Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBC-Ansatzes

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

vorgelegt von
Gunnar Maletzki, geboren in Rostock

Rostock, im Juni 2013

urn:nbn:de:gbv:28-diss2014-0109-5

Eingereicht am: 28. Juni 2013

Verteidigt am: 14. März 2014

Begutachtet von: Prof. Dr.-Ing. habil. Dr.h.c. Bernhard Lampe,
Institut für Automatisierungstechnik, Universität Rostock

Prof. Dr.-Ing. Sven Pawletta,
Hochschule Wismar

Ao.Univ.Prof. Dipl.-Ing. Dr. techn. Felix Breitenecker,
Institut für Analysis und Scientific Computing, TU Wien

Danksagung

Die vorliegende Arbeit ist während meiner Tätigkeit an der Hochschule Wismar in der Forschungsgruppe Computational Engineering und Automation im Rahmen eines Kooperationsprojektes mit dem Institut für Automatisierungstechnik der Universität Rostock entstanden.

Mein aufrichtiger Dank gilt meinen Doktorvätern Prof. Dr.-Ing. Thorsten Pawletta und Prof. Dr.-Ing. Sven Pawletta für die Bereitstellung des interessanten Themas, sowie für die Betreuung und stetige Unterstützung während meiner Arbeit. Mein besonderer Dank gilt dabei Sven Pawletta, der viele seiner freien Abende für mich opferte und mich durch seine konstruktiven Ratschläge entscheidend vorangebracht hat.

Besonders danke ich auch Prof. Dr.-Ing. habil. Dr. h.c. Bernhard Lampe, der mir mit seinem umfangreichen Fachwissen unterstützend zur Seite stand.

Auch ein ganz großes Dankeschön an alle Mitarbeiter der Forschungsgruppe Computational Engineering und Automation für die freundschaftliche Zusammenarbeit, die schöne Zeit und Hilfe bei allen kleinen und größeren Problemen. Speziell hervorheben möchte ich Christian Stenzel, Stefan Behrendt und Tobias Pingel, die mich durch viele fachliche aber auch private Gespräche immer wieder motiviert haben.

Nicht vergessen möchte ich an dieser Stelle meine Familie. Meinen Eltern danke ich für die Förderung und Unterstützung meines Studiums. Ein besonders großes Dankeschön geht an meine Frau Claudia, die immer für mich da war und mir zuletzt gemeinsam mit unserer Tochter Lara geduldig und verständnisvoll zur Seite stand.

Kurzfassung

Bereits seit Jahrzehnten ist der Einsatz von Robotern in der Industrie etabliert, da die Robotersysteme zu einem leistungsfähigen und flexiblen Werkzeug geworden sind. Insbesondere in neuen Anwendungsbereichen wie flexiblen Fertigungssystemen kommt der effizienten Entwicklung von Robotersteuerungen eine immer stärkere Bedeutung zu. Diese Tendenz erfordert die Realisierung neuer Methoden zur herstellerunabhängigen und applikationsübergreifenden Programmierung von Roboteranwendungen. Entscheidend sind hierbei moderne Programmiersysteme als Entwicklungswerkzeuge. Gegenwärtig ist eine Vielzahl von zumeist herstellerspezifischen Roboterprogrammiersystemen auf Basis teilweise sehr unterschiedlicher Konzepte im Gebrauch, da sich existierende Standards im Bereich der Steuerungsentwicklung für Roboter bisher nicht durchsetzen konnten.

Darüber hinaus verfügen heutige Roboteranwendungen zunehmend über umfangreiche externe Aktorik und Sensorik und stellen somit komplexe Automatisierungslösungen dar, die nur im Rahmen eines systematischen Entwicklungsprozesses effizient beherrschbar sind. Rechentechnische Unterstützung sowie Methoden der Modellbildung und Simulation spielen dabei eine wichtige Rolle. In der Literatur wird in diesem Zusammenhang vom Rapid Control Prototyping (RCP) gesprochen. Gegenwärtig sind jedoch RCP-basierte Techniken noch nicht breit in der industriellen Robotik eingeführt.

Die vorliegende Arbeit hat das Ziel, einen Beitrag zur stärkeren Etablierung von RCP-Techniken im Bereich der Steuerungsentwicklung von komplexen und flexiblen Roboteranwendungen zu leisten. Ausgangspunkt ist die Vorstellung der aktuellen Roboterprogrammiermethoden und deren Einordnung in einer für diese Arbeit geeignete Klassifikation. Es wird gezeigt, dass für den Bereich der komplexen Roboteranwendungen die Hauptklasse der Offline-Methoden eine dominierende Stellung einnehmen. Übliche Offline-Entwicklungsmethoden werden hinsichtlich der gegenwärtig verwendeten Entwurfs- und Inbetriebnahme-Methodik analysiert. Dabei wird deutlich, dass die Forderungen des RCP in der industriellen Robotik noch nicht erfüllt werden. Als Lösung wird der Simulation Based Control (SBC) Ansatz vorgestellt und dessen Verwendung auf dem Gebiet der Industrierobotik diskutiert. Es wird gezeigt, dass die vom SBC-Ansatz gestellten Anforderungen an die Entwicklungsplattform durch die RCP-Umgebung Matlab erfüllt werden können und somit eine durchgängige Entwicklung von komplexen Robotersteuerungen möglich ist.

Eine weitere Zielstellung der vorliegenden Arbeit besteht darin, einen Beitrag zur Realisierung von flexiblen aufgabenorientierten Robotersteuerungen zu leisten. Dabei wird aufgezeigt, wie in einen auf dem SBC-Ansatz basierenden Entwicklungsprozess eine aufgabenorientierte Roboterprogrammierung erfolgen kann. Als Ergebnis wird ein Konzept zur Realisierung einer flexiblen aufgabenorientierten Robotersteuerung vorgestellt und an einem konkreten Anwendungsproblem demonstriert.

Abstract

Using robots in industrial application is established for decades because robot systems have become highly productive and flexible. A great importance is attached to an efficient development of robot controls especially in new fields of application such as flexible manufacturing systems. This trend requires realisation of new methods for manufacturer and application independent programming of robot controls. Modern programming systems including powerful development tools are essential in this case. Currently used robot programming systems are basing on different concepts and they are usually manufacturer specific because existing standards in robot control development could not assert themselves.

Furthermore, modern robot applications are increasingly equipped with a lot of external actuators and sensors. They represent complex automation solutions that can be handled efficiently only within a systematic development process. Therefore, methods for computer aided modelling and simulation are given a great importance. In this regard, the term Rapid Control Prototyping (RCP) is used in literature. However, RCP based techniques are not yet common in industrial robotic applications.

Making a contribution to a greater integration of RCP techniques in the field of robot control development is the objective of the present written work. Starting point is the presentation of actual robot programming methods and its appropriate classification. It will be shown that offline methods are dominating in the field of complex robot applications. General offline development methods are analysed with regard to RCP techniques and problems of currently used methods for design and commissioning are presented. As a suggested solution the Simulation Based Control (SBC) approach is introduced and its usage in the field of industrial robotic application is discussed. It will be shown that Matlab as a potential development platform meets the requirements of the SBC approach. Thus making it possible to support a continuous manufacturer independent development of complex robot controls.

Another objective target of the presented work is to make a contribution for realising flexible task oriented robot controls. In this context it is shown how task oriented robot programming being based on SBC approach could be implemented. As a result, a concept for realisation of flexible task oriented robot controls is introduced and subsequently it is demonstrated by using an application example.

Inhaltsverzeichnis

1 Einleitung

Der Einsatz von Robotern ist in der Industrie bereits seit Jahrzehnten etabliert. Robotersysteme stellen heute leistungsfähige und flexible Werkzeuge dar. Aufgrund sinkender Anschaffungskosten ist der Robotereinsatz inzwischen nicht mehr auf die Massenfertigung beschränkt. In neuen Anwendungsbereichen wie flexiblen Produktionssystemen kommt der effizienten Entwicklung von Robotersteuerungen eine immer stärkere Bedeutung zu. Entscheidend sind hierbei moderne Programmiersysteme als Entwicklungswerkzeuge.

Standardisierungsbemühungen im Bereich der Robotersteuerungsprogrammierung erfuhren bisher nur eine geringe Akzeptanz. Dadurch ist der gegenwärtige Stand der Technik auf diesem Gebiet durch eine Vielzahl zumeist herstellerspezifischer Programmiersysteme mit teilweise sehr unterschiedlichen Konzepten gekennzeichnet. Vor diesem Hintergrund besteht die erste Zielstellung dieser Arbeit darin, auf Basis verschiedener in der Literatur vorgestellter Systematiken eine für die weiteren Untersuchungen geeignete Klassifikation der Roboterprogrammiermethoden zu erarbeiten.

Abgesehen vom relativ einfach strukturierten Robotereinsatz in der Massenfertigung stellen heutige Roboteranwendungen zunehmend komplexe Automatisierungslösungen dar. Beispielsweise sind Roboter für anspruchsvolle Fertigungsaufgaben mit einer Vielzahl externer Sensoren und Aktoren ausgestattet und müssen mit ebenfalls gesteuerten Komponenten ihrer Umgebung kooperieren. Der Prozess des Entwurfs und der Inbetriebnahme solcher Roboteranwendungen mit den Mitteln der herkömmlichen Roboterprogrammierung stößt zunehmend an Grenzen.

Eine zweite Zielstellung dieser Arbeit besteht deshalb darin, die Realisierung von Roboteranwendungen aus der Perspektive der allgemeinen Entwicklung diskreter Steuerungen innerhalb der Automatisierungstechnik zu betrachten. In der Automatisierungstechnik ist man seit langer Zeit bemüht, mittels systematischer Entwurfs- und Inbetriebnahmeverfahren die Entwicklungszeiten und Kosten von komplexen Automatisierungslösungen zu senken. Nach dem Vorbild großer informationstechnischer Projekte werden inzwischen auch Automatisierungsprojekte nach dem sogenannten *V-Modell* in systematischer Weise von der Spezifikation bis zum Betrieb der Anlagen geplant und durchgeführt.

Durch Computerunterstützung und insbesondere Simulationstechniken können diese Entwicklungsprozesse erheblich beschleunigt werden. In der Literatur wird in diesem Zusammenhang vom *Rapid Control Prototyping* (RCP) gesprochen. Für regelungstechnische Problemstellungen existieren die softwaretechnischen Voraussetzungen für Rapid Control Prototyping bereits seit längerem. In der steuerungstechnischen Praxis und damit auch bei der Entwicklung von Roboteranwendungen sind RCP-basierte Techniken dagegen noch nicht breit eingeführt. Im Bereich der Forschung ist man jedoch seit mehr als einem Jahrzehnt bemüht, RCP-Techniken auch für Steuerungsprobleme zu entwickeln und bereitzustellen.

In diesem Kontext soll die vorliegende Arbeit einen Beitrag zur stärkeren Etablierung von RCP-Techniken im Bereich der Steuerungsentwicklung für komplexe Roboteranwendungen leisten.

In komplexen Roboteranwendungen, die eine Strukturierung der Steuerung in Teilaufgaben erlauben, kann eine *aufgabenorientierte* Spezifikation und Realisierung der Steuerung erfolgen. Bei anspruchsvollen Problemen ist die Ausführungsabfolge der Teilaufgaben unter Umständen nicht a priori spezifizierbar. In solchen Fällen ergibt sich diese erst während des Betriebes in Abhängigkeit von aktuellen Prozesszuständen. Dies erfordert eine sehr flexible Steuerungslogik. So spricht man bei derartigen Roboteranwendungen auch von einer *flexiblen aufgabenorientierten Robotersteuerung*. Flexible Fertigungs- beziehungsweise Steuerungssysteme sind gegenwärtig noch aktuelle Forschungsgegenstände. In diesem Bereich soll mit der Vorstellung eines Konzeptes zur Spezifikation und Generierung flexibler Robotersteuerungen ebenfalls ein Beitrag geleistet werden.

In den ersten beiden Kapiteln der Arbeit werden die notwendigen Grundlagen eingeführt. Kapitel 2 befasst sich mit den Methoden der Roboterprogrammierung. Es werden zunächst eine Anzahl teilweise sehr unterschiedlicher Systematiken aus der Literatur analysiert. Anschließend wird eine Klassifikation entwickelt, die die zuvor diskutierten Arbeiten in sich widerspruchsfrei vereint. Beispielhaft werden dann verschiedene konkrete Programmiersysteme unter Verwendung der vorgeschlagenen Klassifikation betrachtet und eingeordnet. Abschließend wird aufgezeigt, welche Klassen der Roboterprogrammierung für die Entwicklung komplexer und flexibler Robotersteuerungen relevant sind.

Im Kapitel 3 werden Roboteranwendungen aus der Sicht des Entwicklungsprozesses von diskreten Steuerungen betrachtet. Dazu wird einleitend das systematische Vorgehen nach dem V-Modell allgemein besprochen. Anschließend werden die zentralen Aspekte eines RCP-basierten Entwicklungsprozesses behandelt. Entsprechend der Klassifikation aus Kapitel 2 werden dann der Stand der Technik bezüglich der gegenwärtig eingesetzten Methodik bei der Offline- und Online-Entwicklung von Robotersteuerungen analysiert. Im letzten Abschnitt des Kapitels wird mit dem *Simulation Based Control* (SBC) ein RCP-Ansatz für diskrete Systeme eingeführt, der sich insbesondere auch für die Entwicklung von Robotersteuerungen eignet.

Im Kapitel 4 wird die weitverbreitete Entwicklungsumgebung *Matlab* als eine geeignete Softwareplattform für die RCP-basierte Entwicklung komplexer Robotersteuerungen betrachtet. Dabei wird insbesondere auf notwendige Erweiterungen eingegangen.

Anhand eines konkreten materialtechnischen Bestückungs- und Bearbeitungsproblems wird im Kapitel 5 ein kompletter Entwicklungsprozess nach dem SBC-Ansatz für eine komplexe Robotersteuerung beschrieben.

Das sechste Kapitel widmet sich der Entwicklung aufgabenorientierter Robotersteuerungen. Nach einer kurzen Einführung in die grundlegenden Aspekte der Aufgabenorientierung wird untersucht, wie diese im Rahmen einer SBC-basierten Steuerungsentwicklung realisiert werden kann. Anhand von zwei Beispielproblemen werden dann aufgabenorientierte Steuerungsentwürfe sowohl in einer Top-Down- als auch Bottom-Up-Vorgehensweise demonstriert.

Im siebten Kapitel wird schließlich das Problem der flexiblen aufgabenorientierten Ro-

botersteuerungen betrachtet, das sich oftmals im Kontext von flexiblen oder rekonfigurierbaren Fertigungssystemen (FMS, RMS) stellt. In vielen Arbeiten wird der Weg verfolgt, den Flexibilitätsanforderungen durch adaptive Steuerungsansätze zu genügen. Eine Anzahl von Arbeiten wird zu Beginn des siebten Kapitels analysiert und kategorisiert. Eines der Hauptprobleme beim Entwurf hoch flexibler Steuerungen stellt bereits die Art und Weise der Spezifikation dar. Diesbezüglich wird im dritten Abschnitt des Kapitels die *System Entity Structure* (SES) nach Zeigler als ein geeignetes deklaratives Spezifikationsmittel vorgestellt. Anschließend wird gezeigt, wie auch die automatische Generierung ausführbarer Steuerungen auf Basis einer SES-ähnlichen Beschreibung realisiert werden kann. An einem Montageproblem wird der Einsatz dieser Techniken zum Entwurf und zur Inbetriebnahme einer flexiblen aufgabenorientierten Robotersteuerung demonstriert.

Im abschließenden achten Kapitel werden die wesentlichen Aspekte der Arbeit zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

2 Methoden der Roboterprogrammierung

Ein Robotersystem besteht wie jede automatisierungstechnische Anlage aus mindestens einer Steuerungseinheit und einer Anzahl von Aktoren und Sensoren. Abbildung 2.1 zeigt den prinzipiellen Aufbau eines Robotersystems, wie er insbesondere in der industriellen Robotik typisch ist. Aktorische Elemente des Roboters sind beispielsweise Gelenkantriebe. Zur Realisierung der erforderlichen Antriebsregelungen muss ein Roboter sogenannte *interne* Sensoren besitzen. Die Implementierung der Regler erfolgt entweder wie in Abbildung 2.1 dargestellt auf der zentralen Steuerungseinheit oder verteilt auf mehreren Mikrocontrollern. Die dafür erforderliche Software inklusive eines Kinematikmoduls[1] ist Bestandteil der Firmware eines Robotersystems und ist damit *kein* Gegenstand der Roboterprogrammierung.

Unter einem Roboterprogramm wird im einfachsten Fall eine Abfolge von Steuerungsanweisungen bezüglich Bewegung und Positionierung des Roboterarms sowie eines am Endeffektor montierten Greifers verstanden. Da reale Robotersysteme wie in Abbildung 2.1 dargestellt häufig auch über *externe* Sensoren und Aktoren verfügen sowie Schnittstellen zu einem eventuellen Bediener oder zu einer Leitebene besitzen, sind im weiteren Sinne auch diese Komponenten Gegenstand der Roboterprogrammierung.

Die Entwicklung von Steuerungsprogrammen für Robotersysteme wird durch Programmiersysteme unterstützt. Hierbei handelt es sich in erster Linie um Software-Entwicklungsumgebungen. Traditionell spielen aber auch spezielle Bedieneinrichtungen zur manuellen Roboteransteuerung als Programmiersysteme eine Rolle. In modernen Systemen werden in der Regel beide Techniken in integrierter Form unterstützt. Da sich existierende Standards im Bereich der Steuerungsentwicklung für Roboter bisher nicht durchsetzen konnten, ist der gegenwärtige Stand der Technik vor allem durch herstellerspezifische Programmiersysteme geprägt, die teilweise auf sehr unterschiedlichen Konzepten beruhen.

In diesem Kapitel sollen deshalb zunächst verschiedene Klassifikationen von Roboterprogrammiermethoden vorgestellt werden. Darauf aufbauend wird im Anschluss eine Klassifikation vorgeschlagen, die die vorangegangenen in sich vereint. Nachfolgend wird eine Anzahl konkreter Programmiersysteme betrachtet und in die vorgeschlagene Klassifikation eingeordnet. Abschließend erfolgt eine Bewertung der Relevanz der Methodenklassen für die verschiedenen Robotikanwendungsbereiche.

[1] Softwaremodul zur Umrechnung der Gelenkkoordinaten in die Positionskoordinaten des Endeffektors und umgekehrt.

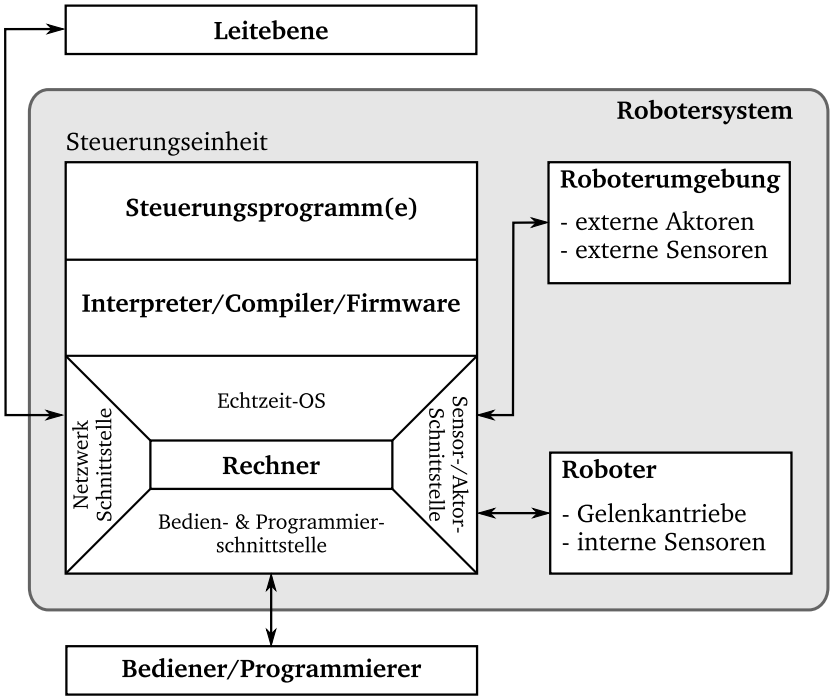


Abbildung 2.1: Prinzipieller Aufbau eines Robotersystems

2.1 Überblick und Klassifikation

In der Literatur finden sich verschiedenartige Ansätze, die Roboterprogrammiermethoden einzuordnen. Eine frühe Darstellung unterschiedlicher Methoden der Roboterprogrammierung enthält der Beitrag von Lozano-Pérez aus dem Jahr 1982 ([Loz82]). Die zu diesem Zeitpunkt existierenden Ansätze werden dort in die Hauptkategorien:

- *Guiding*,
- *roboterorientierte Programmierung* und
- *aufgabenorientierte Programmierung* unterteilt.

Unter Guiding werden in [Loz82] in erster Linie die Methoden eingeordnet, bei denen der Roboter manuell mit Hilfe einer Steuerkonsole an die gewünschten Positionen verfahren wird. Die angefahrenen Positionen werden abgespeichert und bilden in ihrer Abfolge das Steuerungsprogramm, welches die automatische Wiederholung der Bewegungsabläufe ermöglicht. Bei der roboterorientierten Programmierung werden die Bewegungsabläufe mittels einer expliziten Roboterprogrammiersprache beschrieben. Diese Sprachen sind allgemeinen Programmiersprachen ähnlich, verfügen jedoch über roboterspezifische Sprachelemente. Die aufgabenorientierte Programmierung basiert dagegen auf einem höheren Beschreibungsniveau. Das heißt, es wird nicht spezifiziert wie ein Roboter arbeitet, sondern *was* ein Roboter zu tun hat, beispielsweise ein Objekt greifen und platzieren.

Siegert und Bocionek unterteilen in [Sie96] die Roboterprogrammierung in nur zwei Hauptkategorien:

- *Online-Methoden* und
- *Offline-Methoden.*

Zur ersten Kategorie zählen sie alle Ansätze, bei denen der Roboter unmittelbar während des Programmiervorganges benutzt wird. Bei den Offline-Methoden wird der Roboter hingegen erst beim Test benötigt. Neben diesen Hauptkategorien benennen Siegert und Bocionek vier Verfahrensklassen zur Einordnung konkreter Programmieransätze:

- *Programmierung durch Beispiele*,
- *Programmierung durch Training*,
- *roboterorientierte Programmierung* und
- *aufgabenorientierte Programmierung.*

Unter Programmierung durch Beispiele fassen sie alle Verfahren zusammen, bei denen der Roboter an die gewünschten Positionen verfahren wird und diese dann zur Wiederholung gespeichert werden. Dabei unterscheiden sie auf welche Art und Weise der Roboter während der Programmierung an die Positionen verfahren wird. Unter anderem bilden sie so die Unterklassen:

- *manuelle Programmierung:*
  Führung des Roboters durch den Bediener am Endeffektor,
- *Teach-In-Programmierung:*
  Ansteuerung des Roboters durch ein Bediengerät und
- *Master-Slave-Programmierung:*
  manuelle Programmierung eines Master-Roboters als Stellvertreter und Wiederholung durch den Slave-Roboter – beispielsweise ein schwerer Industrieroboter.

Die Verfahrensklasse der Programmierung durch Training wird von Siegert und Bocionek 1996 noch als reiner Forschungsgegenstand charakterisiert. Sie ordnen hier Ansätze ein, bei denen die Bewegungsbahnen und Aktionen zu Beginn der Programmierung nur in sehr geringer Genauigkeit vorgegeben werden; beispielsweise durch Ableitung aus Konstruktionsdaten oder mittels „Vormachen“ durch einen Menschen (Demonstration). Der Roboter führt anschließend die noch unvollkommenen Abläufe aus, wobei die Abweichungen zum Sollverhalten sensorisch erfasst werden. Durch lernfähige Algorithmen und wiederholte Ausführung des Vorganges soll die Genauigkeit schließlich bis auf das gewünschte Maß erhöht werden. Die Charakterisierung der Verfahrensklassen roboterorientierte und aufgabenorientierte Programmierung entspricht bei Siegert und Bocionek vollständig den bereits zuvor besprochenen gleichnamigen Klassen nach Lozano-Pérez. Darüber hinaus betrachten

Siegert und Bocionek die Simulation von Robotersteuerungen in einem sehr engen Kontext mit der Roboterprogrammierung. So weisen sie darauf hin, dass die durch Offline-Methoden erstellten Steuerungsprogramme vorab in der Simulation getestet und verbessert werden können, wodurch sich die Test- und Inbetriebnahmephase am realen Roboter verkürzt. Weiterhin sprechen sie die Möglichkeit an, die klassischen Online-Methoden ebenfalls zur Programmierung des „simulierten Roboters“ zu nutzen. Damit werden diese zu simulationsgestützten Offline-Methoden. Siegert und Bocionek verwenden in diesem Zusammenhang den Begriff der *grafisch-interaktiven Programmierung*.

Eine weitere Klassifikation zur Roboterprogrammierung enthält der Beitrag von Biggs und MacDonald ([Big03]). Sie unterscheiden zwischen

- *automatischen Methoden* und
- *manuellen Methoden*.

In die Klasse der automatischen Methoden ordnen sie alle Ansätze ein, bei denen das Roboterprogramm ohne explizite Codierung von Steuerungsanweisungen durch den Programmierer entsteht. Die bei Siegert und Bocionek als Programmierung durch Beispiele und Training bezeichneten Verfahren sind nach dieser Klassifikation automatische Methoden. Bei den manuellen Methoden wird das Roboterprogramm unter expliziter Verwendung einer Programmiersprache erstellt. Neben der klassischen *textuellen Programmierung* führen Biggs und McDonald als alternative Unterklasse die *grafische Programmierung* zum Beispiel auf Basis von Flussdiagrammen und Icons auf.

Die nachfolgend nur kurz erläuterte Klassifikation von Craig ([Cra05]) aus dem Jahr 2005 ist zeitlich gesehen die jüngste der hier betrachteten Klassifikationen. Bemerkenswert ist, dass sie wie viele andere Arbeiten auf den drei Hauptklassen: Guiding beziehungsweise Teach-In-Programmierung, roboterorientierte und aufgabenorientierte Programmierung beruht, wie sie bereits 1982 von Lozano-Pérez vorgeschlagen wurden. Die Neuerungen dieser Klassifikationen beschränken sich somit auf die weitere Untergliederung in den bereits bestehenden Kategorien. Bei Craig betrifft dies die Hauptklasse der roboterorientierten Programmierung. Er differenziert die hier zum Einsatz kommenden Sprachen in:

- *erweiterte allgemeine Programmiersprachen* und
- *spezielle Roboterprogrammiersprachen*.

Mit allgemeinen Programmiersprachen sind beispielsweise C, Pascal oder Basic gemeint, die mit Erweiterungen zur Roboterprogrammierung eingesetzt werden. Die roboterspezifischen Erweiterungen werden gewöhnlich als Softwarebibliotheken bereitgestellt. Die speziellen Roboterprogrammiersprachen sind vor allem dadurch gekennzeichnet, dass sie in der Regel von einem Roboterhersteller entwickelt wurden und nur für dessen Robotersystem eingesetzt werden können. Synonym können sie deshalb auch als *herstellerspezifische Roboterprogrammiersprachen* bezeichnet werden.

Im Rahmen der vorliegenden Arbeit wird zur Einordnung von Roboterprogrammiermethoden die in Abbildung 2.2 dargestellte Systematik benutzt. Sie verwendet als Hauptklassen und Unterklassen der ersten Ebene die Hauptkategorien und deren Unterteilung in Verfahrensklassen nach Siegert und Bocionek ([Sie96]).

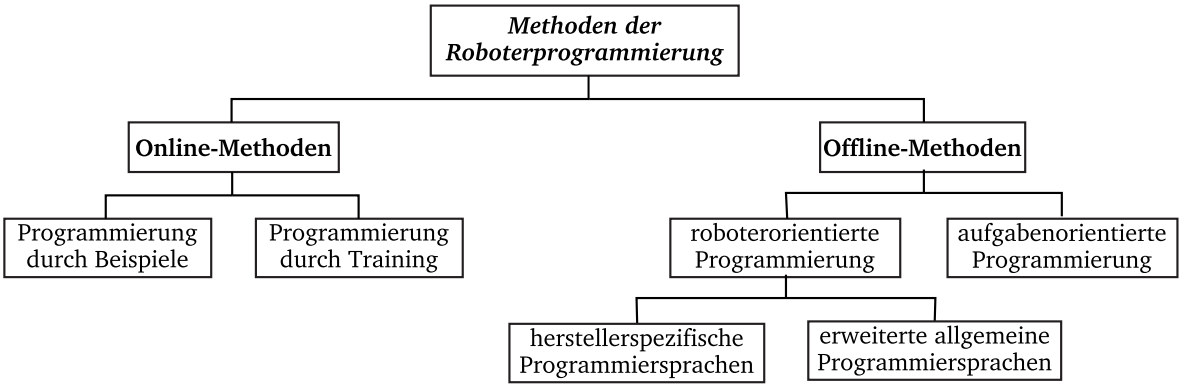


Abbildung 2.2: Klassifikation der Roboterprogrammiermethoden

Die Verfahrensklasse roboterorientierte Programmierung wird in Anlehnung an Craig ([Cra05]) in zwei Unterklassen gegliedert. Dabei wird die bei Craig als spezielle Roboterprogrammiersprachen benannte Klasse hier mit herstellerspezifische Programmiersprachen bezeichnet. Die sehr häufig referenzierten Klassen nach Lozano-Pérez ([Loz82]) bleiben in der Systematik vollständig erhalten. Die beiden Hauptklassen automatische und manuelle Methoden nach Biggs und McDonald ([Big03]) sind unter den Bezeichnungen Online- und Offline-Methoden ebenso enthalten. Die von Biggs und McDonald vorgeschlagene Differenzierung der Offline-Methoden in textuelle und grafische Programmierung wurde zwecks besserer Übersichtlichkeit nicht in die Systematik aufgenommen. Grundsätzlich kann diese Unterscheidung aber als Eigenschaft eines konkreten Programmiersystems berücksichtigt werden.

Die bereits in den Beiträgen von Lozano-Pérez sowie Siegert und Bocionek erwähnte Bedeutung der Simulation im Zusammenhang mit der Roboterprogrammierung ist in den letzten Jahren noch erheblich gestiegen. Eine explizite Berücksichtigung in der dargestellten Systematik erscheint jedoch nicht sinnvoll, da davon praktisch alle Klassen, aber teilweise in sehr unterschiedlicher Art und Weise betroffen sind. Die Rolle der Simulationstechnik insbesondere für die Offline-Methoden ist darüber hinaus ein Kerngegenstand der vorliegenden Arbeit und wird in späteren Kapiteln noch ausführlich behandelt.

In den folgenden Abschnitten wird die Arbeitsweise beispielhafter Programmiersysteme analysiert und unter Verwendung der vorgeschlagenen Systematik eingeordnet.

2.2 Online-Methoden

Nach der Klassifikation im vorangegangenen Abschnitt muss bei den Online-Methoden zwischen der Programmierung durch Beispiele und der Programmierung durch Training unterschieden werden. Während die Verfahren der ersten Klasse zu den am frühesten eingesetzten Methoden der Roboterprogrammierung überhaupt gehören, stellt die zweite Klasse immer noch ein aktuelles Forschungs- und Entwicklungsthema dar.

Als eigenständige Online-Methode ist die Teach-In-Technik das heute noch am häufigsten anzutreffende Verfahren der Roboterprogrammierung durch Beispiele. Insbesondere bei Roboteranwendungen in der Massenfertigung stellt diese Art der Programmierung nach wie vor den Stand der Technik dar. Wie bereits im Abschnitt 2.1 erwähnt, können ursprünglich als Online-Methoden entwickelte Techniken aber auch im Rahmen von Offline-Systemen Verwendung finden. Ein Beispiel dafür ist die in [Sie96] genannte grafisch-interaktive Programmierung.

Bei den meisten veröffentlichten Arbeiten zur zweiten Klasse der Online-Methoden, der Programmierung durch Training, zeichnet sich ab, dass diese nicht als eigenständige Verfahren bereitgestellt werden, sondern als Bestandteil komplexer Offline-Systeme.

Nachfolgend soll anhand zwei konkreter Systeme beispielhaft gezeigt werden, wie die Online-Methoden der Programmierung durch Beispiele und der Programmierung durch Training in Verbindung mit Offline-Systemen eingesetzt werden.

Im Beitrag von Ekvall, Aarno und Kragic ([Ekv06]) wird ein Offline-System für die aufgabenorientierte Programmierung von Robotern beschrieben. Die Steuerungslogik wird ohne direkte Verwendung des Roboters durch den Aufbau von Flussdiagrammen aus vordefinierten Aufgabenmodulen implementiert. Die konkreten Bewegungsabläufe zur Abarbeitung einer Aufgabe werden unter direkter Verwendung des Roboters mittels einer komfortablen Teach-In-Technik programmiert. Dazu ist der Roboter am Endeffektor mit einem Kraft-Momenten-Sensor ausgestattet, mit dessen Hilfe der Programmierer den Roboter in der gewünschten Weise verfährt. Die aufgezeichneten Daten werden dann zur Parametrierung der Aufgabenmodule benutzt. Nach der Klassifikation gemäß Abschnitt 2.1 wird hier die Programmierung durch Beispiele im Rahmen einer aufgabenorientierten Offline-Programmierung eingesetzt. Es sei angemerkt, dass das in [Ekv06] beschriebene System noch weitere Funktionalitäten besitzt, die hier aber nicht weiter betrachtet werden sollen.

Von Meyer, Hollman, Parlitz und Hägele wird in [Mey07] ein System zur Programmierung von Schweiß- und Kleberobotern vorgestellt. Auch hier sind die Roboter mit Kraft-Momenten-Sensoren am Endeffektor ausgestattet. Im Unterschied zu [Ekv06] beginnt hier die Programmierung jedoch online, indem der Roboter mit dem Kraft-Momenten-Sensor entlang der Schweiß- beziehungsweise Klebebahn verfahren wird. Für die finale Robotersteuerung sind die aufgezeichneten Daten allerdings noch zu ungenau. Daher werden diese in einem Offline-System nachbearbeitet bis die geforderten Toleranzen eingehalten werden können. Zusätzlich werden im Offline-System mittels der roboterorientierten Programmierung die Geschwindigkeitsprofile der Roboterbewegung sowie die Ansteuerung der externen Komponenten wie Schweißanlage beziehungsweise Klebepistole implementiert. Gemäß Abschnitt 2.1 kann dieses System als eine Kombination von Online-Programmierung durch

Training und roboterorientierter Offline-Programmierung angesehen werden.

2.3 Offline-Methoden

Charakteristisch für alle Offline-Methoden ist, dass im Prozess der Steuerungsentwicklung der physische Umgang mit dem Robotersystem erst in der Test- und Inbetriebnahmephase erforderlich ist. Gemäß der Klassifikation in Abschnitt 2.1 wird zwischen roboterorientierter und aufgabenorientierter Programmierung unterschieden. Im Folgenden wird eine Anzahl konkreter Offline-Systeme betrachtet und in die Klassifikation eingeordnet.

Bekannte historische Offline-Systeme sind *WAVE*, *SAIL* und *AL*. Sie unterstützten die roboterorientierte Programmierung und wurden bereits in den 1970er Jahren an der Stanford Universität entwickelt. Von besonderer Bedeutung ist das AL-System ([Muj79]). Neben anderen Komponenten stellt dieses System die roboterorientierte Programmiersprache AL (*Assembly Language*) bereit. Alle heutigen herstellerspezifischen roboterorientierten Programmiersysteme sind stark an AL angelehnt.

Wichtige AL-Funktionalitäten, die in praktisch allen heutigen Systemen bereitgestellt werden, sind:

- roboterspezifische Datentypen für die Beschreibung der Roboterposition, zum Beispiel Transformationsmatrizen (Frames),
- Befehle für die logische Verkettung von Koordinatensystemen,
- Befehle für die Programmablaufsteuerung,
- Befehle für die Spezifikation der Roboterbewegung und
- Befehle für die Auswertung und Ansteuerung von externen Sensoren und Aktoren.

Eine der ersten herstellerspezifischen Sprachen nach dem Vorbild von AL war VAL (*Variable Assembly Language*) von Unimation für die PUMA-Roboter (*Programmable Universal Machine for Assembly*) dieser Firma. Beispiele heutiger roboterorientierter Programmiersprachen großer Roboterhersteller sind:

- KRL (KUKA Robot Language),
- AS (Kawasaki-Roboter),
- RAPID (ABB-Roboter).

Trotz des gemeinsamen Vorgängers AL der heutigen herstellerspezifischen Sprachen sind diese in keiner Weise miteinander kompatibel und damit nicht austauschbar. Standardisierungsbemühungen wie beispielsweise die Definition der *Industrial Robot Language* ([Deu93]) blieben bis heute erfolglos.

Eine andere Möglichkeit der roboterorientierten Programmierung besteht in der Nutzung erweiterter allgemeiner Programmiersprachen. Solche Systeme orientieren sich ebenfalls am

Vorbild AL, indem sie um wichtige Funktionalitäten wie roboterspezifische Sprachelemente beziehungsweise Prozeduren erweitert werden. Das Programmiersystem PASRO (*PAScal for RObots*, [Blu87]) ist hierfür ein typisches Beispiel. PASRO ist vollständig in Pascal eingebettet und stellt demzufolge alle charakteristischen Elemente dieser Sprache für die Roboterprogrammierung zur Verfügung. Alternativ wurde auf Basis der Programmiersprache C das System PASRO-C entwickelt.

Im Bereich der roboterorientierten Programmierung finden neben den textbasierten auch grafische Programmiersysteme ihren Einsatz. Bischoff, Kazi und Seyfarth stellen in [Bis02] eine Gestaltungsrichtlinie für die icon-basierte roboterorientierte Programmierung vor (*MORPHA Style Guide*). Danach sind wichtige Bedienfunktionen und Roboterbefehle in Form von Icons bereitzustellen. Durch Verknüpfung der Icons in einem Flussdiagramm kann dann eine Robotersteuerung programmiert werden. Eine prototypische Realisierung dieses Ansatzes erfolgte in Zusammenarbeit mit der KUKA Roboter GmbH.

Im Gegensatz zur roboterorientierten Programmierung, die auf der Spezifikation expliziter Bewegungsbefehle basiert, nutzen aufgabenorientierte Programmiersysteme höhere Beschreibungsformen, um komplexe Aufgaben zu formulieren. Da durch aufgabenbezogene Spezifikationen keine konkreten Bahnkurven, Aktionsfolgen und Ähnliches beschrieben werden, können diese nicht unmittelbar als Steuerung ausgeführt werden, sondern müssen mittels einer sogenannten Aufgabentransformation in ein roboterorientiertes Steuerungsprogramm überführt werden.

Simmons und Apfelbaum berichten in [Sim98] über Forschungsarbeiten zu einem aufgabenorientierten Programmiersystem. Zur Aufgabenbeschreibung wird hier die objektorientierte Sprache TDL (*Task Description Language*) benutzt. Mittels der objektorientierten Paradigmen können die zu spezifizierenden Aufgaben in Teilaufgaben zerlegt werden. Die Darstellung der Relationen zwischen den Teilaufgaben erfolgt in Form eines Aufgabenbaums (engl. *task tree*). Die Transformation der aufgabenorientierten Beschreibung in eine roboterorientierte Steuerung erfolgt durch einen speziellen Compiler, der C++ Code erzeugt. Dieser kann dann zusammen mit einer Programmbibliothek, in der roboterorientierte Funktionen hinterlegt sind, in eine ausführbare Steuerung übersetzt werden.

Ein weiteres Projekt zur aufgabenorientierten Roboterprogrammierung namens IRNApp stellen Kronreif und Fürst in [Kro01] vor. Anders als bei der TDL besteht hier das Ziel in der Bereitstellung einer aufgabenorientierten Umgebung zur Steuerungsentwicklung für mehrere kooperierende Roboter. Die Aufgabenbeschreibung erfolgt mittels einer Skriptsprache. Der Transformationsprozess in ausführbare Roboterbefehle erfolgt inkrementell durch einen Interpreter.

Sowohl beim Projekt TDL als auch bei IRNApp liegt der primäre Fokus auf Anwendungen im Bereich der mobilen Robotik und autonomer Systeme. Über die Anwendung aufgabenorientierter Ansätze in der industriellen Robotik berichten dagegen Ehrmann und Seckner in [Ehr06]. Hier erfolgt die aufgabenorientierte Spezifikation in grafischer Form auf Basis von Aufgabenmodulen, die in einer Bibliothek zur Verfügung stehen. Die Transformation in ein ausführbares Steuerungsprogramm erfolgt durch einen Postprozessor. Zu Demonstrationszwecken wurden Postprozessoren für die herstellerspezifischen Sprachen V+ (Adept Technology GmbH) und KRL (KUKA Roboter GmbH) realisiert.

2.4 Zusammenfassende Bewertung

Am Anfang des Kapitels wurde das Klassifikationsproblem von Roboterprogrammiermethoden für einen weitgefassten Bereich der Industrierobotik behandelt. Anschließend wurden eine Anzahl konkreter Programmiersysteme unter Verwendung der zuvor vorgeschlagenen Klassifikation betrachtet und eingeordnet.

Zusammenfassend kann festgestellt werden, dass die Online-Methoden der Programmierung durch Beispiele – insbesondere Teach-In-Techniken – bereits seit den 1970er Jahren den Stand der Technik für Robotikanwendungen in der Massenfertigung darstellen. Charakteristisch für diese Technik ist, dass der Roboter während der gesamten Steuerungsentwicklung physisch zur Verfügung stehen muss. Der erforderliche Zeitaufwand für die Programmierung ist in der Regel vergleichsweise hoch. Häufig sind umfangreiche Sicherungsvorkehrungen zu treffen, wenn sich der Programmierer im Verlauf der Steuerungsentwicklung im Arbeitsraum des Roboters aufhalten muss. Aufgrund dieser Charakteristik ist diese Art der Online-Programmierung als eigenständige Methode bei komplexen und flexiblen Roboteranwendungen ungeeignet. Im Abschnitt 2.1 wurde aber auch darauf hingewiesen, dass beispielsweise Teach-In-Techniken im Rahmen von Offline-Systemen zum Einsatz kommen können.

Die zweite Klasse der Online-Methoden – die Programmierung durch Training – ist immer noch aktueller Forschungsgegenstand. Perspektivisch kann man davon ausgehen, dass diese Technik nach Erlangung der Praxisreife die Programmierung von Robotern durch Beispiele auch im Bereich der Massenfertigung weitgehend ablösen wird.

Auf dem Gebiet der Offline-Methoden ist zur Zeit die roboterorientierte Programmierung mittels herstellerspezifischer Programmiersprachen weit verbreitet. Diese Technik ist auch für komplexe Roboteranwendungen geeignet und wird heute beispielsweise für die Steuerungsentwicklung kompletter Roboterzellen mit umfangreicher externer Aktorik und Sensorik verwendet. Problematisch ist hier jedoch die geringe Durchsetzung der existierenden Standards. Anwender sind dadurch nicht nur bezüglich des konkret eingesetzten Roboters an einen Hersteller gebunden, sondern auch bezüglich der verwendbaren externen Komponenten. So kann beispielsweise ein KUKA-Roboter durchaus mit einer Kamera sowie dem dazugehörigen Bildverarbeitungssystem, welches auf dem KUKA-Controller mit der restlichen Steuerung in integrierter Form läuft, ausgestattet werden, jedoch wird diese Unterstützung nur für die Komponenten bestimmter kooperierender Anbieter geboten ([Ste08]). Insgesamt muss deshalb festgestellt werden, dass die Möglichkeiten zur herstellerunabhängigen Integration von externer Sensorik und Aktorik in die roboterorientierte Programmierung bei industriellen Anwendungen heute noch unzureichend ist.

Wesentlich günstigere Voraussetzungen für die Integration externer Komponenten bestehen, wenn erweiterte allgemeine Programmiersprachen für die roboterorientierte Programmierung eingesetzt werden. Dies wird in Forschungs- und Hochschulprojekten auch häufig getan, bei industriellen Anwendungen jedoch nur selten.

Für sehr anspruchsvolle Roboteranwendungen wie in flexiblen Fertigungsanlagen für sehr variantenreiche Produktspektren stößt die roboterorientierte Programmierung aufgrund der zu implementierenden umfangreichen Steuerungslogik grundsätzlich an ihre Grenzen.

Herstellerspezifische Programmiersprachen verfügen in der Regel nicht über die erforderlichen Funktionalitäten, um solche Steuerungen zu realisieren. Mit erweiterten allgemeinen Programmiersprachen ist die Umsetzung solcher Steuerungen zwar möglich, aber sehr zeitaufwendig und fehleranfällig. Für diesen Bereich von Roboteranwendungen gewinnen zunehmend die Offline-Methoden der aufgabenorientierten Programmierung kombiniert mit weiteren Techniken an Bedeutung. Diesem Problemkreis widmet sich die vorliegende Arbeit noch weitergehend in den Kapiteln 6 und 7.

3 Entwurf und Inbetriebnahme von Robotersteuerungen

Industrielle Roboteranwendungen können aus steuerungstechnischer Sicht in drei Kategorien eingeteilt werden: *(i) einfache*, *(ii) komplexe* sowie *(iii) flexible Applikationen*. Charakteristisch für einfache Roboteranwendungen ist, das meist keine externen Komponenten vorhanden sind und der Roboter nur eine feste Abfolge von Arm- und Greiferbewegungen wiederholt auszuführen hat. Bei komplexen Roboteranwendungen sind dagegen externe Aktoren und Sensoren vorhanden, die in die Robotersteuerung integriert werden müssen und in der Regel nebenläufig anzusprechen beziehungsweise auszulesen sind. Häufig erfolgt die Abfolge der Roboterbewegungen nicht mehr nach einem starren Zyklus, sondern wird durch Ereignisse in der Roboterumgebung bestimmt. Flexible Roboteranwendungen wie zum Beispiel Montageroboter für variantenreiche Produkte sind durch eine sehr komplexe Steuerungslogik gekennzeichnet. In der Literatur wird hier auch von intelligenten Robotersteuerungen gesprochen.

Roboteranwendungen der beiden letzten Kategorien stellen anspruchsvolle Entwurfs- und Inbetriebnahmeprobleme der Steuerungstechnik dar, die nur im Rahmen eines systematischen Entwicklungsprozesses effizient beherrschbar sind.

In diesem Kapitel soll deshalb zuerst allgemein in die in der Automatisierungstechnik verbreiteten Entwicklungsprozesse eingeführt werden. Der Schwerpunkt liegt dabei auf modernen computergestützten Prozessen unter Nutzung der Simulationstechnik. Anschließend wird sowohl der Bereich der Online- als auch der Offline-Entwicklung von Robotersteuerungen hinsichtlich der gegenwärtig verwendeten Entwurfs- und Inbetriebnahme-Methodik analysiert. Im letzten Abschnitt des Kapitels wird die Entwicklung von Robotersteuerungen auf Basis des SBC-Ansatzes (***S**imulation **B**ased **C**ontrol*) eingeführt und beschrieben.

3.1 Entwicklungsprozesse der Automatisierungstechnik

Ohne computergestützte Entwurfswerkzeuge muss die Realisierung von Automatisierungslösungen direkt am zu regelnden oder am zu steuernden Prozess erfolgen. Ein systematischer Entwurf ist bei dieser Vorgehensweise entweder gar nicht oder nur sehr beschränkt möglich. Damit handelt es sich hier um einen weitgehend heuristischen Ansatz.

Im Bereich der Regelungstechnik ist diese Vorgehensweise inzwischen historisch und wird heute nur noch bei sehr einfachen Problemstellungen mit geringen Güteanforderungen angewandt. Im Bereich der steuerungstechnischen Praxis ist der heuristische Entwurf nach [Abe06] dagegen noch häufig anzutreffen.

Für komplexe automatisierungstechnische Problemstellungen ist jedoch ein systematischer und mehrschrittiger Entwurfs- und Inbetriebnahmeprozess erforderlich. Rechentechnische Unterstützung sowie Methoden der Modellbildung und Simulation spielen dabei eine wichtige Rolle. Ein etabliertes Vorgehensmodell, welches die erforderlichen Schritte von der Aufgabenspezifikation bis hin zur Inbetriebnahme der Gesamtlösung beschreibt, ist das sogenannte *V-Modell*. Dieses Modell wurde ursprünglich für die systematische Durchführung von IT-Projekten entwickelt und durchlief für diesen Bereich bis heute mehrfache Standardisierungsprozeduren. Im Bereich der Automatisierungstechnik wurde das V-Modell in unterschiedlichen Detaillierungen und Ausprägungen zum Beispiel mit Fokus auf regelungstechnische oder steuerungstechnische Probleme übernommen und adaptiert.

