Aktionsprimitiv-basierte Steuerungsarchitektur für Anwendungen in der Robotik und Fertigungstechnik

Primitive action based control architecture for applications in robotics and manufacturing engineering

Dipl.-Wirtsch.-Ing. Matthias Hennig, Prof. Dr. techn. Klaus Janschek
Technische Universität Dresden, Institut für Automatisierungstechnik, 10162 Dresden, Deutschland
matthias.hennig@tu-dresden.de

Kurzfassung

Der vorliegende Beitrag stellt einen Entwurf für eine flexible und robuste Steuerungsarchitektur für Roboter- und Fertigungssysteme vor. Dabei wurde versucht ein offenes Konzept zu realisieren, welches einen vereinfachten Engineeringprozess ermöglicht. Hierzu wird innerhalb der Steuerung eine Trennung zwischen einem funktionellen verhaltensbasierten und einem ablauforientierten Modell vorgeschlagen. Dieser Ansatz wird durch die Verwendung von Aktionsprimitiven innerhalb einer hybriden Robotersteuerung ermöglicht. Diese garantieren durch ihre ausgeprägte Modularität eine hohe Flexibilität und Erweiterbarkeit des entstandenen Systems. Im Beitrag wird sowohl der entstandene Entwurf diskutiert als auch eine prototypische objektorientierte Implementierung vorgestellt sowie erste Ergebnisse präsentiert.

Abstract

This paper presents a framework for a flexible and robust control architecture for robotic systems. The design incorporates an application independent system concept which allows a simplified engineering process. For this purpose a distinction between a functional behavioural and a sequential control system model is proposed. This approach is based on the utilisation of action primitives within a hybrid control architecture. The use of these primitives affords a high level of modularity through increasing flexibility and expandability of the resulting system. In this paper the proposed framework will be discussed as well as a prototypical object-oriented implementation and first results.

1 Einleitung

Mechatronische Systeme der Robotik und Fertigungstechnik sollen heutzutage in zunehmendem Maße in dynamischen nichtdeterministischen Umgebungen arbeiten. Mithilfe von geeigneten Steuerungsarchitekturen sollen solche Systeme komplexe Aufgaben lösen und dabei speziell auf unvorhergesehene Ereignisse reagieren können. Aus diesen Aufgaben können einige zentrale *Anforderungen an Steuerungsarchitekturen* abgeleitet werden: *(i)* Flexibilität und Adaptivität gegenüber Ereignissen aus der Umwelt zur Erhöhung der Robustheit, *(ii)* Erweiterbarkeit und Modularität zur benutzerfreundlichen Ausweitung der Handlungsfähigkeit des Systems sowie *(iii)* Transparenz der Architektur als effizienter Engineering-Ansatz.

Im vorliegenden Beitrag werden anfänglich aktuelle Robotersteuerungsarchitekturen analysiert und evaluiert. Anschließend wird die Verwendung von Aktionsprimitiven innerhalb solcher Architekturen diskutiert und anhand eines formalen Konzeptes erläutert. Darauf aufbauend wird die in diesem Beitrag beschriebene Aktionsprimitiv-basierte Steuerungsarchitektur umfassend vorgestellt und innerhalb eines objektorienterten Softwareentwurfs vertieft. Abschließend wird über durchgeführte Experimente anhand eines Anwendungsbeispiels berichtet.

2 Robotersteuerungsarchitekturen

Steuerungsarchitekturen sind in erster Linie Softwarearchitekturen und legen die Organisationsform fest, in der aus den sensorischen Wahrnehmungen des Systems Handlungsanweisungen für die Aktuatorik erzeugt werden. Dabei existieren drei Ansätze (**Bild 1**) für die Realisierung von Steuerungsarchitekturen (vgl. [1], [2]): *(i)* Beim so genannten *Sense-Plan-Act-Paradigma* liegt der Fokus auf einer vollständigen Modellierung der Welt. Solche *funktionsorientierten (deliberativen) Architekturen* nutzen eine komplexe Planungskomponente, um aus Sensordaten mithilfe eines Weltmodells methodisch Handlungsanweisungen für das Robotersystem zu erzeugen (**Bild 1a.**). Solche Systeme, wie in [3] und [4] vorgestellt, scheitern jedoch aufgrund unflexibler und rechenzeitintensiver Planungskomponenten und unvollständiger Weltmodelle häufig in dynamischen und nichtdeterministischen Umgebungen. *(ii)* Beim so genannten *Sense-Act-Paradigma* finden *verhaltensbasierte (reaktive) Architekturen* Verwendung, welche durch eine direkte Kopplung der Sensorik über vorgegebene Regeln an die Aktuatorik gekennzeichnet sind (**Bild 1b.**). Solche Regeln bilden jeweils ein Verhalten des Systems ab. Verhalten können dabei parallel arbeiten und untereinander über verschiedene Mechanismen interagieren. Ein übergeordnetes

Weltmodell existiert hier nicht. Auf dynamische Ereignisse in der Umwelt kann dabei flexibel reagiert werden. Protoypische Realisierungen finden sich in [5-7], weitere grundlegende Anwendungen in [8-14]. Solche Systeme haben jedoch den entscheidenden Nachteil, dass zielorientiertes und geplantes Verhalten auf höheren Ebenen nur schwer realisiert werden kann.

