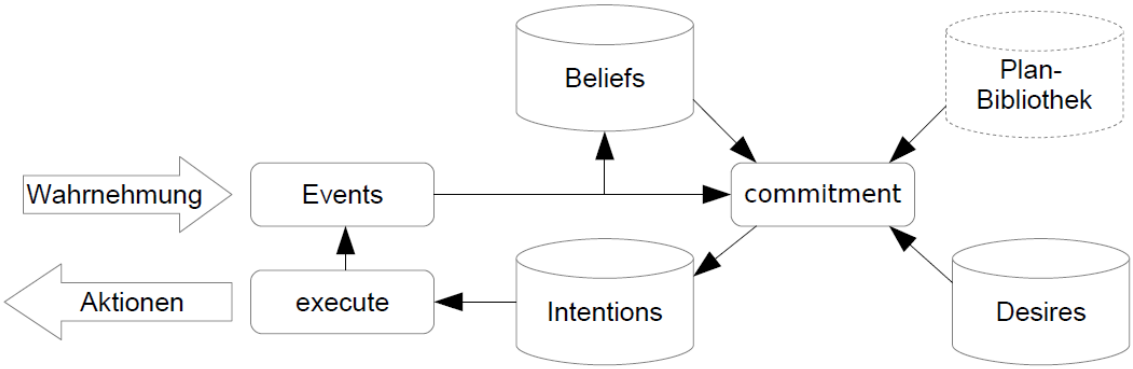

Abbildung 3.2: Ablauf des *practical reasoning*

3.1.3 Probleme und Kritikpunkte

Ein Problem des Konzepts stellt die Verwaltung von *Intentions* dar. Weil sich die Umgebung und somit die *Beliefs* ständig ändern, könnten Bedienungen für einen Plan aus Absichten nicht mehr erfüllbar sein. So eine *Intention* soll von dem Stack entfernt werden. Die Veränderungsrate der Welt wirkt somit auf Erfolgsrate eines Plans. Eine ständige Überprüfung auf Erfüllbarkeit benötigt zusätzliche Ressourcen, wird aber ein Ziel zu lange verfolgt, könnte ein besserer Weg nicht gefunden oder das Ziel nicht erreicht werden.

Weil die Ziele und Pläne unverändert bleiben, ist BDI-Ansatz nicht auf Lernen und Verhaltensanpassung ausgelegt. Eine Lösung ist neue Pläne aus erfolgreichen *Intentions* extrahieren lassen (SOAR Architektur [Kaczmarczyk (2007)]).

Insgesamt ist das sehr interessanter Ansatz, denn man weiter verfolgen sollte.

3.2 KAMRO

Karlsruher Autonomer Mobiler Roboter (KAMRO) wurde am Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe entwickelt (Abbildung 3.3). KAMRO ist ein Versuch die Vorteile von reaktiven (Kollisionsvermeidung) und deliberativen (Planung) Verhalten in einem System zu vereinen.