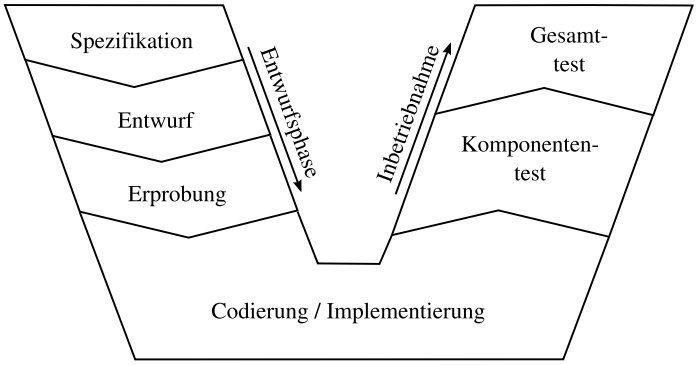


Abbildung 3.1: Realisierung einer Automatisierungslösung nach dem V-Modell

Abbildung 3.1 zeigt eine vereinfachte Darstellung des V-Modells, die sich an [Ort04] anlehnt. Sie soll nachfolgend aus steuerungstechnischer Perspektive erläutert werden. Der linke absteigende Pfad markiert die Entwurfsphase, die mit einer Beschreibung der Steuerungsaufgabe beginnt. Es folgt der Entwurf der Steuerungsalgorithmen sowie deren simulative Erprobung auf dem Entwicklungssystem. Die Entwurfsphase findet ihren Abschluss mit der Codierung der Steuerungsalgorithmen in einer auf der Zielhardware lauffähigen Form. Mit dieser Implementierung des Steuerungsprogramms beginnt die Inbetriebnahme nach dem rechten aufsteigenden Pfad des V-Modells in Form von Komponententests. Den Abschluss der Inbetriebnahme stellt der Test des Steuerungsprogramms am Gesamtprozess dar. Die abgebildeten Hauptentwicklungsschritte bestehen je nach Problemkomplexität aus mehreren Teilschritten. Andererseits können Schritte auch entfallen, beispielsweise der Komponententest bei kleineren Steuerungsproblemen. Darüber hinaus werden die Entwicklungsschritte in realen Projekten nicht nur einmal ausgeführt, sondern es erfolgen Rücksprünge in der Entwicklungskette, um iterativ Korrekturen vornehmen zu können.

Heute werden praktisch alle Schritte des V-Modells mit rechentechnischer Unterstützung durchgeführt. So benutzen zum Beispiel die am Anfang eines größeren Projektes tätigen Planungsingenieure spezialisierte Softwaresysteme zur Spezifikation sowie Simulationssysteme zur Prozessmodellierung und zum Entwurf von Steuerungsstrategien. Mittels Simulationsstudien können verschiedene Steuerungsstrategien erprobt werden. Für die anschlie-

ßende Implementierung der Steuerungsprogramme und deren Inbetriebnahme stehen wiederum rechnergestützte Entwicklungs- und Testumgebungen zur Verfügung, beispielsweise spezialisierte Systeme für die SPS-Programmierung.

3.2 Rapid Control Prototyping (RCP)

Bei dem im vorangegangenen Abschnitt beschriebenen Entwicklungsprozess nach dem V-Modell stehen praxistaugliche Softwarewerkzeuge für alle Entwurfs- und Inbetriebnahmeschritte zur Verfügung. Insbesondere bei großen Projekten ist der Übergang zwischen den einzelnen Entwicklungsschritten und deren Bearbeitern jedoch häufig problematisch, wenn die verwendeten Softwarewerkzeuge nicht ausreichend aufeinander abgestimmt sind. Daten müssen dann unter hohem Zeitaufwand und Fehlerrisiko manuell übertragen und konvertiert werden. Berücksichtigt man, dass der ganze Entwicklungsprozess hochgradig iterativ ist (nach [Joh96] haben 60 bis 80% aller Fehler ihre Ursache in der frühen Spezifikations- und Entwurfsphase, werden aber erst bei der Inbetriebnahme entdeckt), wird klar, welche erheblichen Zeitverluste und damit Kosten sich aus der Übergabeproblematik ergeben können. Zur Entschärfung dieses Problems wurde in einer Vielzahl von Projekten versucht, die Kompatibilität zwischen den eingesetzten Werkzeugen zu verbessern beziehungsweise Werkzeuge zu entwickeln, die den gesamten Entwicklungsprozess von der Spezifikation bis zum operativen Betrieb in geschlossener Form unterstützen.

So berichten Wysk, Joshi und Pegden bereits 1992 über das DARPA-Projekt *Rapid Prototyping of Control Software for Computer Integrated Manufacturing* (*RapidCIM*, [Wys92]) in dem der gesamte Entwicklungsprozess und der anschließende Betrieb von Fertigungsanlagen auf Basis einer Softwareplattform (Arena/SIMAN mit Erweiterungen) erfolgte.

In [Kre03] wird über Arbeiten der Gruppe um Pawletta berichtet, die ebenfalls die Ausführung des gesamten Entwicklungsprozesses sowie des operativen Betriebes von Materialflusssystemen in einer einheitlichen Werkzeugumgebung zum Ziel haben. Als Softwareplattform kommen Matlab sowie zahlreiche Erweiterungsprodukte zum Einsatz. Die Konzeption wird in [Mal08] als *simulationsmodellbasiertes Rapid Control Prototyping* und später in [Sch10] verkürzt als SBC-Ansatz (*Simulation Based Control*) bezeichnet.

Am Institut für Automatisierungstechnik der Universität Rostock widmet sich die Gruppe um Thurow und Stoll dem Entwicklungsprozess für Anwendungen im Bereich der Laborautomation. So wird in [Li 12] ein System vorgestellt, das eine Arbeitsablaufplanung in einem automatisierten Analyselabor vom Entwurf bis zur Inbetriebnahme mittels einer durchgängigen Softwarekette ermöglicht.

Beispiele für weitere Arbeiten mit ähnlicher Zielrichtung sind die von Abel, Bollig, Schloßer und Orth ([Sch03], [Ort04], [Ort05]) sowie die von Morton, Troy und Pizza ([Mor03]).

In ihrem 2006 erschienenen Buch *Rapid Control Prototyping – Methoden und Anwendungen* ([Abe06]) widmen sich Abel und Bollig umfassend dieser Thematik. Sie arbeiten heraus, welche zentralen Voraussetzungen erfüllt sein müssen, um einen zeit- und kosteneffizienten Entwicklungsprozess zu ermöglichen. Darüber hinaus analysieren sie kritisch den tatsächlichen Stand der Technik sowohl im Bereich der Regelungstechnik als auch in der

Steuerungstechnik.

Als erste Voraussetzung für Rapid Control Prototyping (RCP) wird die Lösung der Schnittstellenproblematik zwischen den Entwicklungsschritten genannt. Dies kann entweder durch aufeinander abgestimmte Softwaresysteme (Toolkette) mit kompatiblen Schnittstellen zur automatischen Übernahme und Konvertierung von Daten erreicht werden oder durch integrierte Entwicklungssysteme, wie sie im Bereich der Softwaretechnik bereits seit langem üblich sind. Von besonderer Bedeutung ist für Abel und Bollig dabei der Übergang von der Entwurfs- zur Inbetriebnahmephase. In den konventionellen Entwicklungsprozessen erfolgt hier die Codierung der Regelungs- beziehungsweise Steuerungsalgorithmen nach den Ergebnissen des Entwurfs auf der Zielhardware (z.B. Mikro-Controller, SPS oder Industrie-PC) auf dem Niveau der größten Detaillierung. Die manuelle Ausführung dieser Codierung ist demzufolge häufig der zeit- und kostenintensivste sowie fehleranfälligste Schritt im gesamten Entwicklungsprozess. Für Abel und Bollig ist deshalb eine *automatische Codegenerierung* eine zwingende Voraussetzung für RCP. Als zweite wesentliche Anforderung wird die Unterstützung weiterer Simulationstechniken genannt, namentlich der *Software-in-the-Loop Simulation (SiL)* und der *Hardware-in-the-Loop Simulation (HiL)*.

Wie im vorangegangenen Abschnitt ausgeführt, spielt die Simulation bereits in den bisherigen Entwicklungsprozessen nach dem V-Modell eine wichtige Rolle. Ihr Einsatz ist dort aber auf die Entwurfsphase beschränkt. Sowohl Regler beziehungsweise Steuerung als auch der Prozess sind als Modelle auf dem Entwurfssystem vorhanden und ihr Zusammenspiel kann simulativ erprobt werden. Zur klaren begrifflichen Abgrenzung wird diese Simulationstechnik als *Systemsimulation* bezeichnet.

Bei der *SiL-Simulation* wird der Regel- beziehungsweise Steuerungsalgorithmus wie bei der Systemsimulation auf dem Entwurfssystem ausgeführt, allerdings nunmehr in Realzeit. Über geeignete Schnittstellen bestehen Verbindungen zwischen dem Entwurfssystem und den Aktoren und Sensoren des realen Prozesses. Auf diese Weise können bereits die Entwürfe von Reglern und Steuerungen am realen Prozess erprobt werden, bevor diese überhaupt auf einer Zielhardware implementiert wurden.

Die *HiL-Simulation* ist eine zur SiL komplementäre Technologie. Wie in der Systemsimulation liegt der Prozess als Modell auf dem Entwurfssystem vor und wird dort ausgeführt. Der Regler beziehungsweise die Steuerung ist jedoch bereits auf der Zielhardware implementiert. Die Zielhardware ist mit dem Entwurfssystem so verbunden, dass sie im Prozessmodell sensorische und aktorische Variablen in Realzeit lesen und schreiben kann. Diese Technologie ist insbesondere in Situationen hilfreich, in denen der Prozess noch gar nicht real existiert, sondern nur im Entwurf vorliegt. Für den späteren Einsatz geplante Regel- und Steuerungseinrichtungen können so bereits zu einem sehr frühen Zeitpunkt auf Eignung geprüft werden.

Abel und Bollig bezeichnen SiL und HiL als RCP-Strukturen mit denen im Entwicklungsprozess nach dem V-Modell auch horizontale Iterationen ermöglicht werden. Abbildung 3.2 soll diesen Aspekt veranschaulichen.

Für die erläuterten Technologien werden in der Literatur auch zahlreiche synonyme Bezeichnungen verwendet, wie zum Beispiel *Soft-Commissioning*, *Reality-in-the-Loop* ([Aui99]) und *virtuelle Inbetriebnahme* ([Kai08]). In der regelungstechnischen Literatur werden vor-

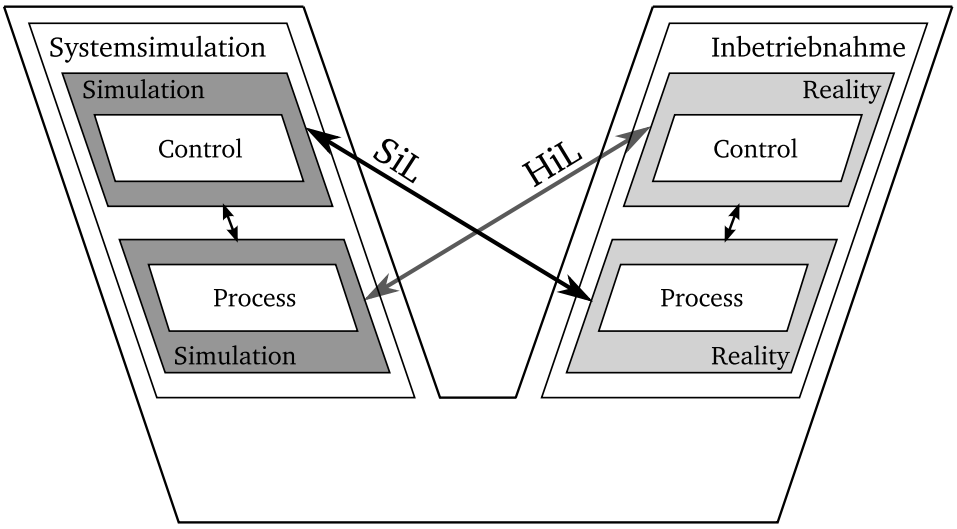


Abbildung 3.2: Einordnung von SiL und HiL in einen Entwicklungsprozess nach dem V-Modell

wiegend die Bezeichnungen SiL und HiL benutzt, häufig aber mit genau umgekehrter Bedeutung als hier beschrieben.

Weiter oben wurde bereits erwähnt, dass Abel und Bollig in [Abe06] auch den tatsächlichen Stand der Entwicklungsprozesse in der Praxis kritisch analysieren. Ihre Ergebnisse von 2006 dürften zumindest in der Tendenz für den heutigen Stand immer noch zutreffend sein. Das heißt, für viele regelungstechnische Anwendungen stehen Werkzeugketten bereit, die die zentralen RCP-Techniken wie SiL, HiL und automatische Codegenerierung unterstützen. In innovativen Anwendungsfeldern wie der Automobilindustrie werden diese Techniken in der Praxis inzwischen auch erfolgreich eingesetzt. Für den Bereich der steuerungstechnischen Praxis wird allerdings konstatiert, dass hier die Entwicklungsprozesse noch weitgehend heuristisch durchgeführt werden. Die Ursache hierfür liegt weniger in einer mangelhaften Bereitstellung geeigneter Softwarewerkzeuge, sondern in der formalen Spezifikation und Modellierung des ungesteuerten Prozessverhaltens. Zu regelnde Prozesse sind meist kontinuierlicher Natur. Ihre Modellierung durch Differentialgleichungen ist daher unproblematisch. Ein zu steuernder Prozess ist dagegen ereignisdiskreter Natur. Modellierungsmethoden für diese Systemklasse existieren zwar auch, ihre Anwendung im Bereich der Steuerungstechnik ist aber noch begrenzt. Häufig wird auf eine Prozessmodellierung noch gänzlich verzichtet und die Steuerungsentwicklung erfolgt direkt am Prozess.

Aus den unterschiedlichen Modellierungsmethoden ergeben sich nach [Abe06] weitere wesentliche Unterschiede zwischen der regelungstechnischen und steuerungstechnischen Praxis. Für Regelungen linearer Systeme existieren eine Vielzahl von Entwurfsverfahren, die einen konkreten Reglerentwurf als Ergebnis zur Folge haben. Durch die formalisierte Darstellung des Übertragungsverhaltens von Regler sowie Regelstrecke eröffnet sich außerdem die Möglichkeit, die an den Regelkreis gestellten Forderungen, zum Beispiel nach Stabilität,

sicherzustellen. Im Bereich der Steuerungstechnik ist ein vergleichbarer Nachweis schwierig. Die Spezifikation von Steuerungszielen und die Modellierung von Steuerungen und zu steuernden Prozessen sind heute zwar möglich, aber in der Praxis noch nicht verbreitet. Deshalb ist auch eine Verifikation, also die Beweisführung, dass das aus Steuerung und Prozess bestehende Gesamtsystem der Spezifikation genügt, derzeit immer noch Forschungsgegenstand.

3.3 Online-Entwicklung einfacher Robotersteuerungen

Nach Kapitel 2 ist bei den Online-Methoden der Roboterprogrammierung zwischen den Klassen Programmierung durch Beispiele und Programmierung durch Training zu unterscheiden.

Die Programmierung durch Beispiele – insbesondere die Teach-In-Technik – ist heute noch dominierend für einfache Roboteranwendungen. Die Entwicklung der Steuerung erfolgt ausschließlich unter direkter Verwendung des Roboters – also unmittelbar am Prozess. Damit ist diese Technik nach Abschnitt 3.1 eindeutig als heuristische Steuerungsentwicklung einzuordnen.

Die zweite Online-Methodenklasse der Programmierung durch Training wird mit einiger Wahrscheinlichkeit die Programmierung durch Beispiele bei einfachen Roboteranwendungen in der Zukunft verdrängen. Im Kapitel 2 wurde aber bereits ausgeführt, dass diese Methoden zur Zeit immer noch Forschungsgegenstand sind. Es zeichnet sich jedoch ab, dass sie Bestandteile künftiger Offline-Systeme sein werden, die einen systematischen modellbasierten Entwurfs- und Inbetriebnahmeprozess unterstützen.

3.4 Offline-Entwicklung komplexer Robotersteuerungen

Gemäß Kapitel 2 wird bei der Offline-Programmierung von Robotern zwischen der roboterorientierten und der aufgabenorientierten Programmierung unterschieden. Im Abschnitt 2.3 wurde erläutert, dass ein aufgabenorientiertes Steuerungsprogramm zur Ausführung immer in roboterorientierte Steuerungsanweisungen überführt wird. Aufgrund dieser Tatsache konzentriert sich die Betrachtung der Offline-Programmierung bezüglich des zugrundeliegenden Entwicklungsprozesses in diesem Abschnitt auf die roboterorientierte Programmierung. Auf weitergehende Betrachtungen zur aufgabenorientierten Programmierung sei an dieser Stelle auf das Kapitel 6 verwiesen.

Die Entwicklung komplexer Roboteranwendungen beginnt mit der Spezifikation der Steuerungsaufgaben. Dabei ist der Roboter und seine gesamte Umgebung zu berücksichtigen, beispielsweise eine komplette Roboterzelle. Die ersten Entwurfsschritte werden bei komplexen Problemen in der Regel durch Planungsingenieure ausgeführt. Dabei wird der einzelne Roboter als Element eines Materialflusssystems betrachtet und nach geeigneten Steuerungsstrategien gesucht. Auf dieser Ebene kommen heute diskret-ereignisorientierte Simulationssysteme zum Einsatz, mit denen die Leistungsfähigkeit verschiedener Steue-

rungsstrategien überprüft werden kann. Nach Abschnitt 3.2 handelt es sich dabei um Systemsimulationen, dass heißt sowohl die Steuerungsalgorithmen als auch der Prozess werden auf dem Entwurfssystem modelliert und ausgeführt. Auf der Ebene der Planungssimulation ist eine Trennung von Steuerung und Prozess jedoch nicht zwingend notwendig. Ebenfalls handelt es sich in dieser Phase noch um Steuerungsstrategien – nicht um konkrete Robotersteuerungsanweisungen.

Die Erstellung der konkreten Robotersteuerungen erfolgt anschließend durch Steuerungsspezialisten. In der Industrierobotik werden dazu die herstellerspezifischen Offline-Systeme benutzt in denen die Steuerung unter Verwendung einer roboterorientierten Programmiersprache implementiert wird. Bei diesem Schritt ist die größte Detaillierung erreicht und es erfolgt der Übergang von der Entwurfs- zur Inbetriebnahmephase.

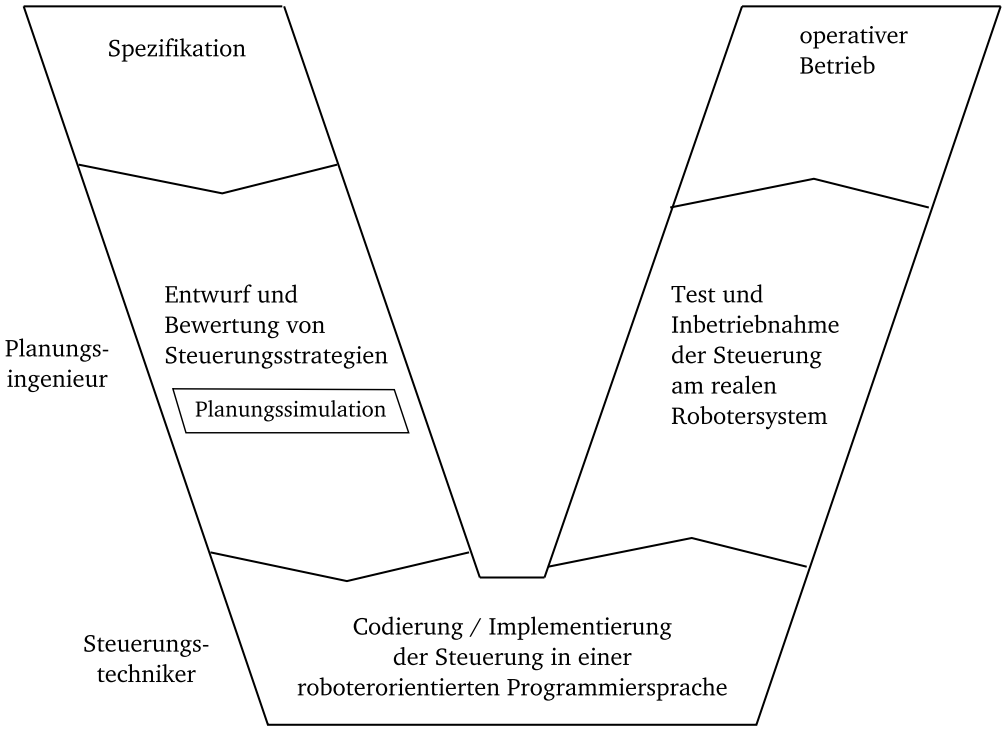


Abbildung 3.3: Entwicklungsprozess einer komplexen Robotersteuerung

Abbildung 3.3 zeigt, dass der beschriebene Entwicklungsprozess der systematischen Vorgehensweise gemäß dem V-Modell entspricht. Kritisch ist allerdings der Übergang von der Planungssimulation zur Implementierung der konkreten Robotersteuerung. Im Ergebnis der Planungssimulation liegt die zu realisierende Steuerungsstrategie vor und wird vom Planungsingenieur an den Steuerungstechniker übergeben. Letzterer hat diese mittels einer roboterorientierten Programmiersprache zu implementieren. Einerseits besteht das Potential einer fehlerhaften Umsetzung der Steuerungstrategie in ein konkretes Steuerungsprogramm und zum anderen sind die Möglichkeiten der Steuerungserprobung trotz Offline-Programmierung begrenzt. Ohne Verwendung des realen Roboters beschränkt sich die Er-

probung des Steuerungsprogramms im Offline-System im Wesentlichen auf die Prüfung der syntaktischen Korrektheit des Robotersteuerungsprogrammes.

Betrachtet man den Entwicklungsprozess aus der engeren Perspektive des Steuerungstechnikers, ist dieser sogar als heuristisch einzuordnen. Ein auf dem Entwurfssystem ausführbares Robotermodell liegt nicht vor und damit ist die simulative Erprobung der Robotersteuerung ausgeschlossen. Somit verbleibt nur der Test der Steuerung am realen Robotersystem.

Die Klassifikation im Kapitel 2 differenziert bei der roboterorientierten Programmierung zwischen der Verwendung herstellerspezifischer und allgemeiner erweiterter Programmiersprachen. Bezüglich des Entwicklungsprozesses ergeben sich daraus aber keine Unterschiede. Das Robotersteuerungsprogramm ist jeweils manuell durch den Steuerungstechniker zu erstellen. Im Sinne von Abschnitt 3.2 kann dies als eine manuelle Codegenerierung angesehen werden.

Wie bereits im Abschnitt 2.3 ausgeführt, werden in der industriellen Robotik zur Implementierung von Steuerungsprogrammen fast ausschließlich herstellerspezifische Sprachen verwendet. Diese können auf dem Steuerrechner des Robotersystems direkt zur Ausführung gebracht werden. Bei der Verwendung einer allgemeinen erweiterten Programmiersprache ist zwar eine Übersetzung in ein ausführbares Programm notwendig, da als Zielhardware bei Industrierobotern aber ausschließlich Industrie-PCs verwendet werden, erfolgt dies durch einen gewöhnlichen Compiler. Hierbei handelt es sich nicht um eine automatische Codegenerierung im Sinne des im Abschnitt 3.2 beschriebenen RCP.

Zur Verbesserung der Offline-Erprobung von konkreten Robotersteuerungen werden seit längerem Robotersimulationssysteme zur Verfügung gestellt. Wie bei den roboterorientierten Programmiersprachen handelt es sich dabei in erster Linie um untereinander inkompatible herstellerspezifische Produkte, beispielsweise:

- *KUKA.Sim* von der KUKA Roboter GmbH ([Kuk09]),
- *PC-Roset* von Kawasaki-Robotics ([Kaw09]) und
- *RobotStudio* von ABB-Robotics ([Abb09]).

In der Literatur wird für diese Softwaresysteme der Begriff *Computer Aided Robotics* (CAR, [Mö96]) verwendet. CAR-Systeme verfügen über Bibliotheken, in denen Robotermodelle der jeweiligen Hersteller bereitgestellt werden, sowie über Schnittstellen zu 3D-CAD-Systemen. Damit wird der Aufbau realitätsnaher Modelle von Robotern und ihrer gesamten Umgebung unterstützt. Als Steuerungsmodelle werden in CAR-Systemen unmittelbar die in einer roboterorientierten Programmiersprache implementierten Steuerungen benutzt. Dem Steuerungstechniker wird es somit ermöglicht, ein konkretes Robotersteuerungsprogramm mittels einer Systemsimulation zu erproben und gegebenenfalls zu korrigieren und zu verbessern.

Im Unterschied zu den Systemsimulationen des Planungsingenieurs zum Entwurf geeigneter Steuerungsstrategien, spielt die 3D-Visualisierung der Simulationsergebnisse auf der Ebene eines CAR-Systems eine wesentliche Rolle. Sie ist beispielsweise eine Voraussetzung

für die effiziente Bahnplanung von Roboterbewegungen sowie zur Darstellung eventueller Kollisionen, die im Rahmen der Simulation automatisch erkannt werden.

Abbildung 3.4 veranschaulicht die Einordnung von CAR-Systemen in den Gesamtentwicklungsprozess komplexer Robotersteuerungen. Im Vergleich zu der in Abbildung 3.3 dargestellten Vorgehensweise wird deutlich, dass die Implementierung der konkreten Steuerung sowie deren modellbasierte Erprobung nun eindeutig der Entwurfsphase zuzuordnen ist.

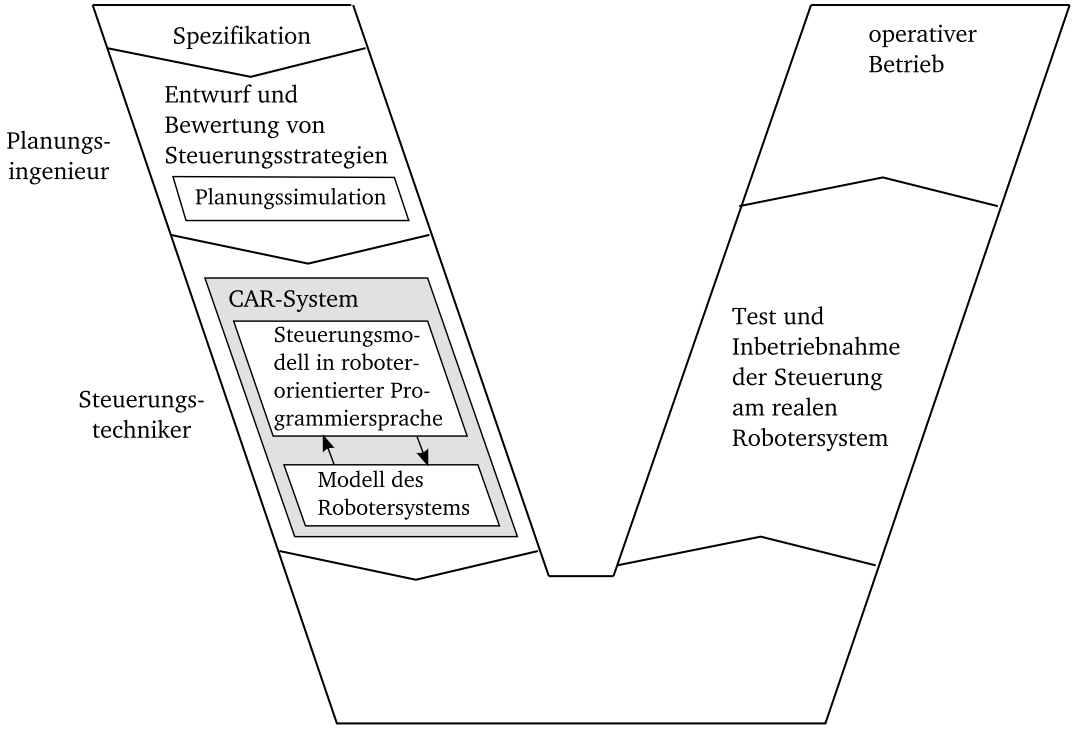


Abbildung 3.4: Entwicklungsprozess bei Verwendung eines CAR-Systems

Darüber hinaus fällt in Abbildung 3.4 auf, dass der Übergang von der Entwurfs- zur Inbetriebnahme quasi „leer“ erscheint. Im allgemeinen V-Modell nach Abschnitt 3.1 ist hier die Codierung der Steuerungsalgorithmen auf der Zielhardware angesiedelt. Die korrekte Interpretation dieses Aspekts der Abbildung 3.4 ist stark kontextabhängig.

Aus der engeren Perspektive des Steuerungstechnikers stellt die vom Planungsingenieur übergebene und zu realisierende Steuerungsstrategie die Aufgabenspezifikation dar. Danach wird die konkrete Steuerung unter Nutzung des CAR-Systems simulativ entworfen und erprobt. Beim nachfolgenden Übergang zur Inbetriebnahmephase ist *keine* explizite Codierung der Steuerungsalgorithmen für die Zielhardware erforderlich, weil das Modell der Steuerung aus der Entwurfsphase ein in einer roboterorientierten Sprache implementiertes Programm ist, welches ohne weitere Bearbeitung auf dem Steuerrechner des Robotersystems ausführbar ist. Im Sinne des RCP kann dies als eine implizite automatische Form der Codegenerierung aufgefasst werden.

Aus der Perspektive des Gesamtentwicklungsprozesses wie sie beispielsweise vom Planungsingenieur wahrgenommen wird, sind dagegen die Voraussetzungen für RCP nicht erfüllt. Es ist keine Durchgängigkeit der Softwarekette vorhanden, weil die Ergebnisse der Planungssimulation manuell an den Steuerungstechniker übergeben werden. Aus dieser Perspektive erscheint auch die Erstellung des Steuerungsprogramms unter Zuhilfenahme eines CAR-Systems nicht als automatischer, sondern als manueller Vorgang.

Nach den bisherigen Betrachtungen scheint die explizite automatische Codegenerierung, wie sie beispielsweise in [Abe06] als eine zentrale Voraussetzung für RCP genannt wird, in der industriellen Robotik keine Rolle zu spielen. Dies ist nur bedingt richtig. Der bisherige Fokus lag auf klassischen Industrierobotern bei denen wie bereits erwähnt fast ausnahmslos Industrie-PCs als Steuerungshardware eingesetzt werden, auf denen in roboterorientierten Sprachen implementierte Steuerungsprogramme direkt ausgeführt werden können. Neben den klassischen stationären Robotern spielen in industriellen Anwendungen aber auch zunehmend mobile Roboter, beispielsweise als intelligente Transporteinheiten, eine Rolle. Hier werden als Steuerungshardware in der Regel spezielle Controller eingesetzt, so dass sich beim Übergang vom Entwurfssystem auf das Zielsystem das Problem der expliziten Codegenerierung stellt.

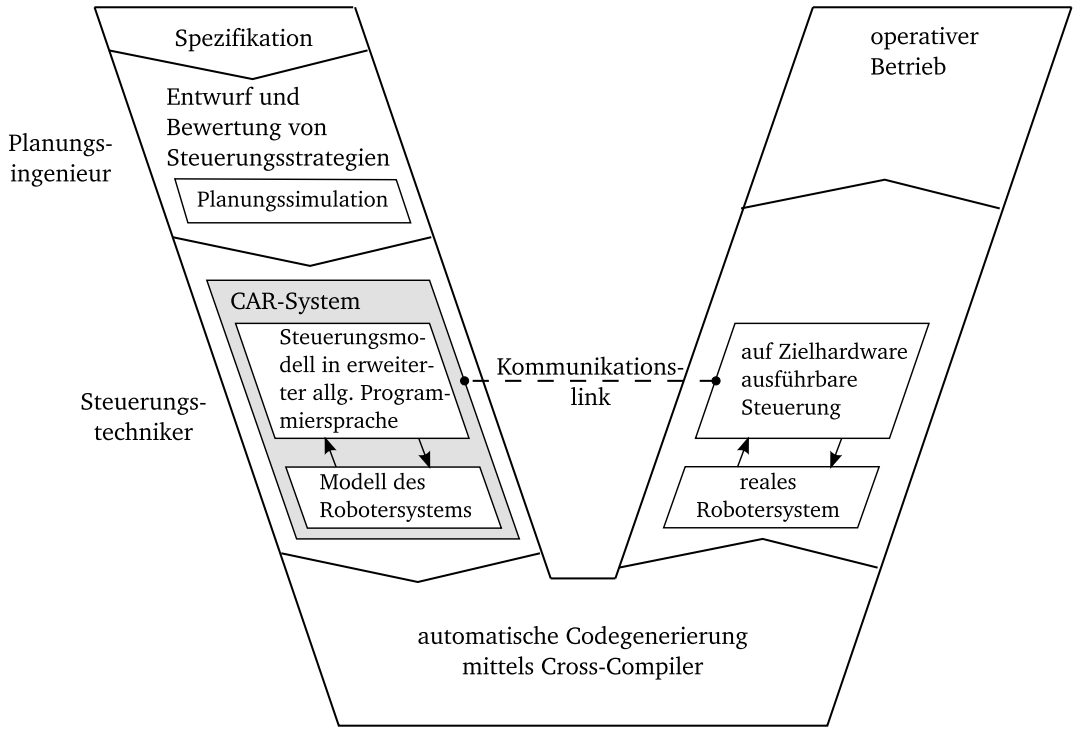


Abbildung 3.5: Entwicklungsprozess bei Robotersystemen mit spezieller Steuerungshardware

Abbildung 3.5 veranschaulicht den Entwicklungsprozess für Robotersysteme mit spezieller Steuerungshardware. Wie bei stationären Industrierobotern werden auch hier CAR-

Systeme zum Entwurf und zur simulativen Erprobung der konkreten Steuerung benutzt. Allerdings werden bei diesen Systemen in der Regel erweiterte allgemeine Programmiersprachen zur Implementierung des Steuerungsmodells verwendet. Beim Übergang zur Inbetriebnahmephase muss das Steuerungsmodell übersetzt und ausführbarer Code für die Zielhardware erzeugt werden. Dieser Schritt erfolgt automatisch durch einen sogenannten Codegenerator, im Kern handelt es sich dabei um einen Cross-Compiler.

Zur Unterstützung der Inbetriebnahme existiert häufig eine Kommunikationsmöglichkeit zwischen der Steuerung auf der Zielhardware und dem CAR-System. Über diese Verbindung können im Testbetrieb sensorische Größen im CAR-System angezeigt werden und umgekehrt auch Parameter der Steuerung verändert werden. Das CAR-System agiert dabei nicht mehr als Simulationssystem, sondern als Hilfssystem zur Inbetriebnahme (Monitor-System).

Prinzipiell wäre über die Kommunikationsverbindung auch die Durchführung von SiL- und HiL-Simulationen denkbar. In den heutigen CAR-Systemen werden diese RCP-Techniken jedoch nur sehr rudimentär beziehungsweise gar nicht unterstützt.

3.5 Entwicklung von Robotersteuerungen nach dem SBC-Ansatz

Die Betrachtung der gegenwärtig in der industriellen Robotik üblichen Entwicklungsprozesse in den beiden vergangenen Abschnitten zeigt, dass in diesem Anwendungsgebiet der Steuerungstechnik noch nicht von einer umfassenden Durchsetzung von RCP-Prozessen gesprochen werden kann. Anliegen dieses Abschnitts der Arbeit ist es deshalb, einen konkreten RCP-Ansatz, der sich bereits in anderen steuerungstechnischen Anwendungsfeldern bewährt hat, in seinen Grundzügen vorzustellen und konzeptionell auf das Gebiet der industriellen Robotik zu übertragen.

Im Abschnitt 3.2 wurde auf Arbeiten hingewiesen, die sich bereits seit den 1990er Jahren mit der durchgängigen Unterstützung von Entwicklungsprozessen für komplexe Steuerungsprobleme befassen. Dazu gehören auch eine Reihe von Projekten, die seit 1994 in einer Kooperation des Instituts für Automatisierungstechnik der Universität Rostock und der Forschungsgruppe Computational Engineering und Automation der Hochschule Wismar durchgeführt wurden ([Sch01], [Paw02], [Paw11]). Im Rahmen dieser Arbeiten wurde ein RCP-Konzept entwickelt, welches zur Erreichung der Durchgängigkeit von Entwicklungsprozessen von der Spezifikation bis zum operativen Betrieb primär auf eine gemeinsame Softwareplattform für alle erforderlichen Phasen abstellt. Darüber hinaus orientiert sich dieses Konzept stark an RCP-Prozessen, die sich bereits im regelungstechnischen Bereich beziehungsweise für kontinuierliche Systeme bewährt haben. In [Mal08] wird dieses Konzept erstmals als *simulationsmodellbasiertes Rapid Control Prototyping*[1] bezeichnet. In dieser

[1] Die Bezeichnung soll darauf hinweisen, dass im Unterschied zu anderen steuerungstechnischen RCP-Ansätzen, hier das Prozessmodell als Bestandteil der Steuerung auch im operativen Betrieb erhalten bleibt. Auf dieser Basis ist die Realisierung von Zustandsbeobachtern, Diagnoseverfahren und ähnlichen

Arbeit wird im Folgenden bevorzugt die verkürzte Bezeichnung SBC-Ansatz (*Simulation Based Control*) verwendet.

Im Abschnitt 3.2 wurde auch betont, dass selbst ein modellgestützter Steuerungsentwurf in der steuerungstechnischen Praxis noch nicht als Stand der Technik angesehen werden kann. In diesem Zusammenhang sei an dieser Stelle erwähnt, dass sich die vorliegende Arbeit diesem Problemfeld widmet. Ein Schwerpunkt besteht in der Simulation und Validierung von Steuerungsentwürfen – Verfahren zur Verifikation werden jedoch nicht betrachtet.

Im Abschnitt 3.4 wurde gezeigt, dass der Gesamtentwicklungsprozess einer Roboteranwendung, die wiederum Bestandteil einer größeren Fertigungsanlage ist, eine mehrstufige Entwurfsphase erfordert. In allen Stufen ist die Spezifikation und Modellierung ereignisdiskreter[2] Problemgegenstände notwendig. Eine Ursache für die bisher fehlende Durchgängigkeit und damit für notwendige manuelle Übergaben zwischen den Entwurfsstufen ist, dass innerhalb der Stufen verschiedene Simulationssysteme mit unterschiedlichen Beschreibungsmitteln eingesetzt werden. Einen Überblick über die üblichen Beschreibungsmittel für diskret-ereignisorientierte Systeme gibt die Abbildung 3.6.

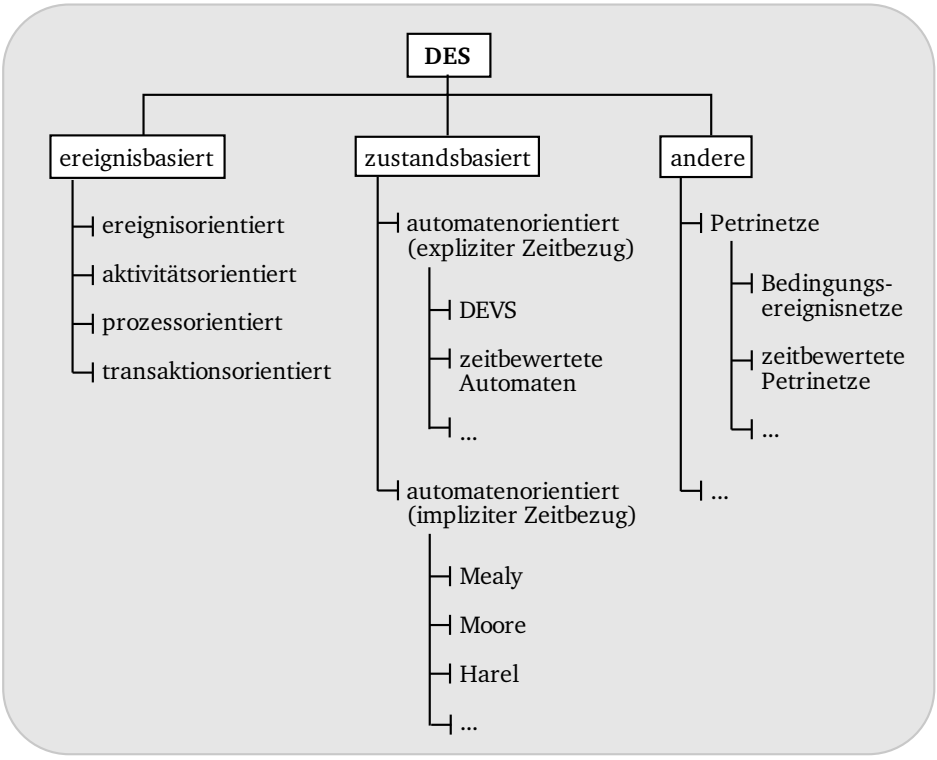


Abbildung 3.6: Beschreibungsmittel für diskret-ereignisorientierte Systeme nach [Ste09]

Ansätzen möglich, wie sie bereits von zu regelnden kontinuierlichen Systemen bekannt sind.

[2]In dieser Arbeit werden die Begriffe *ereignisdiskret* und *diskret-ereignisorientiert* in synonymer Bedeutung verwendet.

Der Versuch, eine Durchgängigkeit im Entwurfsprozess durch Verwendung nur eines Beschreibungsmittels und damit eines Simulationssystems in allen Schritten des Entwurfs herzustellen, wäre nicht zielführend, weil in den verschiedenen Stufen unterschiedliche Anforderungen an die Beschreibungsmittel gestellt werden. So erfordern die anfänglichen Planungssimulationen des Materialflusses im Entwurfsprozess geeignete ereignisbasierte Beschreibungsmittel zur transaktions-, prozess- oder aktivitätsorientierten Modellierung. In den weiteren Entwurfsschritten sind dann Steuerungsstrategien aufzustellen und zu verfeinern. Am Ende des Entwurfsprozesses sind schließlich konkrete Steuerungsalgorithmen vollständig getrennt vom materialflusstechnischen Prozess zu modellieren. Hierzu sind vor allem zustandsbasierte Beschreibungsmittel geeignet.

Um eine Durchgängigkeit des gesamten Entwurfsprozesses auf einer einzigen Softwareplattform zu erzielen, muss diese also die Modellierung diskret-ereignisorientierter Systeme mit einem breiten Beschreibungsmittelspektrum unterstützen. Darüber hinaus müssen die Beschreibungsmittel weitgehend in Kombination miteinander verwendbar sein, damit die Modelle von einem Entwurfsschritt zum anderen übertragen werden können und innerhalb der Schritte der Detaillierungsgrad der Modelle sukzessive erhöht werden kann. Eine bekannte Entwicklungsumgebung, die diese Forderungen weitgehend erfüllt, ist das System *Matlab*. Zusammen mit den relevanten Erweiterungskomponenten wird dieses System im Kapitel 4 als RCP-Plattform näher betrachtet.

Nachfolgend sollen der Entwurfsprozess einer komplexen Roboteranwendung nach dem SBC-Ansatz sowie der kritische Übergang von der Entwurfs- zur Inbetriebnahmephase besprochen werden.

Wie bereits im Abschnitt 3.4 erläutert, wird ein einzelner Roboter in den ersten Schritten des Entwurfs einer größeren Fertigungsanlage vom Planungsingenieur gewöhnlich als Transport- oder Bedienelement eines Materialflusssystems aufgefasst. Im SBC-Ansatz wird davon ausgegangen, dass die RCP-Plattform die transaktionsorientierte Modellierung oder andere ereignisbasierte Beschreibungsmittel zum Aufbau eines Materialflussmodells, wie in Abbildung 3.7 dargestellt, unterstützt. Im Rahmen der Planungssimulation werden mit

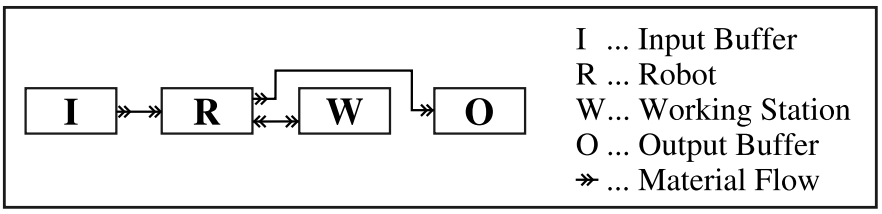


Abbildung 3.7: Prinzipdarstellung eines Materialflussmodells mit einem Roboter als Transportelement

solchen Modellen geeignete Topologien und Dimensionierungen des Materialflusssystems sowie bereits grundsätzliche Steuerungsstrategien entworfen und hinsichtlich ihrer Leistungsfähigkeit überprüft. Wie aus Abbildung 3.7 ersichtlich, sind die Materialflussmodelle modular aufgebaut und bilden damit bereits die wesentlichen realen Prozesselemente als

Komponenten ab.

Nach Abschluss der Planungsphase wird der Modellstand an den Steuerungstechniker übergeben. Die im Rahmen der Planungssimulation festgelegte Steuerungsstrategie ist in impliziter Form im Materialflussmodell abgebildet. Der erste Schritt in der nun beginnenden Automatisierungsphase besteht in der Separierung von Steuerungs- und Materialflussebene, wie in Abbildung 3.8 dargestellt.

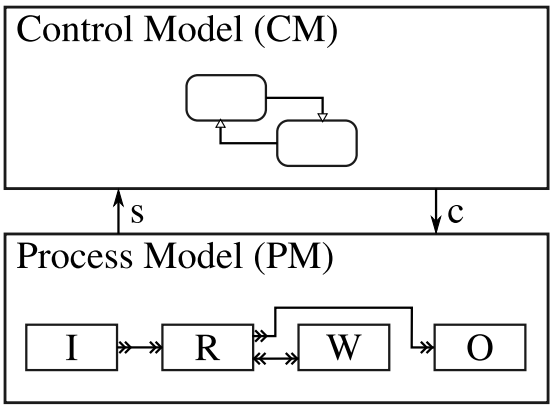


Abbildung 3.8: Separierung des Entwurfsmodells in Steuerungs- und Prozessebene

Im Steuerungsmodell (CM) wird die konkrete Steuerungslogik zur Umsetzung der vorgegebenen Steuerungsstrategie implementiert. Dies erfolgt zweckmäßiger Weise unter Verwendung eines zustandsbasierten Beschreibungsmittels. Besonders geeignet sind hierfür Zustandsdiagramme nach Harel ([Har87]), weil diese für komplexe Problemstellungen auch den Aufbau modularer und hierarchischer Steuerungsmodelle erlauben.

Das Prozessmodell (PM) ist in gleicher Weise wie das Planungsmodell gemäß den realen Prozesselementen strukturiert und bildet die ereignisdiskrete Prozessdynamik sowie den Materialfluss zwischen den Prozesselementen ab.

Die Schnittstelle zwischen Steuerungs- und Prozessmodell wird über die *s* - und *c* - *Größen* realisiert. Die s - Größen[3] sind steuerungsrelevante Informationen aus dem Prozessmodell, wie Zustandsgrößen oder Ereignisse. Die c - Größen[4] sind aktorische Informationen des Steuerungsmodells an das Prozessmodell. Auf diese Art und Weise können prozessgeführte Steuerungen gemäß dem SBC-Ansatz im ersten Schritt der Automatisierungsphase modelliert und mittels Systemsimulationen erprobt werden. Die Dynamik der aktiven Prozesselemente, wie zum Beispiel die des Roboters (R) und der Bearbeitungsstation (W) in Abbildung 3.8, basiert im Wesentlichen auf den aus der Planungsphase übernommenen Bedienzeiten. Die Komponenten des Prozessmodells besitzen im ersten Schritt der Automatisierungsphase damit noch *generischen* Charakter, das heißt sie sind noch nicht von der konkreten Sensor-/Aktorschnittstelle der im realen Prozess verwendeten herstellerspezifischen Prozesselemente abhängig.

[3]steht hier für engl. *state value* oder *signal* (Ereignis).
[4]steht hier für engl. *control value*.

Abbildung 3.9 zeigt den anschließenden zweiten Schritt der Automatisierungsphase, in der das Prozessmodell bis auf die Sensor-/Aktorebene der im Prozess verwendeten herstellerspezifischen Prozesselemente zu detaillieren ist. Der bisherige Modellstand der Steuerungslogik (CM) wird dabei *ohne Änderung* übernommen. Der bisherige Modellstand für die ereignisdiskrete Prozessdynamik (PM) wird ebenfalls als *PM-High-Level-Ebene* weiter verwendet. In den Modellkomponenten der aktiven Prozesselemente werden nunmehr die bisherigen Bedienzeiten jedoch durch eine c/s - Schnittstelle zu einer *PM-Low-Level-Ebene* ersetzt. Die PM-LL-Komponenten modellieren die herstellerspezifischen Eigenschaften der konkret verwendeten Prozesselemente und sind ihrerseits in der Lage auf die jeweilige Sensor-/Aktorebene über ein Prozessinterface abzubilden. Im SBC-Ansatz wird davon ausgegangen, dass für die herstellerspezifischen Prozesselemente physikalische Modelle verfügbar sind. Im Kontext von Roboterapplikationen sind dies die im Abschnitt 3.4 besprochenen CAR-Systeme. Damit ist die Erprobung mittels Systemsimulationen auch im zweiten Schritt der Automatisierungsphase möglich.