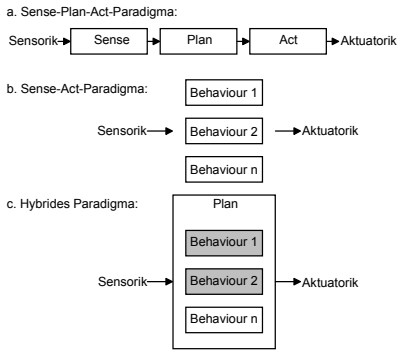

Bild 1 Die drei grundlegenden Paradigmen von Robotersteuerungsarchitekturen.

(iii) Um nun die Vorteile beider Paradigmen zu nutzen, kann ein hybrider Lösungsansatz genutzt werden. Bei so genannten *hybriden Architekturen* werden die reaktiven Fähigkeiten einer verhaltensbasierten Steuerung mit einem aufgabenorientierten Modul vereint (**Bild 1c.**). Dadurch kann auf Umweltereignisse dynamisch und zeitnah mittels parallel agierender Verhalten reagiert werden, während eine übergeordnete Planungs- und Ausführungskomponente diese Basisfähigkeiten aufgabenabhängig kombiniert und aktiviert. Dies wurde beispielsweise in [15-18] für mobile Roboter umgesetzt. Auch Robotersteuerungen für Industrieroboter sind heutzutage nach [19] und [20] oft nur durch einfache Architekturen zur Abarbeitung einer Bearbeitungsaufgabe mittels einer Bahnsteuerung bzw. Multi-Punktsteuerung gekennzeichnet. Solche *roboterorientierten Ansätze* sind jedoch vergleichsweise unflexibel bei sich verändernden Anforderungen an das Robotersystem (vgl. [21]). Dem steht der *aufgabenorientierte Ansatz* gegenüber, bei dem die Steuerung nach [22] die Roboteraufgabe in einzelne Schritte einer hierarchisch unterlagerten Ebene zerlegt und letztendlich ausführt. Beide Ansätze reagieren jedoch kaum auf dynamische Ereignisse und sind meist aufgrund fehlender Transparenz nur schwer erweiterbar.
Hybride Ansätze für Robotersteuerungsarchitekturen sind demnach Stand der Technik. Zur flexiblen und robusten Steuerung solcher Systeme existieren jedoch derzeit nur Insellösungen. Verallgemeinerbare Konzepte und Methoden dazu sind noch nicht weit entwickelt. Dies soll im hier vorgestellten Ansatz durch die Nutzung von Aktionsprimitiven in einer transparenten und modularen Architektur hybriden Ursprungs erreicht werden.

3 Ansatz über Aktionsprimitive

Für die Nutzung von modularen vereinheitlichten Programmbausteinen zur Realisierung gewünschter Funktionalität innerhalb einer Steuerungsarchitektur lässt sich der Aktionsprimitivansatz nutzen. Unter einem Aktionsprimitiv wird dabei ein *klar definiertes Softwaremodul zur Lösung genau einer Aufgabe* verstanden. Es handelt sich demnach um ein Makro, welches autonom *sensorgeführt* und *sensorüberwacht* eine Minimalaktion ausführt. Die Realisierung verschiedener Aufgaben in Form von standardisierten Aktionsprimitiven ist dabei flexibel möglich. Robotersteuerungen auf Basis von Aktionsprimitiven wurden erstmals in [23] vorgestellt und später durch [24] bzw. [25] vertieft. Das Konzept wurde dabei für den Einsatz von Industrierobotern bei Montagearbeiten durch sequentielle Abarbeitung von Aktionsprimitiven vorgeschlagen. Das Aktionsprimitivkonzept wurde in [26] auch innerhalb der mobilen Robotik angewendet. Jedem Aktionsprimitiv wird hierbei ein diskreter Zustand des aufgabenspezifischen Systems zugewiesen.
Der im hier vorgestellten Entwurf verwendete Formalismus zur Beschreibung von Aktionsprimitiven wird in Anlehnung an [27] wie folgt definiert:

$$R = AP(HA,\ \tau,\ \lambda) \tag{1}$$

Ein Aktionsprimitiv AP besteht somit aus den folgenden Komponenten: *(i)* Die hybride Aktion HA beschreibt eine konkrete Aktion des Roboters, welche zur Ausführungszeit des Primitivs aktiviert wird (z.B. Hindernisvermeidung). *(ii)* Durch τ wird eine Menge von so genannten Werkzeugkommandos definiert. Hier wird die zur Laufzeit benötigte Aktuatorik festgelegt (z.B. Greifer). *(iii)* Aktionsprimitive werden immer mit dem Erreichen eines parametrierten Zielzustandes beendet. Dieser wird durch die Abbruchbedingungen λ definiert (z.B. Zielposition, Timeout). Am Ende seiner Aktivierungsphase gibt ein Aktionsprimitiv den Rückgabewert R zurück, mit dessen Hilfe der aktuelle Systemzustand ermittelt werden kann.
Die zuvor beschriebenen Implementierungen realisieren die Robotersteuerung mit Aktionsprimitiven als Zustandsautomaten, wobei zu jedem Zeitpunkt immer nur genau ein Aktionsprimitiv aktiv sein darf. Eine weitergehende Möglichkeit ist jedoch die Nutzung von parallel arbeitenden Primitiven in einer verhaltensbasierten Steuerung unter Zuhilfenahme einer hierarchisch überlagerten Steuerung. Dieses Konzept wird im nachfolgend vorgestellten Ansatz realisiert.

4 Engineering-orientierter Architekturentwurf

Grundanliegen der vorgestellten Steuerungsarchitektur ist die Realisierung einer transparenten und flexiblen modularen Steuerung auf Basis von Aktionsprimitiven. Die daraus hervorgehende offene Softwarearchitektur lässt sich in verschiedenen Domänen der Robotik nutzen.

4.1 Frameworkdesign

Durch die Realisierung eines transparenten Ansatzes soll ein erleichtertes Engineering für mögliche Anwender ge-

schaffen werden. Dazu wird im vorliegenden Ansatz eine Trennung zwischen einem *funktionellen verhaltensbasierten Modell* zur Ausführung von Aktionsprimitiven und einem *ablauforientiertem Modell* zur aufgabenorientierten Konfiguration derselben vorgeschlagen. Mit dieser Herangehensweise ist es möglich, *zwei unterschiedliche Nutzergruppen* mit verschiedenem Systemwissen zuzuordnen: den Operator sowie den Entwickler. In **Bild 2** werden die Komponenten der vorgestellten Architektur gezeigt und nachfolgend erläutert:

Aktionsprimitiv-Bibliothek: Aktionsprimitive bilden das Fundament des vorgestellten Systementwurfs. Einzelne Aktionsprimitive (AP) werden von einem *Entwickler* einmal implementiert und sind dann in einer Bibliothek (AP-Bibliothek) gespeichert.

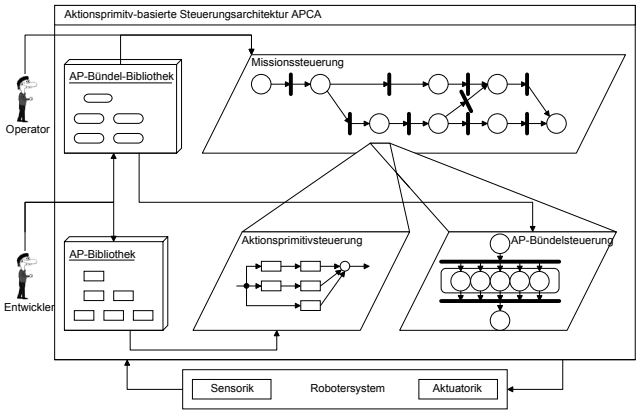

Bild 2 Gesamtansicht der Aktionsprimitiv-basierten Steuerungsarchitektur.

Dabei können neue Aktionsprimitive jederzeit flexibel hinzugefügt werden. Jedes Aktionsprimitiv bildet hierbei eine konkrete Funktionalität des Robotersystems ab.

Aktionsprimitiv-Steuerung: Alle in der Bibliothek hinterlegten Primitive werden in der Aktionsprimitivsteuerung in ein *funktionelles Modell* überführt. Somit werden die in der Bibliothek hinterlegten Aktionsprimitive zu einem Verhaltensnetzwerk verschaltet. Dadurch entsteht eine verhaltensbasierte Schicht mit der Möglichkeit Aktionsprimitive parallel konkurrierend bzw. kooperierend auszuführen. Das funktionelle Modell erlaubt den kontinuierlichen Betrieb der Robotersteuerung auf dieser Ebene.

Aktionsprimitivbündel-Bibliothek: Da es nicht zweckmäßig wäre, zu jedem Zeitpunkt alle in der Bibliothek hinterlegten Aktionsprimitive gleichzeitig auszuführen, wird immer nur ein aufgabenorientierter Satz von Primitiven aktiviert. Ein solcher Satz von zur Laufzeit des Systems gleichzeitig aktivierten Primitiven wird Aktionsprimitivbündel (AP-Bündel, APB) genannt. Solche Bündel werden vom *Entwickler* zusammengestellt und in einer zweiten Bibliothek abgelegt (AP-Bündel-Bibliothek). Aktionsprimitivbündel erfüllen demnach bereits komplexere Roboteraufgaben als einzelne Primitive.