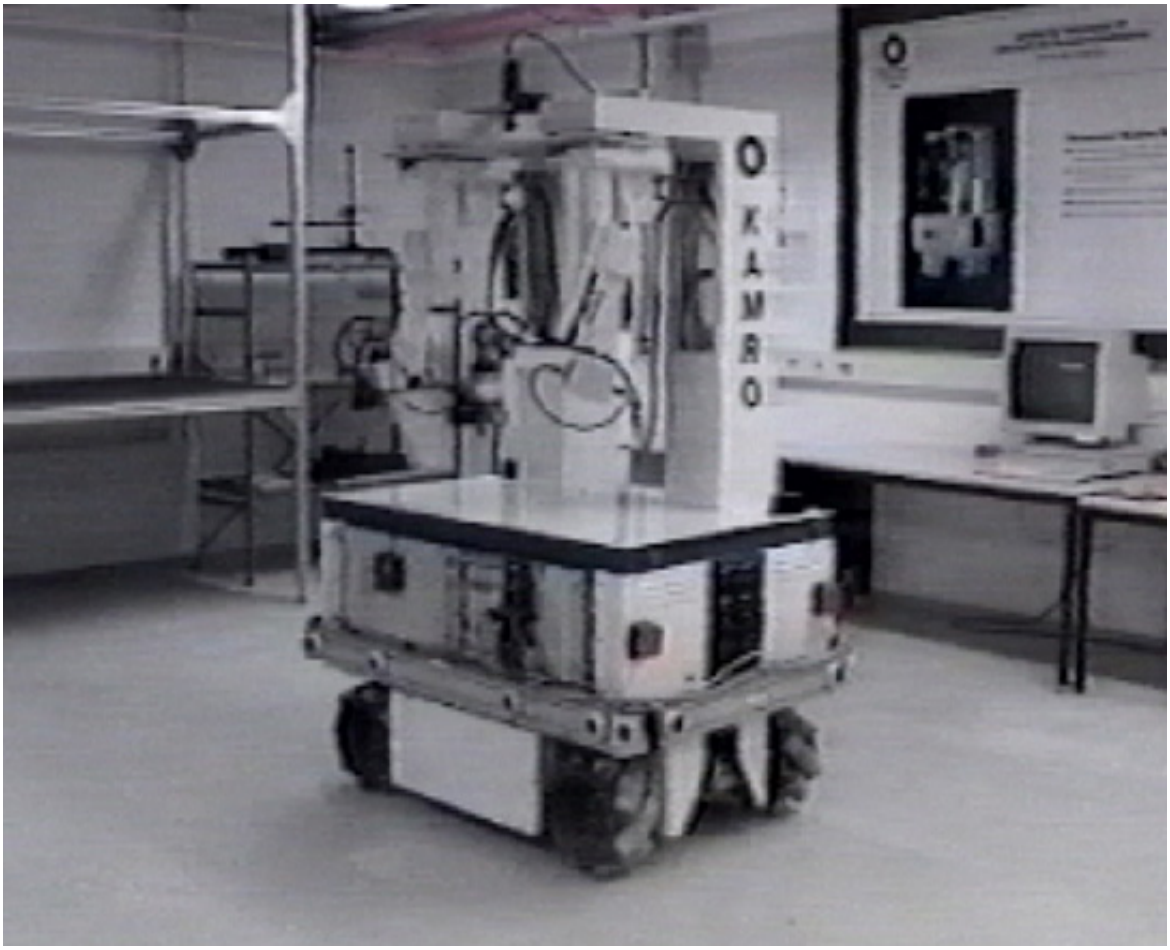

Abbildung 3.3: KAMRO

Der Roboter verfügt über eine hierarchische deliberative Planungskomponente und eine Robotersteuerungskomponente. Die Schnittstelle zwischen Komponenten ist durch explizite Elementaroperationen (EEOs) definiert. Eine EEO ist kein direkter Roboterbefehl, sondern eine Struktur (Trajektorie, Zielposition, etc.) für bestimmte Bewegungen (z.B greifen, bewegen, loslassen) ohne konkrete Parameter. In der Robotersteuerungskomponente werden EEOs mit Unterstürzung von Sensoren reaktiv verarbeitet und Resultate zurückgegeben (reaktives Verhalten).

Hierarchische Planungsarchitektur:

- Auswahl von ausführbaren Montageoperationen
- Vervollständigung der impliziten Montageoperationen
- Bewegungsplanung expliziter Roboteraktionen
- Überwachung und Synchronisation der Ausführung

Eine Montageaufgabe ist durch eine Bedingung/Ereignis-Petrinetz dargestellt, dabei entspricht eine Transition einer Pick- oder Place-Operation. In der ersten Ebene werden die Operationen mit erfühlten Bedienungen ausgewählt. In der zweiten Ebene werden ausgewählte implizite Operationen mit roboterspezifischen Parametern vervollständigt und notwendige Objekte erzeugt.

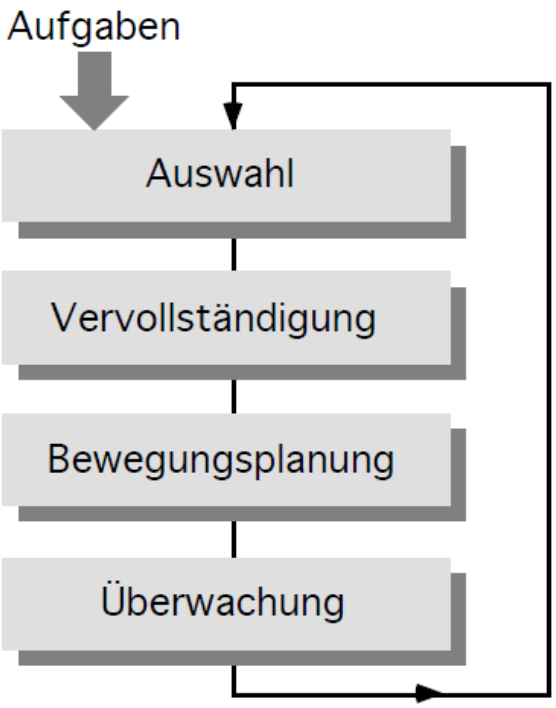

Abbildung 3.4: [Xiaoqing Cheng (1994)]

Die dritte Ebene erstellt aus Operationen eine Reihenfolge von EEOs. Anschließend werden EEOs an die entsprechende Steuerungen weitergegeben und die Ausführung überwacht und synchronisiert. Die Ergebnisse werden zurück an die erste Ebene gereicht.