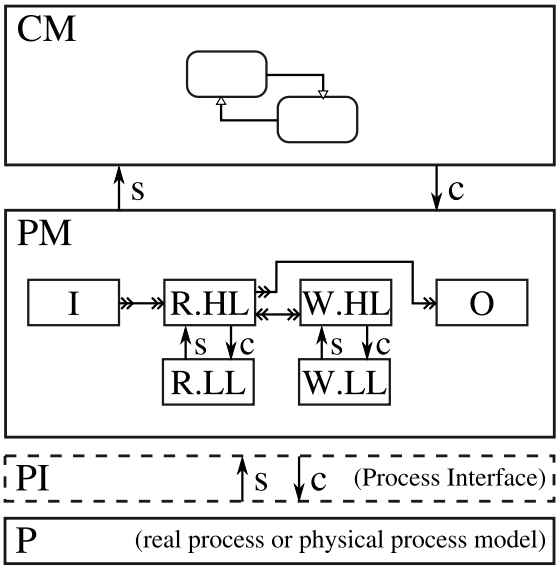


Abbildung 3.9: SiL beziehungsweise operativer Betrieb nach dem SBC-Ansatz

Wegen der prozesselementeorientierten Strukturierung des Prozessmodells ist nach erfolgreicher Systemsimulation ein inkrementeller Übergang zur Inbetriebnahme mittels SiL-Simulation in Bezug auf einzelne reale Prozesselemente möglich. Umfasst die SiL-Simulation schließlich alle realen Prozesselemente und liefert bezüglich der Anforderungsspezifikation befriedigende Ergebnisse, so kann die letzte Konfiguration der SiL als ausgetestete Steuerung für den operativen Betrieb benutzt werden.

In Bezug auf industrielle Roboteranwendungen weist die hier dargestellte Anwendung des SBC-Ansatzes eine besondere Spezifik auf. Weiter oben wurde erläutert, dass die PM-LL-Komponenten auf die konkrete Sensor-/Aktorschnittstelle der realen Prozesselemente abbilden. Im Falle einer Roboterkomponente ist dies die durch eine herstellerspezifische

Roboterprogrammiersprache definierte Schnittstelle. Wie schon im Abschnitt 3.4 im Zusammenhang mit Abbildung 3.4 ausgeführt wurde, ist ein Steuerungsmodell auf Basis einer roboterorientierten Programmiersprache wegen der fast ausnahmslosen Verwendung von Industrie-PCs als Steuerungshardware für Industrieroboter ohne Änderungen als Steuerungssoftware verwendbar. Die Steuerungsentwicklung für Industrieroboteranwendungen nach dem SBC-Ansatz realisiert damit ebenso die bereits im Abschnitt 3.4 angesprochene implizite automatische Codegenerierung.

3.6 Zusammenfassende Bewertung

Am Anfang des Kapitels wurde allgemein in die in der Automatisierungstechnik verbreiteten Entwicklungsprozesse eingeführt. Anschließend wurde sowohl der Bereich der Online- als auch der Offline-Entwicklung von Robotersteuerungen hinsichtlich der gegenwärtig verwendeten Entwurfs- und Inbetriebnahme-Methodik analysiert.

Dabei wurde festgestellt, dass bei den in der industriellen Robotik derzeit üblichen Entwicklungsprozessen noch nicht von einer umfassenden RCP-Methodik gesprochen werden kann. Deshalb wurde im letzten Abschnitt des Kapitels der SBC-Ansatz eingeführt und dessen Verwendung auf dem Gebiet der Industrierobotik diskutiert. Mit Hilfe des SBC-Ansatzes ist ein durchgängiger RCP-Entwicklungsprozess auch für Industrieroboteranwendungen möglich. Der SBC-Ansatz stellt jedoch hohe Anforderungen bezüglich der durch die Entwicklungsplattform zu unterstützenden Beschreibungsmittel für diskret-ereignisorientierte Systeme und deren kombinierte Verwendung in den einzelnen Entwurfsschritten bis hin zur inkrementellen Inbetriebnahme.

Im nachfolgenden Kapitel wird mit dem System Matlab und seinen Erweiterungskomponenten eine für den SBC-Ansatz geeignete Entwicklungsplattform näher betrachtet.

4 Matlab als RCP-Plattform

Im Abschnitt 3.5 wurde als notwendige Voraussetzung für einen durchgängigen Entwicklungsprozess komplexer Robotersteuerungen die Verfügbarkeit eines breiten Beschreibungsmittelspektrums für diskret-ereignisorientierte Systeme herausgestellt. Unter den gegenwärtig verbreiteten Entwurfssystemen wird diese Forderung derzeit nur vom System Matlab im Verbund mit seinen zahlreichen Zusatzprodukten weitgehend erfüllt. Eine Übersicht dazu soll im ersten Abschnitt dieses Kapitels gegeben werden.

Ebenfalls im Abschnitt 3.5 wurde herausgearbeitet, dass ein RCP-Entwicklungsprozess nach dem SBC-Ansatz erst im zweiten Schritt der Automatisierungsphase eine besondere Spezifik hinsichtlich dem Anwendungsgebiet der Industrierobotik besitzt. Zur Umsetzung dieses Schrittes muss die RCP-Plattform die Spezifikation von Low-Level-Roboterkomponenten mittels herstellerspezifischer Roboterprogrammiersprachen unterstützen. Da diese Voraussetzung zur Zeit auch in der Entwicklungsumgebung Matlab nicht a priori erfüllt ist, wird im zweiten Abschnitt die nachträgliche Integration der herstellerspezifischen roboterorientierten Programmierung betrachtet.

Darüber hinaus ist für komplexe Roboteranwendungen gemäß Kapitel 2 und 3 die Integration umfangreicher externer Sensorik und Aktorik erforderlich. Dieser Problematik widmet sich der dritte Abschnitt des Kapitels. Im vierten Abschnitt werden die für Roboteranwendungen wichtigen Möglichkeiten zur CAR-Unterstützung innerhalb der Matlab-Umgebung diskutiert. Abschließend werden die gegenwärtigen Eigenschaften der Entwicklungsplattform Matlab im Kontext des SBC-Ansatzes zusammenfassend bewertet.

4.1 Verfügbare Modellierungsmittel

Das Matlab-Grundsystem unterstützt tatsächlich nur die Modellierung von kontinuierlichen Systemen durch Differentialgleichungen und deren Simulation durch verschiedene Integrationsverfahren. Das Grundsystem wird aber durch zahlreiche Zusatzkomponenten erweitert, die alle für ereignisdiskrete Systeme bekannten Beschreibungsformen unterstützen. Tabelle 4.1[1] gibt einen Überblick zu den von Zusatzkomponenten unterstützten Beschreibungsmitteln.

Die Zusatzkomponenten sind teilweise Produkte des Matlab-Herstellers *The Mathworks Inc.*, darüber hinaus aber auch Produkte von Drittanbietern oder aus Forschungsprojekten resultierende Softwarelösungen.

Neben diskret-ereignisorientierten Beschreibungsmitteln sind auch einige für kontinuierliche Systeme angegeben, die mit ersteren in Kombination benutzt werden können. Dies

[1] Die Übersicht stellt eine Auswahl dar und erhebt keinen Anspruch auf Vollständigkeit.

Tabelle 4.1: Modellierungsmittel in der Matlab-Umgebung

DES/ereignisorientiert:	
Matlab-GPSS-Toolbox, SimEvents	transaktions-, aktivitäts- und prozessorientiert ([Dre99], [Mat12c])
DES/PDES-Toolbox	ereignisorientiert (auch parallel, [Ste09])
DES/zustandsbasiert:	
Stateflow	automatenorientiert (Harel, [Har87], [Mat10])
Matlab-DEVS-Toolbox	automatenorientiert (DEVS, [Paw06])
kontinuierlich:	
ODE-Toolbox	Differentialgleichungen ([Mat12b])
Simulink	Signalflussgraphen ([Mat12e])
Simscape	physikalische Modellierung ([Mat12d])

illustriert den hohen Integrationsgrad des Grundsystems und der Zusatzkomponenten, der damit also auch die hybride Modellierung und Simulation von gemischter ereignisdiskreter und kontinuierlicher Dynamik erlaubt.

4.2 Anbindung konkreter Robotersysteme

Matlab unterstützt derzeit nicht die Programmierung konkreter Robotersysteme mittels herstellerspezifischer roboterorientierter Programmiersprachen. Unter Mitwirkung des Autors dieser Arbeit wurden aber derartige Erweiterungskomponenten für KUKA- und Kawasaki-Robotersysteme realisiert ([Mas04], [Mal09], [Chr11]). Stellvertretend soll die Unterstützung für das Robotersystem KUKA KR3 in diesem Abschnitt näher dargestellt werden.

Das Robotersystem KUKA KR3 besteht aus den Basiskomponenten *(i) KR 3 Knickarm-Industrieroboter*, *(ii) KR C3 Controller* und dem *(iii) KCP (KUKA Control Panel)*. Eine Darstellung der Basiskomponenten zeigt Abbildung 4.1.

Der KR3 Knickarm-Roboter verfügt über sechs rotatorische Achsen und ist für den Einsatz in industriellen Anwendungen konzipiert. Der KR C3 Controller ist der Steuerrechner des Robotersystems und enthält ebenfalls die Leistungselektronik zur Ansteuerung der Gelenkantriebe. Die Systemsoftware besteht aus dem Echtzeit-Betriebssystem *VxWorks* und dem Standardbetriebssystem *MS-Windows*. Für die Offline-Programmierung auf dem Steuerrechner oder einem beliebigen Entwurfssystem steht die roboterorientierte Sprache KRL (KUKA Robot Language) zur Verfügung. Mit dem Bediengerät KCP kann der Roboter manuell gesteuert und online programmiert werden (Teach-In-Programmierung).

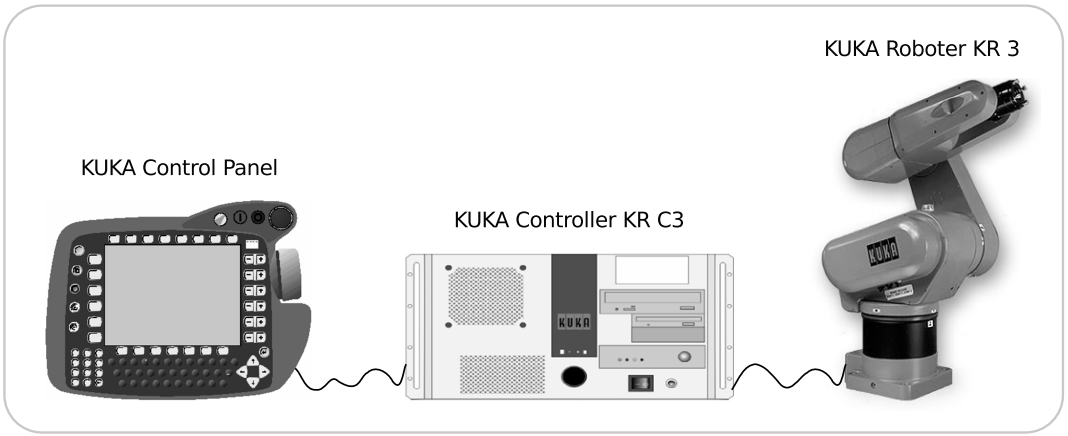


Abbildung 4.1: Basiskomponenten des Robotersystems KUKA KR3 nach [Kuk02]

Zur Realisierung einer Matlab-Anbindung des Robotersystems wäre eine Installation der Matlab-Software direkt auf dem Steuerrechner möglich. Aufgrund der beschränkten Leistungsparameter des KR C3 ist eine solche Konfiguration aber nicht sehr sinnvoll. Flexibler und leistungsfähiger ist eine Matlab-Installation auf einem Standard-PC, der über eine Schnittstelle mit dem Steuerrechner KR C3 verbunden wird. Der Standard-PC mit der Matlab-Installation wird im Folgenden verkürzt als Matlab-PC bezeichnet. Das KCP spielt für die Matlab-Anbindung keine Rolle. Die sich ergebende physische Konfiguration illustriert Abbildung 4.2.

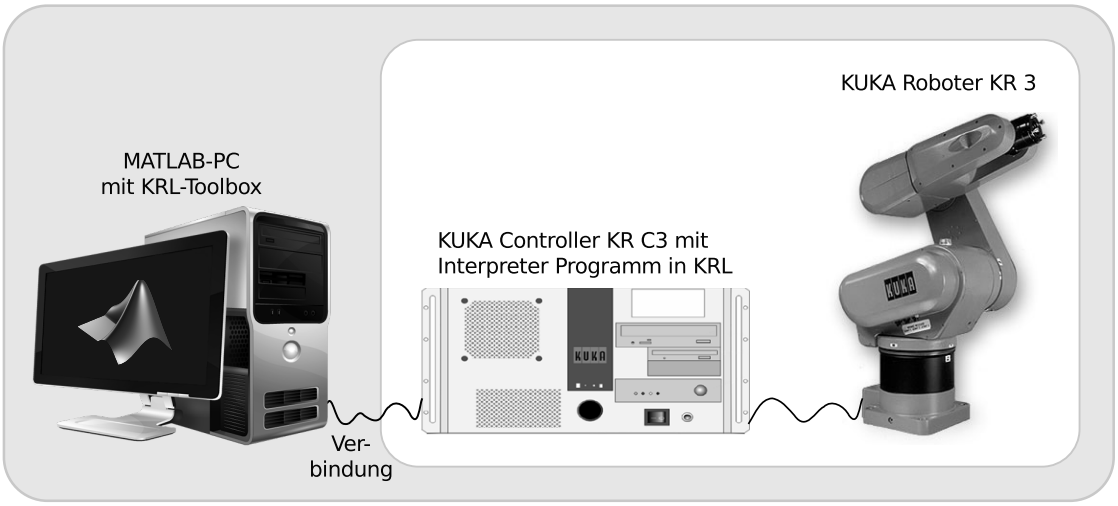


Abbildung 4.2: Physische Konfiguration zur Matlab-Anbindung des Robotersystems

Für die erforderliche bidirektionale Kommunikation zwischen dem Matlab-System und dem KR C3 über die physische Schnittstelle sind auf beiden Seiten Erweiterungskompo-

nenten zu implementieren, die nachfolgend zusammengefasst als Matlab-KRL-Schnittstelle bezeichnet werden.

Auf der Matlab-Seite ist die Softwareschnittstelle als KRL-Toolbox realisiert. Sie besteht aus Funktionen, die in Matlab aufrufbar sind und jeweils eine KRL-Anweisung auf dem KR C3 zur Ausführung bringen. Die Syntax der Matlab-Funktionen entspricht weitgehend den originalen KRL-Anweisungen. Die Programmfragmente 4.1 und 4.2 stellen zum Vergleich einen in Matlab und in Original-KRL implementierten Steuerungsausschnitt gegenüber.

Listing 4.1: Steuerprogramm in Matlab

```
%example.m

% Initialisierung
[hnd,st]=ComInit('COM1',19200,8,0,1);

% Bewegungsbefehle
ptp(hnd,445,0,615,0,90,0);
lin(hnd,400,250,380,0,180,0);

st=ComClose(hnd);
```

Listing 4.2: Steuerprogramm in KRL

```
DEF EXAMPLE()

; Initialisierung
BAS (#INITMOV,0)

; Anweisungsteil - Bewegungsbefehle
PTP {X 445,Y 0,Z 615,A 0,B 90,C 0}
LIN {X 400,Y 250,Z 380,A 0,B 180,C 0}

END
```

Die Initialisierungsfunktion *ComInit* im Programmbeispiel 4.1 baut die Kommunikation von Matlab zum KR C3 auf und gibt ein Handle *hnd* auf die Schnittstelle zurück. Die Handle-Variable ist Voraussetzung, um anschließend Befehle abzusetzen und eine fehlerfreie Kommunikation zu gewährleisten. Verdeutlicht wird das am Beispiel der Bewegungsbefehle *ptp* (Point-to-Point Bewegung) und *lin* (Linearbewegung). Neben den Positionsdaten wird dort als erster Parameter die Handle-Variable übergeben. Die Funktion *ComClose* beendet eine bestehende Verbindung zum KR C3.

Im Vergleich zur originalen KRL erfolgt die Parameterübergabe an die entsprechenden Funktionen der KRL-Toolbox in Matlab in etwas vereinfachter Form. Grundsätzlich beschreiben die Parameter der Bewegungsbefehle *ptp* und *lin* die anzufahrende Zielposition. Im hier dargestellten Fall werden die Zielpositionen im kartesischen Koordinatensystem angegeben. Die ersten drei Parameter (x,y,z) beschreiben die Lage des Zielpunktes als translatorische Distanz zum Koordinatenursprung in *mm*. Die anschließenden drei Parameter (a,b,c) beschreiben die Orientierung des Zielpunktes als Rotationen um die Koordinatenachsen in *Grad*.

Die im originalen KRL-Programm enthaltende Initialisierungsfunktion *BAS* ist verantwortlich für die Initialisierung wichtiger Roboterparameter, wie zum Beispiel die maximalen Geschwindigkeiten und Beschleunigungen. In der Matlab-KRL-Toolbox existieren entsprechende Funktionen, um die Roboterparameter zu beeinflussen ([Mas04]).

Auf der Seite des KR C3 ist für die Realisierung der Matlab-Schnittstelle ein Interpreterprogramm zu implementieren, welches die Steuerungsbefehle von der Matlab-Seite empfängt und als KRL-Anweisungen auf dem KR C3 zur Ausführung bringt. In umgekehrter Richtung sind die Ergebnisparameter der KRL-Anweisung an die Matlab-Seite zu senden. Der rechte Teil der Abbildung 4.3 zeigt die Struktur des Interpreterprogramms,

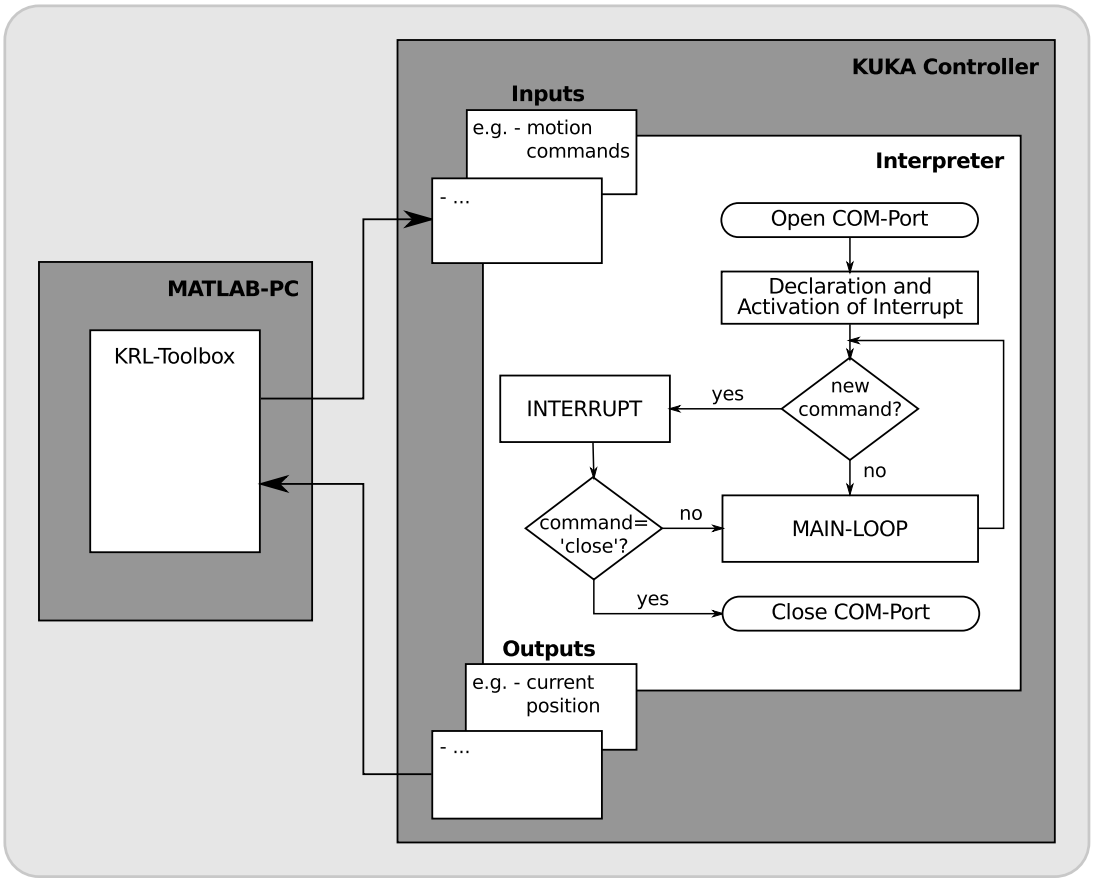


Abbildung 4.3: Matlab-KRL Schnittstelle

die im Wesentlichen aus zwei Teilen besteht. Neben dem Hauptprogramm (*Main-Loop*) existiert eine Interrupt-Serviceroutine, die immer dann die Main-Loop unterbricht, wenn eine neue Anweisung von Matlab über die physische Verbindung eintrifft. Die empfangenen Daten werden in der Interrupt-Serviceroutine auf ihre Korrektheit überprüft und in einem Puffer gespeichert. Die Main-Loop ist eine Endlosschleife, die für die Ausführung der gepufferten Anweisungen verantwortlich ist.

Der Implementierungsaufwand für die beschriebene Matlab-KRL-Schnittstelle ist relativ gering. Unter direkter Verwendung der Schnittstelle ist eine roboterorientierte Programmierung gemäß Kapitel 2 auf der Softwareplattform Matlab möglich. Die durch die Firmware des KR C3 bereitgestellten Sicherheitsfunktionen wie Arbeitsraumüberwachung, Endschalterkontrolle und Momentenüberwachung der Roboterachsen bleiben für die in Matlab spezifizierten Steuerungsprogramme wirksam. Im Sinne des Kapitels 3 kann damit bereits ein systematischer Entwicklungsprozess für einfache Roboteranwendungen durchgeführt werden. Weitere Schnittstellen zu Robotersystemen anderer Hersteller können nach dem selben Muster bereitgestellt und in Kombination miteinander benutzt werden.[2]

[2]Die Funktionen der KRL-Toolbox sind im Anhang A.1 aufgeführt. Weitere Informationen zur Anbindung

4.3 Integration externer Komponenten

Für komplexe Roboteranwendungen muss die Steuerung neben dem eigentlichen Robotersystem auch verschiedene externe sensorische und aktorische Zusatzkomponenten miteinbeziehen. Die Möglichkeiten dazu innerhalb eines KRL-Programms auf dem KR C3 sind sehr begrenzt und beschränken sich auf bestimmte von der KUKA Roboter GmbH unterstützte Drittanbieter. Auf der Matlab-Ebene stehen dagegen die Integrationsmöglichkeiten einer allgemeinen erweiterbaren Programmiersprache zur Verfügung. So kann über die Matlab-External-Schnittstelle (MEX-Interface, [Mat12a]) Programmcode in den Sprachen C, C++, Fortran und Java eingebunden werden. Damit kann die Treibersoftware nahezu jeder externen Komponente integriert werden. Darüber hinaus bietet das Matlab-Grundsystem beziehungsweise spezialisierte Toolboxen umfangreiche Algorithmensammlungen zur Weiterverarbeitung sensorischer Daten, beispielsweise für die Bildverarbeitung.

4.4 CAR-Unterstützung

Im Abschnitt 3.4 wurde die Rolle von CAR-Systemen für den effizienten Entwurf von Robotersteuerungen erläutert, die den Aufbau realitätsnaher Modelle von Robotern und deren Umgebung unterstützen. CAR-Systeme eröffnen damit die Möglichkeit, konkrete Robotersteuerungen bereits während der Entwurfsphase mittels Systemsimulationen zu erproben. Um diese Technik auch für die Entwicklungsumgebung Matlab zur Verfügung zu stellen, existieren zwei prinzipielle Wege.

Der erste Weg besteht in der Anbindung eines bereits existierenden herstellerspezifischen CAR-Systems. Im hier betrachteten Fall wäre dies das System *KUKA.Sim*. Die physische Konfiguration sowie die zu realisierende Softwareschnittstelle würde der bereits im Abschnitt 4.2 erläuterten Anbindung eines KUKA-Roboters entsprechen und soll deshalb an dieser Stelle nicht weitergehend dargestellt werden.

Der zweite Weg besteht in der direkten Bereitstellung von CAR-Funktionalitäten innerhalb der Entwicklungsplattform Matlab. Dazu müssen einerseits Modelle der kontinuierlichen Dynamik sowie Kinematikmodule für die konkreten Robotersysteme bereitgestellt werden. Die Voraussetzung zur Implementierung dieser Komponenten sind im Matlab-Grundsystem vorhanden (ODE-Toolbox, matrizenorientierte Programmiersprache). Ebenso können – wenn erforderlich – Algorithmen zur Kollisionserkennung problemlos mit den Mitteln des Matlab-Grundsystems implementiert werden.

Zum anderen ist wie bereits im Abschnitt 3.4 ausgeführt die 3D-Visualisierung des Robotersystems und seiner Umgebung eine wesentliche CAR-Funktionalität. Bis zu mittleren Ansprüchen an die photorealistische Qualität können solche Visualisierungen ebenfalls mit den Mitteln des Matlab-Grundsystems realisiert werden. Abbildung 4.4 zeigt beispielhaft die Visualisierung des KUKA KR3 auf Basis der Matlab-Handle-Graphics-Object-Technik.

von Kawasaki-Robotern sind in [Chr11] verfügbar.

Für sehr hohe Qualitätsansprüche sind VRML[3]-basierte Techniken besser geeignet. Diese stehen mit der Zusatzkomponente Simulink-3D-Animation ebenfalls zur Verfügung.

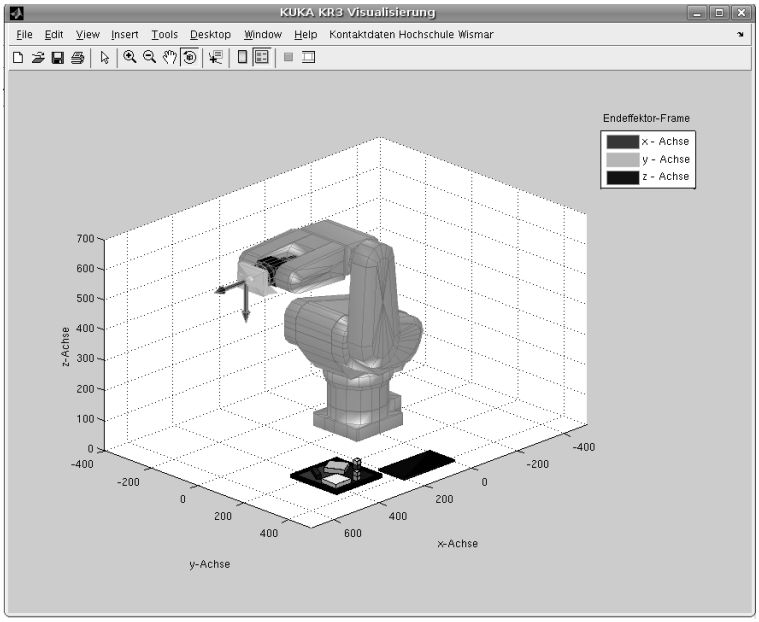


Abbildung 4.4: Visualisierung des KUKA KR3 Roboters

Der Implementierungsaufwand für die direkte Bereitstellung von CAR-Funktionalitäten innerhalb der Entwicklungsplattform Matlab ist im Vergleich zur Anbindung eines herstellerspezifischen CAR-Systems wesentlich größer. Es wird auf diese Weise jedoch die Möglichkeit eröffnet, komplexe Roboteranwendungen bestehend aus Robotern verschiedener Hersteller und externer Komponenten, die in den kommerziellen CAR-Systemen nicht unterstützt werden, in der Entwurfsphase simulativ zu erproben.

4.5 Zusammenfassende Bewertung

In den vorangegangenen Abschnitten wurde gezeigt, welche Beschreibungsmittel die RCP-Plattform Matlab und ihre Erweiterungskomponenten für diskret-ereignisorientierte Systeme zur Verfügung stellen. Darüber hinaus wurde dargestellt, wie die Unterstützung der roboterorientierten Programmierung für konkrete Robotersysteme bereitgestellt werden kann.

Gegenwärtig wird diese nicht durch die großen Roboterhersteller zur Verfügung gestellt. Ob in Zukunft die untereinander konkurrierenden Hersteller eine integrierende Entwicklungsplattform wie Matlab unterstützen werden, ist derzeit schwer abschätzbar. Für eine

[3]Die Virtual Reality Modeling Language (VRML) ist eine Programmiersprache, die unter anderem das Erzeugen und Darstellen von 3D-Szenen unterstützt.

solche Entwicklung könnte sprechen, dass für den Bereich regelungstechnischer Anwendungen bereits seit vielen Jahren von den kommerziellen Anbietern Integrationsmöglichkeiten in Matlab für ihre Produkte angeboten werden.

5 Entwurf und Inbetriebnahme einer komplexen Robotersteuerung nach dem SBC-Ansatz

Im Abschnitt 3.5 wurden die grundlegenden Aspekte des SBC-Ansatzes eingeführt. Mit den im vorangegangenen Kapitel besprochenen roboterspezifischen Anbindungen steht mit dem System Matlab einschließlich seiner Erweiterungskomponenten eine Softwareplattform für die praktische Durchführung von RCP-Projekten zur Verfügung. In diesem Kapitel soll anhand eines konkreten Beispiels der komplette Entwicklungsprozess einer komplexen Robotersteuerung nach dem SBC-Ansatz erläutert werden.

Abbildung 5.1 zeigt die schematische Darstellung einer Roboterzelle für ein materialtechnisches Bearbeitungsproblem. Die Zelle besteht aus einem Eingangspuffer, der Werkstücke auf einer Palette mit nicht festgelegten Positionen und in beliebiger Lage enthält, einem Knickarm-Industrieroboter mit bildgebendem Sensor, einer Bearbeitungsstation mit einem positionierbaren Bohrwerk und zwei Bohrplätzen sowie einem Ausgangspuffer.

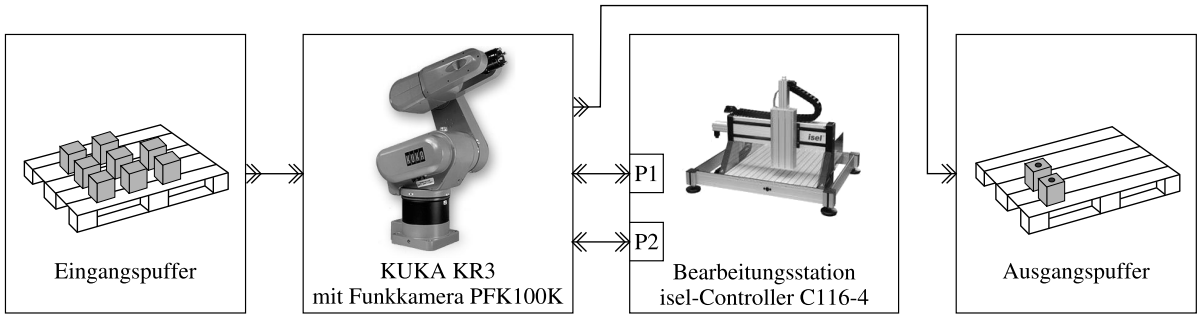


Abbildung 5.1: Aufbau der Roboterzelle

Vor der Entnahme eines Werkstückes aus dem Eingangspuffer muss dessen Position und Lage mit Hilfe einer am Robotergreifer befindlichen Kamera bestimmt werden. Nach Aufnahme eines Werkstückes ist dieses durch den Roboter zur Bearbeitungsstation zu transportieren und auf einem der Bohrplätze abzulegen. Wegen des nur einmal vorhandenen Bohrwerkes kann eine Bearbeitung von Werkstücken auf beiden Bohrplätzen nicht zeitgleich erfolgen. Nachdem die Bearbeitung abgeschlossen ist, entnimmt der Roboter das Werkstück vom Bohrplatz und transportiert dieses zum Ausgangspuffer.

Trotz der relativ einfachen Struktur des Problems handelt es sich im Sinne des dritten

Kapitels um eine komplexe Roboteranwendung. Aufgrund der externen Sensorik (Kamera) kann das Problem nicht allein mit den Mitteln der konventionellen Roboterprogrammierung gelöst werden. Darüber hinaus muss die Robotersteuerung mit einer ebenfalls gesteuerten Komponente der Umgebung – der Bearbeitungsstation – kooperieren. Für den nachfolgend dargestellten Entwicklungsprozess der Gesamtsteuerung der Roboterzelle soll folgende Anforderungsspezifikation gelten:

1. Die Durchlaufzeit der Werkstücke soll möglichst klein sein.
2. Die Auslastung der Bearbeitungsstation soll möglichst hoch sein.

Nachfolgend wird die Planung und Bewertung von Steuerungsstrategien, der anschließende Entwurf einer konkreten Steuerung sowie der Übergang zur Inbetriebnahme unter Nutzung der RCP-Plattform Matlab beschrieben.

5.1 Planung und Bewertung von Steuerungsstrategien

Zu Beginn des Entwicklungsprozesses einer komplexen Fertigungsanlage wird das Problem durch den Planungsingenieur in der Regel als Materialflusssystem aufgefasst, dessen Topologie, Dimensionierung und Steuerungsstrategie entsprechend der Anforderungsspezifikation zu entwerfen ist. Ein effizientes Mittel zum Auffinden und Prüfen geeigneter Lösungen ist die simulative Analyse und vergleichende Bewertung unterschiedlicher Varianten im Rahmen der Planungssimulation.

Zur Modellierung von Materialflusssystemen werden bevorzugt ereignisbasierte Beschreibungsmittel verwendet. Gemäß Tabelle 4.1 existieren für die RCP-Umgebung Matlab verschiedene Erweiterungskomponenten, die solche Beschreibungsformen unterstützen. Als Beispiel wird in diesem Abschnitt die Verwendung der Komponente *SimEvents* ([Mat12c]) zur Lösung der Entwurfsaufgabe betrachtet. SimEvents unterstützt eine komponentenorientierte Modellierung und ermöglicht ereignisbasierte Modellbeschreibungen gemäß der sogenannten transaktionsorientierten Weltsicht. Die Ausführung von SimEvents-Modellen kann in Kombination mit anderen Modellformen innerhalb von *Simulink* erfolgen. Über Simulink ([Mat12e]) ist ebenfalls die Integration mit dem Matlab-Grundsystem realisiert.

Beim hier betrachteten Bearbeitungsproblem ist die Materialflusstopologie durch die Anforderungsspezifikation bereits festgelegt. Das heißt als Systemeingang und -ausgang ist jeweils ein Puffer vorhanden. Es existiert zudem nur eine Bearbeitungsstation und der Transport der Werkstücke vom Eingangspuffer sowie zum Ausgangspuffer erfolgt durch ein einzelnes Transportelement – den Roboter. Auch die Dimensionierung der Bedienzeiten der Prozesskomponenten sei hier durch die Anforderungsspezifikation als festgelegt angenommen. Danach benötigt der Bearbeitungsvorgang auf einem Bohrplatz der Bearbeitungsstation 60 Sekunden. Der An- und Abtransport der Werkstücke durch den Roboter beansprucht jeweils 10 Sekunden.[1]

[1] Zur leichteren Nachvollziehbarkeit der Simulationsergebnisse wurden deterministische Bedienzeiten gewählt. In realen Problemen sind dies häufig stochastische Größen.

Gegenstand der Planungssimulation ist damit nur das Auffinden einer geeigneten Steuerungsstrategie. Offensichtlich sind im betrachteten Beispiel zwei grundsätzlich verschiedene Ansätze möglich. Aus sicherheitstechnischen Erwägungen kann es von Vorteil sein, keine Nebenläufigkeit der aktiven Prozesskomponenten zuzulassen. Bei dieser Strategie dürfen der Roboter und die Bearbeitungsstation also nicht zeitgleich arbeiten. Dagegen soll die zweite Steuerungsstrategie nebenläufige Aktivitäten in Erwartung einer geringeren Gesamtdurchlaufzeit erlauben.

Um die Strategien simulativ zu erproben, müssen zwei Planungsmodelle erstellt werden. Dafür steht in SimEvents eine umfangreiche Bibliothek zur Verfügung, die unter anderem Komponenten zur Generierung von Entitäten und zur Modellierung von Materialflusskomponenten wie beispielsweise Bedieneinrichtungen und Puffer enthält. Der Materialfluss wird dabei durch die Entitäten abgebildet, die sich analog zu den realen Fördereinheiten durch das Materialflusssystem bewegen. Abbildung 5.2 zeigt exemplarisch das SimEvents-Modell zur Simulation des Materialflusses nach der zweiten Steuerungsstrategie, die eine parallele Aktivität von Roboter und Bearbeitungsstation zulässt.

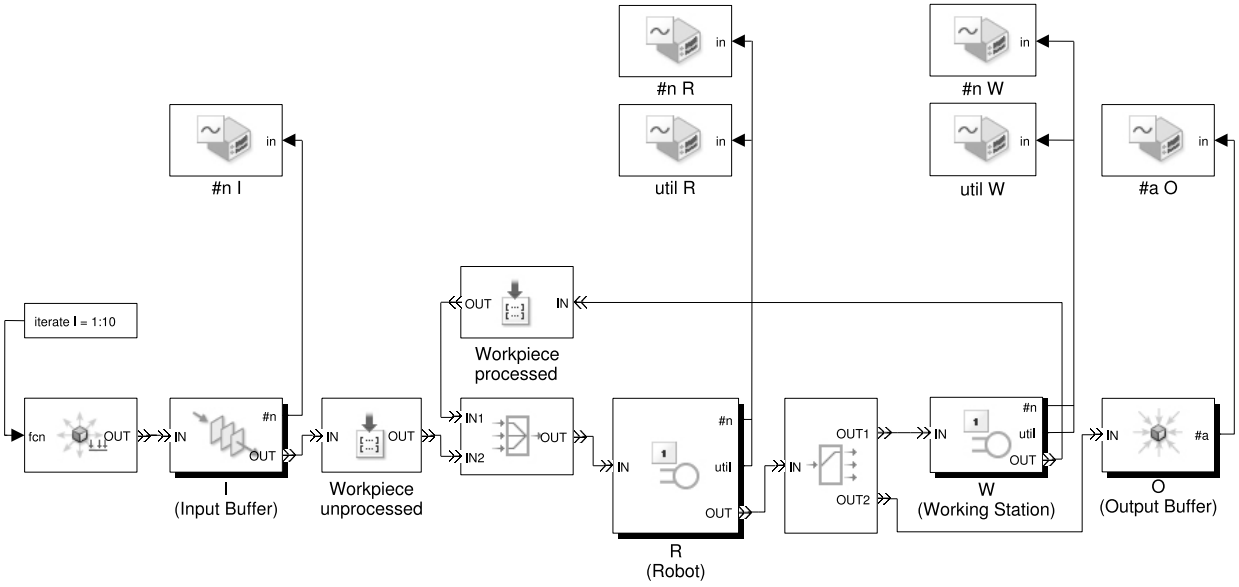


Abbildung 5.2: Planungsmodell in SimEvents

Im unteren Teil der Abbildung sind die einzelnen Materialflusskomponenten dargestellt. Der Roboter und die Bearbeitungsstation sind als separate Bedieneinrichtungen modelliert, die jeweils nur eine Fördereinheit bedienen können (engl. *Single Server*). Werkstücke, die den Eingangspuffer verlassen, bekommen das Attribut *unbearbeitet (unprocessed)* und gelangen damit durch den Roboter zur Bearbeitungsstation. Nach Verlassen der Bearbeitungsstation erhalten die Werkstücke das Attribut *bearbeitet (processed)* und erreichen dann nach erneuter Verzögerung auf dem Roboter den Ausgangspuffer. Die Komponenten im oberen Teil der Abbildung 5.2 dienen der Aufzeichnung und Darstellung von Simulati-

onsergebnissen.

Ein Bearbeitungszyklus umfasst 10 Werkstücke.[2] Abbildung 5.3 zeigt den simulativ ermittelten Ankunftsverlauf der bearbeiteten Werkstücke im Ausgangspuffer. Der Gesamtdurchlauf ist abgeschlossen, wenn sich alle 10 Fördereinheiten im Ausgangspuffer befinden. Die Gesamtdurchlaufzeit der Steuerungsstrategie mit Nebenläufigkeit (620 Sekunden) fällt dabei erwartungsgemäß geringer aus als bei Nebenläufigkeitsverbot (800 Sekunden).

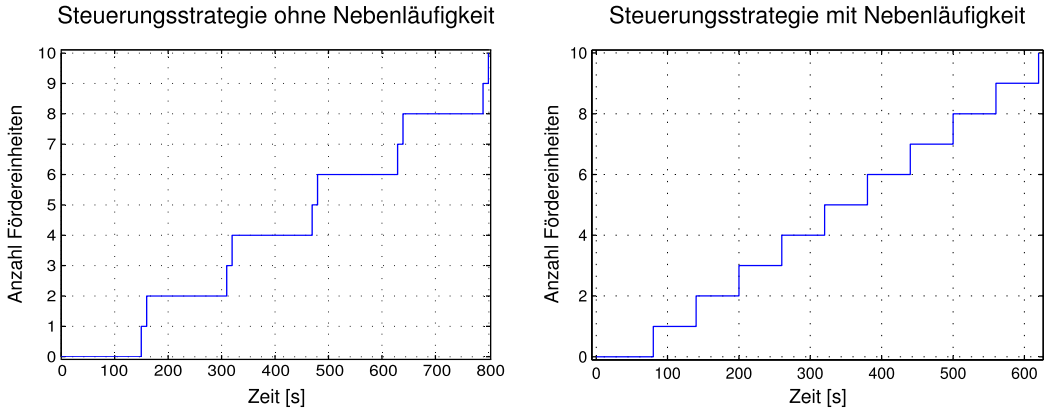


Abbildung 5.3: Zeitlicher Verlauf der Ausgangspufferbelegung

In Abbildung 5.4 ist der zeitliche Verlauf der Belegung beziehungsweise Aktivität des Roboters und der Bearbeitungsstation dargestellt. Die Aktivität einer Komponente wird durch den Wert 1 und Inaktivität durch den Wert 0 angezeigt. Die Auslastung des Roboters und der Bearbeitungsstation kann aus dem Verhältnis von Aktivitätszeit zur Gesamtzeit ermittelt werden ([Kos95]). Sie ist zusammen mit der Gesamtdurchlaufzeit in Tabelle 5.1 aufgeführt.

Tabelle 5.1: Ergebnisvergleich der untersuchten Steuerungsstrategien

	Steuerungsstrategie ohne Nebenläufigkeit	Steuerungsstrategie mit Nebenläufigkeit
Gesamtdurchlaufzeit in s	800	620
Auslastung Roboter	0.25	0.32
Auslastung Bearbeitungsstation	0.75	0.97

Erwartungsgemäß liefert die Steuerungsstrategie mit Nebenläufigkeit bessere Ergebnisse hinsichtlich Gesamtdurchlaufzeit und Maschinenauslastung. Bemerkenswert ist aber, dass die Auslastung des Roboters auch bei der Steuerungsstrategie mit Nebenläufigkeit deutlich unter 50% bleibt. Dies könnte die Prüfung einer veränderten Topologie beziehungsweise

[2] Nach Bereitstellung einer mit 10 Werkstücken belegten Palette im Eingangsbereich der Roboterzelle.

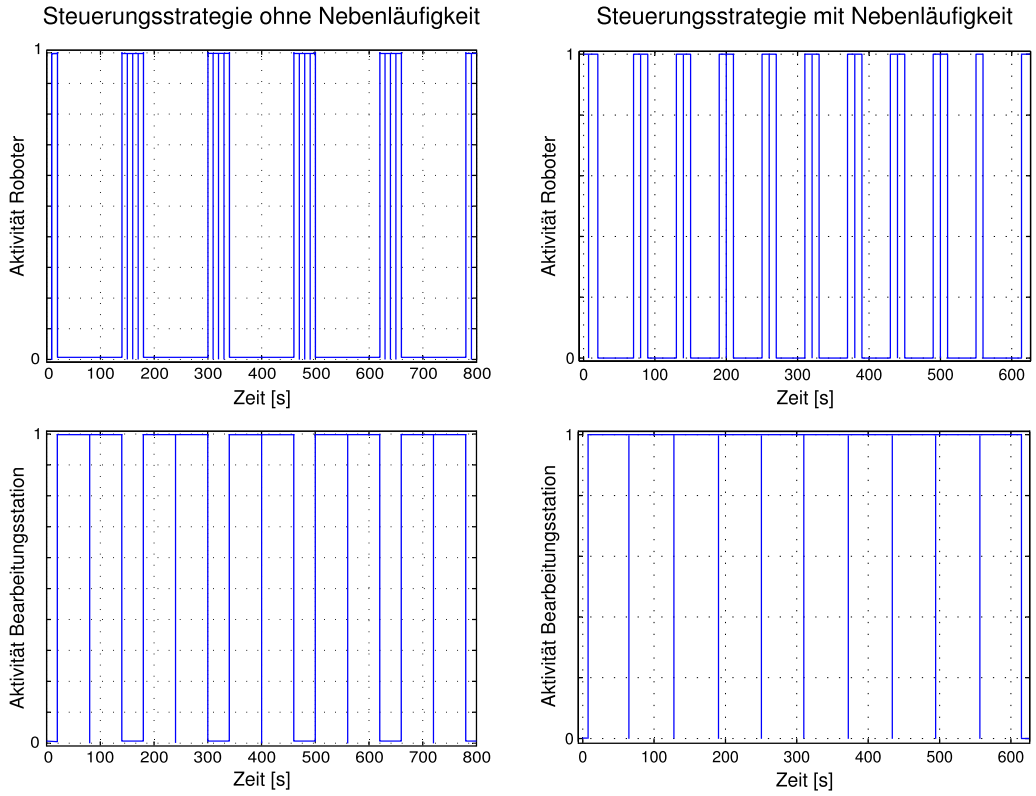


Abbildung 5.4: Zeitlicher Belegungsverlauf der Bedieneinrichtungen

Dimensionierung nahelegen. So könnte der Bearbeitungsprozess um weitere Bearbeitungsstationen ergänzt werden. Alternativ wäre aber auch die Erhöhung der Arbeitsplatzanzahl auf einer Bearbeitungsstation möglich. In der hier zugrunde gelegten Anforderungsspezifikation sollen aber allein die Gesamtdurchlaufzeit und die Maschinenauslastung in der vorgegebenen Topologie und Dimensionierung entscheidend sein. Danach wäre die Steuerungsstrategie mit Nebenläufigkeit als die geeignetere für den nachfolgenden Entwurf einer konkreten Steuerung zu übergeben.

5.2 Entwurf einer konkreten Steuerung

Nachdem im Ergebnis der Planungssimulationen eine der Anforderungsspezifikation entsprechende Steuerungsstrategie vorliegt, ist diese in einem konkreten Steuerungsentwurf umzusetzen. Dazu wird die Steuerungslogik vom Materialfluss getrennt modelliert. Bei dem aus der Planungsphase übernommenen Modellstand ist dies in der Regel noch nicht der Fall. Die Separierung und Detaillierung der Steuerungsebene ist üblicherweise mit einem Wechsel der Modellbeschreibung auf eine zustandsbasierte Form verbunden.

In der RCP-Umgebung Matlab stehen dafür wiederum verschiedene Erweiterungskomponenten zur Verfügung (vgl. Tab. 4.1). Als Beispiel wird in diesem Abschnitt die Kom-

ponente *Stateflow* ([Mat10]) zur weiteren Lösung der Entwurfsaufgabe betrachtet. Stateflow untersützt die zustandsbasierte Modellierung in Form von Statecharts, die weitgehend der Theorie des Harel-Automaten ([Har87]) entsprechen. Stateflow-Diagramme können wie SimEvents-Modelle in Kombination mit anderen Modellbeschreibungen innerhalb der Matlab-Erweiterung Simulink ausgeführt werden.