AP-Bündelsteuerung: Die zur Laufzeit ausgewählten Aktionsprimitive in einem Bündel und die zugehörigen Parameter bleiben für die Zeit bis zum Auslösen einer Abbruchbedingung konstant. So entsteht ein diskreter Roboterzustand innerhalb eines *Ablaufmodells*. Ein Zustand innerhalb des Modells bildet somit bereits eine konkrete auszuführende Aufgabe des Robotersystems ab. Damit entsteht ein hybrides System mit einer aufgabenorientierten Ablaufsteuerung und einer kontinuierlich arbeitenden verhaltensbasierten funktionellen Schicht auf Aktionsprimitivebene.

Missionssteuerung: Eine überlagerte Missionssteuerung übernimmt die *aufgabenorientierte Aktivierung und Parametrierung* der Aktionsprimitivbündel. Dabei werden auch die Abbruchbedingungen der einzelnen Aktionsprimitive, welche im jeweiligen AP-Bündel kombiniert werden, festgelegt (*Parametrierung*). Nachdem ein Aktionsprimitivbündel abgearbeitet wurde, wird in Abhängigkeit von der ausgelösten Abbruchbedingung ein Folgezustand eingenommen und das nächste AP-Bündel aktiviert sowie parametriert.
Aufgaben, welche durch das Robotersystem ausgeführt werden sollen, bestehen somit typischerweise aus einer Anzahl von verschiedenen bzw. unterschiedlich parametrierten Aktionsprimitivbündeln, welchen diskrete Zustände zugeordnet sind. Die daraus resultierenden Missionspläne liegen als Sequenz von Aktionsprimitivbündeln vor und werden durch einen *Operator* zusammengestellt (*Konfiguration*). Dabei bedient sich der Operator an den in der Bibliothek enthaltenen AP-Bündeln.

Akteure/Nutzerkonzept: Der Engineering-orientierte Entwurf unterstützt demnach zwei Typen von Akteuren: Ein *Operator* stellt eine Aufgabe an den Roboter und entwirft dafür gegebenenfalls einen Missionsplan. Durch die Vorgabe der Aktionsprimitivbündel als Interface zum Robotersystem wird von ihm kein Wissen über die unterlagerte Robotersteuerung verlangt. Die auszuführenden Robotersystemaufgaben werden demnach aus AP-Bündeln zusammengesetzt. Der Operator *konfiguriert* und *parametriert* lediglich die Missionssteuerung.
Ein *Entwickler* entwirft und implementiert Funktionalität innerhalb der Architektur. Er stellt demnach benötigte Aktionsprimitive zur Verfügung und kombiniert neue AP-Bündel. Wenn keine Sequenz von Aktionsprimitivbündeln eine an das Robotersystem gestellte Aufgabe lösen kann, wird der Entwickler aus den vorhandenen Aktionsprimitiven die benötigten neuen AP-Bündel zusammensetzen. Wenn auch dieses Vorgehen nicht zu Lösung der Roboteraufgabe führt, dann werden durch den Entwickler neue Aktionsprimitive innerhalb der Steuerung hinterlegt. Der Entwickler *programmiert* demzufolge die der Missionssteuerung unterlagerten Ebenen.
Die transparente Struktur der Steuerungsarchitektur mit Nutzergruppen unterschiedlichen Systemwissens sowie die aus Sicht der Missionssteuerung gekapselten Aktionsprimitive erleichtern somit den Akteuren den Engineeringprozess für die Projektierung und die Konfiguration

einer anwendungsspezifischen Steuerung für mechatronische Systeme der Robotik bzw. Fertigungstechnik.

4.2 Funktionelles Modell der Aktionsprimitivsteuerung

Aktionsprimitive werden, wie zuvor beschrieben, zur Laufzeit aufgabenabhängig von der Missionsteuerung ausgewählt. Dies bedeutet, dass ein Aktionsprimitiv entweder aktiviert wird oder inaktiv bleibt. Dieser Sachverhalt ist in **Bild 3** verdeutlicht. Wird ein Aktionsprimitiv zur Lösung der aktuellen Aufgabe benötigt, so wird es aktiv geschalten und parametriert. Ein zur Laufzeit nicht benötigtes Primitiv bleibt inaktiv.

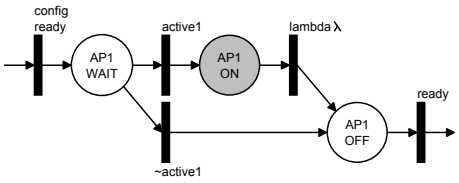

Bild 3 Ablaufsicht eines einzelnen Aktionsprimitivs.

Die im aktuell auszuführenden AP-Bündel enthaltenen Aktionsprimitive werden zu einem Zeitpunkt gleichzeitig abgearbeitet. Die einzelnen Primitive werden dabei von der überlagerten Missionssteuerung konfiguriert (z.B. Zielposition). Außerdem werden die Abbruchbedingungen parametriert (z.B. Zielposition erreicht). Diese Abbruchbedingungen werden anschließend zur Laufzeit durch die Aktionsprimitive selbst überwacht und bei Auslösung an die Missionsteuerung gemeldet.