Weil diese Architektur geschlossen ist, wurde ein Umstieg von hierarchischen zu verteilten System geplant. Dieser Lösungsansatz kann sich als sehr nützlich erweisen.

4 aktuelle Projekte

Derzeit existiert große Anzahl aus verschiedenen Projekten entstandener Systeme, die den Entwicklern eine Weiternutzung von erarbeiteten Konzepte und Architekturen ermöglichen. Die Beispiele für diese Systeme sind: *Player*, *YARP*, *Orocos*, *ROS*, *CARMEN*, *Orca*, *MOOS*, *Microsoft Robotics Studio*. Jeder dieser Systeme hat seine eigenen Stärken und Schwächen. So wurde *Player* für mobile Plattformen gedacht, *YARP* für humanoide Roboter, *Orocos* legt den Schwerpunkt auf Real-time Steuerung und *ROS* setzt auf Wiederverwertbarkeit von Code.

Außer bestimmten Aufgabenfeldern jedes Systems, haben sie auch Gemeinsamkeiten. Alle sind verteilte funktionsorientierte Architekturen mit modularem Aufbau. Es werden von Systemen mehrere Programmiersprachen unterstützt, so kann C++, Java, Lisp, Python und andere Programmiersprachen für Benutzung bei der Implementierung ausgewählt werden. Manche Systeme sind Cross-Plattform fähig, was als sehr nützlich erweisen könnte. Auch gemeinsame Einsatz mehrere Systeme in einem Projekt ist bei Umständen möglich. Die Plattformen dieser Systeme bieten viele Möglichkeiten für eine Community ihr Wissen auszutauschen. So können Lösungsansätze anderer Nutzer in eigenes Projekt einfließen.

4.1 TUM-Rosie mit CRAM-Architektur

Von der Intelligent Autonomous Systems Group am Technischen Universität München wurde *Cognitive Robot Abstract Machine* (CRAM) erstellt und im Roboter TUM-Rosie (Abbildung 4.1) eingesetzt. Die CRAM-Architektur ist an BDI-Koncept angelehnt. CRAM-Architektur ist modular aufgebaut (Abbildung 4.2) und mit ROS, Player, Yarp und Orocos realisiert. Diese Middleware kann zusammen genutzt werden, so wird ROS auf höherer Abstraktionsebene für asynchronen Transfer der Pläne und Daten zwischen Modulen eingesetzt. Die Ansteuerung von elektrischen Motoren läuft über Real-time fähigen Orocos.

Abbildung 4.1: TUM Rosie

4.1.1 Hauptmodule

CRAM plan language (CPL) verwaltet miteinander konkurrierende Pläne. Diese Pläne haben sensorunterstützten reaktiven Verhalten und können parallel ausgeführt werden. CPL ermöglicht eine Fehlerbehandlung der Pläne durch semantische Annotation.

KnowRob ist für die Verarbeitung und Verstand zuständig. Die Beschreibung logischer Zusammenhänge ist mit OWL realisiert. KnowRob besitzt einen Mechanismus um externe Prozeduren zu starten und auf Daten zuzugreifen. Dadurch können verschieden externe Module angeschlossen werden. So sind Open Mind Common Sense (OMCS) Datenbank und ProbCoge mit Wahrscheinlichkeitsverteilung über verschiedene Alternativen eingebunden. Außerdem sind spezifische Methoden für Schlussfolgerungen enthalten.