Abbildung 5.5 zeigt für das betrachtete Problem ein Konzeptmodell für den Steuerungsentwurf. Die untere Ebene ist das Prozessmodell PM mit den bereits aus der Planungsphase

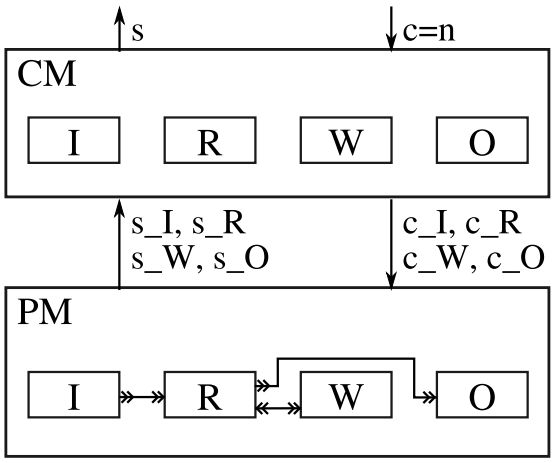


Abbildung 5.5: Konzeptmodell des Steuerungsentwurfs

bekannten Prozesselementen Eingangspuffer: PM.I, Roboter: PM.R, Bearbeitungsstation: PM.W und Ausgangspuffer: PM.O. Die Steuerungsebene CM soll modular realisiert werden und sieht für jedes Prozesselement eine zugehörige Steuerungskomponente: CM.I, CM.R, CM.W und CM.O vor. Die Schnittstelle zwischen Steuerungs- und Prozessebene wird durch entsprechende s- und c-Größen gebildet. Darüber hinaus ist eine Schnittstelle zwischen der Steuerungsebene und einer darüber liegenden Leit- oder Bedienebene vorgesehen. Sie besteht aus den Größen *c* und *s*. Mit c wird der Steuerung die Bereitstellung einer Palette mit zu bearbeitenden Werkstücken sowie die Anzahl der auf der Palette befindlichen Werkstücke mitgeteilt. Über die Größe s wird der Leit- oder Bedienebene der aktuelle Bearbeitungsstand berichtet.

Für die Umsetzung des Konzeptmodells innerhalb von Simulink in Verbindung mit Stateflow sind verschiedene Ansätze möglich. In Abbildung 5.6 ist eine Umsetzung unter Verwendung der Simulink-Submodell-Technik und von parallelen Stateflow-Zuständen sowie mehrfachen Stateflow-Diagrammen dargestellt.

Die Simulink-Submodell-Technik wird zur klaren Trennung der Steuerungs- und Prozessebene in ein CM- und ein PM-Submodell genutzt. Die Steuerungslogik innerhalb des CM-Submodells ist in einem Stateflow-Diagramm abgebildet. Der prozesselementeorientierte Entwurf der enthaltenen Steuerungskomponenten ist mittels paralleler Zustände innerhalb des CM-Stateflow-Diagramms umgesetzt und in Abbildung 5.7 dargestellt. Die Modellierung der Prozesselemente innerhalb des PM-Submodells ist durch jeweils ein Stateflow-

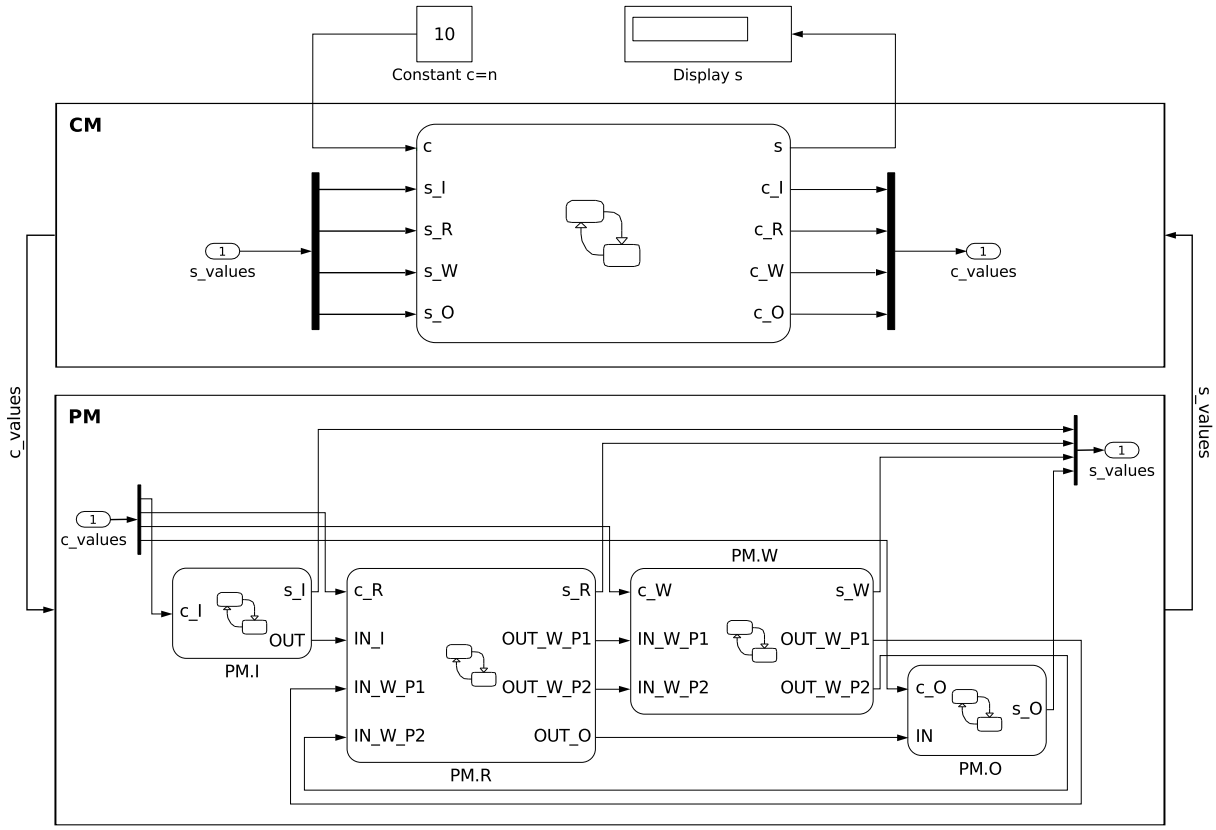


Abbildung 5.6: Umsetzung des Konzeptmodells in Simulink unter Verwendung von Stateflow-Diagrammen

Diagramm pro Prozesselement realisiert. Im Unterschied zum SimEvents-Planungsmodell in Abbildung 5.2 steht für die Materialflusskopplungen in Simulink/Stateflow kein separater Konnektortyp zur Verfügung. Diese Kopplungen werden im Simulink/Stateflow-Modell durch gewöhnliche Signal-Konnektoren dargestellt. Anhand des eingesetzten Bezeichnerschemas, welches für aus- und eingehende Materialflüsse jeweils den Präfix *OUT* beziehungsweise *IN* verwendet, kann der Materialfluss innerhalb des PM-Submodells dennoch leicht nachvollzogen werden.

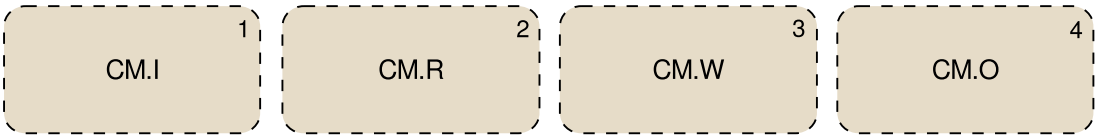


Abbildung 5.7: Umsetzung des prozesselementeorientierten Steuerungsentwurfs mittels paralleler Zustände

Nachfolgend soll die Umsetzung der Steuerungslogik sowie die Modellierung der Prozess-

dynamik im ersten Schritt der Automatisierungsphase am Beispiel der Roboterkomponente näher erläutert werden. Für eine Darstellung des vollständigen Entwurfsmodells in dieser Phase sei auf den Anhang A.2 verwiesen.

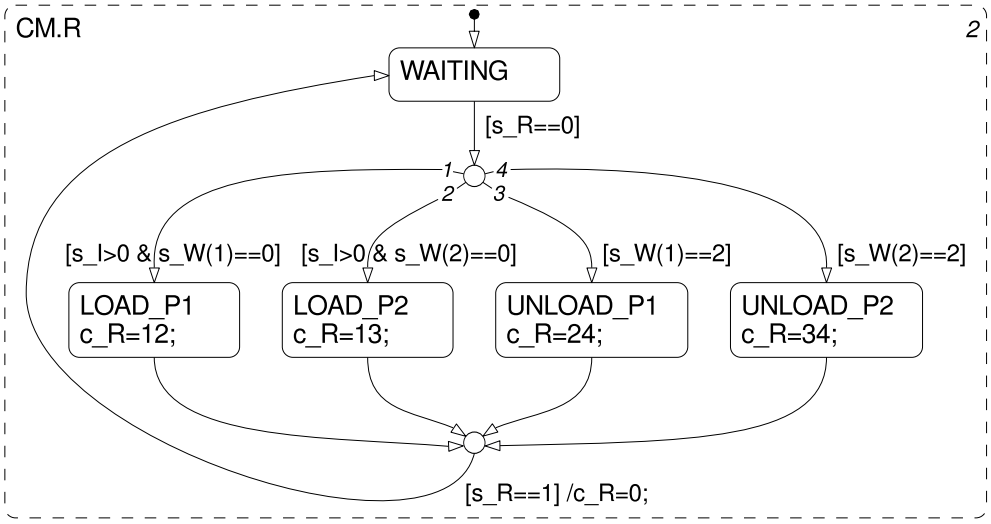


Abbildung 5.8: Modell der Steuerungslogik für die Roboterkomponente

Der in Abbildung 5.8 dargestellte parallele Elternzustand CM.R[3] wertet zur Steuerung des Roboters die innerhalb der vektoriellen Variable *s_values* vom PM-Submodell gesendeten Zustandswerte *s_I* (Anzahl Werkstücke im Eingangspuffer), *s_R* (Belegungszustand des Roboters) und *s_W* (Belegungszustand der Bearbeitungsstation) aus. Als Ausgangsgröße wird der Steuerungswert *c_R* generiert und als Bestandteil der vektoriellen Variablen *c_values* an das Prozessmodell gesendet. Der Initialzustand der Steuerungslogik ist *WAITING*. Wird vom Prozessmodell mit *s_R = 0* signalisiert, dass der Roboter einen Transportvorgang abgeschlossen hat, kann der *WAITING*-Zustand verlassen werden. In welchen der möglichen vier Folgezustände die Steuerung übergeht, hängt von den Prozesszuständen *s_I* und *s_W* ab. Die Zustandsgröße *s_W* besteht aus zwei Elementen, die den Belegungszustand der Bohrplätze P1 und P2 auf der Bearbeitungsstation angeben. Bei einem Wert von 0 ist der Bohrplatz unbelegt. Der Wert 2 zeigt an, dass der Bohrplatz mit einem Werkstück belegt und die Bearbeitung abgeschlossen ist. Geht die Steuerung in einen der Transportzustände über, wird ein Steuerungswert generiert und über die Variable *c_R* an das Prozessmodell gesendet. Der Steuerungswert besteht aus zwei Ziffern. Die erste Ziffer codiert den Entnahmeort und die zweite Ziffer den Zielort des auszuführenden Transports (1 = Eingangspuffer, 2 = Bohrplatz P1, 3 = Bohrplatz P2, 4 = Ausgangspuffer). Signalisiert das Prozessmodell mit *s_R = 1*, dass der Transportvorgang begonnen wurde, geht die Steuerung wieder in den Zustand *WAITING* über.

[3] Die hier zur besseren Lesbarkeit verwendete Punktnotation von Zustandsbezeichnern wird von den derzeitigen Stateflow-Versionen nicht unterstützt. Für eine lauffähige Implementierung wäre CM.R deshalb zur Zeit durch CM_R zu ersetzen.

In der Abbildung 5.9 ist die Modellierung der Prozessdynamik der Roboterkomponente als separates Stateflow-Diagramm PM.R dargestellt. Im Unterschied zu parallelen Zuständen werden eigenständige Stateflow-Diagramme *ohne* gestrichelte Umrandung dargestellt. Die Schnittstelle des Stateflow-Diagramms PM.R besteht einerseits aus den *IN-*

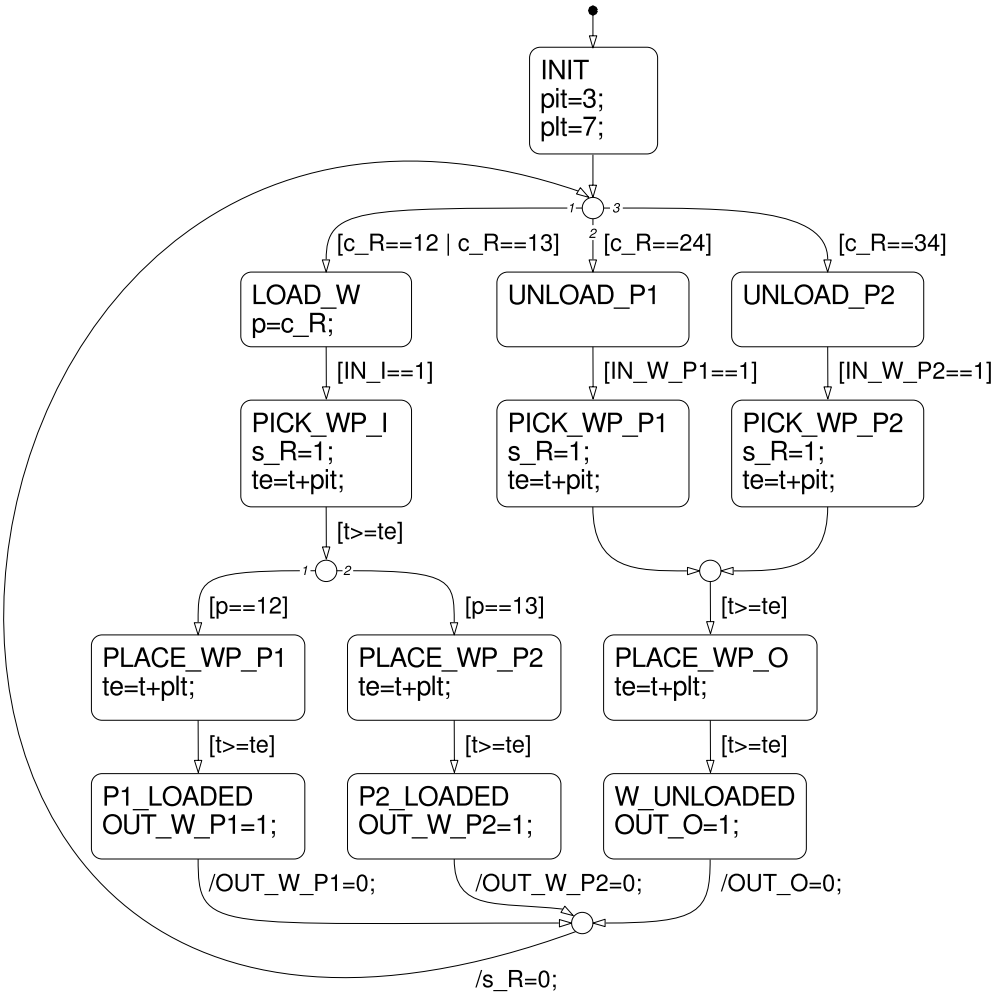


Abbildung 5.9: Modell der Prozessdynamik für die Roboterkomponente (PM.R)

und *OUT-Variablen*, über die die materialflusstechnische Verknüpfung mit den anderen Prozesskomponenten realisiert wird und den *c* - und *s* - *Variablen* zum Empfang von Steuerungsanweisungen beziehungsweise zum Versand von Zustandsinformationen an das Steuerungsmodell. Bei Einnahme des Initialzustandes *INIT* werden zunächst die Bedienzeiten des Roboters gesetzt. Die Variable *pit (pick time)* gibt eine Zeitspanne an, die der Roboter benötigt, um ein Werkstück aus dem Eingangspuffer oder von einem der beiden Bearbeitungsplätze zu entnehmen. Die Variable *plt (place time)* definiert eine Zeitspanne, die zur Abgabe eines Werkstückes erforderlich ist. Während sich die Roboter-Komponente im Initialzustand befindet wird der Steuereingang *c_R* überwacht. Die Steuerungswerte

bestehen wie oben beschrieben aus zwei Ziffern, wobei die erste Ziffer den Entnahmeort und die zweite Ziffer den Zielort des auszuführenden Transportes codiert. Die vom Initialzustand ausgehende Transitionsverzweigung bildet mit den drei Folgezuständen die drei möglichen Entnahmeorte ab. Der von der Verzweigung ausgehende linke Pfad beschreibt die Zustandsübergänge für den Teiletransport[4] vom Eingangspuffer zu einem der beiden Bohrplätze auf der Bearbeitungsstation. Die verbleibenden zwei Pfade beschreiben den Teiletransport von einem der beiden Bohrplätze zum Ausgangspuffer. In den Entnahme-Zuständen *PICK_WP* wird der Steuerung über *s_R = 1* signalisiert, dass der Roboter in einen Belegungszustand übergegangen ist. Der Abschlusszeitpunkt des Entnahmevorganges *te* wird aus der aktuellen Simulationszeit *t* und der Bedienzeit *pit* bestimmt. Nach der Entnahme eines Werkstückes geht die Roboter-Komponente in einen der Abgabe-Zustände *PLACE_WP* über. Der Abschlusszeitpunkt der Abgabe wird anhand der Bedienzeit *plt* bestimmt. Mit der Abgabe eines Werkstückes ist der Transportvorgang abgeschlosssen und die Roboter-Komponente nimmt einen der *LOADED*-Zustände beziehungsweise den *UNLOADED*-Zustand ein. In diesen Zuständen werden über die Ausgangsschnittstelle *OUT* materialflusstechnische Informationen an die nachfolgende Prozesskomponente übermittelt und mit der abschließenden Transitionsaktion *s_R = 0* wird der Steuerung signalisiert, dass der Roboter nunmehr unbelegt ist und erneut in den Initialzustand übergeht.

Die im ersten Schritt der Automatisierungsphase separierte CM- und PM-Ebene kann im Rahmen einer Systemsimulation in der Regel sehr gut mittels den aus der Planungsphase vorliegenden Ergebnissen verifiziert werden, da die in den PM-Komponenten angesetzten Bedienzeiten denen der Planungsphase entsprechen.

5.3 Steuerungsdetaillierung und Inbetriebnahme

Der im vorangegangenen Abschnitt beschriebene Modellstand ermöglicht die simulative Erprobung einer konkreten Steuerung für das materialtechnische Bearbeitungsproblem. Er ist zugleich aber noch generisch, das heißt unabhängig von den später zum Einsatz kommenden konkreten herstellerspezifischen Prozesselementen. Zur Erprobung der entworfenen Steuerung im Rahmen einer SiL-Simulation und schließlich zur Anwendung dieser im operativen Betrieb ist eine weitere Detaillierung des Entwurfsmodells bis auf die Sensor- und Aktor-Ebene der realen Prozesselemente in einem zweiten Schritt der Automatisierungsphase erforderlich. Es ist offensichtlich vorteilhaft bei der weiteren Detaillierung des Modellstandes unter Nutzung des Modularisierungskonzeptes in generische und herstellerspezifische Modellkomponenten zu separieren.

Bei Anwendung des SBC-Ansatzes bleibt das bisher entworfene Steuerungsmodell von den weiteren Detaillierungen vollständig unberührt. Die bisherigen generischen Teile des Prozessmodells werden als *High-Level-Komponenten* weiter benutzt. Alle weiteren herstellerspezifischen Detaillierungen werden in sogenannten *Low-Level-Komponenten* beschrieben. Die Low-Level-Komponenten sind nur für jene Teile des Entwurfsmodells erforderlich, die in der realen Prozesskonfiguration über Sensoren beziehungsweise Aktoren verfügen.

[4] WP steht in den Zustandsbezeichnern für *workpiece*.

Beim hier betrachteten Problem trifft dies auf den Roboter sowie auf die Bearbeitungsstation zu. Abbildungen 5.10 zeigt ein Konzeptmodell zur Erweiterung des in Abschnitt 5.2 beschriebenen Modellstandes um die erforderlichen Low-Level-Komponenten. Dessen konkrete Umsetzung mittels Simulink/Stateflow ist in der Abbildung 5.11 dargestellt.

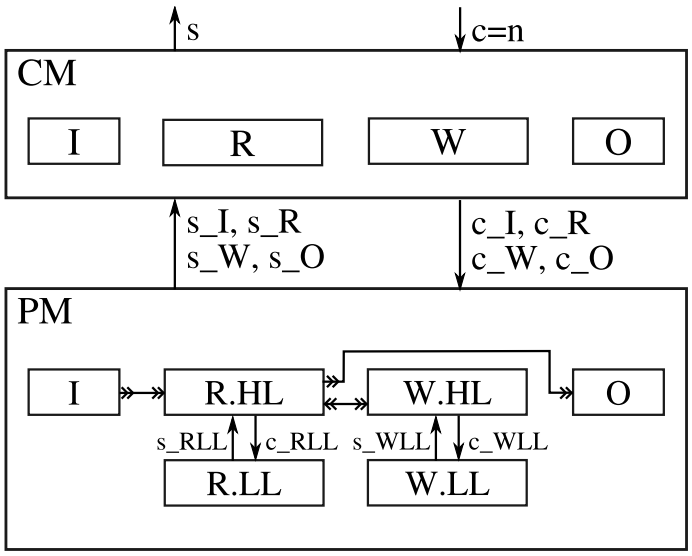


Abbildung 5.10: Konzeptmodell zur Detaillierung des Entwurfsmodells

Die Kommunikation zwischen den High-Level- und Low-Level-Komponenten eines Prozesselementes erfolgt wiederum über *c* - und *s* - *Größen*. Gegenüber dem Modellstand aus Abschnitt 5.2 ermöglichen diese Größen in den High-Level-Komponenten die Ersetzung der dort bisher angesetzten Bedienzeiten durch Werte, wie sie tatsächlich durch die in den Low-Level-Komponenten modellierten herstellerspezifischen Prozesselementen realisiert werden. Die dafür notwendigen Modifikationen werden nachfolgend ebenfalls nur für die Roboterkomponente exemplarisch erläutert. Für eine vollständige Darstellung des zweiten Schrittes der Automatisierungsphase des betrachteten Anwendungsproblems sei auf den Anhang A.2 verwiesen.

Die Abbildung 5.12 zeigt im oberen Teil das aus dem Zustandsdiagramm PM.R des ersten Schrittes der Automatisierungsphase hervorgegangene Diagramm PM.R.HL. Es ist ersichtlich, dass die bisherigen Bedienzeiten *pit* und *plt* nunmehr über den Austausch der Schnittstellengrößen *c_RLL* und *s_RLL* mit der hinzugefügten Low-Level-Komponente PM.R.LL realisiert werden. Die Struktur des Diagramms PM.R.LL entspricht im Wesentlichen der der High-Level-Komponente PM.R.HL. Im Zustand *PICK_WP_I* wird in PM.R.LL mit **ml.ptp(*posI*)** eine Point-to-Point-Bewegung des KUKA KR3 Roboters ausgelöst, wodurch dieser in eine Position über den Eingangspuffer verfährt[5]. Der Aufruf

[5]Die Angabe der Koordinaten der Zielposition ist hier zwecks besserer Übersichtlichkeit mit Hilfe des Pseudocodeparameters *posI* nur angedeutet. Für eine syntaktisch vollständige Darstellung der KR3-Schnittstellenfunktionen sei auf Abschnitt 4.1 sowie Anhang A.2.3 verwiesen.

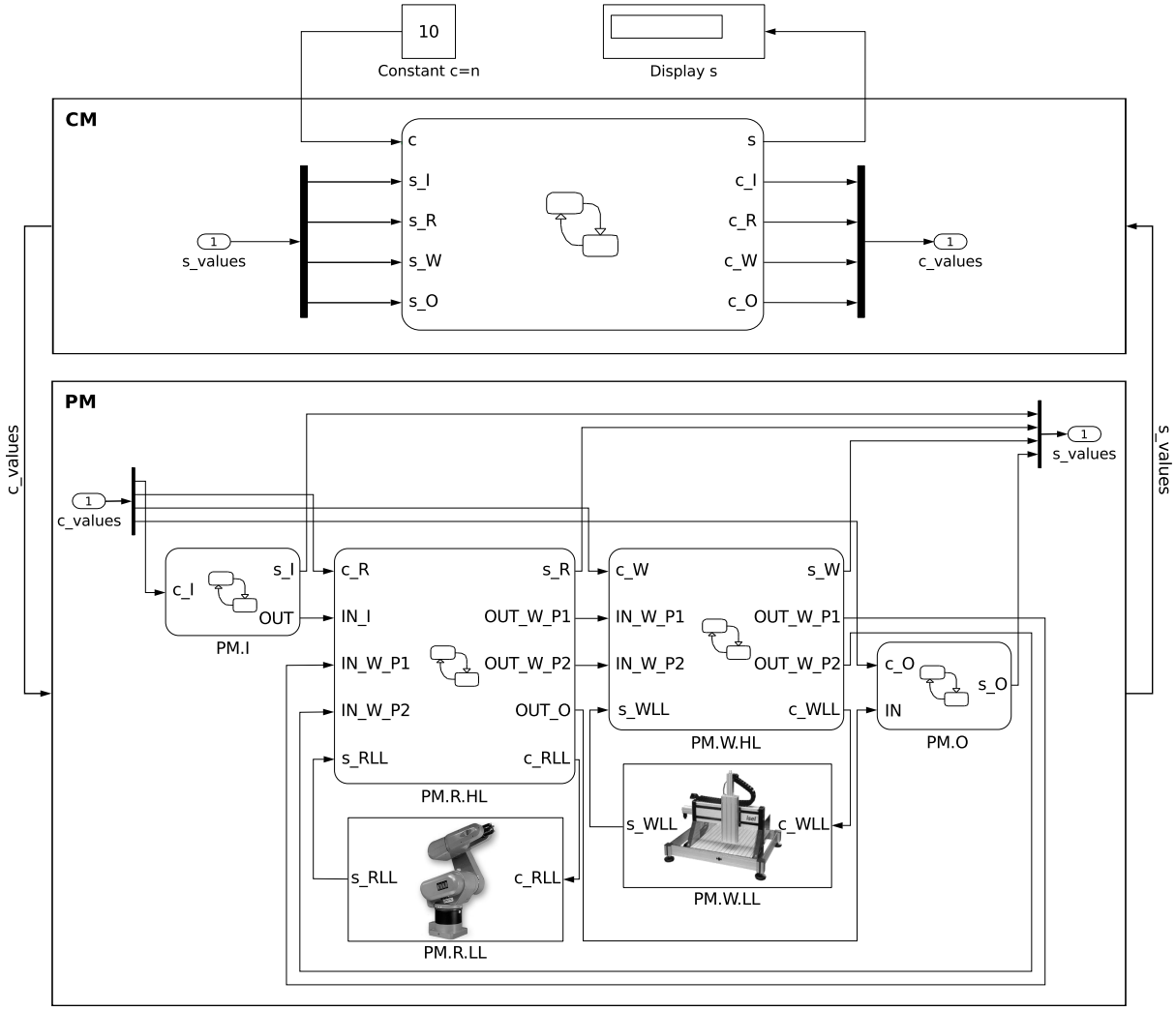


Abbildung 5.11: Detaillierung des Entwurfsmodells mittels Low-Level-Komponenten in Simulink

von ml.ptp(posI) hat neben seiner offensichtlichen aktorischen Wirkung auch eine sensorische Funktion, da der Aufruf erst nach Abschluss der Bewegung zurückkehrt.[6] Nachdem sich der Roboter über dem Eingangspuffer befindet, wird mittels ***posWP*=ml.getpos()** die Funkkamera *PFK 100K* zur Aufnahme angesteuert sowie die erforderliche Bildverarbeitung zur Objekt- und Positionserkennung ausgeführt. Die ermittelten Positionsdaten des zu transportierenden Werkstückes werden in der Variablen posWP übergeben. Der anschließende Aufruf von **ml.ptp(*posWP*)** deutet die Ansteuerung des Roboters zur Aufnahme des Werkstückes an. Tatsächlich sind dazu jedoch mehrere Bewegungs- und Greifer-Befehle

[6] Wegen der noch fehlenden Multi-Thread-Unterstützung in Matlab dürfen in lauffähigen Stateflow-Diagrammen derzeit nur nicht-blockierende Schnittstellenfunktionen aufgerufen werden. Nähere Informationen dazu gibt Anhang A.2.3.

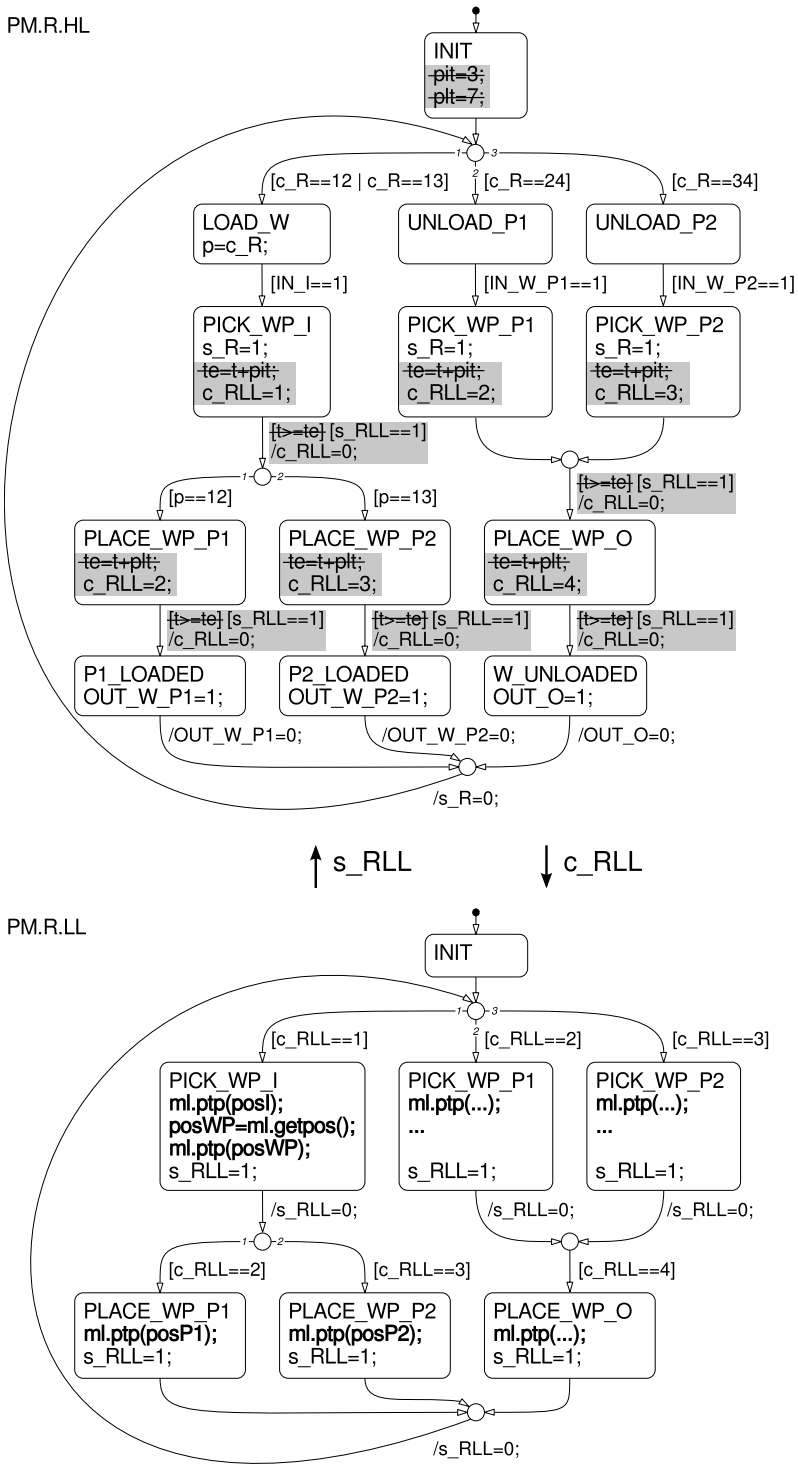


Abbildung 5.12: Modifikationen von PM.R zu PM.R.HL (grau unterlegt) und die Detaillierung PM.R.LL für einen konkreten KUKA-Roboter

erforderlich, auf deren Darstellung zu Gunsten der besseren Übersichtlichkeit hier verzichtet wurde. Dies trifft auch auf alle anderen Passagen der Abbildung 5.12 zu, in denen eine unmittelbare Roboteransteuerung erfolgt. Detailliertere Informationen hierzu werden im Anhang A.2.3 gegeben.

Die Realisierung der Schnittstellen zu den realen herstellerspezifischen Prozesselementen in der RCP-Umgebung Matlab wurde im Kapitel 4 beschrieben. Im gleichen Kapitel wurde dargestellt, dass ein bis auf die Sensor/Aktor-Ebene (Prozess-Interface) detailliertes Entwurfsmodell mittels CAR-Unterstützung simulativ erprobt werden kann. Nach Abschnitt 3.4 stellen diese Systemsimulationen den Abschluss der Entwurfsphase dar. Für das hier betrachtete Beispiel bestehend aus einem KUKA KR3 Roboter mit Funkkamera und einer ISEL-Bearbeitungsstation wurde zur Erprobung des detaillierten Modellstandes eine in Matlab realisierte CAR-Toolbox benutzt. Abbildung 5.13 zeigt eine Visualisierungsszene aus der abschließenden Systemsimulation.

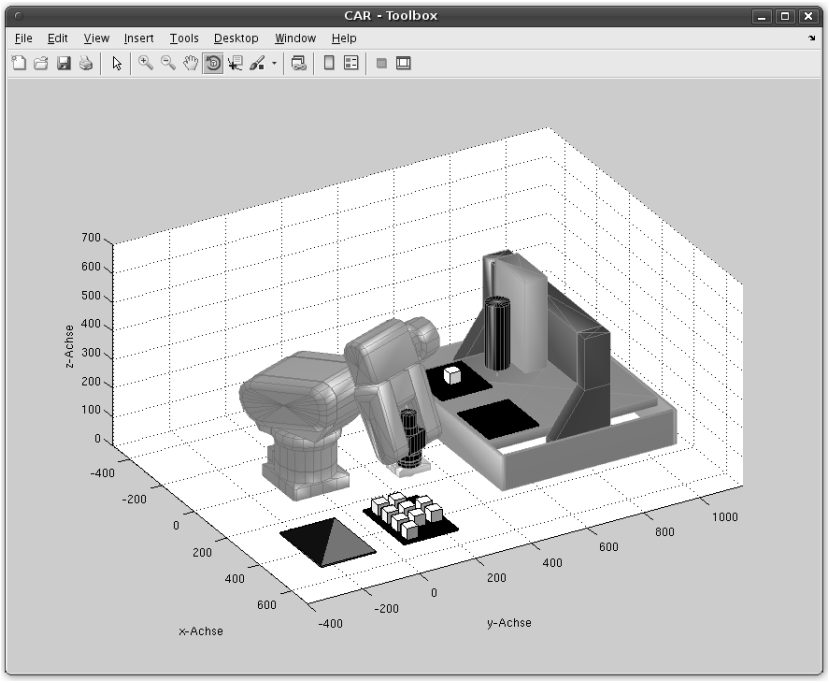


Abbildung 5.13: Visualisierung der abschließenden Systemsimulation

Beim Übergang zur Inbetriebnahmephase wird anstelle des CAR-Systems nun die Verbindung zu den realen Prozesselementen hergestellt. Das Entwurfsmodell kann dabei ohne Modifikationen im Rahmen einer SiL-Simulation am realen Prozess getestet werden. Da beim hier betrachteten Beispiel bezüglich der Steuerung nur geringe Echtzeitanforderungen bestehen, kann auch die bisher eingesetzte Entwurfsplattform (PC mit Standardbetriebssystem) als Steuerungsplattform weiter benutzt werden.[7] Im Unterschied zu den bisherigen

[7] Die Regelung und Steuerung der internen Sensorik und Aktorik mit höheren Echtzeitanforderungen wird durch den KUKA Controller KR C3 realisiert (s. Abb. 4.2).

Systemsimulationen wird die Ausführung des nun als Steuerung agierenden Entwurfsmodells innerhalb von Matlab mit der Realzeit synchronisiert. Nach erfolgreichem Abschluss der Inbetriebnahmetests kann die letzte SiL-Konfiguration als Steuerung für den operativen Betrieb eingesetzt werden. Entsprechend dem SBC-Ansatz realisiert die Steuerungssoftware damit neben der eigentlichen Steuerungslogik ebenfalls ein zum realen Prozess mitlaufendes Prozessmodell, welches zur Auswertung nicht gemessener Prozesszustände (Zustandsbeobachtung) oder zu Diagnosezwecken (zum Beispiel Erkennung von Sensor-Ausfällen) benutzt werden kann. Diese Aspekte des SBC-Ansatzes sind jedoch kein Gegenstand des hier betrachteten Beispielproblems.

5.4 Zusammenfassung

In diesem Kapitel wurde anhand eines konkreten Beispiels der gesamte Entwicklungsprozess einer komplexen Roboteranwendung auf Basis des SBC-Ansatzes innerhalb der RCP-Umgebung Matlab demonstriert.

Hinsichtlich der im Kapitel 2 eingeführten Klassifikation der Roboterprogrammierung ist die dargestellte Steuerungsentwicklung in die Kategorie der Offline-Verfahren einzuordnen.

Bei der Beschreibung der Roboter-Low-Level-Komponente im Abschnitt 5.3 wurde gezeigt, dass der Ansatz auf unterer Ebene die roboterorientierte Programmierung mittels einer herstellerspezifischen Sprache integriert. In den darüber liegenden Ebenen erfolgt die Programmierung des Roboters und anderer Prozesskomponenten dagegen mit den Mitteln der matrizenorientierten Programmiersprache von Matlab sowie mit Zustandsdiagrammen der Erweiterungskomponente Stateflow. Dies entspricht gemäß Kapitel 2 der roboterorientierten Programmierung auf Basis einer erweiterten allgemeinen Programmiersprache. Die hier verwendeten, dem Harel-Automaten entsprechenden, Zustandsdiagramme weisen gegenüber anderen für die Roboterprogrammierung verwendeten allgemeinen Programmiersprachen wie C oder Pascal entscheidende Vorteile auf. Dazu gehören insbesondere die komfortablen Möglichkeiten zur Modularisierung und Hierarchisierung auf Basis einer gemischten graphischen und textuellen Codierung sowie die einfache Realisierung von Nebenläufigkeiten durch parallele Zustände oder separate Stateflow-Diagramme.

Trotz dieser Vorteile sind der in diesem Kapitel demonstrierten Methodik hinsichtlich der Realisierung flexibler und adaptiver Steuerungen Grenzen gesetzt. Für solche Anwendungsprobleme erscheint der Ansatz der aufgabenorientierten Roboterprogrammierung in Kombination mit weiteren Techniken vielversprechend. Die beiden nachfolgenden Kapitel sollen sich deshalb diesem Problemkreis im Kontext eines SBC-basierten Entwicklungsprozesses von Robotersteuerungen widmen.

6 Aufgabenorientierte Robotersteuerungen

Gemäß der im Kapitel 2 eingeführten Klassifikation gehört die aufgabenorientierte Programmierung neben der roboterorientierten Programmierung zur Klasse der Offline-Methoden. Einige Systeme zur Unterstützung der aufgabenorientierten Programmierung wurden bereits im Abschnitt 2.3 beispielhaft vorgestellt. Nach [Hau07] und [Web09] handelt es sich bei diesen Projekten noch jeweils um Einzellösungen. Eine einheitliche Methodik der aufgabenorientierten Steuerungsentwicklung hat sich nach [Hau07] noch nicht herausgebildet. Im Wesentlichen basieren die Arbeiten zur Aufgabenorientierung aber zumindest auf einigen gemeinsamen Grundprinzipien. Diese werden im ersten Abschnitt des Kapitels erläutert. Anschließend wird gezeigt, wie die Aufgabenorientierung in einen auf dem SBC-Ansatz basierenden Entwicklungsprozess einzuordnen ist. Anhand von zwei Beispielen wird dann die Realisierung aufgabenorientierter Robotersteuerungen auf Basis der RCP-Plattform Matlab demonstriert.

6.1 Grundlegende Aspekte der Aufgabenorientierung

Die Grundidee der Aufgabenorientierung besteht in der Zerlegung eines komplexen Problems in Teilprobleme, die jeweils als zu lösende Aufgaben aufgefasst werden. Ein aufgabenorientiertes Roboterprogramm besteht damit im einfachsten Fall aus einer Sequenz von zu lösenden Aufgaben. Nach Haun ([Hau07]) beschreibt ein solches Programm nicht *wie* ein Robotersystem anzusteuern ist, um ein Problem zu lösen, sondern allein *welche* Aufgaben *in welcher* Reihenfolge durch das Robotersystem zu bewerkstelligen sind. Dabei sind neben einfachen Aufgabensequenzen auch bedingungsabhängige Verzweigungen in alternative Aufgabenfolgen und Wiederholungen sowie die nebenläufige Lösung von Aufgaben möglich.

Nach Kruth et al. ([Kru01]) ist der entscheidende Gegenstand einer Aufgabe, das zu erzielende Ergebnis. Eine typische Aufgabe kann zum Beispiel lauten: *Identifiziere Objekt*. Ergebnis der gelösten Aufgabe ist eine Datenstruktur *Objekt* mit den ermittelten Informationen wie beispielsweise Objekttyp und Position in einem Pufferbereich. Wie die Objektidentifikation durch den Roboter oder Komponenten seiner Umgebung erfolgt, ist nicht Gegenstand eines aufgabenorientierten Programms.

Häufig erweist es sich als zweckmäßig, eine Parametrierung von Aufgaben zu ermöglichen. Bei einer textuellen aufgabenorientierten Programmierung kann dies über Aufrufargumente realisiert werden:

Objekt = Identifiziere Objekt (Ort)

Mittels der Parametrierung könnte in diesem Beispiel festgelegt werden, in welchem von mehreren vorhandenen Eingangspufferbereichen einer Roboterzelle die Objektidentifikation erfolgen soll. Ohne Parametrierung wären beim gleichen Problem mehrere ähnliche Aufgaben erforderlich (Identifiziere Objekt im Puffer A, Identifiziere Objekt im Puffer B usw.).

In Aufgabenfolgen können die Ergebnisse einer Aufgabe zur Parametrierung von nachfolgenden Aufgaben benutzt werden:

Objekt = Identifiziere Objekt (Ort)

Status = Transportiere Objekt (Objekt, Zielort)

In diesem Beispiel wird die Aufgabe *Transportiere Objekt* mit dem Ergebnis der Aufgabe *Identifiziere Objekt* parametriert. Mit den in der Datenstruktur *Objekt* enthaltenen Informationen und dem zweiten Parameter *Zielort*, wird die allgemeine Aufgabe *Transportiere Objekt* zur parametrierten Aufgabe *Transportiere Objekt X vom Entnahmeort Y zum Ablageort Z*.

Das Ergebnis der Aufgabe Transportiere Objekt ist die Datenstruktur *Status*. In einer Umgebung in der die Abarbeitung nebenläufiger Aufgaben zulässig ist, kann über die Datenstruktur Status auch angezeigt werden, dass die Bearbeitung der Aufgabe Transportiere Objekt noch nicht abgeschlossen ist. Für andere Aufgaben könnte dies dann als Erlaubnis zur parallelen Ausführung ausgewertet werden.

Im Unterschied zu einem roboterorientierten Steuerungsprogramm werden bei der aufgabenorientierten Programmierung Details wie die Lage und Orientierung eines aufzunehmenden Objektes in einem konkreten Koordinatensystem oder dessen geometrische Abmessungen, die für einen tatsächlichen Greifvorgang erforderlich sind, zunächst außer Acht gelassen. Auf diese Art und Weise können auch sehr komplexe Probleme in überschaubarer Form auf einer vergleichsweisen abstrakten Ebene spezifiziert werden. Eine direkte Ausführung einer solchen aufgabenorientierten Steuerungsspezifikation ist jedoch aufgrund der fehlenden Detaillierung nicht möglich.

In der Literatur wird zur Lösung dieses Problems meist die Spezifikation der detaillierten Informationen – beispielsweise der geometrischen Daten aller Werkstücke, die Koordinaten von Aufnahme- und Ablageplätzen von Werkstücken und Ähnlichem – in einem von der Aufgabenspezifikation getrennten *Weltmodell* vorgeschlagen ([Sie96], [Hau07], [Web09]). In einem möglichst automatisch auszuführenden Prozess soll dann ein *Aufgabentransformator* die aufgabenorientierte Steuerungsspezifikation unter Zuhilfenahme des Weltmodells in ausführbare Roboterbefehle umsetzen. In frühen Arbeiten zur Aufgabenorientierung wie beispielsweise in [Loz82] ist die Überführung der aufgabenorientierten Spezifikation in eine ausführbare Form noch eindeutig in der Entwurfsphase der Steuerungsentwicklung angesiedelt. Das heißt unter Zuhilfenahme eines Compilers und der Informationen des Weltmodells wird die gesamte aufgabenorientierte Steuerungsspezifikation in ein roboterorientiertes Steuerungsprogramm übersetzt, welches dann, nach bestandenen Inbetriebnahmetests, im operativen Betrieb eingesetzt werden kann.

In neueren Arbeiten, wie in Abbildung 6.1 dargestellt, wird die Aufgabentransformation als Funktionalität der operativen Steuerung verstanden. Damit stehen für die Transformation einer Aufgabe in ausführbare Roboterbefehle neben den Informationen aus dem Weltmodell auch aktuelle Zustandsinformationen des realen Prozesses zur Verfügung. Bei diesem Ansatz kann die Aufgabentransformation nicht mehr als Offline-Prozess realisiert werden, sondern muss als Bestandteil der operativen Steuerung in Echtzeit erfolgen.

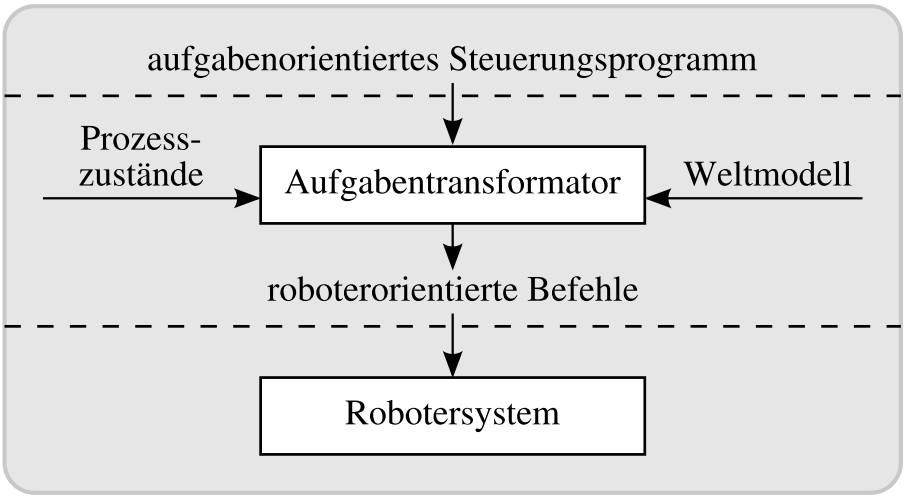


Abbildung 6.1: Schichtenmodell einer aufgabenorientierten Steuerung in Anlehnung an [Web09]

6.2 Aufgabenorientierung im Kontext des SBC-Ansatzes

Der SBC-Ansatz wurde im Abschnitt 3.5 als ein Verfahren für das Rapid Control Prototyping ereignisdiskreter Steuerungen vorgestellt und seine Anwendung im Entwicklungsprozess einer komplexen Robotersteuerung im Kapitel 5 an einem konkreten Beispiel demonstriert. Aus diesen Teilen der Arbeit ist bekannt, dass beim SBC-Ansatz zwischen einer Planungs- und einer Automatisierungsphase unterschieden wird.

In der anfänglichen Planungsphase wird die Entwurfsaufgabe als Materialflussproblem betrachtet und modelliert. Mittels Systemsimulationen werden geeignete Topologien und Dimensionierungen des Materialflusssystems bestimmt. Wie am Beispiel im Kapitel 5 gezeigt wurde, kann auch die Untersuchung unterschiedlicher Steuerungsstrategien bereits Gegenstand dieser Phase sein. Hierbei handelt es sich aber noch nicht um konkrete Steuerungsentwürfe, auch wird in der Regel noch nicht zwischen einer Steuerungs- und Prozessebene unterschieden. Die Planungsphase ist damit hinsichtlich einer später verfolgten roboterorientierten oder aufgabenorientierten Steuerungsentwicklung noch unspezifisch.

Der Entwurf einer konkreten Steuerung ist erst Gegenstand der anschließenden Automatisierungsphase. Nach dem SBC-Ansatz besteht diese Phase aus zwei wesentlichen

Schritten. Im ersten Schritt wird die Separierung von Steuerungsebene und Prozessebene eingeführt, was zu einer getrennten Modellierung der konkreten Steuerungslogik in einem Steuerungsmodell (CM) und der ereignisdiskreten Prozessdynamik in einem Prozessmodell (PM) führt. Das Prozessmodell hat im ersten Schritt der Automatisierungsphase noch generischen Charakter, das heißt es ist noch unabhängig von den im Prozess konkret verwendeten Prozesselementen. Die Detaillierung des Prozessmodells bis auf die Sensor-/Aktorebene der konkreten Prozesselemente ist Gegenstand des zweiten Schrittes der Automatisierungsphase. Das Steuerungsmodell (CM) bleibt in diesem Schritt unverändert. Das Prozessmodell (PM) wird in die Ebenen PM-HL (generischer Teil) und PM-LL (prozesselementeabhängiger Teil) untergliedert.