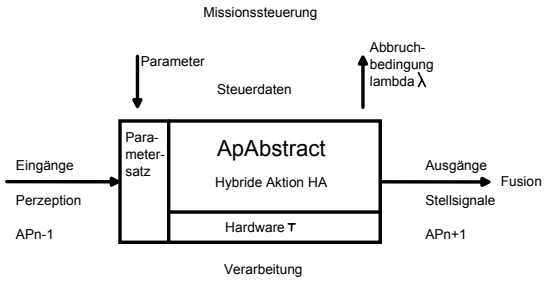

Bild 4 Darstellung des funktionellen Modells eines abstrakten Aktionsprimitivs.

Das funktionelle Modell der zuvor beschriebenen Aktionsprimitive (**Bild 4**) zeigt, dass Sensordaten ebenso als Eingang von Aktionsprimitiven dienen können wie Ausgangsdaten anderer (hierarchisch übergeordneter) Primitive. Ausgangsdaten können folglich an nachfolgende Primitive weitergegeben oder direkt an die Aktuatorik geleitet werden. Die überlagerte Missionssteuerung parametriert das Primitiv dabei aufgabenorientiert.
In **Bild 5** wird die resultierende *funktionelle Wirkstruktur* veranschaulicht. Sie enthält ebenso parallele wie auch hierarchisch angeordnete Aktionsprimitive. Es entsteht so eine regelungstechnisch-orientierte Kaskadenstruktur in der unterschiedliche Verhalten strukturiert angeordnet und ausgeführt werden können.

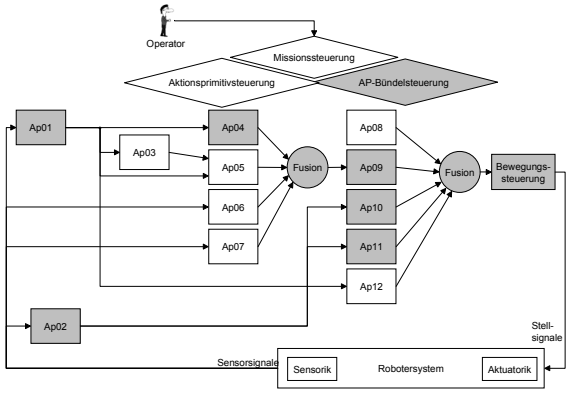

Bild 5 Funktionelle Schicht der Steuerungsarchitektur. Im Schaltbild ist eine aktuell genutzte aufgabenspezifische Konfiguration (AP-Bündel) beispielhaft grau hinerlegt.

4.3 Koordinierung

Wenn parallel arbeitende Aktionsprimitive mit gleichen Ausgangsdaten konkurrierend arbeiten, müssen diese geeignet koordiniert werden. Hier wird eine Fusion von Einzelverhalten benötigt. Aktionsprimitive, die auf gleiche Hardwareelemente zugreifen, durchlaufen demnach zusätzlich eine Fusionsstufe, um eine resultierende Ausgangsvariable v_r zu erhalten.

$$v_r = \sum_{i=1}^{n} \left(v_{AP,i} \cdot W_{AP,i} \right) \tag{2}$$

Dies wird nach Gleichung (2) kooperativ über eine Wichtung der Ausgangsdaten $v_{AP,i}$ erreicht, deren Wert $W_{AP,i}$ individuell im jeweiligen Primitiv i berechnet wird.

4.3 Ablaufmodell der Aktionsprimitivbündelsteuerung

Zur Ausführung einer Roboteraufgabe sind immer mehrere Primitive gleichzeitig aktiv. Eine solche Kombination entspricht einem Bündel von Aktionsprimitiven, die in der funktionellen Schicht gemeinsam parametriert wurden und nun selbstständig aktiv sind, bis eine der parametrierten Abbruchbedingungen erfüllt ist. Aktionsprimitivbündel überführen demnach den Roboter von einem Vorzustand in einen gewünschten Folgezustand.

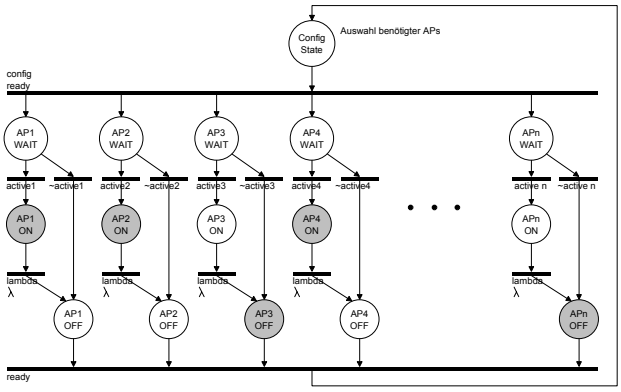

Bild 6 Ablaufmodell der Steuerungsarchitektur.