CRAM kernel realisiert Roboterpläne und macht Überlegungen über *Belief*-Zustand und Be-

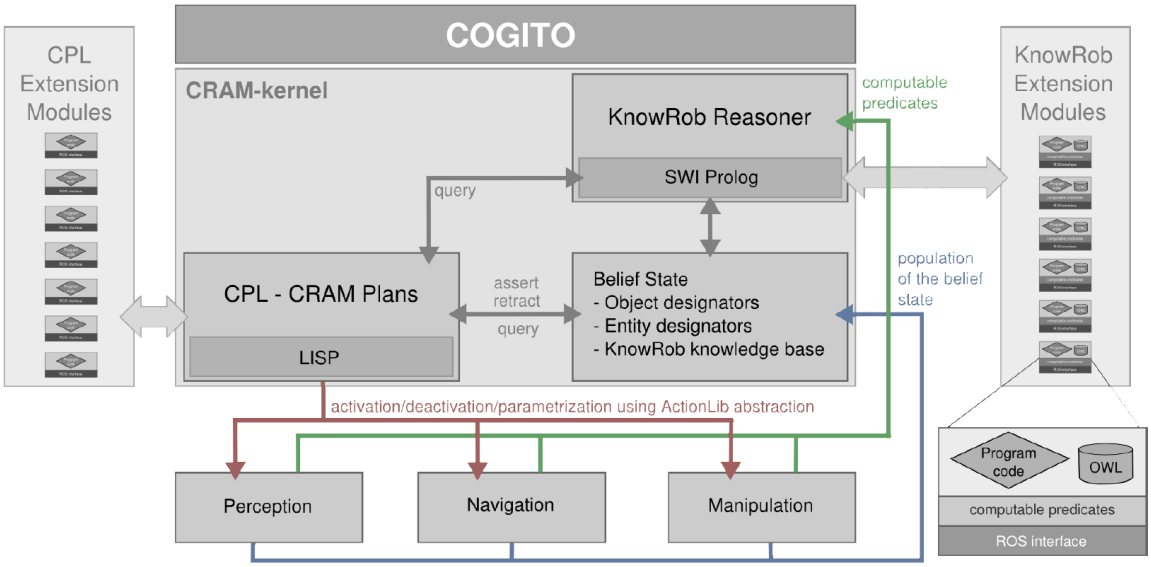

Abbildung 4.2: CRAM-Architektur [Beetz (2010)]

obachtungen der Umgebung. Kernel besitzt Schnittstellen und Methoden, um die Informationen von anderen Modulen zu verarbeiten. Auf den Kernel ist COGITO aufgesetzt. COGITO befasst sich mit Performanceprüfung auszuführender Pläne, sucht nach Problemen und Mängeln, versucht diese zu beheben und damit die vorhandene Pläne zu verbessern.

4.1.2 CRAM Erweiterungsmodule

Zu den Hauptmodulen können auch weitere Module eingebunden. *Designator* versucht mit Rete-Algorithmus die Eigenschaften und Parameter für Objekte und Pläne bei geänderten Umgebungsinformationen zu propagieren. Mit *AM-EvA* werden die Bewegungen bei Menschen interpretiert. Robot Learning Language ist für Protokollierung der Resultate für Verbesserung der Leistung von Planungssystem zuständig. Für Transformationen der Pläne wird CPL mit Methoden von *TRAINER* erweitert. Über Web können neue Pläne und Objektmodelle ins System importiert werden. Auch Prognosen aus eine detaillierten physikalischen Simulation sind möglich.

4.1.3 Abschlusswort

Der Projekt TUM-Rosie ist weit fortgeschritten und kann als eine gute Unterstützung dienen. Im Projekt TUM-Rosie angewendete Lösungsansätze können HAW ScitosG5- Team bei Problemen und Schwierigkeiten weiter bringen.

Literaturverzeichnis

[Artifcial Intelligence Center Ravenswood 2001] Artifcial Intelligence Center Ravenswood (Veranst.): *Procedural Reasoning System User's Guide*. 2.0. 2001

[Beetz 2010] BEETZ, Michael: *CRAM - A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments*. 2010. – Intelligent Autonomous Systems Group

[Kaczmarczyk 2007] KACZMARCZYK, Peter P.: *SOAR Eine Kognitive Architektur*. 2007. – AI Tools

[Schönmann 2003] SCHÖNMANN, Frank: *BDI-Architektur (Beliefs-Desires-Intentions)*. 2003. – Seminar: Kognitive Modellierung animierter Agenten

[Steinmüller 2009] STEINMÜLLER, Dr. J.: *Robotik*. 2009. – Vorlesung an der TU Chemnitz

[Xiaoqing Cheng 1994] XIAOQING CHENG, Tim L.: Reaktive Planung und Planausführung in dem intelligenten Robotersystem KAMRO Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe (Veranst.), 1994

[Zweigle] ZWEIGLE: Roboter- und MAS-Architekturen