Damit muss ein aufgabenorientierter Steuerungsentwurf im SBC-Ansatz offensichtlich im ersten Schritt der Automatisierungsphase erfolgen. Das bereits im Kapitel 5 benutzte Beschreibungsmittel der Zustandsdiagramme erfüllt dazu alle Voraussetzungen, das heißt es können Aufgabensequenzen, Verzweigungen, Wiederholungen als auch nebenläufige Aufgaben spezifiziert werden. Entsprechend den im Abschnitt 6.1 dargestellten Grundprinzipien ist die Ausführung eines aufgabenorientierten Steuerungsmodells aufgrund der fehlenden Detaillierung für sich allein nicht möglich. Im SBC-Ansatz kann ein aufgabenorientiertes Steuerungsmodell aber im Verbund mit dem Prozessmodell im Rahmen einer Systemsimulation ausgeführt werden. Daraus folgt, dass das Prozessmodell im SBC-Ansatz sowohl die Funktion des Weltmodells als auch die des Aufgabentransformators gemäß Abschnitt 6.1 realisiert. Bei der weiteren Detaillierung des Prozessmodells im zweiten Schritt der Automatisierungsphase in die PM-HL- und PM-LL-Ebene gilt die Trennung zwischen generischem und prozesselementespezifischen Teil dann auch für die Daten im Sinne des Weltmodells als auch für die Aufgabentransformation.

Mit der Verwendung von Zustandsdiagrammen nach Harel eröffnet sich auch bei aufgabenorientiert entworfenen Steuerungen die Möglichkeit der Hierarchisierung, das heißt aus bereits vorhandenen Aufgabenspezifikationen können komplexere Aufgaben aggregiert werden. Im SBC-Ansatz werden Aufgaben, die nicht aggregiert sind und somit direkt auf Komponenten des Prozessmodells abbilden, als *Basisaufgaben* bezeichnet. In einigen Anwendungsfeldern spielt die Wiederverwendbarkeit von Basisaufgaben und eventuell der Aufbau aggregierter Aufgaben eine Rolle. In diesen Fällen ist dann eine separate Speicherung von Aufgabenspezifikationen in einer sogenannten Aufgabenbibliothek erforderlich. Wegen der Abhängigkeit der Basisaufgaben von ihrer Aufgabentransformation durch die Komponenten eines Prozessmodells ist die Realisierung eines solchen Bibliothekskonzeptes nicht trivial. Im Kapitel 7 wird das Konzept der Aufgabenbibliothek benutzt und an dieser Stelle noch weitergehend erläutert.

6.3 Entwurf und Inbetriebnahme aufgabenorientierter Robotersteuerungen

Im vorangegangenen Abschnitt wurde ausgeführt, dass die Planungsphase bezüglich einer aufgabenorientierten Steuerungsentwicklung unspezifisch ist. Gemäß dem SBC-Ansatz

erfolgt ein aufgabenorientierter Steuerungsentwurf im ersten Schritt der Automatisierungsphase. Anhand von zwei Beispielproblemen soll der aufgabenorientierte Steuerungsentwurf detailliert betrachtet und die konkrete Durchführung auf Basis der RCP-Plattform Matlab demonstriert werden.

Beim ersten Beispielproblem handelt es sich um das bereits aus Kapitel 5 bekannte materialtechnische Bearbeitungsproblem. Damit ist ein unmittelbarer Vergleich zwischen der dort betrachteten roboter- beziehungsweise prozesselementeorientierten Entwurfstechnik mit dem in diesem Kapitel verfolgten aufgabenorientierten Ansatz möglich.

Das zweite Beispielproblem widmet sich dem Anwendungsfall einer Bauteilsortierung. Dabei wird gezeigt, wie in einem aufgabenorientierten Steuerungsentwurf das Konzept der Parametrierung von Basisaufgaben vorteilhaft genutzt werden kann.

6.3.1 Materialtechnisches Bearbeitungsproblem

Abbildung 6.2 zeigt die Prozessstruktur des bereits im Kapitel 5 betrachteten materialtechnischen Bearbeitungsproblems, für das zu Vergleichszwecken nun ein aufgabenorientierter Steuerungsentwurf durchgeführt werden soll.

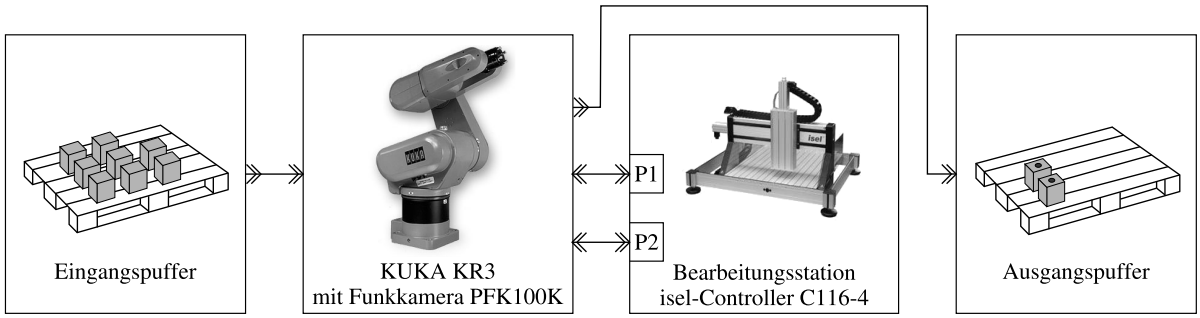


Abbildung 6.2: Beispielproblem 1 – materialtechnisches Bearbeitungsproblem

In Abschnitt 6.2 wurde ausgeführt, dass die Grundstruktur des Entwurfsmodells nach dem SBC-Ansatz unabhängig vom konkret angewandten Entwurfsansatz der Steuerung ist. Das heißt, das Entwurfsmodell besteht definitiv aus einem Steuerungsmodell (CM) und einem Prozessmodell (PM). Beide Modelle sind über eine c/s - Schnittstelle verbunden. Das Steuerungsmodell verfügt zusätzlich über eine c/s - Schnittstelle zu einer übergeordneten Leit- beziehungsweise Bedienebene. Abbildung 6.3 zeigt diese Grundstruktur nochmals anhand des Konzeptmodells zum prozesselementeorientierten Entwurf aus Kapitel 5.

Die Struktur des Prozessmodells ist unabhängig von der angewandten Art des Steuerungsentwurfs. Das Prozessmodell bildet in jedem Fall die Struktur des realen Materialflusssystems ab.

In einem modularen prozesselementeorientierten Steuerungsentwurf existiert in der Regel zu jeder Prozesskomponente eine korrespondierende Komponente im Steuerungsmodell. Beim aufgabenorientierten Steuerungsentwurf löst man sich von dieser Regel vollständig.

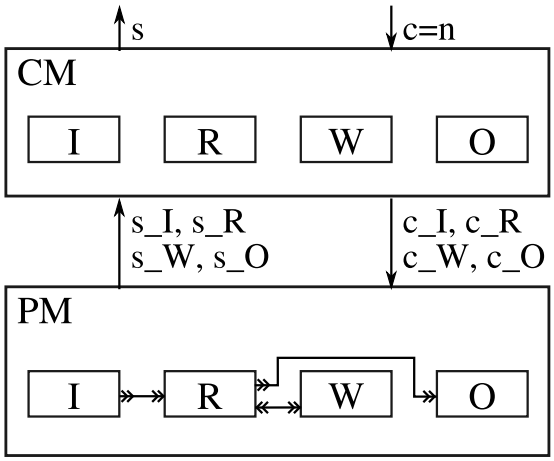


Abbildung 6.3: Konzeptmodell mit prozesselementeorientiertem Steuerungsmodell aus Kapitel 5

Hier steht die Frage im Mittelpunkt: *Welche Aufgabe beziehungsweise Aufgaben sind zu lösen?* Antworten auf diese Frage der Aufgabenorientierung können allein auf Basis der Prozessstruktur (PM) nicht gefunden werden. Für einen aufgabenorientierten Entwurf ist vor allem die aus der Planungsphase übergebene und in der Automatisierungsphase zu realisierende Steuerungsstrategie von zentraler Bedeutung. Im hier betrachteten Beispiel besteht diese darin, dass der Roboter R als Transportelement die Bearbeitungsstation W mit Werkstücken aus dem Eingangspuffer I zu bestücken hat. Nach erfolgter Bearbeitung ist die Bearbeitungsstation zu entstücken und die bearbeiteten Werkstücke sind in den Ausgangspuffer O zu transportieren. Da die Bearbeitungsstation zwei Bohrplätze für je ein Werkstück besitzt, soll die Be- und Entstückung parallel zur Bearbeitung erfolgen. Aus dieser Anforderungsspezifikation der Steuerungsstrategie können unmittelbar Aufgaben, die durch die Steuerung zu realisieren sind, abgeleitet werden:

$R_{LOAD} = LOAD$ – Bestücken von W durch R,

$R_{PROCESS} = PROCESS$ – Bearbeiten von Werkstücken durch W,

$R_{UNLOAD} = UNLOAD$ – Entstücken von W durch R.

Auf dieser Basis kann ein erstes Konzeptmodell, wie in Abbildung 6.4 gezeigt, entworfen werden.

Die Ergebnisvariablen R_{LOAD}, $R_{PROCESS}$ und R_{UNLOAD} sind im Konzeptmodell nicht aufgeführt, weil die dauerhaft und nebenläufig zu erledigenden Aufgaben LOAD, PROCESS und UNLOAD potentiell nicht zum Abschluss kommen. Das heißt, nach Abschluss eines Auftrages $c = n$, der mittels s an die Leit- beziehungsweise Bedienebene signalisiert wird, ist die Aufgabenausführung nur vorübergehend blockiert, bis ein neuer Auftrag an das Steuerungsmodell ergeht.

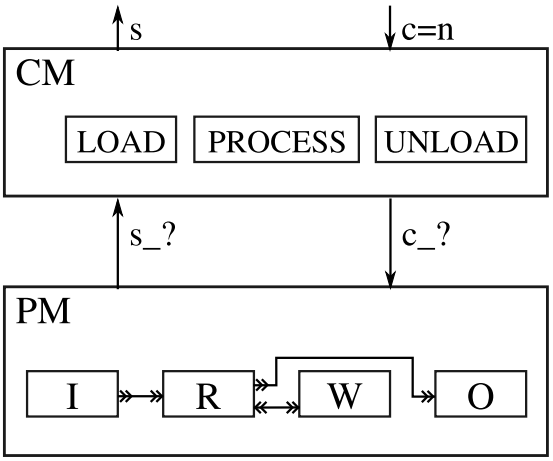


Abbildung 6.4: Erstes Konzeptmodell des aufgabenorientierten Entwurfs

Im Unterschied zum prozesselementeorientierten Steuerungsentwurf gibt es zwischen den spezifizierten Aufgaben der Steuerung und den Komponenten des Prozessmodells keine eindeutige Zuordnung mehr. Die konkrete Gestalt der c/s - Schnittstelle zwischen Steuerungs- und Prozessmodell ist damit noch unklar, was durch c_? und s_? in Abbildung 6.4 ausgedrückt wird.

Da die bisher spezifizierten Aufgaben relativ komplex sind, erscheint es sinnvoll dem Top-Down-Ansatz folgend, nach angemessenen Untergliederungen in Teilaufgaben zu suchen. Für die Lösung der Aufgabe LOAD ist der Roboter R von zentraler Bedeutung. Dieser ist mit bildgebender Sensorik ausgestattet und ist damit das Prozesselement, welches in der Lage ist, die genaue Position von Werkstücken im Eingangspuffer I zu bestimmen. Erst nach dieser Positionsbestimmung ist eine Werkstückaufnahme möglich und schließlich der Transport des Werkstückes und seine Platzierung auf einem der zwei Bohrplätze der Bearbeitungsstation W. Eine geeignete Zerlegung der Aufgabe LOAD wäre damit:

LOAD { R_{IOBJ} *= IOBJ;* *- Identify Object(s)*
R_{PIO_I} *= PIO_I;* *- Pick Object out of I*
R_{PLO_W} *= PLO_W;* *- Place Object in W*
}

Die Teilaufgaben IOBJ, PIO_I und PLO_W sind zur Lösung der Aufgabe LOAD sequentiell und in Wiederholung auszuführen.

Die Aufgabe PROCESS ist von der Bearbeitungsstation W durchzuführen und zwar für sämtliche Werkstücke aller Aufträge c = n. Hier scheint offensichtlich eine Zerlegung in eine einzelne Teilaufgabe sinnvoll, die sich auf die Bearbeitung *eines* Werkstückes auf einem der beiden Bohrplätze beschränkt und wiederholt ausgeführt werden kann:

$$PROCESS\ \{\ R_{PRO} = PRO;\ \ \text{- Process Workpiece}$$
$$\}$$

Für die Lösung der Aufgabe UNLOAD ist wiederum der Roboter R in Koordination mit der Bearbeitungsstation W verantwortlich. Anders als beim Bestücken ist beim Entstücken die Position des aufzunehmenden Werkstückes a priori bekannt. Dies gilt auch für die Ablage im Ausgangspuffer O. Somit ist folgende Unterteilung von UNLOAD möglich:

$$UNLOAD\ \{\ R_{PIO\_W} = PIO\_W;\ \ \text{- Pick Object out of W}$$
$$R_{PLO\_O} = PLO\_O;\ \ \text{- Place Object in O}$$
$$\}$$

Die Bearbeitung der beiden Teilaufgaben von UNLOAD muss sequentiell und in Wiederholung erfolgen.

Im Ergebnis der obigen Top-Down-Zerlegung der Aufgaben LOAD, PROCESS und UNLOAD ergeben sich insgesamt *sechs* Teilaufgaben. Eine weitere Zerlegung dieser Aufgaben erscheint nicht vorteilhaft. Damit bilden sie die Menge der zu realisierenden Basisaufgaben. Nach Abschnitt 6.1 sind diese durch Aufgabentransformation in eine ausführbare Form zu überführen. Im SBC-Ansatz ist dazu gemäß Abschnitt 6.2 das Prozessmodell verantwortlich. Aus dieser Vorgabe lässt sich eine zweckmäßige Gestaltung der zuvor noch unklaren c/s - Schnittstelle zwischen Steuerungs- und Prozessmodell ableiten. Es erscheint offensichtlich sinnvoll, für jede Basisaufgabe eine c - Größe zur Signalisierung der Aufgabenbearbeitung an das Prozessmodell und eine s - Größe zur Rückmeldung der Aufgabenerledigung zu spezifizieren. Ein entsprechend verfeinertes Konzeptmodell des Problems ist in Abbildung 6.5 dargestellt.

Im Unterschied zur Abbildung 6.4 sind nunmehr die Ergebnisvariablen der Basisaufgaben im Konzeptmodell mit aufgeführt, obwohl diese ebenfalls nur die Statusinformation über die erfolgreiche Erledigung einer Aufgabe enthalten. In Verbindung mit den gepunkteten Pfeilen soll dies die Sequenzrelation zwischen den Basisaufgaben verdeutlichen. Das heißt, erst nach dem Vorliegen eines Ergebnisses ist eine Aufgabe beendet und erst dann kann die Bearbeitung der nachfolgenden Aufgabe an das Prozessmodell angewiesen werden.

Auf Basis des Konzeptmodells kann nun eine konkrete Spezifikation des Steuerungsmodells unter Nutzung von Zustandsdiagrammen erfolgen. Die nebenläufigen Aufgaben LOAD, PROCESS und UNLOAD werden naheliegender Weise als parallele Elternzustände dargestellt, die die Basisaufgaben in Form von Unterzuständen enthalten. Abbildung 6.6 zeigt das auf diese Art aufgebaute Entwurfsmodell.

Im direkten Vergleich der Abbildungen 6.3 und 6.5 wird sichtbar, dass wie bereits angemerkt die Grundstruktur des Prozessmodells von der für das Steuerungsmodell angewandten Entwurfstechnik unabhängig ist. Allerdings ist aus den beiden Abbildungen auch ersichtlich, dass dies nicht für die c/s - Schnittstelle zwischen dem Steuerungs- und dem

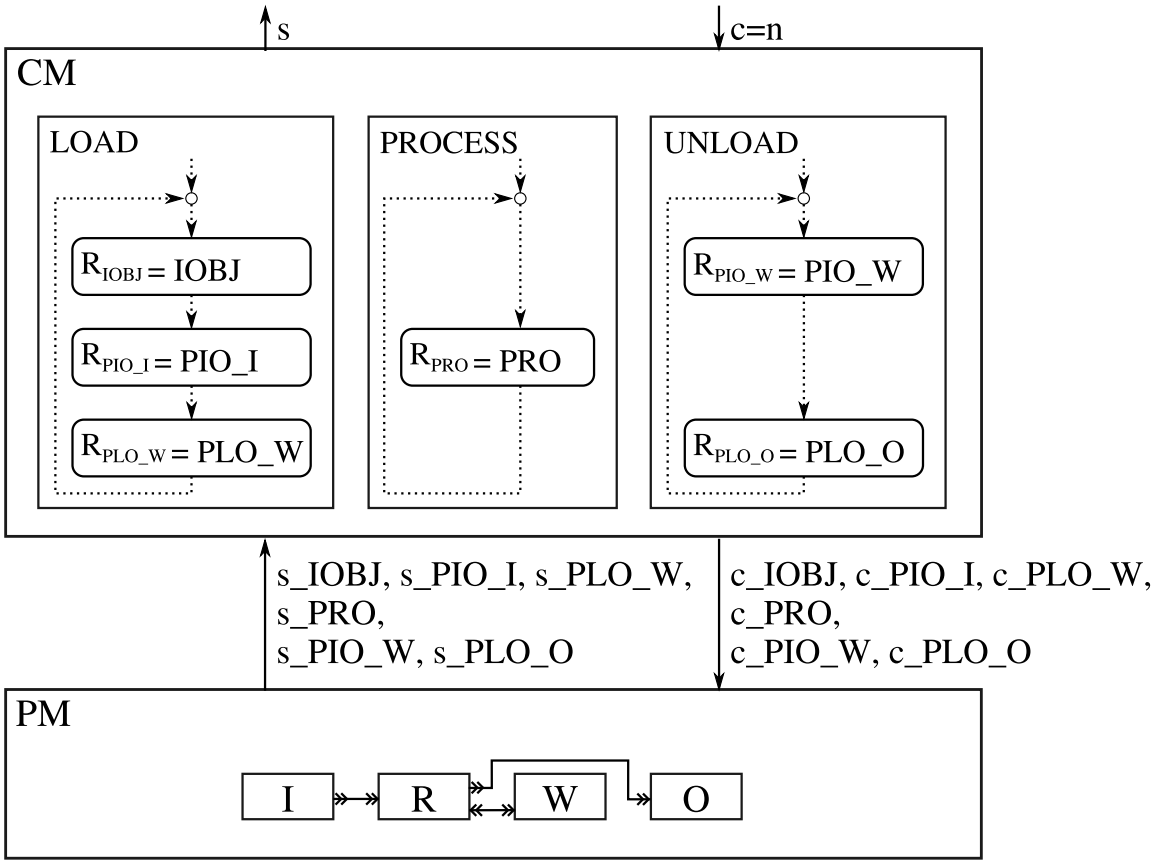


Abbildung 6.5: Verfeinertes Konzeptmodell des aufgabenorientierten Entwurfs

Prozessmodell gilt. Damit ist es nicht möglich, ein eventuell vorhandenes Prozessmodell aus einem prozesselementeorientierten Steuerungsentwurf ohne Anpassungen in einem aufgabenorientierten Entwurf zu übernehmen.

Der Aufbau eines gemäß Abbildung 6.6 passenden Prozessmodells soll nachfolgend beispielhaft für das Prozesselement Roboter demonstriert werden. Für eine vollständige Darstellung des Prozessmodells nach dem ersten aufgabenorientierten Entwurfsschritt der Automatisierungsphase sei an dieser Stelle auf den Anhang A.3.1 verwiesen.

Aus den oben erfolgten Aufgabenspezifikationen kann abgeleitet werden, dass der Roboter nur an der Ausführung der Basisaufgaben IOBJ, PIO_I, PLO_W, PIO_W und PLO_O beteiligt ist. Damit ist die c/s-Schnittstelle der Roboterkomponente als Untermenge der in Abbildung 6.6 gezeigten Gesamtschnittstelle zwischen Prozess- und Steuerungsmodell eindeutig bestimmt. Die Implementierung der Komponenten des Prozessmodells soll wie im Abschnitt 5.2 in Form von Zustandsdiagrammen erfolgen. Eine mögliche Lösung für die Roboterkomponente ist in Abbildung 6.7 dargestellt.

Mit der Implementierung aller Komponenten des Prozessmodells (s. Anhang A.3.1) ist der erste Schritt der Automatisierungsphase abgeschlossen und der erreichte Entwurfs-

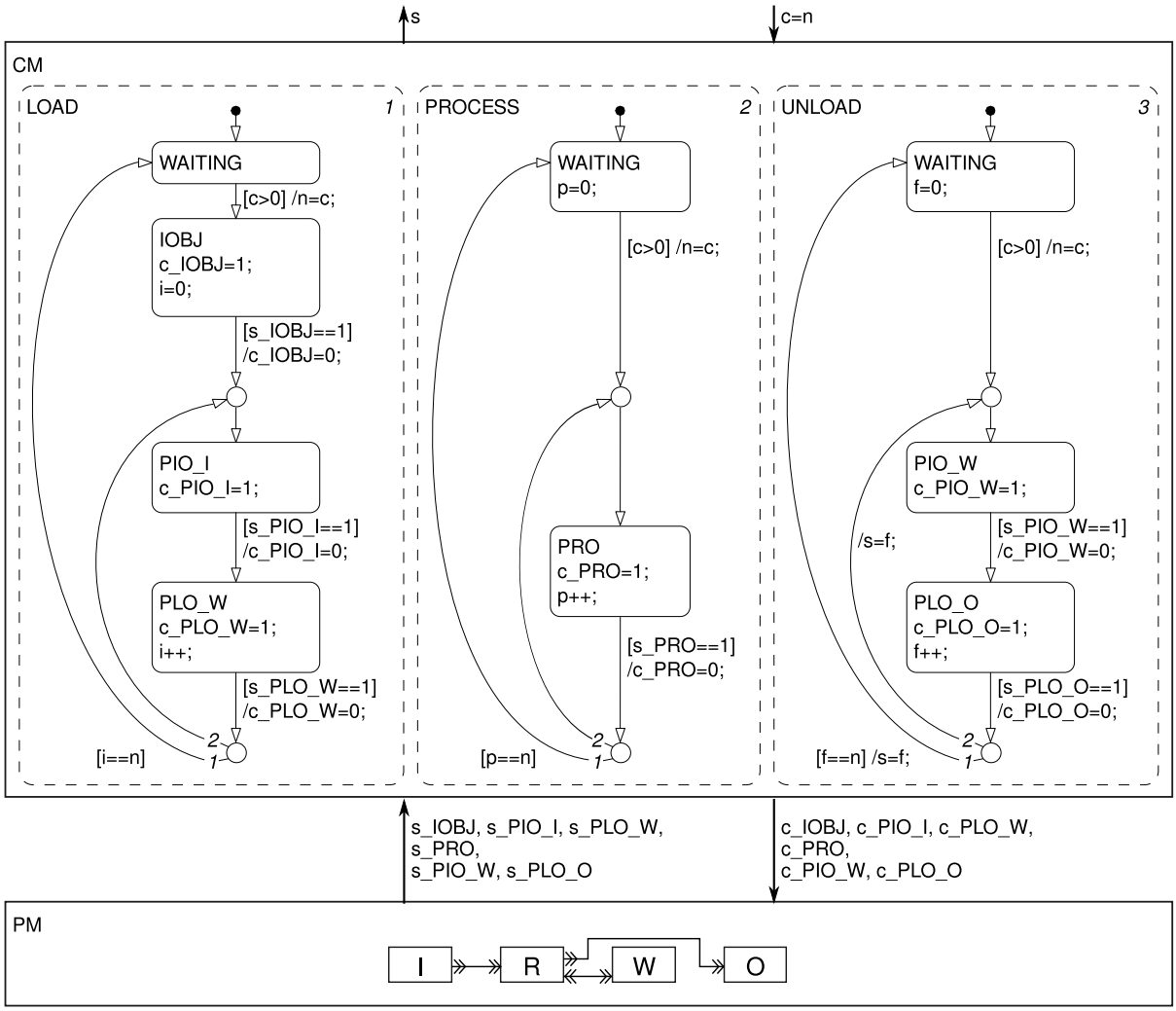


Abbildung 6.6: Entwurfsmodell mit aufgabenorientiertem Steuerungsmodell

stand kann mittels Systemsimulationen erprobt werden. Im zweiten Schritt der Automatisierungsphase ist das Prozessmodell, wie bereits im Abschnitt 5.3 demonstriert, bis auf die Sensor-/Aktor-Ebene der realen Prozesselemente zu detaillieren. Abbildung 6.8 zeigt die Struktur des Entwurfsmodells in dieser Phase als Konzeptmodell. Die Implementierungen der HL- und LL-Komponenten als Zustandsdiagramme sind im Anhang A.3.2 aufgeführt. Dieser Entwurfsstand kann mit Hilfe von SiL-Simulationen erprobt werden. Ein ausgetesteter Modellstand kann schließlich als Steuerungssoftware für den operativen Betrieb eingesetzt werden.

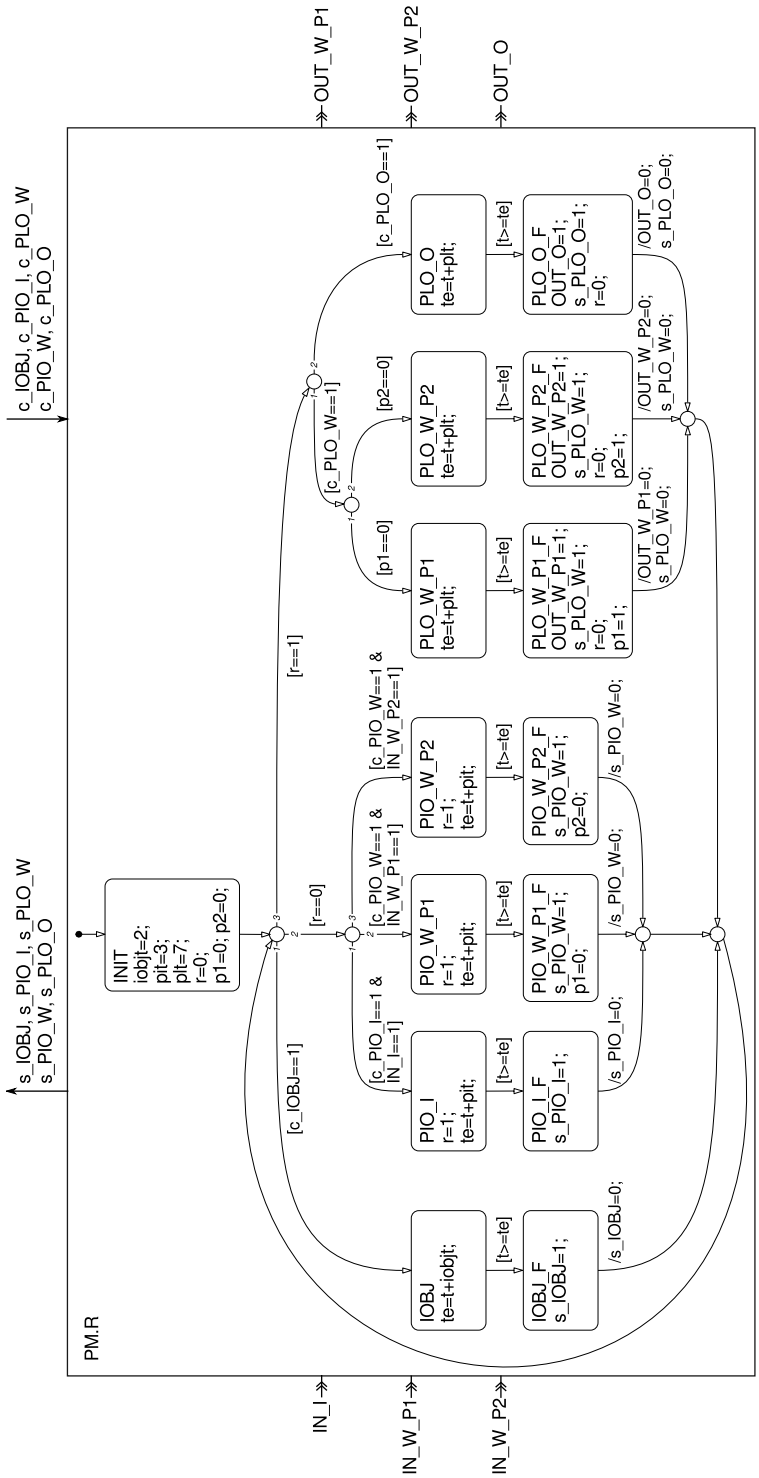


Abbildung 6.7: Zustandsdiagramm der Prozesskomponente PM.R

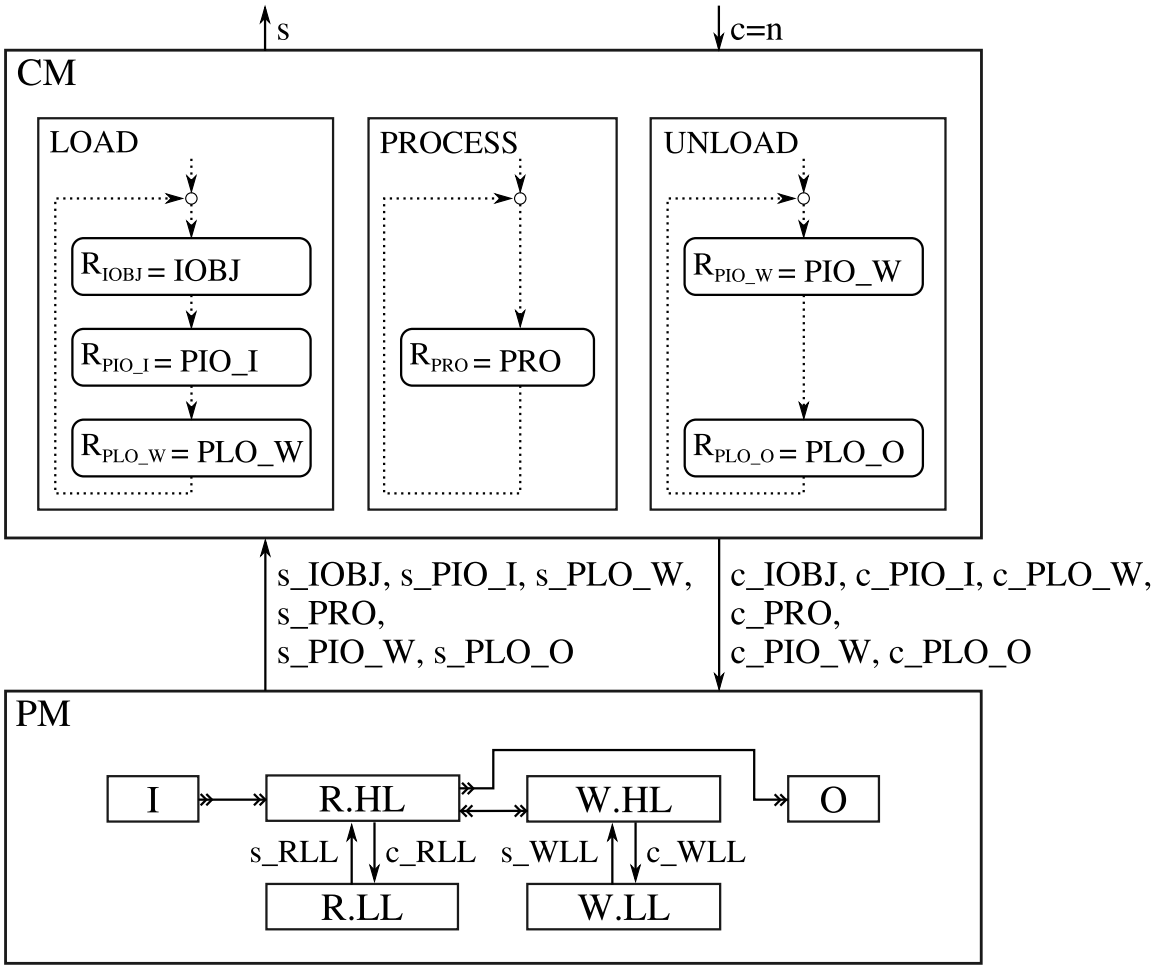


Abbildung 6.8: Konzeptmodell des detaillierten aufgabenorientierten Entwurfsmodells

6.3.2 Bauteilsortierungsproblem

Abbildung 6.9 zeigt die Prozessstruktur einer Bauteilsortierung mit Hilfe eines Knickarm-Industrieroboters. An dieser Anwendung soll der vorteilhafte Einsatz von parametrierbaren Basisaufgaben demonstriert werden.

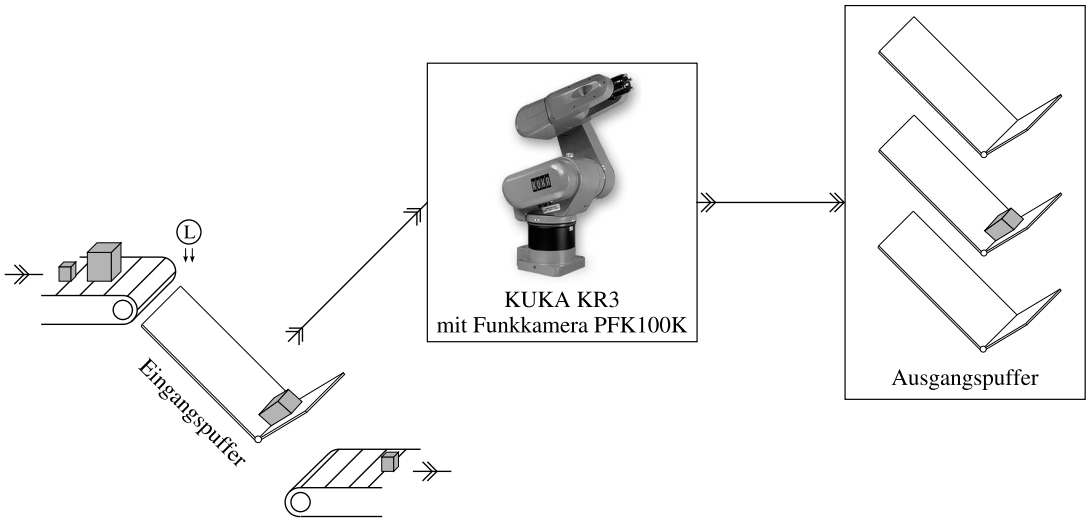


Abbildung 6.9: Beispielproblem 2 – Bauteilsortierung

Gegenüber dem im vorangegangenen Abschnitt betrachteten Bearbeitungsproblem weist die Sortieranwendung zwei wesentliche Unterschiede auf. Die Bauteilsortierung ist Teilsystem eines größeren Anwendungsproblemes mit stärkerer sowohl materialflusstechnischer als auch steuerungstechnischer Verkopplung mit den vor- und nachgelagerten Systemen. Das vorgelagerte System ist hier eine fortlaufende Zuführung eines Einzelteilstromes über eine Förderbandanlage. Aus diesem Teilestrom sollen mehrere Sortierstationen bestimmte Einzelteile entnehmen und in untergliederten Zwischenpuffer an nachgelagerte Montageplätze bereitstellen. Der zweite Unterschied besteht darin, dass in dieser Anwendung Einzelteile sehr verschiedener Typen durch den Roboter zu greifen sind.

Die grundsätzliche Prozessstruktur ist, wie in Abbildung 6.10 dargestellt, sehr einfach.

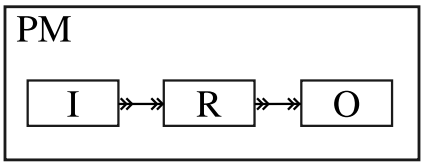


Abbildung 6.10: Prozessstruktur einer Sortierstation

Aufgrund der materialflusstechnischen Kopplung zum vorgelagerten System ist der Eingangspuffer I jedoch eine aktive und dynamische Prozesskomponente. Wie aus Abbildung

6.9 ersichtlich, besitzt der Eingangspuffer als aktorisches Element eine schwenkbare Klappe. Im geöffneten Zustand passieren die Bauteile den Eingangspuffer ungehindert. Bei geschlossener Klappe werden alle ankommenden Bauteile vorübergehend im Eingangspuffer aufgehalten. Mittels einer Lichtschranke im Eingangsbereich des Puffers wird der Belegungszustand registriert. Spätestens bei Erreichen der maximalen Pufferkapazität wird die Klappe wieder geöffnet und die aufgehaltenen Bauteile fließen an nachgelagerte Systeme ab. Während geschlossener Eingangspufferklappe ist die Entnahme von Bauteilen aus dem Eingangsstrom durch den Roboter R möglich. In welcher Reihenfolge welche Typen von Bauteilen aus dem Bauteilstrom zu entnehmen sind, ergibt sich aus dem Bedarf des nachgelagerten Montageprozesses und wird der Steuerung der Sortierstation aus der übergeordneten Ebene mitgeteilt. Dazu gehört auch die Information in welchen Bereichen des Ausgangspuffers O die Sortierstation die gewünschten Bauteile dem Montageprozess bereitzustellen hat.

Im Unterschied zum Bearbeitungsproblem im vorangegangenen Abschnitt soll nachfolgend für die Bauteilsortierung ein aufgabenorientierter Steuerungsentwurf in einer Bottom-Up-Vorgehensweise demonstriert werden. Ausgangspunkt sollen die primär vom Roboter zu lösenden Basisaufgaben sein, die darin bestehen, ein Bauteil vom gewünschten Typ im Eingangsstrom zu *identifizieren*, ein identifiziertes Bauteil aus dem Eingangspuffer *aufzunehmen* und dieses im geforderten Bereich des Ausgangspuffers zu *platzieren*. Damit ergeben sich erneut die schon aus dem ersten Beispielproblem bekannten Basisaufgaben IOBJ, PIO und PLO. Bei der Bauteilsortierung ist allerdings zu beachten, dass diese Aufgaben nunmehr mit verschiedenen Typen von Bauteilen zu lösen sind. Ein Entwurfsansatz bei dem für jeden möglichen Objekttyp eine spezifische Ausprägung der Aufgaben vorgesehen wird, würde zu einer großen Anzahl von Basisaufgaben führen und ist deshalb ungünstig. Dieses Problem lässt sich wesentlich effizienter durch die nachfolgend verwendete Technik der Aufgabenparametrierung lösen:

$$R_{IOBJ} = IOBJ\,(P_{IOBJ}),$$

$$R_{PIO} = PIO\,(P_{PIO}),$$

$$R_{PLO} = PLO\,(P_{PLO}).$$

Durch den Einsatz der Parametrierung werden Aufgaben in einem gewissen Maße flexibilisiert. So wird mit IOBJ (P_{IOBJ}) unter der Voraussetzung einer geeigneten bildgebenden Sensorik die Objektidentifikation auf einen bestimmten in P_{IOBJ} übergebenen Bauteiltyp eingegrenzt. Mit P_{PIO} kann in analoger Weise eine PIO-Aufgabe für einen bestimmten Objekttyp spezifiziert werden. Mittels P_{PLO} ist die Angabe des Ablegebereiches im untergliederten Ausgangspuffer für die PLO-Aufgabe möglich.

Aus wiederholten Sequenzen der Basisaufgaben kann eine vollständige Steuerungsaufgabe der Sortierstation aggregiert werden:

$$
\begin{aligned}
&R_{SP} = SP\,(P_{SP}) \quad \text{- Supply Parts} \\
&\quad i{+}{+} \\
&\{\; R_{IOBJ} = IOBJ\,(P_{SP}(i)); \\
&\quad R_{PIO} = PIO\,(R_{IOBJ}); \\
&\quad R_{PLO} = PLO\,(i); \\
&\}
\end{aligned}
$$

SP steht für *Supply Parts*. Die aggregierte Aufgabe SP ist durch P_{SP} ebenfalls parametrierbar. Aus der übergeordneten Ebene könnte die Sortierstation damit beispielsweise den folgenden Auftrag erhalten:

$$P_{SP} = \begin{bmatrix} OC & OA & OB \end{bmatrix}.$$

Die Sortierstation würde daraufhin zuerst ein Bauteil vom Typ OC aus dem Eingangsstrom entnehmen und im ersten Ablagebereich des Ausgangspuffers platzieren. Anschließend erfolgt die Entnahme eines Bauteils vom Typ OA und dessen Ablage im zweiten Ausgangspufferbereich. Den Abschluss findet die Aufgabe SP schließlich mit der Entnahme eines Bauteils vom Typ OB und dessen Ablage im dritten Ausgangspufferbereich.

Im Unterschied zum Problem der Werkstückbearbeitung im vorangegangenen Abschnitt führt die Sortierstation entsprechend dem Parameter P_{SP} jeweils nur eine bestimmte aus der überlagerten Ebene vorgegebene Aufgabe durch. Konsequenterweise kann die aggregierte Aufgabe SP mit dem gesamten Steuerungsmodell der Sortierstation gleichgesetzt werden und P_{SP} als c-Größe sowie R_{SP} als s-Größe aufgefasst werden. Ein entsprechend aufgebautes Konzeptmodell ist in Abbildung 6.11 dargestellt.

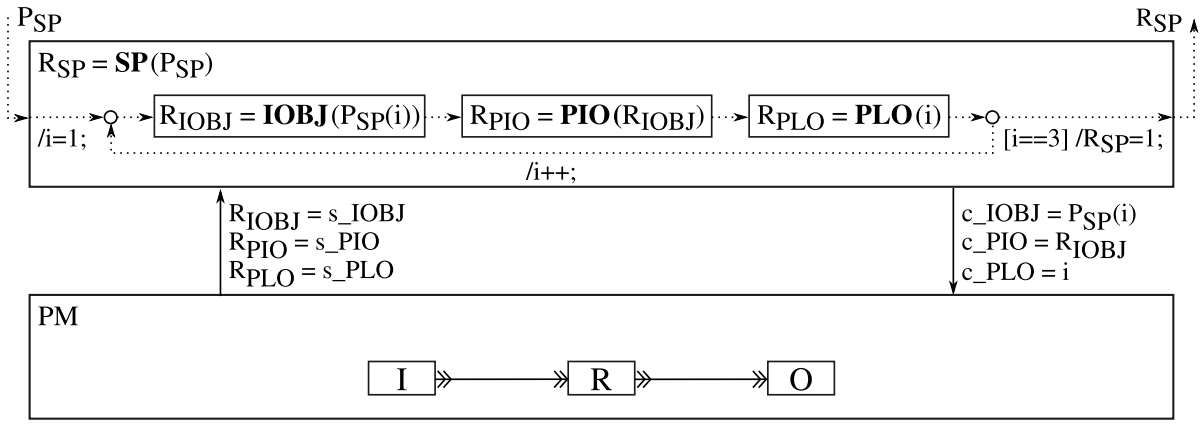


Abbildung 6.11: Konzeptmodell des aufgabenorientierten Steuerungsentwurfs mit Aufgabenparametrierung

Die Umsetzung des Steuerungsmodells $R_{SP} = SP\,(P_{SP})$ in Form von konkreten Zustandsdiagrammen ist im Anhang A.4 dargestellt. Der Entwurf des Prozessmodells sowie dessen Detaillierung in den zwei Schritten der Automatisierungsphase entspricht dem Vorgehen wie bereits im Abschnitt 6.3.1 besprochen und soll deshalb an dieser Stelle für das Problem der Bauteilsortierung nicht weiter ausgeführt werden.

6.4 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie die aufgabenorientierte Roboterprogrammierung in einem auf den SBC-Ansatz aufbauenden Entwicklungsprozess integriert werden kann. Es wurde herausgearbeitet, dass ein aufgabenorientierter Entwurf im ersten Schritt der Automatisierungsphase erfolgt. In dieser Phase wird die Separierung von Steuerung und Prozess eingeführt. Im Unterschied zum prozesselementeorientierten Steuerungsentwurf gemäß Kapitel 5, in dem für jedes Prozesselement eine zugehörige Steuerungskomponente spezifiziert wurde, wird nach dem aufgabenorientierten Steuerungsentwurf das zu lösende Gesamtproblem auf der Steuerungsebene in Teilaufgaben gegliedert. Die Schnittstelle zwischen Steuerungs- und Prozessebene orientiert sich infolgedessen an den Teilaufgaben, den sogenannten Basisaufgaben, die unmittelbar auf Komponenten des Prozessmodells abbilden.

In einem aufgabenorientierten Steuerungsentwurf werden Aufgaben spezifiziert, die in Sequenz, in Wiederholung oder in Nebenläufigkeit auszuführen sind. Da aber aufgrund der fehlenden Detaillierung nicht eindeutig definiert ist, wie die Aufgaben konkret zu lösen sind, ist eine Aufgabentransformation unter Zuhilfenahme eines Weltmodells erforderlich. Diese Funktionalität wird im SBC-Ansatz vom Prozessmodell zur Verfügung gestellt. Am Beispiel im Abschnitt 6.3.1 wurde gezeigt, wie die fehlende Detaillierung durch eine zusätzliche Weltmodellierung im Prozessmodell aufgelöst werden kann. Somit sind alle Voraussetzungen erfüllt, um mit einem aufgabenorientierten Steuerungsmodell im Verbund mit einem Prozessmodell nach dem SBC-Ansatz Systemsimulationen auszuführen. Bei der SiL-Simulation und Inbetriebnahme im zweiten Schritt der Automatisierungsphase besteht zwischen aufgabenorientiert und prozesselementeorientiert entworfenen Steuerungen kein grundsätzlicher Unterschied.

Abschließend wurde im Abschnitt 6.3.2 demonstriert, wie in einem aufgabenorientierten Steuerungsentwurf parametrierbare Basisaufgaben nutzbringend verwendet werden können. Dabei wurde gezeigt, dass die Parametrierung einerseits das Zusammenfassen mehrerer ähnlicher Aufgaben ermöglicht, andererseits können in Aufgabenfolgen die Ergebnisse einer Aufgabe zur Parametrierung von nachfolgenden Aufgaben benutzt werden. Die Aufgabenparametrierung trägt somit dazu bei, die Komplexität einer aufgabenorientierten Steuerung zu reduzieren, indem die Anzahl der Basisaufgaben auf eine überschaubare Menge begrenzt wird. In diesem Sinne stellt die Aufgabenparametrierung auch eine Methode zur Flexibilisierung von Steuerungen in einem begrenzten Rahmen dar. In Kombination mit weiteren Techniken bietet die Aufgabenorientierung noch ein weitaus größeres Potential zur Realisierung hoch flexibler Steuerungen. Ein solcher Ansatz soll im nachfolgenden Kapitel näher betrachtet werden.

7 Flexible aufgabenorientierte Robotersteuerungen

Im Kapitel 6 wurde gezeigt, wie die Entwicklung aufgabenorientierter Robotersteuerungen im Rahmen des SBC-Ansatzes erfolgt. Der aufgabenorientierte Steuerungsentwurf kann gegenüber dem im Kapitel 5 demonstrierten prozesselementeorientierten Entwurf insbesondere für sehr komplexe Problemstellungen vorteilhaft sein. Beiden Entwurfskonzepten ist jedoch gemeinsam, dass die resultierenden Steuerungen fest codiert sind. Flexible oder adaptiv arbeitende Steuerungen sind auf diese Weise nur mit sehr hohem Aufwand realisierbar. Aber genau diese Aufgabe stellt sich zunehmend, wenn Roboter im Rahmen von flexiblen (FMS[1]) und rekonfigurierbaren (RMS[2]) Fertigungssystemen eingesetzt werden.

Zu Beginn dieses Kapitels sollen deshalb die sich aus FMS und RMS ergebenden Anforderungen für die Steuerungsentwicklung näher betrachtet werden. Ferner wird diskutiert wie diese im Rahmen eines SBC-basierten Entwicklungsprozesses realisierbar sind.

Ein wesentliches Konzept zur Erreichung hoher Flexibilität ist das der adaptiven Steuerungen. Einer Betrachtung des Problems flexibler Robotersteuerungen aus dieser Perspektive widmet sich der zweite Abschnitt des Kapitels.