Wie in der *Ablaufsteuerung* in **Bild 6** gezeigt, werden alle aktivierten Aktionsprimitive eines Bündels somit über dieselbe Übergangsbedingung gekoppelt. Aktionsprimitivbündel sind in der Steuerung hinterlegt und helfen dem Operator bei der Zusammenstellung einer aufgabenspezifischen Mission. Die Auswahl von aufgabenspezifischen AP-Bündeln erfolgt durch die Entnahme geeigneter Bündel aus der Bibliothek. Die Kombination von Aktionsprimitiven in den Bündeln der Ablaufsteuerung wird nur in begrenztem Maße geändert, beispielsweise wenn ein neues Primitiv in der AP-Bibliothek hinterlegt wird.

4.4 Missionssteuerung

Eine Kombination von Aktionsprimitiven in einem Bündel, welche zur Laufzeit aktiviert wurden, entspricht genau einem Zustand der übergeordneten Missionssteuerung. Diese Steuerung kann somit, wie in **Bild 7** gezeigt, ebenfalls als Ablaufsteuerung modelliert werden. Anhand der parametrierten Abbruchbedingungen und der vorgegebenen situationsbezogenen Zustandsübergänge erfolgt hierbei die Aktivierung der aufgabenorientierten Aktionsprimitivbündel.

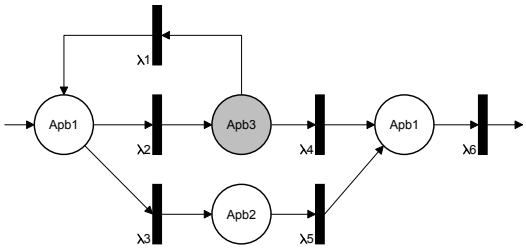

Bild 7 Beispielhaftes Schema der Ablaufsteuerung der Missionssteuerung mit Aktionsprimitivbündeln.

Missionspläne werden durch einen Operator aufgaben- und systemspezifisch anhand der vorhandenen Aktionsprimitivbündel konfiguriert.

5. Implementierung

Die Implementierung der zuvor in ihrer Konzeption beschriebenen Aktionsprimitiv-basierten Architektur erfolgt verteilt auf zwei Phasen. Zum einen resultiert aus dem Entwurf eine allgemeine Implementierung des Frameworks. Sie ist unabhängig vom konkreten Einsatz des Robotersystems. In einem zweiten Schritt werden spezifische Module für den jeweiligen Anwendungskontext hinzugefügt.

5.1 Objektorientierte Realisierung

Da einzelne Aktionsprimitive als gekapselte Minimalaktionen realisiert wurden und auch die daraus resultierenden Aktionsprimitivbündel solche Primitive nur zu aufgabenorientierten Einheiten zusammenfassen, bietet sich eine objektorientierte Implementierung an. In der softwaretechnischen Beschreibung werden Aktionsprimitive demnach durch Klassen beschrieben, wobei die hybride Aktion eines jeden Primitivs als abstrakte Methode im Klassenmodell verbleibt. Wie im Klassendiagramm in **Bild 8** zu erkennen, findet sich der hierarchische Ansatz aus **Bild 2** in der Softwarerealisierung wieder.

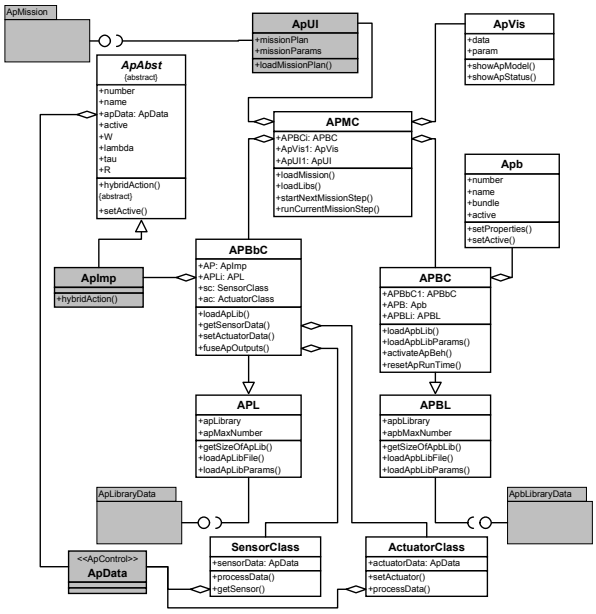

Bild 8 Vereinfachtes Klassendiagramm der entworfenen Aktionsprimitiv-basierten Steuerungsarchitektur.

Den Kern der Implementierung bildet das allgemeine Architekturframework mit den drei Hauptkomponenten AP-Steuerung (APBbC), AP-Bündelsteuerung (APBC) sowie Missionssteuerung (APMC) mit den zugehörigen Basisklassen für Aktionsprimitive bzw. –bündel (ApAbst, ApImp, Apb) sowie Hilfsklassen zum Verarbeiten der Bibliotheken (APL, APBL). Die grau hinterlegten Module realisieren die anwendungsspezifischen Funktionen. Diese müssen kontextabhängig für das genutzte Robotersystem sowie die konkrete Aufgabe angepasst werden.