Als ein geeignetes Mittel zur Realisierung adaptiver und damit flexibler Steuerungskonzepte im Rahmen des SBC-Ansatzes erscheint das *System Entity Structure and Model Base Framework* nach Zeigler [Zei84]. Einer kurzen Darstellung der grundlegenden Aspekte sowie erforderlicher Erweiterungen für den Einsatz dieses Frameworks im Rahmen einer operativen Steuerung widmet sich der dritte Abschnitt des Kapitels.

Im vierten Abschnitt wird anhand einer konkreten Montageaufgabe demonstriert, wie im Rahmen des SBC-Ansatzes unter Verwendung einer vereinfachten, aus dem SES/MB-Framework abgeleiteten, deklarativen Steuerungsspezifikation eine flexible aufgabenorientierte Robotersteuerung generiert werden kann.

7.1 Anforderungen an Robotersteuerungen in FMS und RMS

Aus dem Robotereinsatz in flexiblen und rekonfigurierbaren Fertigungssystemen ergeben sich gegenüber Anwendungen in der industriellen Massenfertigung erweiterte Anforderungen. Diese sollen ausgehend von einer Analyse der wesentlichen Eigenschaften von FMS

[1] Flexible Manufacturing Systems
[2] Reconfigurable Manufacturing Systems

und RMS nachfolgend herausgearbeitet werden. Anschließend wird der SBC-Ansatz einer kritischen Betrachtung unterzogen, inwieweit die erhöhten Anforderungen durch dieses Entwurfskonzept erfüllt werden können.

7.1.1 Charakterisierung von FMS und RMS

Mit den FMS wurde der Begriff der Flexibilität bei Fertigungssystemen eingeführt. Nach [Tuf88] wird darunter das Vermögen einer Fertigung verstanden, die zu fertigenden Werkstücke in beliebiger Reihenfolge und nach Möglichkeit unabhängig von einer Losbildung durch den Betrieb laufen lassen zu können. Daneben ist ein FMS durch eine hohe Produktvielfalt mit zumeist kleinen Stückzahlen gekennzeichnet. Auch wenn sich die technische Realisierung von FMS seit der Einführung des ersten flexiblen Fertigungssystems 1967 mit einem zentralen IBM-Steuerungsrechner namens *Molins 24* in Großbritannien grundlegend verändert hat, gelten die obigen Eigenschaften ([Tol09]) nach wie vor.

Der Begriff und erste Forschungsarbeiten zu RMS stammen aus dem College of Engineering der University of Michigan aus dem Jahr 1999 ([Kor99]). Ein RMS ist gegenüber einem FMS durch eine begrenzte Produktvielfalt mit dafür wesentlich größeren Stückzahlen gekennzeichnet. Der Flexibilitätsgedanke liegt bei RMS auf einer schnellen Umrüstbarkeit des Fertigungssystems ([Kor99], [Hei06]).

Die Automation von FMS und RMS erfolgt heute nach [Gro07] immer in Form eines *dezentralen* Systems gemäß der in Abbildung 7.1 dargestellten Automatisierungspyramide.

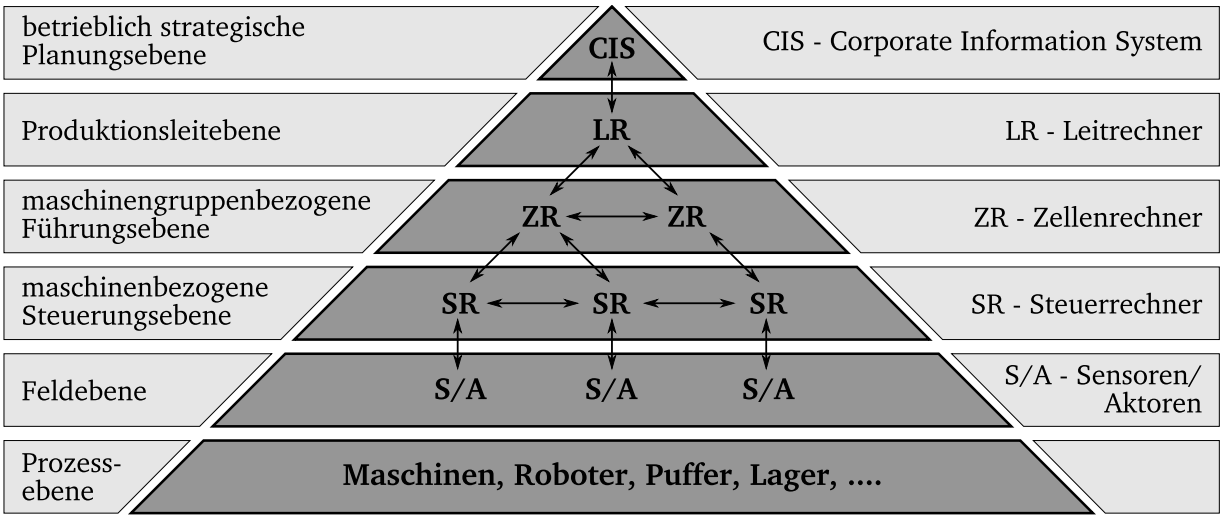


Abbildung 7.1: Automatisierungspyramide von FMS und RMS nach [Gro07]

Hervorzuheben ist, dass weitgehend jedes Element der Prozessebene über einen eigenen Steuerrechner verfügt. Bei umfangreichen Steuerungsaufgaben kann ein Prozesselement auch einen eigenen *Zellenrechner* besitzen, wenn zum Beispiel durch das jeweilige Element selbstständig komplexe Abarbeitungsreihenfolgeprobleme zu lösen sind. Nur durch einen

dezentralen Aufbau können die Anforderungen hinsichtlich der Leistungsfähigkeit und der Skalierbarkeit[3] des Automatisierungssystems erbracht werden.

Betrachtet man das Teilproblem der Verwendung von Robotern in FMS und RMS, so ergeben sich in erster Linie spezifische Anforderungen bezüglich der Sensor- und Aktorintegration sowie der Entwurfszielstellung der Robotersteuerung selbst. Eine Gegenüberstellung der unterschiedlichen Zielstellungen von FMS und RMS und die daraus resultierenden Anforderungen für den Robotereinsatz sind zusammengefasst in Tabelle 7.1 dargestellt.

Tabelle 7.1: Anforderungen an die Robotersteuerungsentwicklung beim Einsatz in FMS und RMS

Systemtyp	Zielstellung	Anforderungen
FMS	• Herstellung teilweise sehr unterschiedlicher Produkte in kleinen Stückzahlen unter variierenden Produktionsbedingungen	**A1:** Einbindung vielfältiger Sensorik und möglichst universeller Aktorik **A2:** Realisierung einer Steuerung mit hoher Anpassungsfähigkeit
RMS	• Kurzfristige Umrüstung auf neue Produktfamilien mit hohen Stückzahlen	**A3:** schnelle steuerungstechnische Einbindung unterschiedlichster Sensor- und Aktortypen **A4:** schnelle Realisierung einer effizienten Steuerung zur Gewährleistung kleiner Taktzeiten

7.1.2 Bewertung des SBC-Ansatzes im Kontext von FMS und RMS

Aus Tabelle 7.1 ist ersichtlich, dass sowohl in FMS als auch in RMS sehr hohe Anforderungen bezüglich der Sensor- und Aktorintegration an die Robotersteuerungen bestehen. Während in FMS der Schwerpunkt vor allem auf einer großen Vielfalt verschiedener Sensoren und Aktoren liegt (Tab. 7.1, A1), besteht bei RMS in erster Linie die Erfordernis, dass bei Umrüstungen die jeweils erforderlichen Sensoren und Aktoren möglichst schnell und problemlos in die Steuerung integriert werden können (Tab. 7.1, A3). Konzeptionell können diese Forderungen durch den SBC-Ansatz grundsätzlich erfüllt werden. Praktisch muss die zum Einsatz kommende RCP-Plattform die softwaretechnischen Voraussetzungen

[3] Unter Skalierbarkeit wird in diesem Zusammenhang die automatisierungstechnische Erweiterbarkeit verstanden ([Lan04]).

zur Einbindung entsprechender Treiber zur Ankopplung der Sensoren und Aktoren besitzen. Wie im Kapitel 4 gezeigt wurde, sind diese Voraussetzungen bei der RCP-Umgebung Matlab erfüllt.

Die in RMS bestehende Forderung der schnellen Realisierbarkeit von effizienten Steuerungen (Tab. 7.1, A4) ist der zentrale Gegenstand des Rapid Control Prototyping und wird damit auch vom SBC-Ansatz erfüllt. Ob ein prozesselementeorientierter Entwurf (vgl. Abschnitt 5.3) oder ein aufgabenorientierter Entwurf (vgl. Abschnitt 6.3) die zeitsparenste Entwurfstechnik ist, hängt vom konkreten Problem ab; realisierbar sind beide Techniken im Rahmen des SBC. Unter Umständen kann die Wiederverwendbarkeit von Steuerungsimplementierungen in RMS vorteilhaft sein. Hierfür bietet insbesondere das Konzept der Basisaufgaben im SBC die notwendigen Voraussetzungen (vgl. Abschnitt 6.2).

Anhand von konkreten Beispielen wurde in den Kapiteln 5 und 6 die Entwicklung komplexer prozessgeführter Robotersteuerungen nach dem SBC-Ansatz demonstriert. Da die Struktur dieser Steuerungen fest codiert ist, ist die erreichbare Flexibilität bezüglich variierender Prozessbedingungen und veränderlicher Vorgaben aus der Leitebene sehr begrenzt. Eine hohe Anpassungsfähigkeit von Steuerungen, wie sie als Anforderung innerhalb eines FMS besteht (Tab. 7.1, A2), könnte mittels einer fest codierten Steuerungsstruktur nur mit sehr hohem Aufwand realisiert werden.

Zusammenfassend ist festzustellen, dass die in Tabelle 7.1 aufgeführten Anforderungen A1, A3 und A4 in FMS beziehungsweise RMS durch prozesselemente- oder aufgabenorientierte Steuerungsentwürfe nach dem SBC-Ansatz, wie sie in den Kapiteln 5 und 6 demonstriert wurden, erfüllt werden können. Die Forderung nach hoher Flexibilität und Anpassungsfähigkeit einer Steuerung (Tab. 7.1, A2) kann dagegen mit den bisher vorgestellten Entwurfstechniken noch nicht hinreichend erfüllt werden, da diese auf fest codierte Steuerungsstrukturen abstellen. Für die Entwicklung von Steuerungen mit variabler Struktur erscheint als Ausgangsbasis das bereits vorgestellte Konzept der Aufgabenorientierung geeignet. Die für ein flexibles Fertigungsproblem erforderlichen Steuerungsfunktionalitäten können mittels einer überschaubaren Menge von Basisaufgaben spezifiziert werden. Diese sind dann im operativen Betrieb in Abhängigkeit von den aktuellen Prozessbedingungen und den Vorgaben aus der Leitebene in einer geeigneten Abfolge zur Ausführung zu bringen.

7.2 Adaptive Steuerungsansätze

Aus den Flexibilitätsanforderungen, wie sie sich im Rahmen von FMS und RMS stellen, ergibt sich die Notwendigkeit einer hohen Anpassungsfähigkeit einer Steuerung an variierende Prozessbedingungen und veränderliche Zielvorgaben der Leitebene. In herkömmlichen prozess- beziehungsweise sensorgeführten Steuerungen, wie sie in den Kapiteln 5 und 6 dieser Arbeit Gegenstand waren, ist die erzielbare Anpassungsfähigkeit sehr begrenzt.

Dieser Sachverhalt kann bereits bei der Betrachtung der formalen Darstellung solcher Steuerungen überzeugend aufgezeigt werden. Abbildung 7.2 zeigt das Prinzip einer ereig-

nisdiskreten, prozessgeführten Steuerung.[4]

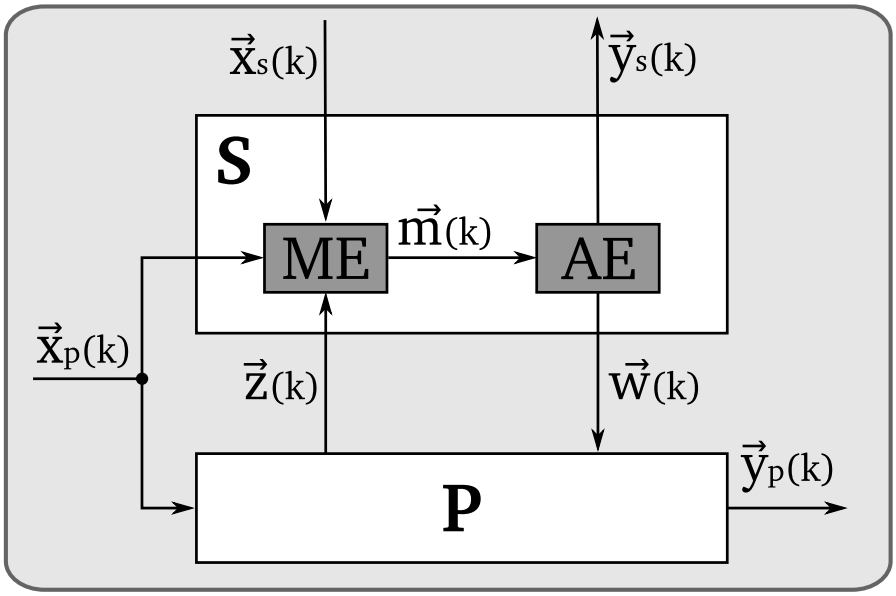


Abbildung 7.2: Steuerungsprinzip der Robotersteuerungen gemäß Kapitel 5 und 6

Auf den Prozess P und die Steuerung S wirkt hier zu einem Zeitpunkt k ein Vektor $\vec{x}_p(k)$ mit Eingangsereignissen ein. Zusätzlich beeinflusst die Steuerung ein Vektor $\vec{x}_s(k)$ mit Ereignissen horizontaler und übergeordneter Steuerungssysteme und ein Zustandsgrößenvektor $\vec{z}(k)$ mit am Prozess gemessenen oder mit einem Prozessmodell berechneten Größen. Aus den einwirkenden Größen bestimmt in S eine Monitoring-Einheit (ME) einen Ereignisvektor $\vec{m}(k)$. Auf dessen Grundlage berechnet eine Ausführungseinheit (AE) einen Steuerungsgrößenvektor $\vec{w}(k)$, der auf P einwirkt und einen Vektor $\vec{y}_s(k)$ mit Ausgangsereignissen an horizontale und übergeordnete Steuerungssysteme. In P werden durch den Einfluss der Steuerungsgrößen $\vec{w}(k)$ die daraus resultierenden Ausgangsereignisse bestimmt und als Vektor $\vec{y}_p(k)$ weitergegeben.

Anpassungsfähigkeit kann hier als Vermögen der Steuerung S aufgefasst werden, für einen bestimmten Definitionsbereich von $\vec{x}_s$ und $\vec{z}$ geeignete Steuerungsgrößen $\vec{w}$ zu erzeugen. In sämtlichen *nicht-adaptiven* Ansätzen ist die Steuerung S *zeitinvariant*, dass heißt sie besitzt eine konstante Struktur und Parametrierung. Hohe Anpassungsfähigkeit hat damit in nicht-adaptiven Ansätzen zwangsläufig komplexe Strukturen von S zur Folge. In der praktischen Anwendung sind der Komplexität von Steuerungen aufgrund des erforderlichen Zeit- und Kostenaufwandes für den Entwurf und die Inbetriebnahme jedoch enge Grenzen gesetzt.

In Analogie zur Regelungstechnik liegt die Idee nahe, durch *zeitvariante* beziehungsweise

[4]Die in Abbildung 7.2 verwendete Notation lehnt sich an [Voi86] an, um eine Vergleichbarkeit mit dem später im Abschnitt dargestellten adaptiven Steuerungsprinzip zu ermöglichen. Eine Zuordnung zu der in den Kapiteln 3, 4 und 5 dieser Arbeit verwendeten Notation sollte dem Leser leicht möglich sein.

adaptive Steuerungskonzepte eine höhere Anpassungsfähigkeit bei vergleichsweise geringerer Strukturkomplexität zu erzielen. Tatsächlich werden in der Literatur eine Vielzahl von Steuerungsansätzen beschrieben, die in diesem Bereich einzuordnen sind. Die jeweils verwendete Begrifflichkeit ist jedoch nicht einheitlich und teilweise widersprüchlich.

Beispielsweise bezeichnet Jacak in [Jac99] eine Klasse von Robotersteuerungen, die sich bei der praktischen Umsetzung zeitvariant verhalten, als *intelligente Steuerungen*:

Definition: „A robotic system and its control are termed intelligent if the system can selfdetermine its decision choices based upon the simulation of needed solutions or upon experience stored in the form of rules in its knowledge base.“

Nach Jacak beruht die Anpassungsfähigkeit beziehungsweise Variabilität der von ihm als intelligent bezeichneten Steuerungen damit entweder auf den Ergebnissen prädiktiver Simulationen während des operativen Betriebes oder auf in einer Wissensbasis hinterlegten Entscheidungsregeln.

Weller fasst in [Wel08] Robotersteuerungen mit Anpassungsfähigkeit hingegen unter dem Begriff *planende Systeme* zusammen. Dabei unterscheidet er zwischen den Unterklassen:

- Problemlösende Systeme,
- Agentensysteme und
- Lernende Systeme.

Die wesentlichen Merkmale problemlösender Systeme sind nach Weller: *(i)* ein Steuerungsprinzip ohne Gedächtnis und *(ii)* die Art und Reihenfolge der auszuführenden Handlungen variieren, obwohl es sich um wiederholt zu lösende gleichartige Problemstellungen handelt. In diese Klasse fallen beispielsweise Roboteranwendungen, bei denen sehr feinstufige Bewegungen ohne exakte Bahnvorgaben auszuführen sind. Die Feinsteuerung erfolgt im operativen Betrieb auf Basis von Sensorinformationen. Die Realisierung solcher Roboteranwendungen, wie zum Beispiel eine Feinmontage unter Verwendung eines Kraft-Momenten-Sensors, kann prinzipiell als herkömmliche Steuerung mit den Mitteln der roboterorientierten Programmierung erfolgen. Die resultierenden Lösungen sind dann aber sehr komplex und erfordern einen entsprechend hohen Entwicklungsaufwand.

Agentensysteme weisen nach Weller wie problemlösende Systeme das Merkmal auf, *(i)* dass das Steuerungsprinzip ohne Gedächtnis arbeitet. Sie sind jedoch in der Lage, *(ii)* ein selbstständiges Planen von Handlungen auszuführen. Als Entscheidungsbasis dienen dazu entweder die Ergebnisse prädiktiver Simulationen oder die in einer Wissensbasis enthaltenen Regeln. Die von Weller als Agentensysteme bezeichneten Ansätze entsprechen damit der oben zitierten Definition von intelligenten Systemen nach Jacak. Ein typisches Anwendungsbeispiel für ein solches System ist die Steuerung eines Montageroboters in Abhängigkeit von der momentanen Bauteilverfügbarkeit. In solchen Steuerungen wird häufig die Technik der aufgabenorientierten Programmierung mit Montagemodellen (Spezifikation möglicher Montagereihenfolgen in Form von Regeln in einer Wissensbasis) kombiniert.

Lernende Systeme unterscheiden sich nach Weller von den vorangegangenen Systemen dadurch, *(i)* dass ihr Steuerungsprinzip auf einem Gedächtnis beruht. Ausgehend von einem initialen Wissensstand kann *(ii)* durch Methoden der künstlichen Intelligenz das Wissen erweitert werden. Auf Basis des erlernten Wissens können diese Systeme ihre Handlungen selbstständig und effizient planen. Anwendungen finden sich heute vor allem im Bereich mobiler, autonomer Roboter. Wie bei Agentensystemen erfolgt die Umsetzung in der Regel auf Basis der aufgabenorientierten Programmierung und einer Wissensbasis. Für die Lernfähigkeit ist aber eine weitere Komponente zur Aneignung von neuem Wissen erforderlich.

Weller charakterisiert in [Wel08] problemlösende Systeme und Agentensysteme als adaptive Steuerungen und lernende Systeme als intelligente Steuerungen. Dagegen verweist Pawletta in [Paw11] darauf, dass alle drei Systemklassen auf einer gemeinsamen Grundstruktur basieren. Formal kann diese Grundstruktur in Anlehnung an [Voi86], wie in Abbildung 7.3 gezeigt, dargestellt werden.

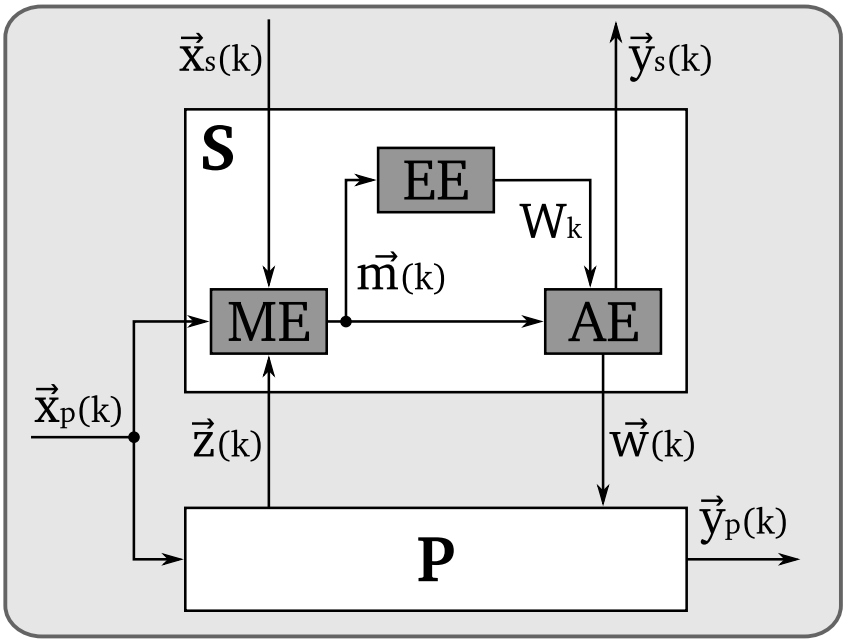


Abbildung 7.3: Grundstruktur einer adaptiven Steuerung nach [Paw11]

Nach Voigt und Cramer ([Voi86]) wird eine Steuerung als adaptiv charakterisiert, wenn sich die Steuerungsstruktur[5] W im Betrachtungszeitraum k ändert. Das heißt die Steuerung kann verschiedene Steuerungsstrukturen W aus einer Menge zulässiger Steuerungsstrukturen $\mathcal{W}$ besitzen. Die aktuelle Steuerungsstruktur $W_k \in \mathcal{W}$ ist festzulegen in Abhängigkeit der aktuellen Eingangsereignisse in den Vektoren $\vec{x}_p(k)$ und $\vec{x}_s(k)$ sowie dem aktuellen Zustandsgrößenvektor $\vec{z}(k)$. Eine Monitoring-Einheit (ME) beobachtet in Echtzeit die auf die

[5] in [Voi86] als Steuerungsstrategie bezeichnet

Steuerung einwirkenden Vektoren $\vec{x}_p(k)$, $\vec{z}(k)$ und $\vec{x}_s(k)$ und generiert einen Ereignisvektor $\vec{m}(k)$. Das Monitoring beinhaltet auch die Schwellwertüberwachung von zeitdiskreten Sensorsignalen. Der Ereignisvektor $\vec{m}(k)$ wird von einer Entscheidungseinheit *EE* ausgewertet, welche eine Steuerungsstruktur W_k festlegt und parametriert beziehungsweise entscheidet, dass die aktuelle Steuerungsstruktur beibehalten wird. Ausgehend von der Steuerungsstruktur W_k und dem Ereignisvektor $\vec{m}(k)$ berechnet eine Ausführungseinheit (*AE*) den auf den Prozess einwirkenden Steuerungsvektor $\vec{w}(k)$.

Folgt man Voigt und Cramer, so ist offensichtlich das Vorhandensein einer Entscheidungseinheit *EE* entscheidend, um einen Steuerungsansatz als adaptiv zu bezeichnen. Nach [Paw11] verfügen alle von Weller als planende Systeme bezeichneten Ansätze über eine Entscheidungskomponente. Ihre Realisierung fällt in den jeweiligen Unterklassen jedoch unterschiedlich aus. So besitzt die Entscheidungseinheit in lernenden Systemen im Unterschied zu problemlösenden Systemen und Agentensystemen beispielsweise eine Gedächtniskomponente. Ausgehend vom in Abbildung 7.3 dargestellten Strukturprinzip wurden im Rahmen dieser Arbeit eine Anzahl von Steuerungsansätzen mit hoher Anpassungsfähigkeit analysiert. Die Ergebnisse dieser Analyse sind in Tabelle 7.2 zusammengefasst.

Im Unterschied zu [Wel08] erfolgt die Untergliederung in Tabelle 7.2 nicht nach Systemklassen, sondern nach Problemklassen. Dabei wird sichtbar, dass die analysierten Steuerungsansätze je nach Problemklasse charakteristische Merkmale bezüglich der Steuerungseinheiten *ME*, *AE* und *EE* aufweisen. Von besonderem Interesse sind dabei die unterschiedlichen Ausprägungen der Entscheidungseinheit *EE*.

Das Ergebnis der durchgeführten Analyse bestätigt die bereits in [Jac99] und [Paw11] getroffenen Aussagen, dass die Anpassungsfähigkeit einer Steuerung entweder *(i)* durch prädiktive Simulationen des Prozessverhaltens bei unterschiedlichen Steuerungsstrukturen oder *(ii)* durch Auswertung einer Wissensbasis oder *(iii)* durch die Kombination beider Methoden realisiert wird. Im Rahmen der vorliegenden Arbeit ergibt sich daraus die Fragestellung, inwieweit diese Methoden im Kontext des vorgestellten SBC-Ansatzes realisierbar sind.

Aus den vorangegangenen Kapiteln ist bereits bekannt, dass der SBC-Ansatz auf der Verwendung von Simulationsmodellen sowohl für die zu realisierende Steuerung als auch für den zu steuernden Prozess beruht. Eine für den SBC-Ansatz geeignete Softwareplattform stellt deshalb im Kern eine leistungsfähige Simulationsumgebung dar (vgl. Kapitel 4). Damit sind grundsätzlich günstige Voraussetzungen für die Integration prädiktiver Simulationen gegeben. In [Paw04] und [Kre06] wurde bereits gezeigt, wie unter Verwendung der Softwareplattform Matlab die Entwicklung einer adaptiven Steuerung auf Basis prädiktiver Simulationen nach dem SBC-Ansatz erfolgen kann.

Die Integration und Auswertung einer Wissensbasis im Rahmen des SBC-Ansatzes zur Realisierung einer adaptiven Steuerung wurde dagegen noch nicht eingehend untersucht. Dieser Thematik widmen sich die nachfolgenden Abschnitte der vorliegenden Arbeit.

Tabelle 7.2: Adaptive Steuerungsansätze

Problemklasse	Adaptiver Steuerungsansatz	Literaturbasis
• Online-Reihenfolgeplanung von Aufträgen zur Kompensation von Prozessstörungen	**ME:** Online-Prozessüberwachung **EE:** Prädiktive Online-Simulation vordefinierter Prozesssteuerungsvarianten und Auswahl der besten Variante anhand quantitativer Kenngrößen **AE:** Automatische Umsetzung der besten Steuerungsvariante	[Kim94], [Dra95], [Cho03]
• Online-Steuerungsanpassung zur Kompensation von Prozessstörungen oder als Reaktion bei Änderung der Auftragspriorisierung	**ME:** Online-Überwachung von Prozessgrößen und Produktionszielen **EE:** Prozessbegleitende Überprüfung der vorgegebenen Produktionszielwerte durch prädiktive Online-Simulation. Bei unzureichenden Produktionszielwerten erfolgt die Berechnung neuer Steuerungsparameter mittels Parameteroptimierung oder eine vergleichende Bewertung unterschiedlicher vordefinierter Steuerungsvarianten auf Basis prädiktiver Simulation. **AE:** Automatische Umsetzung der besten Steuerungsvariante	[Paw04], [Kre06]
• Planung von Fertigungs- oder Montageaufgaben	**ME:** Vorgabe einer fertigungs- oder montagetechnischen Aufgabe **EE:** Wissensbasis mit Detailinformationen zu Bauteilbeschreibungen, Montage- oder Fertigungsprinzipien, um ausführbare Reihenfolgen von Montage- oder Fertigungsoperationen in Form von unterschiedlichen Steuerungsvarianten zu ermitteln. Vergleichende Bewertung der Steuerungsvarianten durch prädiktive Offline-Simulationen. **AE:** Automatische Inbetriebnahme der besten Steuerungsvariante	[Jac93], [Jac98], [Cha85], [Sie96], [Chi91], [Ter08]

7.3 Deklarative Steuerungsspezifikation und automatisierte Steuerungssynthese

Unter einer deklarativen Steuerungsbeschreibung soll im Folgenden eine von der operativen Steuerungsausführung unabhängige Beschreibung von Anwendungswissen in einer

- strukturierten,
- formalisierten und
- durch einen Algorithmus interpretierbaren

Form verstanden werden. Der Autor ist sich bewusst, dass der Term *deklarativ* im Sinne der Informatik ([Rec97]) eine umfassende Bedeutung besitzt. Nach [Dä05] ist die Problemstellung als eine Informationsmodellierung aufzufassen.

Die Analyse von adaptiven Steuerungskonzepten im vorangegangenem Abschnitt zeigt, dass insbesondere für das Anwendungsgebiet der automatisierten Montage deklarative Beschreibungsformen entwickelt wurden, aus welchen automatisiert aufgabenorientierte Robotersteuerungen generiert werden ([Hei06], [Jac99]). Die dort verwendeten Beschreibungsmethoden sind domainenspezifisch. Darüber hinaus sind die beschriebenen Steuerungen nur bedingt anpassungsfähig. Es werden aus einer deklarativen Steuerungsspezifikation unterschiedliche Steuerungsstrukturen und -parametrierungen extrahiert und bewertet. Das generierte Steuerungsprogramm implementiert aber feste Aufgabenfolgen auf Basis der zum Zeitpunkt der Steuerungsgenerierung zugrunde gelegten Prozessbedingungen und Steuerungsvorgaben. Während der Steuerungsausführung kann die Steuerungsstruktur nicht adaptiert werden. Damit unterscheidet sich die Steuerung während der Ausführung prinzipiell nicht von dem im Kapitel 6 betrachteten Ansatz.

Anliegen dieses Abschnittes sind eine domainenunabhängige deklarative Spezifikation von aufgabenorientierten Robotersteuerungen und eine automatisierte Steuerungsgenerierung in Abhängigkeit von Prozesszustandsänderungen und variierenden Steuerungsvorgaben. Das heißt, die operative Steuerung basiert nicht auf einer festen Steuerungsstruktur und -parametrierung. Diese Form von Steuerung wird in diesem Kontext als *flexible* Steuerung bezeichnet.

Im Rahmen des Software-Engineerings wurden bereits Methoden und Werkzeuge zur domainenunabhängigen Informationsmodellierung entwickelt. Das bekannteste Modellierungswerkzeug ist vermutlich die *Unified Modeling Language (UML)*. Ein weniger bekannter Ansatz ist das *System Entity Structure (SES) and Model Base (MB) Framework*, welches ursprünglich aufbauend auf Methoden des System-Engineerings als ein Konzept für den simulationsbasierten Systementwurf entwickelt wurde ([Zei84], [Roz85]). Das SES/MB-Framework wurde kontinuierlich weiterentwickelt ([Zei00], [Zei07]) und stellt nach [Zei00] ein allgemeines und praktisch anwendbares Werkzeug zur System- und Informationsmodellierung dar. Eine aktuelle Recherche in [Paw11] belegt die breite domainenunabhängige Verwendung des SES/MB-Framework insbesondere im Rahmen des simulationsbasierten Systementwurfs.

Simulationsmethoden stellen die zentrale Basis des in dieser Arbeit eingeführten SBC-Ansatzes dar. Demzufolge ist es naheliegend, eine deklarative Steuerungsbeschreibung und die automatisierte Steuerungssynthese auf Basis des SES/MB-Framework im Kontext des SBC-Ansatzes zu untersuchen. Ausgehend von einer Einführung in die grundlegende Syntax und Semantik der SES sowie in die Funktionsweise des Framework wird nachfolgend eine praktische Anwendung im operativen Betrieb diskutiert.

7.3.1 Das System Entity Structure und Model Base Framework

Das SES/MB-Framework wurde in [Zei84] und [Roz85] als allgemeines Konzept zum simulationsbasierten Systementwurf eingeführt. Das grundlegende Funktionsprinzip ist in Abbildung 7.4 dargestellt.

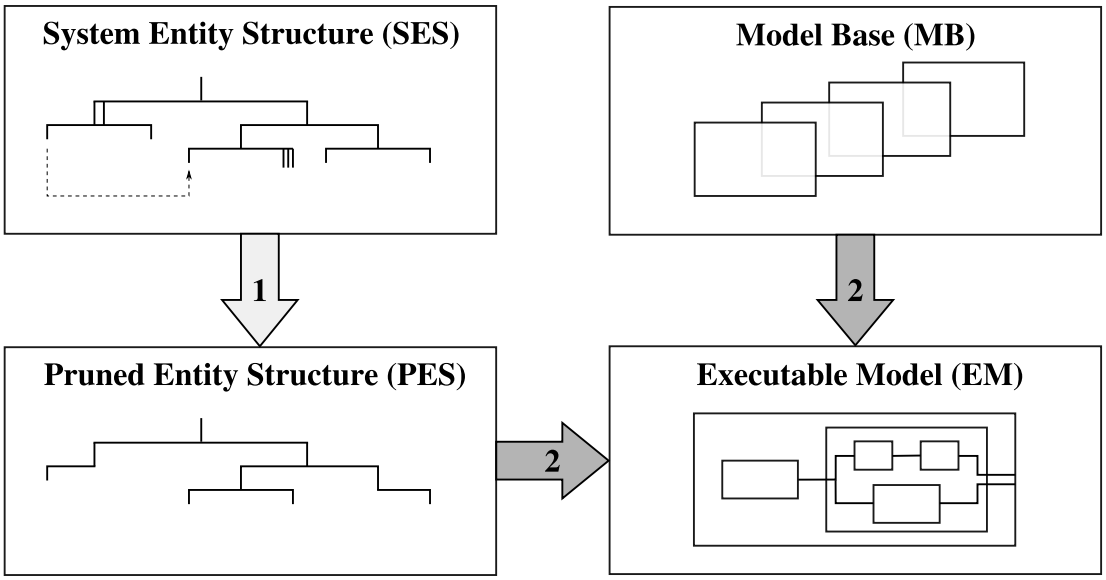


Abbildung 7.4: Prinzip des SES/MB-Framework gemäß [Zei00]

Die Modellbasis (MB) ist eine Bibliothek, die ausführbare Modellkomponenten enthält. Sie beinhaltet sowohl nicht zerlegbare Basiskomponenten (*atomic models*) als auch zusammengesetzte Komponenten (*coupled models*). Die SES ist ein allgemeines struktur- und wissensdarstellendes Schema zur Beschreibung von hierarchischen Systemstrukturen und deren Parametrierung. Alle prinzipiell möglichen Systemstrukturen werden durch die SES abgebildet. Darüber hinaus werden Methoden zur Generierung von ausführbaren Modellen aus einer SES unter Verwendung einer zugehörigen Modellbasis definiert.

Aus einer SES wird durch einen Auswahlprozess, welcher als *Pruning* bezeichnet wird, eine *Pruned Entity Structure (PES)* abgeleitet. Durch den Auswahlprozess werden alle Alternativen in der SES aufgelöst und damit eine eindeutige Systemstruktur synthetisiert und parametriert. Auf Basis der PES wird unter Verwendung der Modellkomponenten der MB ein ausführbares Modell (*Executable Model, EM*) generiert. Die wesentlichen Aspekte

des SES/MB-Framework werden nachfolgend anhand des in Abbildung 7.5 dargestellten Beispiels vorgestellt.

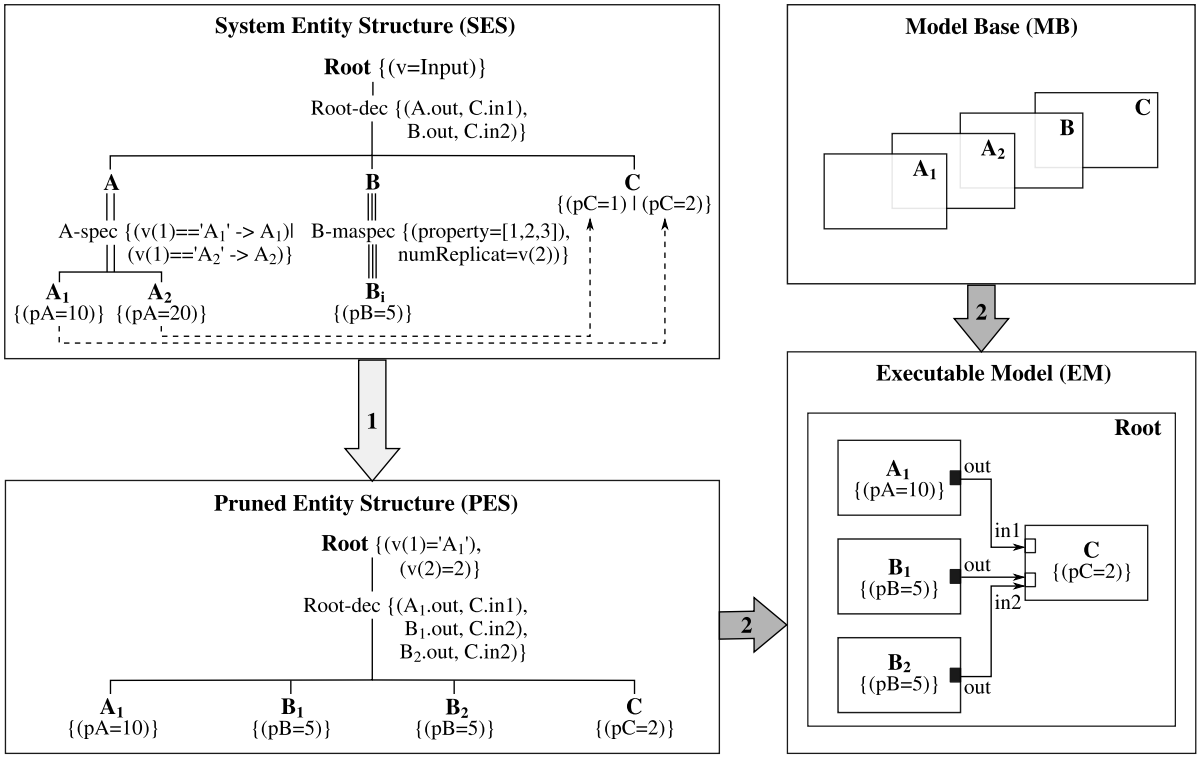


Abbildung 7.5: Anwendungsbeispiel des SES/MB-Framework

Mit Hilfe der SES werden alle möglichen Kompositionsvarianten eines Systems in Form eines Baumes spezifiziert.[6] Im Sinne der Objektorientierung repräsentieren die Knoten einer SES Datentypen. Dabei kann zwischen Knoten-Typen unterschieden werden, die *(i)* eine atomare oder zusammengesetzte *Entität* beziehungsweise *(ii)* einen *Aspekt* oder eine *Spezialisierung* spezifizieren.

Die erste Ebene der in Abbildung 7.5 dargestellten SES ist der Wurzel-Entitäten-Knoten *Root*. In der zweiten Ebene folgt der Aspekt-Knoten *Root-dec*, der den Root-Knoten der ersten Ebene näher charakterisiert. Die dritte Ebene wird erneut durch Entitäten-Knoten gebildet, denen in der vierten Ebene zugehörige Aspekt- und Spezialisierungs-Knoten folgen können. Dieses Prinzip ist in beliebiger hierarchischer Tiefe fortsetzbar und wird als *alternierender Mode* der SES bezeichnet. Die Enden eines SES-Baumes (Blattknoten) müssen jeweils durch Entitäten-Knoten gebildet werden. Nachfolgend sollen die Knoten der in Abbildung 7.5 dargestellten SES kurz erläutert werden.

[6] Die Bezeichnungsweise der Knotentypen der SES ist in der originären Literatur ([Zei84], [Roz85], [Zei00], [Zei07]) nicht einheitlich. Die in dieser Arbeit verwendeten Bezeichnungen orientieren sich primär an [Zei84] und [Roz85].

1. *Wurzel-Entitäten-Knoten: Root*
   Dieser Knoten repräsentiert das Gesamtsystem, welches durch den SES-Baum spezifiziert wird. Am Wurzelknoten können in geschweiften Klammern Konstanten- und Variablenzuweisungen notiert werden, die beim Pruning einer SES ausgewertet werden. Im dargestellten Beispiel ist v eine vektorielle Variable.

2. *Aspekt-Knoten: Root-dec*
   Im Beispiel wird der Entitäten-Knoten *Root* der ersten Ebene durch *Root-dec* charakterisiert. Die Einfachkante zwischen den Knoten *Root* und *Root-dec* sowie der Suffix *-dec* zeigen an, dass es sich um einen *Dekompositions-Aspekt* handelt. Das heißt, *Root* wird als Verknüpfung (Komposition) der in der dritten Ebene folgenden Entitäten-Knoten spezifiziert. Die Kopplungsbeziehungen der Komposition sind in den geschweiften Klammern am Knoten *Root-dec* notiert.

3. *Entitäten-Knoten: A, B, C*
   Die Knoten A und B repräsentieren im dargestellten Beispiel Entitäten, die mittels nachfolgender Aspekt-Knoten noch näher charakterisiert werden. Beim Knoten C handelt es sich dagegen bereits um einen Blattknoten der SES. In den geschweiften Klammern ist die Parametrierung einer zu instantiierenden Modellkomponente C in Abhängigkeit von Zwangsbedingungen (gestrichelten Kanten) spezifiziert. Diese werden unter 5a. noch näher erläutert.

4a. *Spezialisierungs-Knoten: A-spec*
   Mit dem Knoten *A-spec* wird für den Entitäten-Knoten A eine *Spezialisierung* durch nachfolgende alternative Entitäten-Typen spezifiziert. Die Spezialisierung wird in der Syntax der SES durch eine Doppelkante sowie dem Suffix *-spec* im Knotennamen ausgedrückt. In den geschweiften Klammern wird notiert, wie die Auswahl alternativer Spezialisierungen beim Pruning erfolgen soll. Im dargestellten Beispiel wird über die Belegung des ersten Elementes der vektoriellen Variablen v die Auswahl der möglichen Entitäten-Typen A_1 beziehungsweise A_2 bestimmt.

4b. *Multiaspekt-Knoten: B-maspec*
   Der Knoten *B-maspec* spezifiziert für den Entitäten-Knoten B den Aspekt der *Mehrfachheit*. Das heißt, im spezifizierten System kann eine Modellkomponente vom Typ B in Form mehrerer Instanzen enthalten sein. Der Aspekt der Mehrfachheit wird mittels Dreifachkante und Suffix *-maspec* spezifiziert. In den geschweiften Klammern ist am Knoten zu notieren, welche Mehrfachheiten möglich sind und wie eine bestimmte Mehrfachheit beim Pruning auszuwählen ist. Im dargestellten Beispiel wird mit *property = [1, 2, 3]* festgelegt, dass die Modellkomponente vom Typ B im System ein-, zwei- oder dreifach vorhanden ist. Durch *numReplicat = v(2)* erfolgt die Auswahl der konkreten Anzahl von Instantiierungen anhand der Belegung der Variablen *v(2)*.

5a. *Entitäten-Knoten: A_1 und A_2*
   A_1 und A_2 sind Blattknoten der SES und spezifizieren die alternativen Entitäten-

Typen der Spezialisierung *A-spec*. In den geschweiften Klammern sind jeweils Parametrierungen für eine zu instantiierende Modellkomponente vom Typ A_1 beziehungsweise A_2 angegeben. Darüber hinaus sind im Beispiel mit den gestrichelten Kanten von A_1 zu C und von A_2 zu C Zwangsbedingungen spezifiziert. Wird beim Pruning für A die Spezialisierung A_1 ausgewählt, so führt dies am Entitäten-Knoten C zur Auswahl der Parametrierung $pC = 2$ und bei Auswahl von A_2 am Knoten C entsprechen zu $pC = 1$.

5b. *Entitäten-Knoten:* B_i
B_i ist ebenfalls ein Blattknoten der SES und spezifiziert den Entitäten-Typ der Mehrfachheit *B-maspec* sowie die bei der Instantiierung der Modellkomponenten anzuwendenden Parametrierungen.

Der Aufbau einer SES aus den oben beschriebenen Knoten-Typen unterliegt bestimmten Regeln. Für eine gültige SES wird gefordert, dass sie den folgenden Axiomen genügt:

A1. *Alternierender Mode*
Wie bereits ausgeführt müssen in einer SES jeweils Ebenen von Entitäten-Knoten und Ebenen von Aspekt- und Spezialisierungs-Knoten abwechselnd aufeinander folgen.

A2. *Uniformität*
Zwei Knoten mit gleichem Namen müssen identische zugehörige Variablen und isomorphe Sub-Bäume besitzen.

A3. *Strikte Hierarchie*
Ein Knotenbezeichner darf in einem Sub-Baum nicht nochmals auftreten.

A4. *Gültige Brüder*
Zwei horizontal benachbarte Knoten im Baum dürfen nicht den identischen Namen besitzen.

A5. *Zugehörige Variablen*
Die einem Knoten zugehörigen Variablen müssen sämtlich voneinander verschiedene Bezeichner besitzen.

A6. *Vererbung*
Vater und Kinder einer Spezialisierung vereinen beim Pruning ihre Variablen, Aspekte und Spezialisierungen.

Die Beispiel-SES in Abbildung 7.5 beschreibt ein System in insgesamt sechs Ausprägungsvarianten, basierend auf vier atomaren Entitäten-Typen. Die Definition der Entitäten-Typen und deren Ablage als Modellkomponenten-Spezifikationen erfolgt im SES/MB-Framework in der Modellbasis MB. Im dargestellten Beispiel müssen damit in der MB die Spezifikationen für die Modellkomponenten A_1, A_2, B und C bereitgestellt werden. Unter Verwendung der SES und der MB kann dann in zwei Schritten ein konkretes ausführbares Systemmodell synthetisiert werden. Im ersten Schritt – dem Pruning – wird der SES-Baum

mittels einer kombinierten Tiefen- und Breitensuche analysiert. Bei Angabe hinreichender Informationen können beim Pruning alle in einer SES enthaltenen Alternativen eindeutig aufgelöst werden. In Abbildung 7.5 wird dies für eine beispielhafte Variablenbelegung für das Pruning demonstriert. Mit *v(1)* = 'A_1' wird für A die Spezialisierung A_1 mit der zugehörigen Parametrierung ausgewählt. Infolge wird durch Auswertung der Zwangsbedingung die Parametrierung $pC = 2$ für C festgelegt. Mit *v(2)* = *2* wird B durch die Mehrfachheit B_1 und B_2 ersetzt. Damit ist nun für *Root-dec* eine Zusammensetzung bestehend aus den Entitäten-Typen A_1, B_1, B_2 und C inklusive anzuwendender Parametrierung eindeutig festgelegt. Infolgedessen können nunmehr auch die an *Root-dec* spezifizierten Kopplungsbeziehungen konkretisiert werden. Im Ergebnis des Prunings liegt die in Abbildung 7.5 dargestellte PES vor. Im zweiten Schritt werden nach den Vorgaben der PES und unter Verwendung der in der MB enthaltenen Entitäten-Definitionen die Modellkomponenten instantiiert und parametriert sowie die Kopplungsbeziehungen zwischen den Modellkomponenten aufgebaut. Als Resultat liegt ein ausführbares Systemmodell (EM) wie in Abbildung 7.5 dargestellt vor.

7.3.2 Modifikationen des SES/MB-Framework zur automatisierten Generierung von Steuerungsprogrammen

Im vorangegangenen Abschnitt wurde die System Entity Structure nach [Zei84] als allgemeines struktur- und wissensdarstellendes Schema vorgestellt, auf dessen Basis im Rahmen des SES/MB-Framework nach [Zei84], [Roz85] ausführbare Systemmodelle synthetisiert werden können. In diesem Abschnitt soll nun eine neue Methode vorgestellt werden, mit der durch eine Anzahl von Modifikationen des SES/MB-Framework die automatisierte Generierung von Steuerungen ermöglicht wird. Der vorgestellte Ansatz erscheint insbesondere im Kontext flexibler aufgabenorientierter Steuerungen sinnvoll. Die wesentlichen Modifikationen des SES/MB-Framework für den Einsatz zur automatisierten Generierung von Steuerungen sind in Abbildung 7.6 dargestellt.

Die erste Modifikation des SES/MB-Framework besteht in der Einführung eines ?-Operators, mit dem in einer SES der gezielte Abbruch eines Prune-Vorgangs erzwungen werden kann. Das Ergebnis eines solchen abgebrochenen Prunings ist eine *temporäre PES (TPES)*. Auf Basis der TPES und der in der MB bereitstehenden Spezifikationen von Entitäten-Typen, die nunmehr Komponenten eines Steuerungsmodells darstellen, wird ein *temporäres ausführbares Steuerungsmodell (TECM)* generiert. Dieses TECM kann in Verbindung mit einem passenden Prozessmodell (PM), entsprechend Kapitel 5 oder Kapitel 6 dieser Arbeit, für eine Software-in-the-Loop Simulation oder als Steuerung für den operativen Betrieb für einen begrenzten Zeithorizont benutzt werden. Nach Ablauf der Gültigkeit des TECM wird in einer zweiten wesentlichen Modifikation gegenüber dem originären SES/MB-Framework unter Auswertung der aktuellen Prozesszustände und eventuell veränderter Vorgaben aus der Leitebene die Ausgangs-SES durch eine automatisch aktualisierte SES ersetzt, auf deren Basis dann erneut TPES und TECM generiert werden. Dieser Prozess wird solange wiederholt, bis die aktualisierte SES in der dritten Ebe-

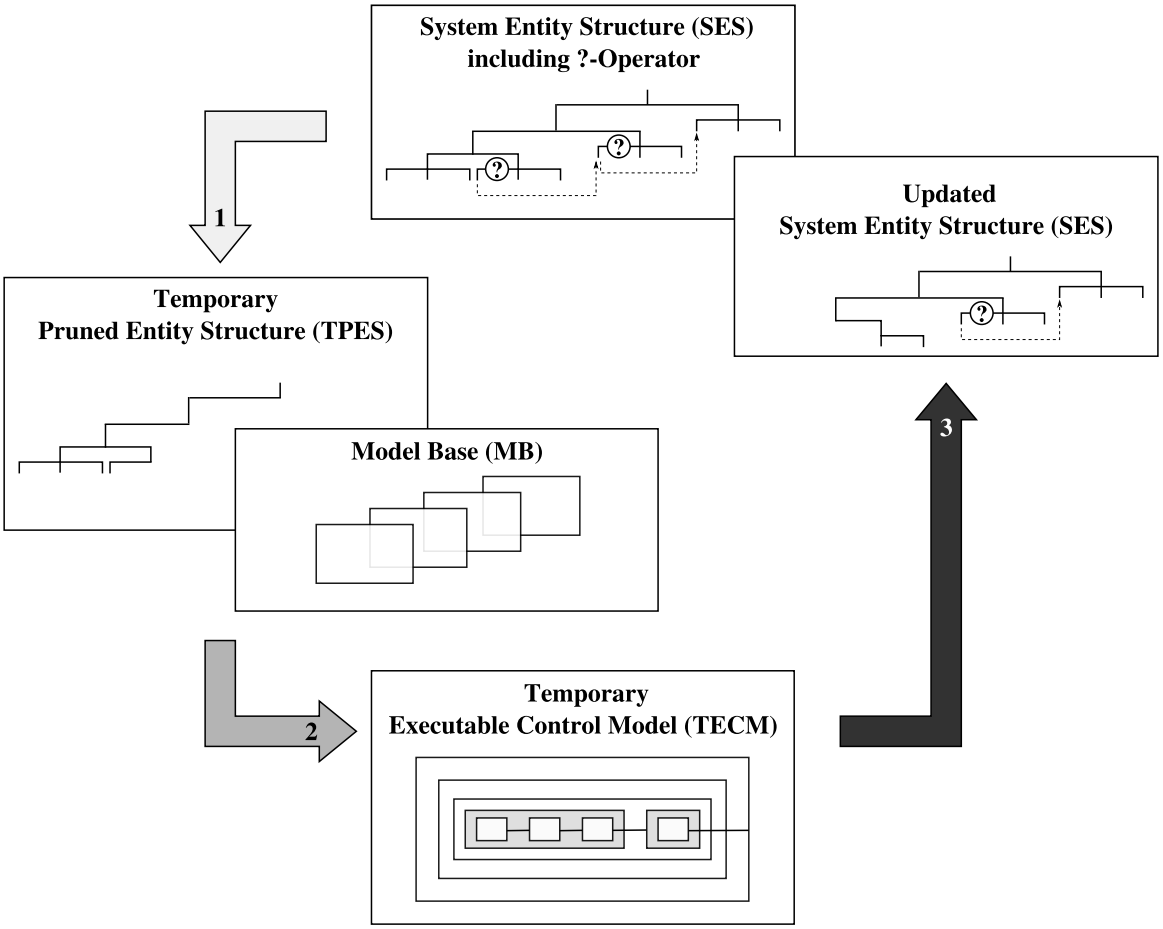


Abbildung 7.6: Modifikationen des SES/MB-Framework zur automatisierten Generierung von Steuerungen

ne keine Entitäten-Knoten mehr enthält, was wiederum die vollständige Abarbeitung der Steuerungsaufgabe anzeigt.

Zur Veranschaulichung der eingeführten Modifikationen des SES/MB-Framework soll deren Verwendung zur Generierung einer aufgabenorientierten Steuerung für ein minimales Montageproblem, wie in Abbildung 7.7 dargestellt, nachfolgend erläutert werden.

Zu Beginn einer Montagevorgangsplanung werden in der Regel mittels graphenbasierter Beschreibungen alle möglichen Montagereihenfolgen spezifiziert ([Lev88], [Hom91], [Sie96]). In der Praxis werden dazu auch häufig sogenannte *Und/Oder-Graphen* ([Kai07]) eingesetzt. Ein Und/Oder-Graph für das minimale Montageproblem aus Abbildung 7.7 zeigt die Abbildung 7.8.

Und/Oder-Graphen setzen sich aus Knoten und gerichteten Kanten zusammen. Der Wurzelknoten *OA+OB+OC* in Abbildung 7.8 steht für das zu montierende Produkt. Die Knoten *OA+OB* und *OA+OC* stellen Baugruppen *(Construction Groups)* dar und die Blattknoten *OA*, *OB* und *OC* repräsentieren die Einzelteile *(Parts)* des Montageproblems.

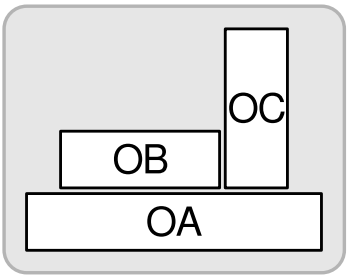


Abbildung 7.7: Minimales Montageproblem bestehend aus drei Einzelteilen

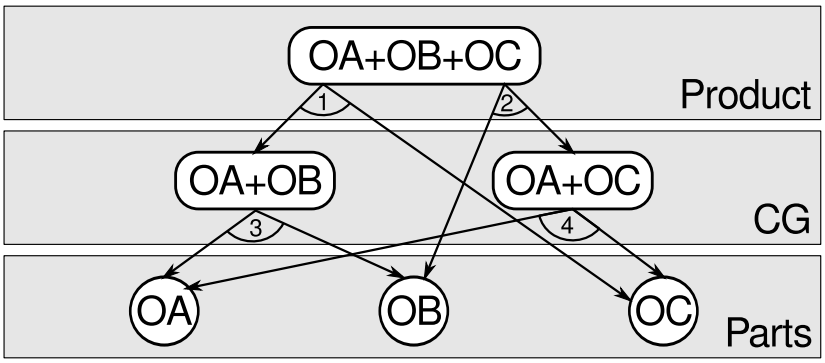


Abbildung 7.8: Und/Oder-Graph für das minimale Montageproblem

Die nach unten gerichteten Kanten, die in Kombination eine Montagemöglichkeit des ausgehenden Knotens spezifizieren, werden als Hyperkanten zusammengefasst und sind in Abbildung 7.8 durch einen nummerierten Kreisbogen markiert. Diese zusammengefassten Hyperkanten werden als *Und-Kanten* bezeichnet. Führen mehrere Enden von Und-Kanten zu einem Knoten, werden diese als *Oder-Alternativen* benannt.

Für das zu montierende Produkt ($OA{+}OB{+}OC$) existieren genau zwei Montagevarianten. Diese werden durch zwei Und-Kanten dargestellt. Die Einzelteile können in zwei alternativen Reihenfolgen zum fertigen Produkt montiert werden:

1. $(\mathrm{OA} + \mathrm{OB}) + \mathrm{OC}$ oder
2. $(\mathrm{OA} + \mathrm{OC}) + \mathrm{OB}$.

Nachfolgend sei angenommen, dass für eine automatisierte Montage eine Roboterzelle zur Verfügung steht. Diese soll aus einem Roboter mit bildgebender Sensorik und einem Teileeingangspuffer bestehen, in dem parallel zum laufenden Montageprozess Einzelteile sequentiell bereitgestellt werden. Darüber hinaus soll sich im Bereich der Roboterzelle ein Montageplatz befinden, der automatisch für die Überführung fertig montierter Produkte in einen Ausgangspuffer sorgt. Eine zu realisierende flexible aufgabenorientierte Steuerung soll in Abhängigkeit von der Einzelteilverfügbarkeit im Eingangspuffer eine Montage

gemäß der im Und/Oder-Graphen spezifizierten Oder-Alternativen ermöglichen. Gemäß der im Kapitel 6 behandelten Aufgabenorientierung ist das Montageproblem mittels drei parametrierbarer Basisaufgaben lösbar:

Identify Object: Die ***IOBJ***-Aufgabe besteht in der Identifikation des Einzelteiltyps sowie der Positionsbestimmung des Einzelteils im Eingangspuffer.

Pick Object: Die ***PIO***-Aufgabe besteht in der Aufnahme des Einzelteils durch den Roboter.

Place Object: Die ***PLO***-Aufgabe besteht im Transport eines aufgenommenen Einzelteils zum Montageplatz sowie dessen dortige montagegerechte Positionierung.

In der Sequenz IOBJ, PIO und PLO können die Basisaufgaben die zusammengesetzte Aufgabe ***AOBJ*** (*Assemble Object*) realisieren. Eine Sequenz von AOBJ-Aufgaben kann wiederum eine zusammengesetzte Aufgabe ***ACG*** (*Assemble Construction Group*) umsetzen.

Die Spezifikation einer Steuerung auf Basis der genannten Aufgaben zur Lösung des Montageproblems mittels einer SES ist in Abbildung 7.9 dargestellt.

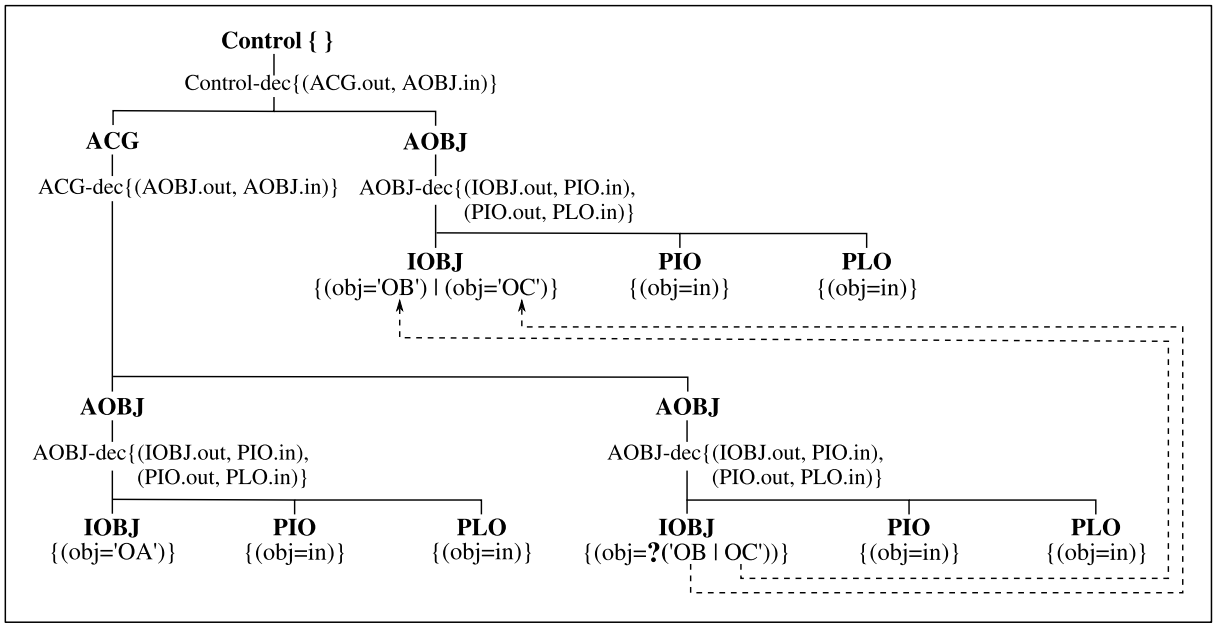


Abbildung 7.9: Spezifikation einer aufgabenorientierten Steuerung zur Lösung des Montageproblems in Form einer SES (unter Verletzung des Axioms A4)

Allerdings verletzt die dargestellte SES auf der dritten Knoten-Ebene mit den benachbarten namensgleichen Entitäten-Knoten AOBJ das Axiom A4 der *Gültigen Brüder*. Auf dieses Problem wird im weiteren Verlauf des Abschnitts noch zurückgekommen. Der Wurzelknoten ***Control*** repräsentiert die Gesamtsteuerungsaufgabe, welche mit dem nachfolgenden

Dekompositions-Aspekt in die Teilaufgaben ACG und AOBJ zerlegt wird. Da es sich bei den Aufgaben ACG und AOBJ ebenfalls um zusammengesetzte Aufgaben handelt, werden auch diese Knoten durch nachfolgende Dekompositions-Aspekte bis auf die Ebene der drei Basisaufgaben IOBJ, PIO und PLO zerlegt. Die leere geschweifte Klammer am Wurzelknoten Control zeigt an, dass für das Pruning keine weiteren Informationen erforderlich sind. Unter der Voraussetzung, dass vom Pruning-Algorithmus die jeweils linken Kanten eines Dekompositions-Aspekts zuerst analysiert werden, wird als erste ausführbare Basisaufgabe IOBJ mit der Parametrierung *obj = 'OA'* ausgewählt. Dies entspricht der Montagevorgangsplanung aus Abbildung 7.8, die OA zwingend als erstes Einzelteil des zu montierenden Produktes vorschreibt. Im Ergebnis der Aufgabe *IOBJ('OA')* ist die Verfügbarkeit eines Einzelteils OA sowie dessen Position im Eingangspuffer bekannt und kann über die Kopplungsbeziehung an PIO weitergegeben und zur Parametrierung dieser Aufgabe verwendet werden. In analoger Weise kommt nach der Beendigung von PIO die Aufgabe PLO zur Ausführung. Danach ist die zusammengesetzte Aufgabe AOBJ mit der Montage des ersten Einzelteils OA abgeschlossen. Gemäß Abbildung 7.8 kann nun das Einzelteil OB *oder* OC montiert werden, was sich in der Parametrierung *obj = ? ('OB|OC')* der Basisaufgabe IOBJ der zweiten zusammengesetzten Aufgabe AOBJ widerspiegelt. Der Fragezeichen-Operator weist darauf hin, dass ein weiteres Pruning der SES erst *nach* Ausführung der Aufgabe *IOBJ('OB|OC')* möglich ist. Ursache sind die von diesem Knoten ausgehenden Zwangsbedingungen. Diese können erst anhand des Ergebnisses von *IOBJ('OB|OC')* aufgelöst werden.

Anhand des hier dargestellten minimalen Montageproblems ist erkennbar, dass ausgehend von einem aus der Montageplanung vorliegenden Und/Oder-Graphen allein unter Nutzung des Aspekts der Dekomposition der systematische Aufbau einer SES zur Spezifikation einer aufgabenorientierten Steuerung möglich ist. Gleichzeitig zeigt das Beispiel aber auch eine Reihe von Problemen auf.

1. Wie bereits ausgeführt verletzt die in Abbildung 7.9 dargestellte SES das Axiom A4 der *Gültigen Brüder*. Damit sind bestimmte formale Prüf- und Verifikationsalgorithmen des SES/MB-Frameworks auf diese SES nicht anwendbar. Die Verletzung des Axioms A4 wäre durch eine Änderung der namensgleichen Knotenbezeichner AOBJ in der dritten Ebene des Baums beispielsweise in $AOBJ_1$ und $AOBJ_2$ zwar leicht behebbar, allerdings wäre diese Bezeichnung im Sinne der Aufgabenorientierung dann nicht mehr plausibel.

2. Bereits am hier betrachteten minimalen Beispiel ist erkennbar, dass es bei der Erhöhung der Komplexität des Montageproblems bei der hier angewendeten Verfahrensweise ebenfalls wegen gleicher Knotenbezeichner zu einer Verletzung des Axioms A3 *Strikte Hierarchie* kommt.

Aus den genannten Problemen ergeben sich für den mit dieser Arbeit angestrebten Nachweis der Generierbarkeit flexibler aufgabenorientierter Robotersteuerungen auf Basis des SES/MB-Frameworks jedoch keine maßgeblichen Einschränkungen. Lösungen für die genannten Probleme sind nach derzeitigem Untersuchungsstand wahrscheinlich insbesondere

durch Nutzung der SES-Spezialisierung möglich. Diese Untersuchungen sind Gegenstand einer weiterführenden Arbeit durch Schwatinski ([Sch12]).

In der hier vorliegenden Arbeit soll der aufgezeigte Weg unter alleiniger Verwendung des SES-Aspekts der Dekomposition bis zu einem Anwendungsbeispiel realer Komplexität weiterverfolgt werden. Zur Vermeidung formaler Widersprüche mit der einschlägigen SES-Literatur wegen der Verletzung der Axiome A4 und A3 werden nachfolgend anstelle der Bezeichner SES und TPES die Bezeichnungen *CS (Control Specification)* und *TPCS (Temporary Pruned Control Specification)* verwendet. Darüber hinaus werden die folgenden gegenüber dem allgemeinen SES/MB-Framework einschränkenden Annahmen zugrunde gelegt:

Die erste Annahme besteht darin, dass der Steuerungsentwurf ausschließlich aufgabenorientiert erfolgt. Damit repräsentieren alle Entitäten-Knoten, einschließlich dem Wurzelknoten, atomare Basisaufgaben beziehungsweise zusammengesetzte Aufgaben der Art:

$$\text{Resultat} = \textbf{Aufgabe}\,(\text{Parameter}).$$

In diesem Sinne repräsentiert der in Abbildung 7.9 mit Control bezeichnete Wurzel-Knoten im hier betrachteten Beispiel die Gesamtsteuerungsaufgabe:

$$R_{AP} = \textbf{AP}\,(\,).$$

AP steht für *Assemble Product* und R_{AP} für das Ergebnis dieser Aufgabe, also für das montierte Produkt.

Die zweite Annahme besteht darin, dass für die Spezifikation zusammengesetzter Aufgaben *nur* der Aspekt der Dekomposition verwendet wird und die in einer Dekomposition enthaltenen Aufgaben von links nach rechts ausschließlich in einfacher Sequenz ausgeführt werden.

Unter diesen Annahmen kann die Angabe der in Abbildung 7.9 dargestellten Aspekt-Knoten einschließlich der an ihnen aufgeführten Kopplungsbeziehungen in einer CS vollständig entfallen. Zur weiteren Erhöhung der Kompaktheit der deklarativen Steuerungsspezifikation werden zusammengehörige Zwangsbedingungen zu einer gestrichelten Linie zusammengefasst. Diese wird erst am Zielknoten in Einzelkanten aufgegliedert. Für die Auflösbarkeit der Einzelkanten ist der Inhalt der Zwangsbedingungen an deren Enden notiert. Die sich unter diesen Annahmen und Vereinbarungen ergebende CS für das minimale Montageproblem zeigt Abbildung 7.10.

Abbildung 7.10 zeigt, dass in der CS für die Bezeichner der Entitäten-Knoten anstelle der Aufgabennamen nunmehr die bereits im Kapitel 6 verwendeten prozeduralen Aufgabennotationen benutzt werden. Aufgrund der oben genannten Annahmen, enthält die CS keinerlei Kopplungsbeziehungen, weil implizit die Regel gilt, dass jede Aufgabe nur einen Ausgang in Form ihres Ergebnisses hat, welcher mit dem Eingang der benachbarten rechts stehenden Aufgabe verbunden ist. Damit steht das Ergebnis der vorangegangenen Aufgabe zur Parametrierung der Nachfolgeaufgabe, wenn erforderlich, zur Verfügung.

Der ?-Operator wurde gegenüber Abbildung 7.9 aus der Parametrierung der IOBJ-Aufgabe auf die Kante zur nachfolgenden Aufgabe PIO verlagert. Dadurch wird die korrekte Lesbarkeit vereinfacht. Das heißt, der ?-Operator zeigt an, dass das Pruning die

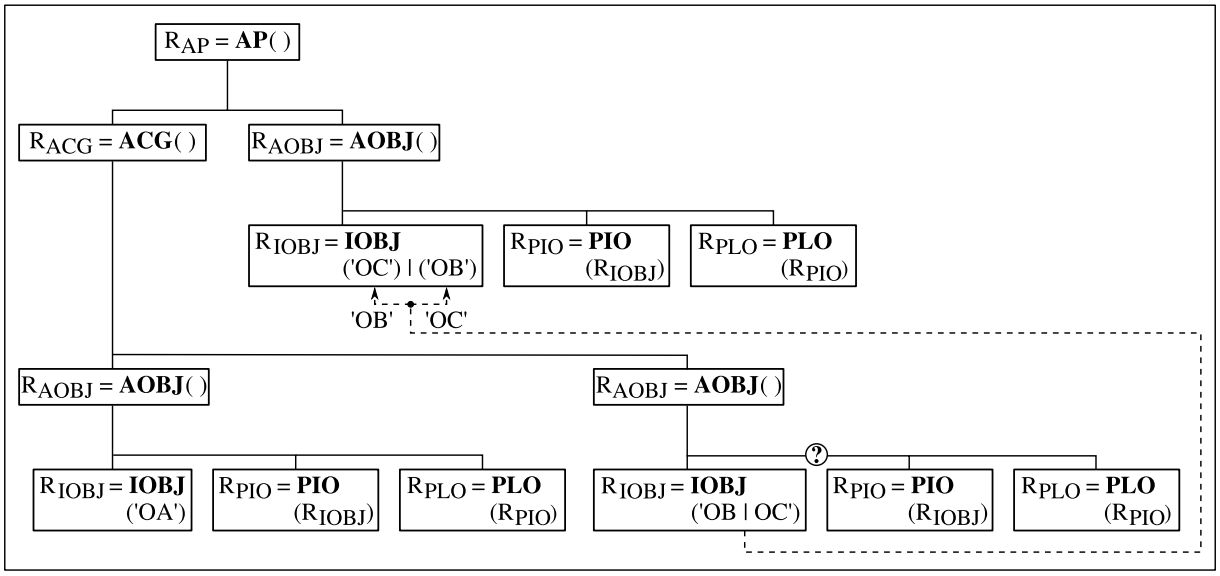


Abbildung 7.10: Deklarative Steuerungsspezifikation des Montageproblems in einer CS

Aufgabe IOBJ noch vollständig einzuschließen hat und erst danach die Generierung der TPCS abgeschlossen wird.

Abbildung 7.11 zeigt die im Ergebnis des ersten am ?-Operator beendeten Prunings generierte TPCS für das betrachtete Montageproblem. Auf Basis dieser nun eindeutigen

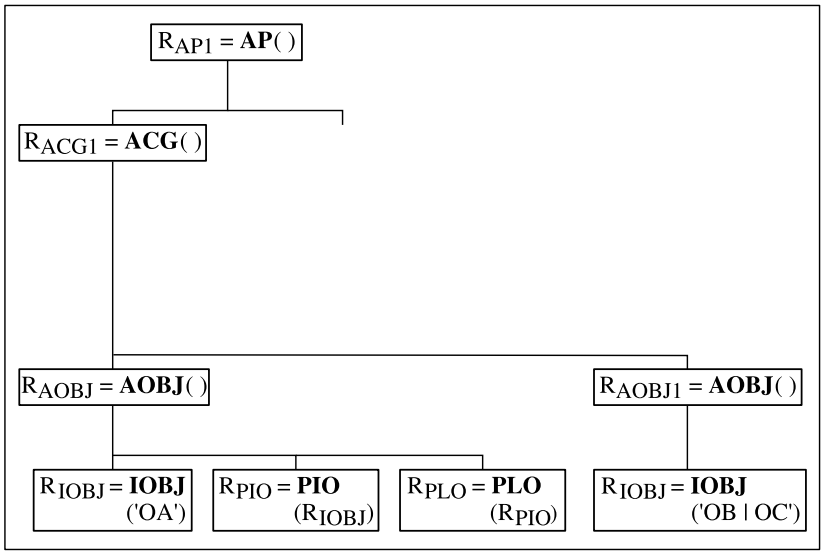


Abbildung 7.11: Erste TPCS des Montageproblems

Steuerungsspezifikation und unter Verwendung der in der Modellbasis bereitstehenden Aufgabenimplementierungen kann ein erstes ausführbares Steuerungsmodell automatisch auf-

gebaut werden. Die Struktur und Parametrierung dieses TECM zeigt Abbildung 7.12. Die Darstellungsweise des aufgabenorientierten Steuerungsmodells ist bereits aus Kapitel 6 bekannt. Entsprechend der TPCS besitzt das TECM keine Eingänge und kann somit ohne Angabe von Parametern gestartet werden.

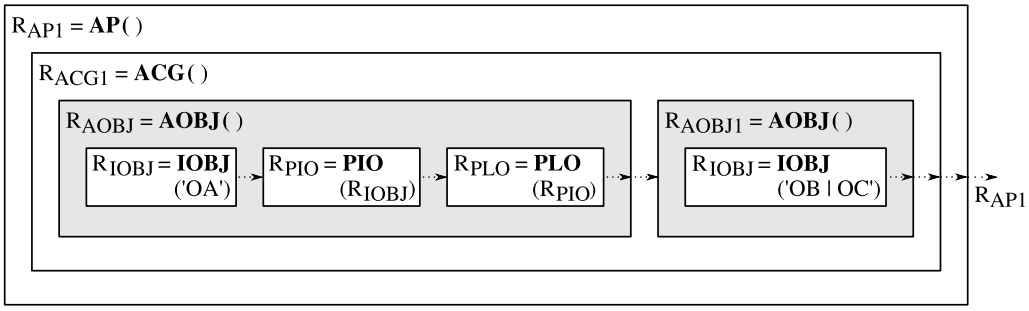


Abbildung 7.12: Erstes TECM des Montageproblems

Aus Abbildung 7.12 wird ersichtlich, dass das erste TECM tatsächlich nur eine Aufgabe AOBJ mit dem Ergebnis $R_{AOBJ} = 'OA'$ *vollständig* ausführt. Die zweite AOBJ-Aufgabe wird mit der Ausführung der Basisaufgabe *IOBJ('OB|OC')* vorläufig nur unvollständig gelöst. $R_{AP1} = R_{ACG1} = R_{AOBJ1} = R_{IOBJ}$ ist damit als erstes Zwischenergebnis der zweiten AOBJ-Aufgabe sowie der darauf aufbauenden Aufgaben ACG und AP zu verstehen.

Zur weiteren Erläuterung des Montageproblems soll angenommen werden, dass nach der Montage von OA im Eingangspuffer ein Einzelteil OC zur Verfügung stand. Nach der Ausführung des ersten TECM ist damit das Ergebnis $R_{AP1} =' OC'$. Der zweite Zyklus der automatischen Steuerungsgenerierung startet mit der Aktualisierung der in Abbildung 7.10 gezeigten Ausgangs-CS. Bei der Aktualisierung werden alle Teile der CS entfernt, die beim vorangegangenen Pruning bereits ausgewählt wurden. Darüber hinaus werden vorhandene Zwangsbedingungen auf Basis der Ergebnisse des zuletzt ausgeführten TECM so weit wie möglich aufgelöst.

Abbildung 7.13 zeigt die aktualisierte CS für das Montageproblem. Es ist ersichtlich, dass die CS bereits nach der ersten Aktualisierung keine Zwangsbedingungen beziehungsweise Alternativen besitzt. Beim Pruning wird diese CS damit ohne Änderungen in eine TPCS überführt, auf deren Basis dann ein zweites TECM generiert wird.

Aus dem in Abbildung 7.14 dargestellten zweiten TECM wird deutlich, das dieses zum Start das Ergebnis der ersten TECM-Ausführung übergeben bekommt. Auf dieser Basis können die im ersten TECM nur teilweise bearbeiteten Aufgaben nunmehr vollständig gelöst werden. Nach Ausführung des zweiten TECM liegt das Ergebnis R_{AP} der Gesamtsteuerungsaufgabe AP vor, was als Beendigung der Steuerungsaufgabe zu interpretieren ist. Formal stellt sich eine erfolgreiche Terminierung der Gesamtsteuerungsaufgabe in einer aktualisierten CS dar, die nur noch den Wurzelknoten als einzigen Entitäten-Knoten besitzt.

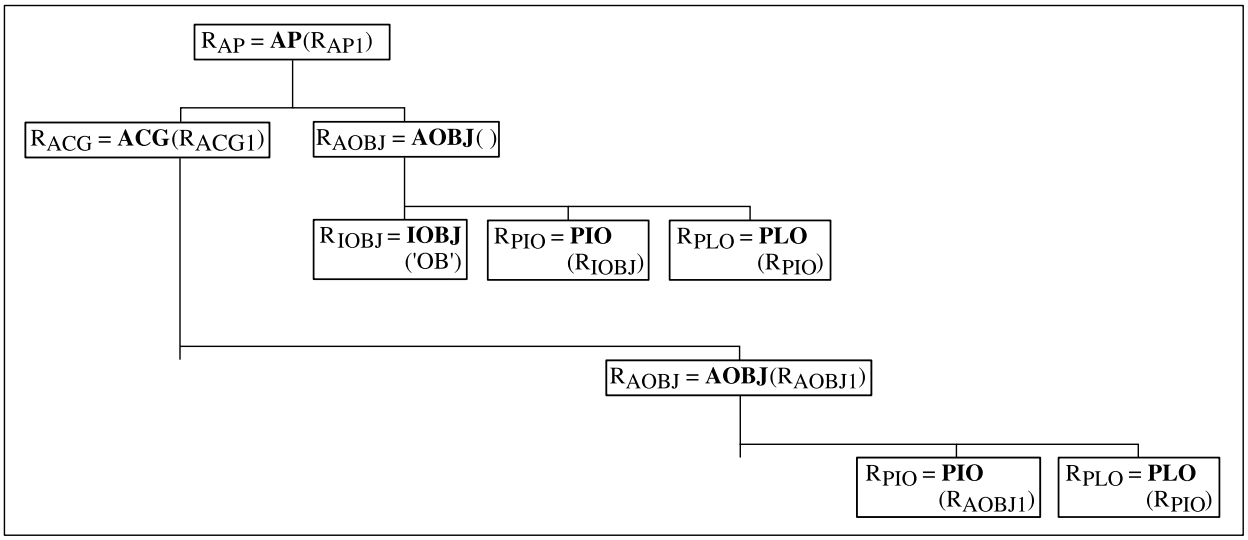


Abbildung 7.13: Erste aktualisierte CS des Montageproblems

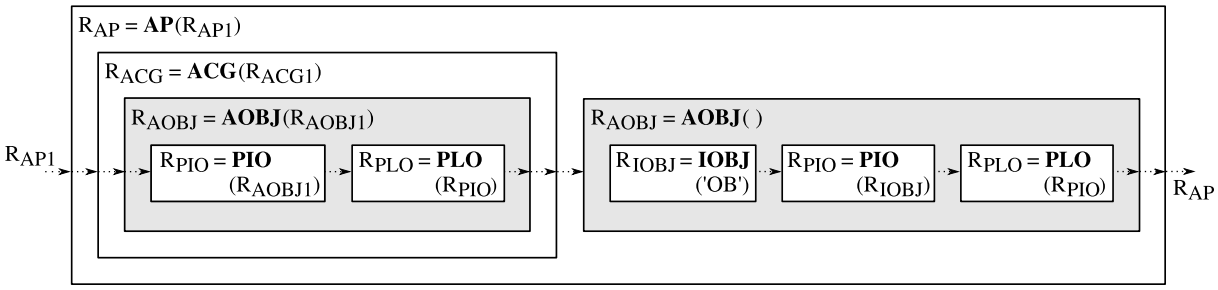


Abbildung 7.14: Zweites TECM des Montageproblems

7.4 Entwurf und Inbetriebnahme einer flexiblen aufgabenorientierten Robotersteuerung

Das im vorangegangenen Abschnitt aus nur drei Einzelteilen bestehende Montageproblem diente der Erläuterung der eingeführten Modifikationen am SES/MB-Framework zur automatischen Generierung von Steuerungsprogrammen. Zur Verdeutlichung der Vorteile flexibler aufgabenorientierter Robotersteuerungen ist ein Problem so geringer Komplexität allerdings wenig überzeugend. In diesem Abschnitt soll deshalb die Anwendung der eingeführten Methodik nochmals an einem realen Montageproblem demonstriert werden.

Abbildung 7.15 zeigt ein Labyrinth-Spiel in einer sogenannten Explosionsdarstellung als zu montierendes Produkt. Diese Form einer technischen Zeichnung ist insbesondere dazu

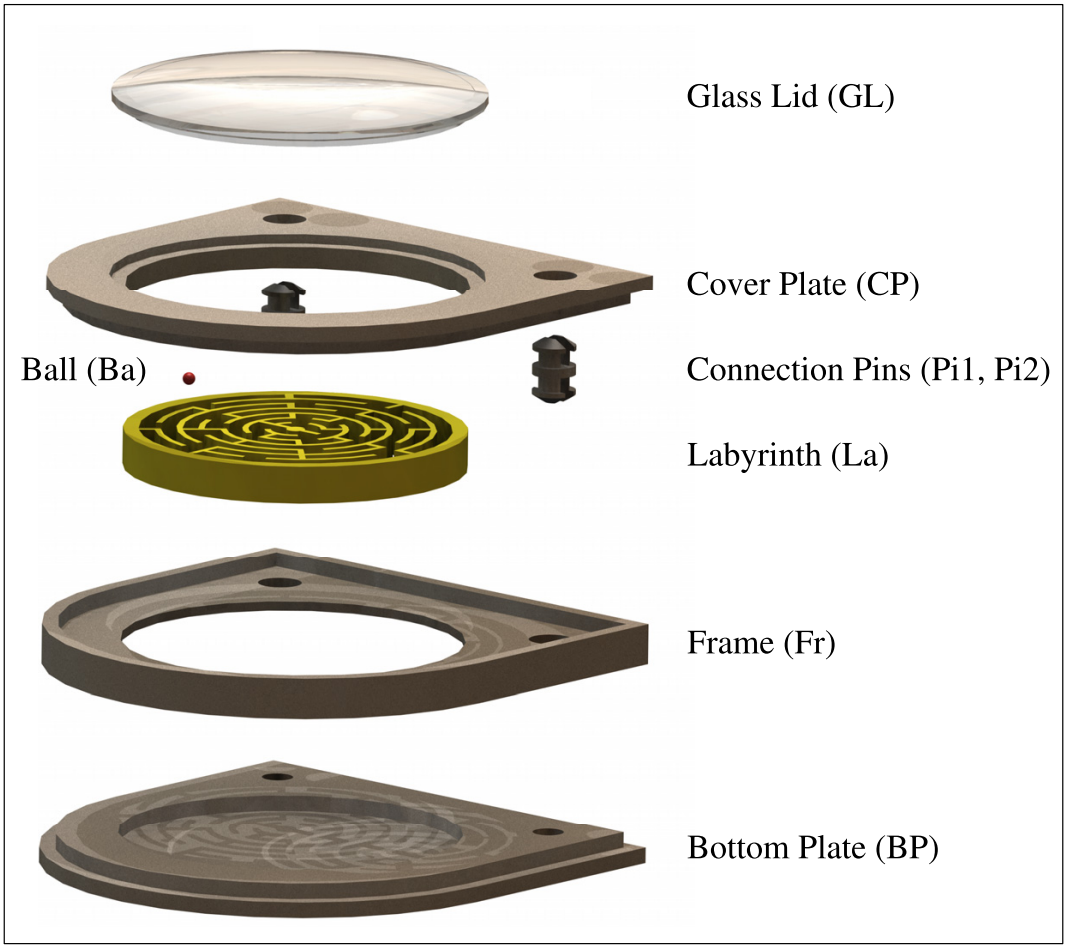


Abbildung 7.15: Explosionsdarstellung des zu montierenden Produktes

geeignet, komplexe Produkte in seine Einzelteile zerlegt darzustellen. Sie verdeutlicht die Lage der Einzelteile zueinander und erlaubt erste Aussagen über mögliche Montagereihenfolgen. Das gezeigte Produkt besteht aus acht Einzelteilen und unter Ausschluss von

nicht realisierbaren Montagemöglichkeiten existierten insgesamt 25 Montagevarianten. Diese müssen im Rahmen der Montagevorgangsplanung in systematischer Weise dokumentiert werden. Abbildung 7.16 zeigt das Ergebnis der Vorgangsplanung in Form eines Und/Oder-Graphen.

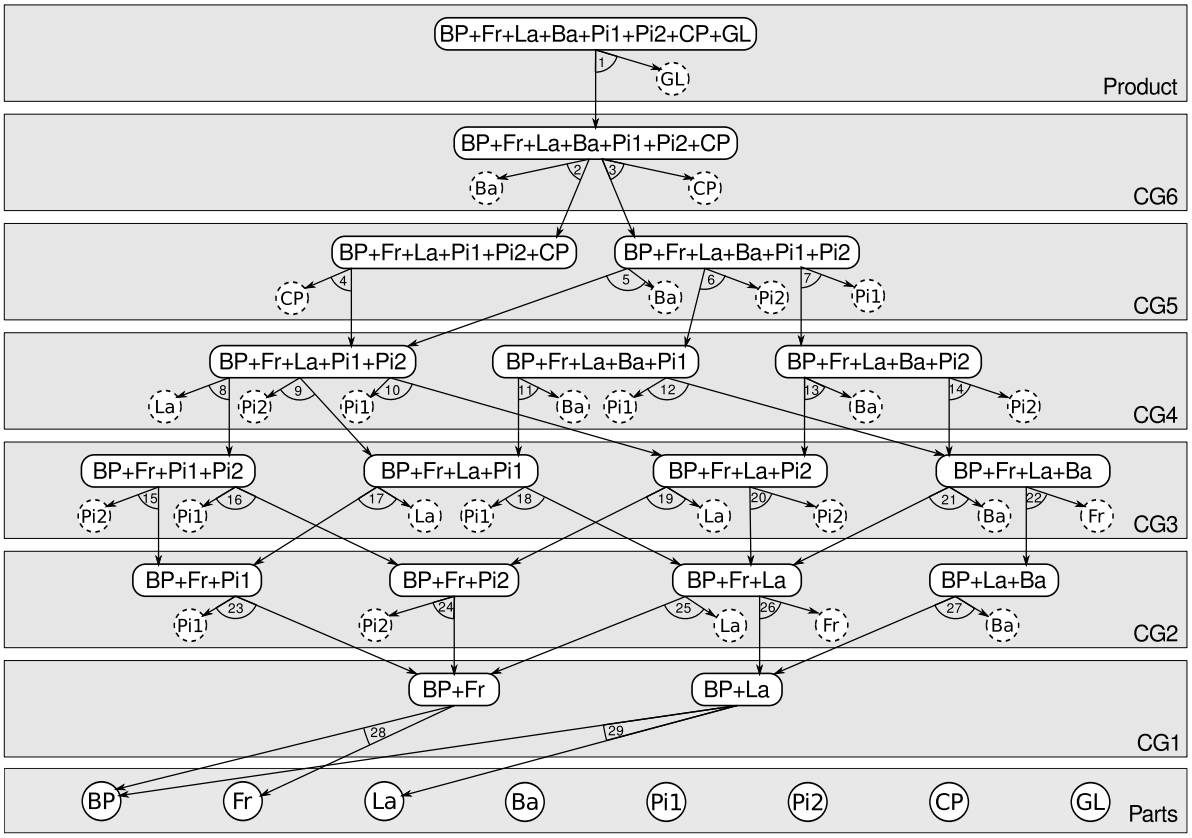


Abbildung 7.16: Und/Oder-Graph

Technisch nicht realisierbare Einzelteilkombinationen wurden bereits eliminiert und sind im Und/Oder-Graph nicht dargestellt. Die Eliminierungsvorschriften werden im Rahmen der Montagevorgangsplanung aufgestellt und sehen im konkreten Montagebeispiel wie folgt aus:

1. Die Bodenplatte (BP) wird stets vor allen anderen Einzelteilen montiert.

2. Einzelteile, die von Verbindungselementen (Pi1, Pi2) fixiert werden, sind vor diesen zu montieren.

3. Einzelteile, die von anderen Teilen umfasst werden, sind vor diesen zu montieren.

4. Der Glasdeckel (GL) wird stets zuletzt montiert.

Die für die automatisierte Montage zur Verfügung stehende Roboterzelle hat wiederum eine Struktur, wie sie bereits für das minimale Montageproblem im vorangegangenen Abschnitt angenommen wurde. Die Zelle setzt sich aus einem Roboter mit bildgebender Sensorik, einem Teileeingangspuffer und einem Montageplatz zusammen. Die Bereitstellung der Einzelteile im Eingangspuffer erfolgt simultan zum Montageprozess. Deshalb wird von der zu entwickelnden Robotersteuerung gefordert, dass die Montagereihenfolge nicht fest vorgegeben ist, sondern von der konkreten Teileverfügbarkeit im Eingangspuffer abhängt. Das Entwurfsproblem der Automatisierungsphase besteht also in der Spezifikation einer flexiblen aufgabenorientierten Steuerung auf Basis des aus der Montagevorgangsplanung übergebenen Und/Oder-Graphen.

Aus Abschnitt 7.3.2 ist bekannt, dass die Spezifikation einer flexiblen aufgabenorientierten Steuerung in Form einer CS in systematischer Weise aus den Informationen eines Und/Oder-Graphen abgeleitet werden kann. Im ersten Entwurfsschritt sind die erforderlichen Basisaufgaben zur Lösung des Montageproblems festzulegen. Das hier betrachtete Realproblem besteht nunmehr aus acht Einzelteilen. Es ist aber offensichtlich, dass deren Identifikation, Transport und Montage mit den bereits im vorangegangenen Abschnitt verwendeten Basisaufgaben IOBJ, PIO und PLO, die in einfacher Sequenz die zusammengesetzte Aufgabe AOBJ bilden, möglich ist.

Im zweiten Entwurfsschritt ist die Struktur der CS aus dem Und/Oder-Graphen abzuleiten. Maßgeblich ist dabei, über wie viele Ebenen von Baugruppen (CG) die Montage des Produktes aus den Einzelteilen erfolgt. Im hier betrachteten Problem sieht die Vorgangsplanung eine Montage über jeweils *sechs* Baugruppenebenen vor. Daraus ergibt sich, dass auch die zu entwerfende CS eine Hierarchie von sechs aufeinander aufbauenden Aufgaben ACG (Assemble Construction Group) aufweisen muss. Im abschließenden dritten Entwurfsschritt sind die im Und/Oder-Graphen vorgesehenen alternativen Montagevarianten in Form bedingter Aufgabenparametrierungen, also unter Zuhilfenahme von Zwangsbedingungen und des ? - Operators in der CS zu spezifizieren. Die im Ergebnis der drei Entwurfsschritte vorliegende CS ist in Abbildung 7.17 dargestellt.

Stehen die Implementierungen der Basisaufgaben IOBJ, PIO, PLO im Rahmen einer Aufgabenbibliothek zur Verfügung, kann die CS, wie im Abschnitt 7.3.2 beschrieben, unmittelbar zur Generierung ausführbarer Steuerungen und damit für den operativen Betrieb eingesetzt werden. Die Anzahl der dabei automatisch ablaufenden Zyklen:

CS ➔ TPCS ➔ TECM ➔ CS-Update

ergibt sich aus der Anzahl von ? - Operatoren, welche die Ausgangs-CS beinhaltet. Im hier betrachteten Fall sind *fünf* ? - Operatoren enthalten. Die Gesamtsteuerungsaufgabe, dass heißt die Montage eines kompletten Produktes, benötigt damit *sechs* Zyklen. Nachfolgend soll beispielhaft ein möglicher Ablauf der flexiblen Steuerung betrachtet werden.

Nach dem Start der Montage mit der Bodenplatte (BP) soll die weitere Einzelteilverfügbarkeit in der Reihenfolge: Frame (Fr), Labyrinth (La), Pin 1 (Pi1) und Pin 2 (Pi2) stattgefunden haben. Zur Montage dieser fünf Einzelteile wurden vier TECM generiert, nach deren Ausführung eine Baugruppe der vierten Ebene vorliegt. Der konkret abgelau-

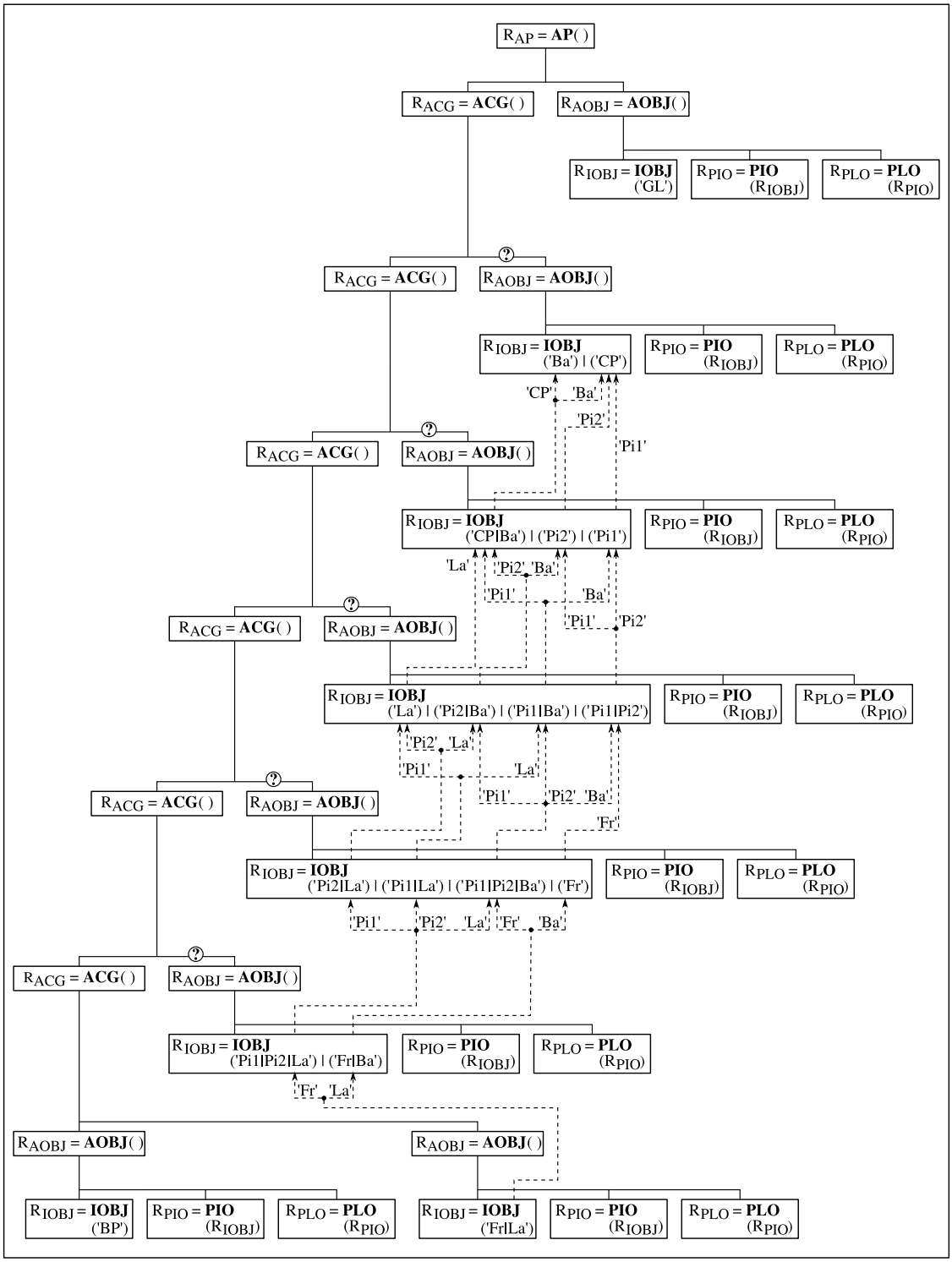


Abbildung 7.17: Spezifikation der flexiblen aufgabenorientierten Steuerung

fene Montageweg bis zur vierten Baugruppenebene lässt sich mit dem in Abbildung 7.18 dargestellten Und/Oder-Graphen gut veranschaulichen.