5.2 Anwendungsbeispiel

Eine repräsentative Anwendung der vorgestellten Steuerungsarchitektur bietet die mobile Robotik. Zur kollisionsfreien Navigation in einer dynamischen Umgebung soll ein mobiler Roboter einem vorgegebenen Pfad folgen (**Bild 9**) und dabei unbekannten statischen und dynamischen Hindernissen ausweichen. Dazu wird der in **Tabelle 1** gezeigte Satz von Aktionsprimitiven in der Steuerung hinterlegt.

Tabelle 1 Anwendungsspezifische Aktionsprimitive

AP-Name	*Eingänge*	*Ausgänge*	*Abb.- Bed. λ*
ApMove	localDist	v	robotStuck
	localDir	ω	timeOut
ApTurn	turnAngle	localDist	angleReached
		localDir	timeOut
ApAvoidObst	obstDist	v	robotStuck
		ω	timeOut
ApGotoXY	poseGlobal	localDist	poseReached
	poseTarget	localDir	timeOut
ApFollowCrdr	poseObj	localDist	objReached
		localDir	objLost
			timeOut

Anschließend werden diese zu Aktionsprimitivbündeln zusammengefasst (**Tabelle 2**).

Tabelle 2 Anwendungsspezifische Aktionsprimitivbündel

APB-Name	*Aktivierte AP*
1 ApbMoveToDir	1 ApMove ∧ 2 ApAvoidObst ∧ 3 ApTurn
2 ApbDriveToObj	1 ApMove ∧ 2 ApAvoidObst ∧ 3 ApFollowCrdr
3 ApbDriveToTarget	1 ApMove ∧ 2 ApAvoidObst ∧ 3 ApGotoXY

Der *Operator* ist nun in der Lage aus der Menge der Aktionsprimitvbündel die Robotermission entlang des Pfades zusammenzustellen. Dazu werden die gegebenen Bündel kombiniert (*Konfiguration*) und anhand der Eingangsparameter sowie Abbruchbedingungen *parametriert*.

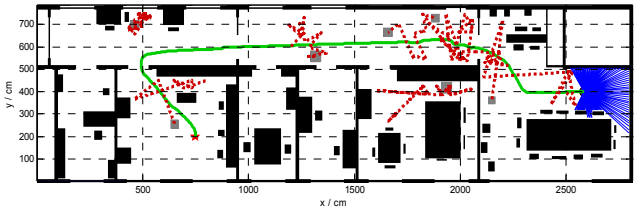

Bild 9 Aufgabenstellung und simulierter Pfad zum Zielpunkt sowie Bewegungsmuster der dynamischen Hindernisse.

5.3 Ergebnisse

Das vorgestellte Szenario wurde innerhalb einer Roboter-Simulationsumgebung in einer Standard-Büroumgebung erfolgreich getestet. Die Sollgeschwindigkeit des Roboters wurde aus der Menge der parallel aktivierten Aktionsprimitive sowie der zugehörigen Wichtungen ermittelt (**Bild 10**). Die vorgestellte Realisierung ermöglicht eine kollisionsfreie Navigation des mobilen Roboters in einer mit Unsicherheiten behafteten dynamischen Umgebung.

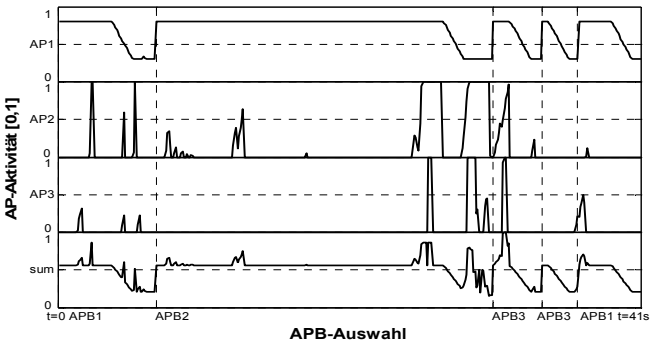

Bild 10 Die Wichtungen der aktiven Aktionsprimitive sowie die resultierende Geschwindigkeitswichtung in den einzelnen AP-Bündeln zur Laufzeit.

Ziel des Experiments war dabei auch die Verifikation des Entwurfs anhand einer üblichen Roboteraufgabe. Dabei wurde festgestellt, dass der simulierte Roboter flexibel für die vom Operator gestellte Aufgaben konfiguriert und parametriert werden kann.

6 Zusammenfassung

Im vorliegenden Beitrag wurde ein neuer Ansatz für eine flexible und transparente Steuerungsarchitektur für Robotersysteme vorgestellt. Dies wurde durch einen offenen Ansatz ermöglicht, welcher in erster Linie vom konkreten Anwendungskontext unabhängig ist. Dazu wurde die Verwendung von Aktionsprimitiven vorgeschlagen, welche durch ihre weitgehende Modularität eine hohe Flexibilität und Erweiterbarkeit der Steuerung garantieren. Innerhalb eines hybriden Entwurfs wurde eine Trennung zwischen einem funktionellen verhaltensbasierten Modell zur Ausführung von parallel arbeitenden Aktionsprimitiven und einem ablauforientiertem Modell zur aufgabenorientierten Konfiguration derselben realisiert. Durch die damit geschaffene Transparenz ist ein sehr benutzerfreundliches Engineering beim Steuerungsentwurf für Systeme der Robotik sowie der Fertigungstechnik auf Ebenen verschiedenen Systemwissens möglich. Erste durchgeführte Experimente verifizierten den Entwurf innerhalb einer simulierten mobilen Anwendung in einer dynamischen nichtdeterministischen Umgebung. Zukünftig sind weitere Experimente innerhalb anderen Anwendungskontexte sowie mit realen Robotersystemen geplant.

7 Literatur

[1] G. A. Bekey, *Autonomous robots: from biological inspiration to implementation and control*. MIT Press, 2005.

[2] B. Siciliano und O. Khatib, *Springer handbook of robotics*. Springer-Verlag New York Inc, 2008.

[3] N. J. Nilsson, *Shakey the Robot*. 1984.

[4] J. Albus und W. Rippey, “RCS: a reference model architecture for intelligent control,” in *From Perception to Action Conference, 1994., Proceedings*, S. 218-229, 1994.

[5] V. Braitenberg, *Vehicles, Experiments in Synthetic Psychology*. MIT Press, 1984.

[6] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE journal of robotics and automation*, Bd. 2, Nr. 1, S. 14-23, 1986.

[7] R. C. Arkin, *Behavior-Based Robotics*. MIT Press, 1998.

[8] P. Maes, “Situated agents can have goals,” *Robotics and Autonomous Systems*, Bd. 6, Nr. 1, S. 49-70, Juni. 1990.

[9] J. K. Rosenblatt, “DAMN: A distributed architecture for mobile navigation,” *Journal of Experimental & Theoretical Artificial Intelligence*, Bd. 9, Nr. 2, S. 339–360, 1997.

[10] J. Connell, *A Colony Architecture for an Artificial Creature*. 1989.

[11] S. Behnke und R. Rojas, “A Hierarchy of Reactive Behaviors Handles Complexity,” in *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, 2001, S. 125-136.

[12] S. Lenser, J. Bruce, und M. Veloso, “A Modular Hierarchical Behavior-Based Architecture,” in *RoboCup 2001: Robot Soccer World Cup V*, 2002, S. 79-99.

[13] M. Proetzsch, T. Luksch, und K. Berns, “The behaviour-based control architecture ib2c for complex robotic systems,” *KI 2007: Advances in Artificial Intelligence*, S. 494–497, 2007.

[14] M. N. Nicolescu und M. J. Matarić, “A hierarchical architecture for behavior-based robots,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, S. 227–233, 2002.

[15] R. J. Firby und M. G. Slack, “Task execution: Interfacing to reactive skill networks,” in *Working notes of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Architectures for Physical Agents*, 1995.

[16] R. P. Bonasso, “Integrating reaction plans and layered competences through synchronous control,” in *Proceedings of the Twelvth International Joint Conference on Artificial Intelligence, Sydney*, Bd. 1, S. 1225–1231, 1991.

[17] E. Gat, “On three-layer architectures,” *Artificial intelligence and mobile robots*, 1997.

[18] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, und H. Das, “The CLARAty architecture for robotic autonomy,” in *Aerospace Conference, 2001, IEEE Proceedings.*, Bd. 1, S. 1, 2002.

[19] S. Hesse und G. Seitz, *Robotik, Grundwissen für die berufliche Bildung*. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1996.

[20] J. Bartenschlager, H. Hebel, und G. Schmidt, *Handhabungstechnik mit Robotertechnik: Funktion, Arbeitsweise, Programmierung*. Vieweg, 1998.

[21] H. J. Siegert und S. Bocionek, *Robotik: Programmierung intelligenter Roboter*. Springer Verlag, 1996.

[22] L. Sciavicco und B. Siciliano, *Modelling and control of robot manipulators*, 6. Aufl. Springer Verlag, 2005.

[23] T. Hasegawa, T. Suehiro, und K. Takase, “A model-based manipulation system with skill-based execution,” *Robotics and Automation, IEEE Transactions on*, Bd. 8, Nr. 5, S. 535-544, 1992.

[24] B. Finkemeyer, T. Kröger, und F. M. Wahl, “Aktionsprimitive: Ein universelles Roboter-Programmierparadigma,” *at- Automatisierungstechnik*, Bd. 53, Nr. 4, S. 189-196, 2005.

[25] U. Thomas, W. An, und F. M. Wahl, “Sensorbasierte Ausführung von Roboteraufgaben auf der Basis von Aktionsprimitiven,” *Im Tagungsband Robotik*, S. 71-77, 2002.

[26] G. Milighetti, H. Kuntze, C. Frey, B. Diestel-Feddersen, B., und J. Balzer, “On a primitive skill-based supervisory robot control architecture,” in *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, S. 141-147, 2005.

[27] B. Finkemeyer, *Robotersteuerungsarchitektur auf der Basis von Aktionsprimitiven*, 1. Aufl. Shaker, 2004.