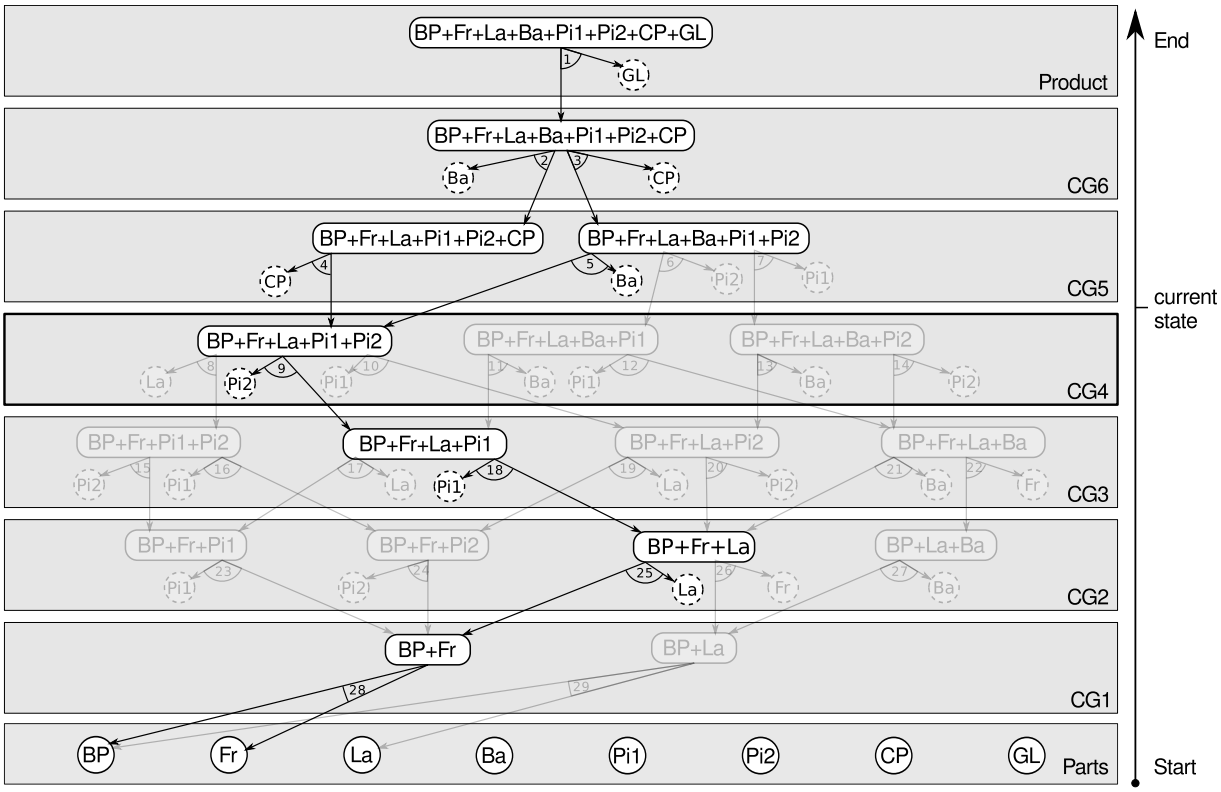


Abbildung 7.18: Und/Oder-Graph einer stattgefundenen Montagevariante

Außerdem ist zu erkennen, dass zu diesem Zeitpunkt nur noch zwei mögliche Varianten für die Fertigstellung des Produktes existieren. Für die Realisierung der Baugruppe der fünften Ebene kann im nächsten Schritt entweder das Einzelteil *CP* oder *Ba* montiert werden. Dementsprechend enthält die für den anschließenden fünften Zyklus aktualisierte CS, wie in Abbildung 7.19 gezeigt, nur noch einen ? - Operator und eine Relation von Zwangsbedingungen. Die daraufhin generierte TPCS sowie das TECM sind in den Abbildungen 7.20 und 7.21 dargestellt.

Unterstellt man, dass im fünften Zyklus im Eingangspuffer eine Abdeckplatte CP zur Montage bereit steht, liegt nach Ausführung des TECM eine montierte Baugruppe der fünften Ebene in der Zusammensetzung: $\boxed{\mathrm{BP+Fr+La+Pi1+Pi2+CP}}$ vor.

Wie die in Abbildung 7.22 dargestellte CS-Aktualisierung für den anschließenden sechsten Zyklus zeigt, sind nunmehr keine ? - Operatoren und Zwangsbedingunen enthalten. Die durch das Pruning generierte TPCS ist somit identisch zur letzten CS (Abb. 7.22). Das für den sechsten Zyklus generierte TECM zeigt Abbildung 7.23. Dieses signalisiert nach erfolgreicher Ausführung mit $R_{AP} = 'Bp + Fr + La + Pi1 + Pi2 + CP + Ba + GL'$ schließlich den Abschluss der gesamten Montageaufgabe AP.

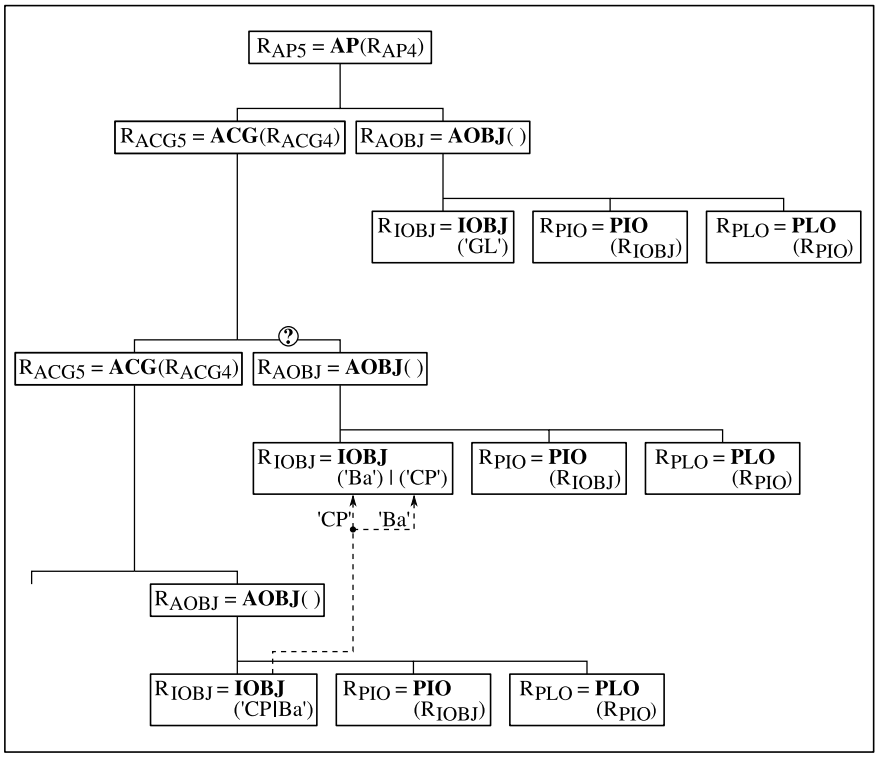


Abbildung 7.19: Aktualisierte CS für den fünften Zyklus

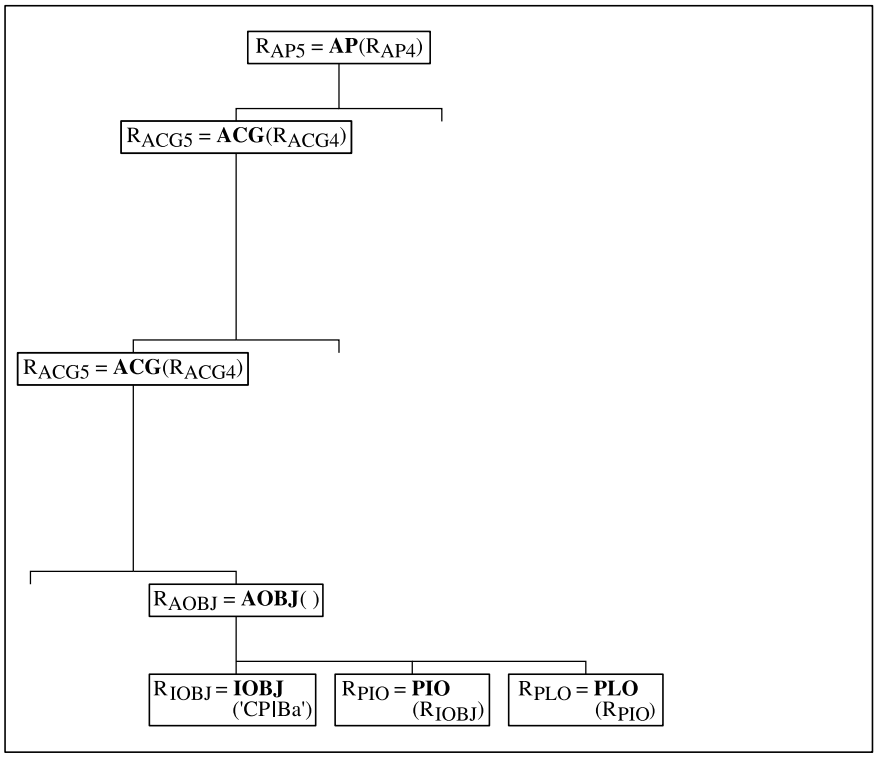


Abbildung 7.20: TPCS im fünften Zyklus

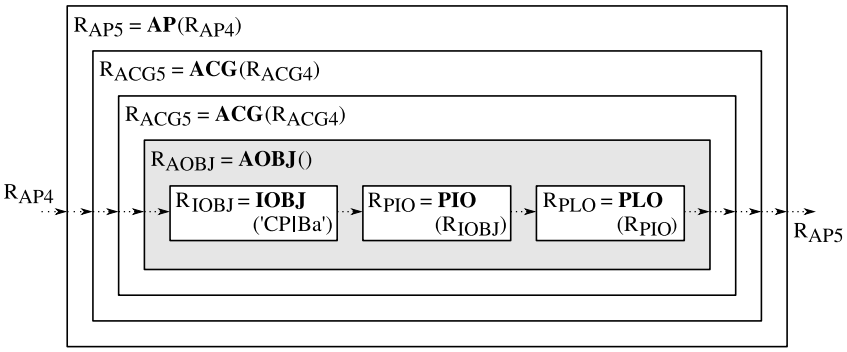


Abbildung 7.21: TECM im fünften Zyklus

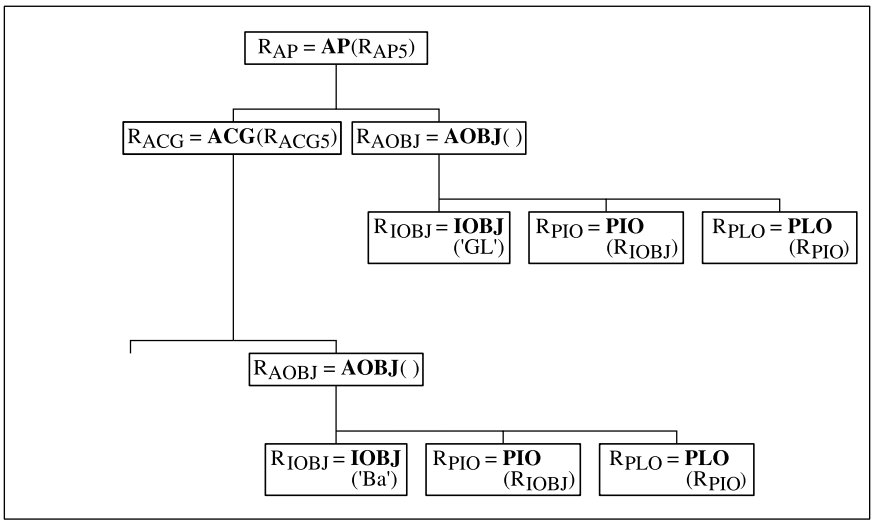


Abbildung 7.22: Aktualisierte CS für den abschließenden sechsten Zyklus

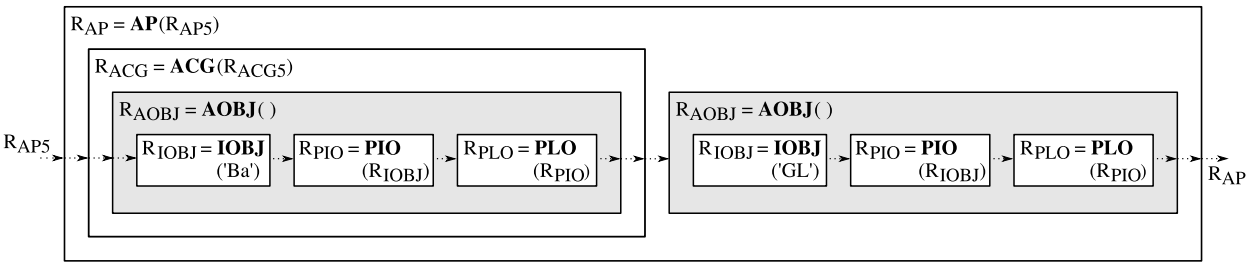


Abbildung 7.23: Letztes TECM im sechsten Zyklus

7.5 Zusammenfassung

In diesem Kapitel wurde anhand eines konkreten Beispiels die Realisierung einer adaptiven und damit flexiblen aufgabenorientierten Robotersteuerung demonstriert.

Es wurde gezeigt, dass die im Abschnitt 7.1 formulierte Flexibilitätsanforderung an eine Steuerung in FMS und RMS durch eine hohe Anpassungsfähigkeit erreicht werden kann. Das setzt voraus, dass sowohl die Struktur als auch die Parametrierung einer Robotersteuerung umfassend modifizierbar ist. Das heißt, im Sinne einer aufgabenorientierten Robotersteuerung sollten sich die konkreten Aufgabenfolgen erst während des operativen Betriebes in Abhängigkeit von den aktuellen Prozessbedingungen und den Zielvorgaben der Leitebene ergeben.

Diese Funktionalität kann mit den in den vorangegangenen Kapiteln vorgestellten Entwurfstechniken nach dem SBC-Ansatz noch nicht hinreichend erfüllt werden, da diese auf fest codierte Steuerungsstrukturen abstellen. Deshalb wurde für die Realisierung eine domainenunabhängige deklarative Spezifikation von aufgabenorientierten Robotersteuerungen in Kombination mit einer automatisierten Steuerungssynthese vorgeschlagen. Es wurde gezeigt, wie diese Methodik im Kontext des SBC-Ansatzes realisierbar ist. Da der SBC-Ansatz auf einer leistungsfähigen Simulationsumgebung basiert, sind grundsätzlich geeignete Voraussetzungen für die Integration von Methoden zur Informationsmodellierung gegeben. Als geeigneter Ansatz wurde hierfür das aus der Simulationstechnik bekannte SES/MB-Framework identifiziert, welches neben einer deklarativen Steuerungsbeschreibung auch Methoden zur Generierung von ausführbaren Modellen definiert. Um während des operativen Betriebes eine anpassungsfähige Steuerung zu gewährleisten, war es erforderlich, die Funktionalität des SES/MB-Framework anzupassen. Die notwendigen Modifikationen wurden anhand von beispielhaften Montageproblemen erläutert und es konnte schließlich die Realisierbarkeit einer operativen flexiblen aufgabenorientierten Robotersteuerung auf Basis des SES/MB-Framework an einem Realproblem nachgewiesen werden.

8 Zusammenfassung und Ausblick auf weiterführende Arbeiten

Die vorliegende Arbeit ist unter der Zielstellung enstanden, einen Beitrag zur stärkeren Etablierung von *Rapid Control Prototyping*-Techniken im Bereich der Steuerungsentwicklung von komplexen und flexiblen Roboteranwendungen zu leisten. Zum grundlegenden Verständnis sollten zuerst die unterschiedlichen Methoden der Roboterprogrammierung vorgestellt und in einer für diese Arbeit geeigneten Klassifikation zusammengetragen werden. Die aus Sicht der aktuellen Robotikanwendungsbereiche relevanten Methodenklassen sollten hinsichtlich der gegenwärtig verwendeten Entwurfs- und Inbetriebnahme-Methodik analysiert und aus der Perspektive der allgemeinen Entwicklung diskreter Steuerungen innerhalb der Automatisierungstechnik betrachtet werden.

Ausgangspunkt der Arbeit war die Lösung des Klassifikationsproblems der unterschiedlichen Roboterprogrammiermethoden. Als Ergebnis wurde eine Klassifikation vorgeschlagen, die verschiedene aus der Literatur bekannte Systematiken in sich vereint. Die Klassifikation umfasst die Hauptklassen: Online-Methoden und Offline-Methoden. Es wurde deutlich, dass die Online-Methoden mit den Unterklassen: Programmierung durch Beispiele (Teach-In-Techniken) und Programmierung durch Training für den Bereich der einfachen Roboteranwendungen eine dominierende Stellung einnehmen. Bei komplexen Roboteranwendungen, die in der Regel mit umfangreicher externer Aktorik und Sensorik ausgestattet sind, werden dagegen Offline-Methoden eingesetzt. Konkret handelt es sich dabei um Anwendungen, die der Unterklasse der roboterorientierten Programmierung zuzuordnen sind. Solche Roboteranwendungen stellen anspruchsvolle Entwurfs- und Inbetriebnahmeprobleme der Steuerungstechnik dar, die nur im Rahmen eines systematischen Entwicklungsprozesses effizient beherrschbar sind. Für eine detailliertere Untersuchung wurde das aus der Automatisierungstechnik bekannte *V-Modell* sowie das Rapid Control Prototyping (RCP) zugrundegelegt. Der Fokus lag dabei auf einem für die RCP-Methodik notwendigen durchgängigen Steuerungsentwicklungsprozess von der Steuerungsspezifikation bis zur Inbetriebnahme mit zusätzlichen Möglichkeiten für SiL- und HiL-Simulationen. Nach einer mit diesem Schwerpunkt durchgeführten Analyse der gegenwärtig in der industriellen Robotik verwendeten Steuerungsentwicklungsmethoden konnte festgestellt werden, dass noch nicht von einer umfassenden RCP-Methodik gesprochen werden kann. Deshalb wurde der *Simulation Based Control (SBC) Ansatz* eingeführt und dessen Verwendung auf dem Gebiet der Industrierobotik diskutiert. Als Ergebnis stellte sich heraus, dass auch für Industrieroboteranwendungen mit Hilfe des SBC-Ansatzes ein durchgängiger RCP-Entwicklungsprozess möglich ist. In diesem Zusammenhang wurde darüber hinaus gezeigt, dass die vom SBC-Ansatz gestellten hohen Anforderungen an die Entwicklungsplattform hinsichtlich der zu unterstützenden

Beschreibungsmittel und deren Verwendung in den einzelnen Entwurfsschritten bis hin zur inkrementellen Inbetriebnahme durch die RCP-Umgebung Matlab erfüllt werden können. Der Nachweis dafür wurde anhand eines konkreten Anwendungsbeispieles erbracht, indem der gesamte Entwicklungsprozess einer komplexen Roboteranwendung auf Basis des SBC-Ansatzes innerhalb der RCP-Umgebung Matlab demonstriert wurde.

Eine weitere Zielstellung der vorliegenden Arbeit bestand darin, einen Beitrag zur Realisierung von flexiblen aufgabenorientierten Robotersteuerungen zu leisten. Für den Bereich der aufgabenorientierten Anwendungsprobleme sollte zuerst gezeigt werden, wie die aufgabenorientierte Roboterprogrammierung, die als Unterklasse neben der roboterorientierten Programmierung zu den Offline-Methoden zählt, in einen auf dem SBC-Ansatz basierenden Entwicklungsprozess einzuordnen ist. Dabei war zu analysieren, welches Potential die aufgabenorientierte Programmierung zur Realisierung von flexiblen Steuerungen besitzt. Ein besonderer Schwerpunkt wurde dabei auf die Umsetzung von adaptiv arbeitenden Steuerungen gelegt, die insbesondere bei flexiblen (FMS) und rekonfigurierbaren Fertigungssystemen (RMS) zum Einsatz kommen.

Anhand von zwei Beispielproblemen wurde der vollständige Entwicklungsprozess für aufgabenorientierte Robotersteuerungen, sowohl in einer Top-Down- als auch Bottom-Up-Vorgehensweise dargestellt. Dabei wurde herausgearbeitet, dass die Spezifik des aufgabenorientierten Entwurfs im ersten Schritt der Automatisierungsphase, also in der Phase der Separierung des Planungsmodells in ein Steuerungs- und ein Prozessmodell, angesiedelt ist. Darüber hinaus wurde gezeigt, dass eine Aufgabenparametrierung die Komplexität einer aufgabenorientierten Steuerung reduzieren kann, indem die Parametrierung einerseits das Zusammenfassen mehrerer ähnlicher Aufgaben ermöglicht, andererseits können in Aufgabenfolgen die Ergebnisse einer Aufgabe zur Parametrierung von nachfolgenden Aufgaben benutzt werden.

Die Aufgabenparametrierung ist ebenfalls von zentraler Bedeutung bei der Flexibilisierung von Steuerungen. Ein wesentliches Konzept zur Erreichung hoher Flexibilität ist das der adaptiven Steuerungen. Nach einer Analyse verschiedener aus der Literatur bekannter adaptiver Steuerungsansätze wurde deutlich, dass eine große Herausforderung in der Spezifikation der flexiblen Steuerung besteht. Zu diesem Zweck wurde als ein geeignetes Mittel das *System Entity Structure (SES) and Model Base (MB) Framework* nach Zeigler identifiziert, welches neben einer deklarativen Steuerungsbeschreibung auch Methoden zur automatischen Generierung von ausführbaren Steuerungen definiert.

In der vorliegenden Arbeit wurden die Untersuchungen zur Anwendung des SES/MB-Frameworks zur Realisierung flexibler aufgabenorientierter Robotersteuerungen auf den SES-Aspekt der Dekomposition beschränkt. Mittels einer anwendungsorientierten Studie konnte der Nachweis erbracht werden, dass die automatische Generierung flexibler Robotersteuerungen auf Basis einer deklarativen aufgabenorientierten Steuerungsspezifikation möglich ist. Inwieweit dieser Ansatz auch bei Verwendung der SES-Spezialisierung und des SES-Multiaspekts tragfähig ist, wird in einer weiterführenden Arbeit von Schwatinski untersucht ([Sch11], [Sch12]). Darüber hinaus werden in dieser Arbeit alternative Beschreibungsmittel für diskret-ereignisorientierte Systeme im Rahmen des SBC-Ansatzes untersucht. Der Fokus wird dabei alternativ zu den hier verwendeten Zustandsdiagrammen auf

dem *Discrete Event System (DEVS) Formalismus* liegen.

Weitergehende Arbeiten werden ebenfalls das Fachgebiet der *parallelen diskret-ereignis-orientierten Simulation (PDES)* in den Steuerungsentwurf miteinbeziehen. In der vorliegenden Arbeit wurde der SBC-Ansatz als grundlegende Entwurfsmethodik eingeführt und dessen Verwendung auf dem Gebiet der Industrierobotik an verschiedenen Anwendungsbeispielen demonstriert. Anwendungsfälle bestehend aus großen verteilten Steuerungsproblemen wurden bisher noch nicht untersucht. Der Fragestellung, welche Geschwindigkeitsvorteile bei solchen Problemen durch PDES-Methoden erzielbar sind, wird sich die Arbeit von Stenzel ([Ste13]) widmen.

Literaturverzeichnis

[Abb09] Abb Automation Technology Products: *RobotStudio*. Data Sheet, März 2009.

[Abe06] Abel D., Bollig A.: *Rapid Control Prototyping - Methoden und Anwendungen*. Springer Verlag, 2006.

[Aui99] Auinger F., Vorderwinkler M., Buchtela G.: *Interface driven domain-independent modeling architecture for Soft-Commissioning and Reality in the Loop*. In *Proc. of the Winter Simulation Conference*, pages 798–805, 1999.

[Big03] Biggs G., MacDonald B.: *A survey of robot programming systems*. In *Proc. of the Australiasian Conf. on Robotics and Automation*, 2003.

[Bis02] Bischoff R., Kazi A., Seyfarth M.: *The MORPHA Style Guide for Icon-Based Programming*. In *Proc. of the 11th IEEE Int. Workshop on Robot and Human interactive Communication*, pages 482–487, Berlin, 2002.

[Blu87] Blume C., Jakob W., Favaro J.: *PasRo: Pascal and C for robots*. Springer Verlag, 1987.

[Cha85] Chang K.-H., Wee W. G.: *A knowledge based planning system for mechanical assembly using robots*. In *Proc. of the 22nd Conference on Design Automation*, pages 330–336, 1985.

[Chi91] Chierotti M., Rozenblit J., Jacak W.: *A framework for simulation design of flexible manufacturing systems*. In *Proc. of the 1991 Winter Simulation Conference*, pages 1106–1114, 1991.

[Cho03] Chong C. S., Sivakumar A. I., Gay R.: *Simulation-based scheduling for dynamic discrete manufacturing*. In *Proc. of the 2003 Winter Simulation Conference*, pages 1465–1473, 2003.

[Chr11] Christern M., Schmidt A., Schwatinski T., Pawletta T.: *KUKA-KAWASAKI-Robotic Toolbox for Matlab*. Website, March 2011. `http://www.mb.hs-wismar.de/cea/KK_Robotic_Tbx/KK_Robotic_Tbx.html`.

[Cra05] Craig J.: *Introduction to Robotics: Mechanics and Control*. Pearson Education, Inc., 2005.

[Dä05] Däubler L.: *Strukturverträgliche Ontologien der Automatisierungstechnik.* Dissertation. TU Braunschweig, 2005.

[Deu93] Deutsches Institut für Normung: *DIN 66312*, 1993. Manipulating industrial robots - Programming language - Industrial Robot Languages (IRL).

[Dra95] Drake G. R., Smith J. S., Peters B. A.: *Simulation as a planning and scheduling tool for flexible manufacturing systems.* In *Proc. of the 1995 Winter Simulation Conference*, pages 805–812, 1995.

[Dre99] Drewelow W., Pawletta T., Pawletta S., Kaehler R., Kirchhoff M.: *Matlab-GPSS-Toolbox.* Website, 1999. `http://www.mb.hs-wismar.de/cea/mgpss/`.

[Ehr06] Ehrmann M., Seckner M.: *Ein Ansatz zur nutzergerechten Programmierung von Industrierobotern.* atp - Automatisierungstechnische Praxis 48, Heft 4:56–62, 2006.

[Ekv06] Ekvall S., Aarno D., Kragic D.: *Task learning using graphical programming and human demonstration.* In *Proc. of the 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 398–403. Univ. of Hertfordshire, UK, 2006.

[Gro07] Groover M. P.: *Automation, production systems and computer-integrated manufacturing.* Prentice Hall, 2007.

[Har87] Harel D.: *Statecharts: A visual formalism for complex systems.* In *Science of Computer Programming*, pages 231–274, 1987.

[Hau07] Haun M.: *Handbuch Robotik.* Springer Verlag, 2007.

[Hei06] Heisel U., Meitzner M.: *Progress in reconfigurable manufacturing systems.* In *Reconfigurable Manufacturing and Transformable Factories*, pages 47–62, 2006.

[Hom91] Homem de Mello L. S., Lee S.: *Computer-aided mechanical assembly planning.* Kluwer Academic Publishers, 1991.

[Jac93] Jacak W., Rozenblit J.: *Virtual process design techniques for intelligent manufacturing.* In *Proc. of the 4th Conference on AI, Simulation and Planning in High Autonomy Systems*, pages 192–198, 1993.

[Jac98] Jacak W., Rozenblit J.: *Model-based workcell task planning and control.* Transactions of the Society for Computer Simulation International, 15(4):167–180, 1998.

[Jac99] JACAK W.: *Intelligent robotic systems design, planning and control.* Klüwer Academic Press, 1999.

[Joh96] JOHNSON D. M.: *The system engineer and the software crisis.* ACM SIGSOFT Sotfware Engineering Notes, 21/2:64–73, 1996.

[Kai07] KAISER C.: *Untersuchungen zur aufgabenorientierten Realisierung flexibler Robotersteuerungen anhand von Montageproblemen.* Master Thesis. Hochschule Wismar, 2007.

[Kai08] KAIN S., HEUSCHMANN C., SCHILLER F.: *Von der virtuellen Inbetriebnahme zur betriebsparallelen Simulation.* atp - Automatisierungstechnische Praxis, Heft 8:48–52, 2008.

[Kaw09] KAWAKAKI ROBOTICS INC.: *PC-ROSET.* Website, März 2009. `http://www.kawasakirobotics.com/products/?page=otherPCROSET`.

[Kim94] KIM M. H., KIM Y.-D.: *Simulation-based Real-Time scheduling in a flexible manufacturing system.* Journal of Manufacturing Systems, 13/2:85–93, 1994.

[Kor99] KOREN Y., JOVANE F., HEISEL U.: *Reconfigurable manufacturing systems.* In *CIRP Annuals*, volume 18, pages 6–12, 1999.

[Kos95] KOSTURIAK J., GREGOR M.: *Simulation von Produktionssystemen.* Springer Verlag, 1995.

[Kre03] KREMP M., PAWLETTA T., PAWLETTA S., COLQUHOUN G.: *Untersuchung verschiedener Strategien zur simulationsmodellbasierten Steuerung von Materialflusssystemen.* In: *17th Symposium Simulationstechnik*, Seiten 373–378, Ghent, 2003. SCS Publishing House.

[Kre06] KREMP M., PAWLETTA T., COLQUHOUN G.: *Simulation-model-based process control of discontinious production processes.* In: *Advances in Manufacturing Technology - XX, 4th International Conf. on Manufacturing Research*, Seiten 49–54, Liverpool, UK, September 2006.

[Kro01] KRONREIF G., FÜRST M.: *TCP-based Communication for Task Oriented Programming and Control of Heterogenous Multi-Robot-Systems.* In *Proc. of the 32th International Symposium on Robotics*, pages 90–95, Korea, 2001.

[Kru01] KRUTH J.-P., VAN GINDERACHTER T., PRIANGGADA I. T., VALCKENAERS P.: *The use of finite state machines for task-based machine tool control.* Computers in Industry, 46:247–258, 2001.

[Kuk02] KUKA ROBOTER GMBH: *Grundlagen der Roboterprogrammierung: Programmierung Experte*, 2002.

[Kuk09] Kuka Roboter GmbH: *KUKA.Sim.* Website, März 2009. http://www.kuka-robotics.com/germany/de/products/software/kuka_sim/kuka_sim_detail/PS_KUKA_Sim_Pro.htm.

[Lan04] Langmann R.: *Taschenbuch der Automatisierung.* Hanser Verlag, 2004.

[Lev88] Levi P.: *Planen für autonome Montageroboter.* Informatik-Fachberichte Nr. 191. Springer Verlag, 1988.

[Li 12] Li Y., Junginger S., Stoll N., Thurow K.: *4D simulation system for laboratory workflow of life science automation.* In *Proc. of the IEEE Int. Instrumentation and Measurement Technology Conf.*, pages 1886–1890, 2012.

[Loz82] Lozano-Pérez T.: *Robot Programming.* Technical Report, Memo No. 698, 1982. MIT AI.

[Mal08] Maletzki G., Pawletta T., Pawletta S., Dünow P., Lampe B.: *Simulationsmodellbasiertes Rapid Prototyping von komplexen Robotersteuerungen.* atp - Automatisierungstechnische Praxis 50, Heft 8:54–60, 2008.

[Mal09] Maletzki G., Christern M., Schmidt A., Pawletta T., Dünow P.: *KUKA-KRL-Toolbox for Matlab and Scilab.* Website, March 2009. http://www.mb.hs-wismar.de/cea/Kuka_KRL_Tbx/Kuka_KRL_Tbx.html.

[Mas04] Maslanka K.: *Matlab toolbox for KUKA-robot remote control.* Diploma Thesis. Hochschule Wismar, 2004.

[Mat10] MathWorks: *Stateflow and Stateflow Coder 7.* User's Guide, September 2010.

[Mat12a] MathWorks: *Matlab - External Interfaces.* User's Guide, March 2012.

[Mat12b] MathWorks: *Matlab - Mathematics.* User's Guide, March 2012.

[Mat12c] MathWorks: *SimEvents.* User's Guide, March 2012.

[Mat12d] MathWorks: *Simscape.* User's Guide, March 2012.

[Mat12e] MathWorks: *Simulink.* User's Guide, March 2012.

[Mey07] Meyer C., Hollmann R., Parlitz C., Hägele M.: *Programmieren durch Vormachen für Assistenzsysteme - Schweiß- und Klebebahnen intuitiv programmieren.* it - Information Technology, 49(4):238–246, 2007.

[Mö96] Möbius F.: *Visuelle Programmierung von Industrierobotern - Ein Beitrag zur bedienergerechten Gestaltung von Programmiersystemen.* Fortschritt-Bericht VDI Reihe 8 Nr. 577. VDI Verlag, 1996.

[Mor03] Morton Y. T., Troy D. A., Pizza G. A.: *A state-based modelling approach to develop component-based control software for flexible manufacturing systems.* Int. J. of Computer Integrated Manufacturing, 16/4-5:292–306, 2003.

[Muj79] Mujtaba S., Goldman R.: *AL users' manual.* Stanford Artificial Intelligence Laboratory, AIM 323, January 1979.

[Ort04] Orth P., Bollig A., Abel D.: *Rapid Control Prototyping diskreter Steuerungen in der Automatisierungstechnik.* In: *SPS/IPC/Drives Kongress*, Seiten 143–152, Nürnberg, 2004.

[Ort05] Orth P.: *Rapid Control Prototyping diskreter Steuerungen mit Petrinetzen.* Fortschritt-Bericht VDI Reihe 8 Nr. 1085. VDI Verlag, 2005.

[Paw02] Pawletta T.: *Untersuchung und Prototypentwicklung zur Steuerung von Materialflusssystemen mittels prozessgekoppelter Simulation.* Abschlussbericht zum BMBF-Projekt (aFuE), 2002. Hochschule Wismar.

[Paw04] Pawletta T., Kremp M., Pawletta S., Colquhoun G.: *Optimisation of manufacturing control stategies using online-simulation.* In: *Proc. of EUROSIM Congress 2004 Paris*, France, September 2004.

[Paw06] Pawletta T., Deatcu C., Hagendorf O., Pawletta S., Colquhoun G.: *DEVS-Based Modeling and Simulation in Scientific and Technical Computing Environments.* In *Proc. of DEVS Integrative M&S Symposium (DEVS'06) - Part of the Spring Simulation Multiconference*, pages 151–158, Huntsville/AL, USA, April 2006.

[Paw11] Pawletta T.: *Entwicklung einer Rapid Control Prototyping-Umgebung für aufgabenorientierte und sprachgesteuerte Roboterapplikationen.* Abschlussbericht zum BMBF-Projekt, September 2011. FKZ: 1747X08, Hochschule Wismar.

[Rec97] Rechenberg P., Pomberger G.: *Informatik-Handbuch.* Hanser Verlag, 1997.

[Roz85] Rozenblit J., Zeigler B. P.: *Concepts for knowledge-based system design environments.* In *Proc. of the 17th Winter Simulation Conference*, pages 223–231, 1985.

[Sch01] Schildmann P., Pawletta T., Lampe B., Pawletta S., Drewelow W.: *Modellierung und Simulation hybrider Systeme mit Strukturdynamik.* DFG-Abschlussbericht, 2001. Hochschule Wismar.

[Sch03] Schlosser A., Bollig A., Abel D.: *Rapid Control Prototyping in der Lehre.* In: *GMA Kongress 2003*, Seiten 1077–1084, Düsseldorf, 2003.

[Sch10] Schwatinski T., Pawletta T., Pawletta S., Kaiser C.: *Simulation-based development and operation of controls on the basis of the DEVS formalism.* In *Proc. of the 7th EUROSIM Congress on Modelling and Simulation*, Prag, Czech Republic, September 2010.

[Sch11] Schwatinski T., Pawletta T., Pawletta S.: *Simulation based implementation of flexible task oriented robot controls using the system entity structure and model base approach.* In *Proc. of the 21th Symposium Simulationstechnik*, 2011.

[Sch12] Schwatinski T., Pawletta T., Pawletta S.: *Flexible task oriented robot controls using the system entity structure and model base approach.* Simulation News Europe (SNE), 22(2):107–114, 2012.

[Sie96] Siegert H.-J., Bocionek S.: *Robotik: Programmierung intelligenter Roboter.* Springer Verlag, 1996.

[Sim98] Simmons R., Apfelbaum D.: *A Task Description Language for Robot Control.* In *Proc. of the Conference on Intelligent Robotics and Systems*, 1998.

[Ste08] Stemmer Imaging Schweiz AG: *Robotern das Sehen beibringen.* MegaLink Precision - Fachzeitschrift für Elektronik und Automation, Heft 5:28–30, 2008. AZ Fachverlage AG, Schweiz.

[Ste09] Stenzel C.: *Modellierung diskret-ereignisorientierter Systeme.* Technical Report, Hochschule Wismar, FG CEA, 2009.

[Ste13] Stenzel C.: *Parallele und verteilte Simulation von diskret-ereignisorientierten Systemen in ingenieurtechnischen Anwendungen.* Promotionsvorhaben. Universität Rostock, 2013. (in Bearbeitung).

[Ter08] Terzic I., Merdan M., Zoitl A., Hegny I.: *Modular assembly machine - ontology based concept.* In *Proc. of the 13th IEEE International Conf. on Emerging Technologies and Factory Automation*, pages 241–244, 2008.

[Tol09] Tolio T.: *Design of flexible production systems - methodologies and tools.* Springer Verlag, 2009.

[Tuf88] Tuffentsammer K. (Hrsg.), Storr A., Lange K., Pritschow G., Warnecke H.-J.: *Flexibles Fertigungssystem - Beiträge zur Entwicklung des Produktionsprinzips. Ergebnisse aus dem Sonderforschungsbereich Fertigungstechnik der Universität Stuttgart.* VCH Verlagsgesellschaft Weinheim, 1988.

[Voi86] Voigt G., Cramer S.: *Diskontinuierliche technologische Prozesse.* Akademie Verlag, 1986.

[Web09] Weber W.: *Industrieroboter - Methoden der Steuerung und Regelung.* Hanser Verlag, 2009.

[Wel08] Weller W.: *Automatisierungstechnik im Überblick.* Beuth Verlag, 2008.

[Wys92] Wysk R. A., Joshi S. B., Pegden C. D.: *Rapid prototyping of shop floor control systems for computer integrated manufacturing.* DARPA project No. 8881, 1992.

[Zei84] Zeigler B. P.: *Multifaceted modelling and discrete event simulation.* Academic Press, 1984.

[Zei00] Zeigler B. P., Praehofer H., Kim T. G.: *Theory of modeling and simulation.* Academic Press, 2000.

[Zei07] Zeigler B. P., Hammonds P. E.: *Modeling and simulation based data engineering: introducing pragmatics into ontologies for net-centric information exchange.* Elsevier Academic Press, 2007.

Publikationsliste des Autors

2005 Maletzki G., Pawletta T., Dünow P., Manemann P.: *Simulationsmodellbasierte Steuerung einer Roboterzelle.* In *18th Symposium Simulationstechnik*, SCS Publishing House, Erlangen, S. 305 - 310, 2005.

2006 Maletzki G., Pawletta T., Dünow P., Kremp M., Lampe B.: *Simulationsmodellbasierte Entwicklung und Realisierung von Robotersteuerungen.* In *Entwurf komplexer Automatisierungssysteme - EKA 2006*, Braunschweig, S. 67 - 77, 2006.

Maletzki G., Pawletta T., Pawletta S., Lampe B.: *A model-based robot programming approach in the Matlab/Simulink environment.* In *Proc. of 4th Int. Conf. on Manufacturing Research*, Liverpool, UK, pp. 377 - 382, 2006.

2007 Maletzki G., Pawletta T., Dünow P., Pawletta S., Lampe B.: *Entwicklung und Realisierung komplexer Robotersteuerungen mit Matlab/Stateflow.* In *12th Symposium Maritime Elektrotechnik, Elektronik und Informationstechnik*, Universität Rostock, S. 71 - 78, 2007.

2008 Maletzki G., Pawletta T., Pawletta S., Dünow P., Lampe B.: *Simulationsmodellbasiertes Rapid Prototyping von komplexen Robotersteuerungen.* atp - Automatisierungstechnische Praxis 50, Heft 8, Oldenbourg Industrieverlag, S. 54 - 60, 2008.

Maletzki G., Stenzel C., Pawletta T., Pawletta S., Lampe B.: *Entwicklung von flexiblen aufgabenorientierten Robotersteuerungen.* In *Entwurf komplexer Automatisierungssysteme - EKA 2008*, ifak, Otto-von-Guericke-Universität Magdeburg, S. 363 - 372, 2008.

2009 Pawletta T., Pawletta S., Maletzki G.: *Integrated Modeling, Simulation and Operation of High Flexible Discrete Event Controls.* In *Proc. of Mathematical Modelling - MATHMOD 2009*, Vienna, Austria, 13 pages, 2009.

Pawletta T., Pawletta S., Maletzki G.: *Spezifikation, Simulation und Generierung hoch flexibler, aufgabenorientierter Robotersteuerungen.* In *Tagungsband ASIM Workshop - Simulation technischer Systeme - Grundlagen, Methoden, Anwendungen*, Dresden, Fraunhofer IRB Verlag, S. 93 - 100, 2009.

2010 Pingel T., Pawletta S., Pawletta T., Maletzki G.: *Manufacturing simulation within a rapid control prototyping approach - a comparative study.* In *Proc. of the 7th EUROSIM 2010 Congress, Vol.2: Full Papers*, Prag, Czech Republic, 6 pages, 2010.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AE ..... Ausführungseinheit
AL ..... Assembly Language
CAD ..... Computer-Aided Design
CAR ..... Computer-Aided Robotics
CM ..... Control Model
CS ..... Control Specification
DES ..... Discrete Event Simulation
DEVS ..... Discrete Event System Specification
EE ..... Entscheidungseinheit
EM ..... Executable Model
FMS ..... Flexible Manufacturing System
HiL ..... Hardware-in-the-Loop
HL ..... High-Level
KRL ..... KUKA Robot Language
LIN ..... Linearbewegung
LL ..... Low-Level
MB ..... Model Base
ME ..... Monitoring-Einheit
PASRO ..... Pascal for Robots
PDES ..... Parallel Discrete Event Simulation
PES ..... Pruned Entity Structure
PM ..... Process Model
PTP ..... Point-to-Point Bewegung
PUMA ..... Programmable Universal Machine for Assembly

RapidCIM .... Rapid Prototyping of Control Software for Computer Integrated Manufacturing

RCP ..... Rapid Control Prototyping

RMS ..... Reconfigurable Manufacturing System

SBC ..... Simulation Based Control

SES ..... System Entity Structure

SiL ..... Software-in-the-Loop

SPS ..... Speicherprogrammierbare Steuerung

TCP ..... Tool Center Point

TDL ..... Task Description Language

TECM ..... Temporary Executable Control Model

TPCS ..... Temporary Pruned Control Specification

TPES ..... Temporary Pruned Entity Structure

VAL ..... Variable Assembly Language

VRML ..... Virtual Reality Modeling Language

Anhang

A.1 Funktionen der KRL-Toolbox

Die nachfolgend aufgelisteten Matlab-Funktionen bringen jeweils eine KRL-Anweisung auf dem KR C3 zur Ausführung. Die Syntax entspricht weitgehend den originalen KRL-Anweisungen. Weitere Informationen sind in [Kuk02], [Mas04] und [Mal09] verfügbar.

1. Functions for communication and safety:

- `[hnd,st]=ComInit(comport,speed,bytesize,parity,stopbits)`
  Command establishes a link to serial interface with given properties (comport, speed, bytesize, parity, stopbits) and returns a handle `hnd` and status variable `st` for failure handling. Subsequently, the handle variable `hnd` is used to execute toolbox commands until the serial communication is closed.

- `[st]=ComClose(hnd)`
  Command closes an established communication via serial interface that is identified by handle `hnd`.

- `[st]=StopKuka(hnd)`
  Command stops a running robot motion immediately.

- `[st]=ExitKuka(hnd)`
  Command closes the serial communication via KUKA controller and moves the robot to HOME-position.

2. Robot motion functions:

- `[st]=ptp(hnd,x,y,z,a,b,c)`
  Command for moving the TCP in the fastest way (ptp-interpolation) from current position to a given position in cartesian coords: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=ptprel(hnd,x,y,z,a,b,c)`
  Command for a relative movement of the TCP in the fastest way (ptp-interpolation) from current position ABOUT given positions in cartesian coords: 3 changes of translational position (x,y,z) and 3 changes of rotation (a,b,c).

- `[st]=ptppos(hnd,x,y,z,a,b,c,s,t)`
  Command for moving the TCP in the fastest way (ptp-interpolation) from current position to a given position in cartesian coords: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptprelpos(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative movement of the TCP in the fastest way (ptp-interpolation) from current position ABOUT given positions in cartesian coords: 3 changes of translational position (x,y,z), 3 changes of rotation (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptpaxis(hnd,a1,a2,a3,a4,a5,a6)`
  Command for moving the TCP in the fastest way (ptp-interpolation) from current position to a given position in axis-specific coords: 6 angles for each robot axis (a1...a6).

- `[st]=ptprelaxis(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a relative movement of the TCP in the fastest way (ptp-interpolation) from current position ABOUT given positions in axis-specific coords: 6 angular adjustments for each robot axis (a1...a6).

- `[st]=ptpcptp(hnd,x,y,z,a,b,c)`
  Command for a continuous motion with approximate positioning according to the cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=ptprelcptp(hnd,x,y,z,a,b,c)`
  Command for a relative continuous motion with approximate positioning according to the cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=ptpposcptp(hnd,x,y,z,a,b,c,s,t)`
  Command for a continuous motion with approximate positioning according to the cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptprelposcptp(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative continuous motion with approximate positioning according to the cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptpaxiscptp(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a continuous motion with approximate positioning according to the cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in axis-specific coords approximately: 6 angles for each robot axis (a1...a6).

- `[st]=ptprelaxiscptp(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a relative continuous motion with approximate positioning according to the

cptp-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative axis-specific coords approximately: 6 angular adjustments for each robot axis (a1...a6).

- `[st]=ptpcptpcdis(hnd,x,y,z,a,b,c)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=ptpcptpcori(hnd,x,y,z,a,b,c)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=ptpcptpcvel(hnd,x,y,z,a,b,c)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=ptprelcptpcdis(hnd,x,y,z,a,b,c)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=ptprelcptpcori(hnd,x,y,z,a,b,c)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=ptprelcptpcvel(hnd,x,y,z,a,b,c)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=ptpposcptpcdis(hnd,x,y,z,a,b,c,s,t)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptpposcptpcori(hnd,x,y,z,a,b,c,s,t)`
  Command for a continuous-path motion with approximate positioning according to the

cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptpposcptpcvel(hnd,x,y,z,a,b,c,s,t)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptprelposcptpcdis(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptprelposcptpcori(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptprelposcptpcvel(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=ptpaxiscptpcdis(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in axis-specific coords approximately: 6 angles for each robot axis (a1...a6).

- `[st]=ptpaxiscptpcori(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in axis-specific coords approximately: 6 angles for each robot axis (a1...a6).

- `[st]=ptpaxiscptpcvel(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation)

from current position via a given intermediate position in axis-specific coords approximately: 6 angles for each robot axis (a1...a6).

- `[st]=ptprelaxiscptpcdis(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cdis-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative axis-specific coords approximately: 6 angular adjustments for each robot axis (a1...a6).

- `[st]=ptprelaxiscptpcori(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cori-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative axis-specific coords approximately: 6 angular adjustments for each robot axis (a1...a6).

- `[st]=ptprelaxiscptpcvel(hnd,a1,a2,a3,a4,a5,a6)`
  Command for a relative continuous-path motion with approximate positioning according to the cptp-criterion and cvel-criterion. The TCP moves in the fastest way (ptp-interpolation) from current position via a given intermediate position in relative axis-specific coords approximately: 6 angular adjustments for each robot axis (a1...a6).

- `[st]=ptp_async(hnd,x,y,z,a,b,c)`
  Command is identical to `ptp`-function however it enables an asynchronous communication.

- `[st]=ptprel_async(hnd,x,y,z,a,b,c)`
  Command is identical to `ptprel`-function however it enables an asynchronous communication.

- `[st]=ptpaxis_async(hnd,a1,a2,a3,a4,a5,a6)`
  Command is identical to `ptpaxis`-function however it enables an asynchronous communication.

- `[st]=ptprelaxis_async(hnd,a1,a2,a3,a4,a5,a6)`
  Command is identical to `ptprelaxis`-function however it enables an asynchronous communication.

- `[st]=ptppos_async(hnd,x,y,z,a,b,c,s,t)`
  Command is identical to `ptppos`-function however it enables an asynchronous communication.

- `[st]=ptprelpos_async(hnd,x,y,z,a,b,c,s,t)`
  Command is identical to `ptprelpos`-function however it enables an asynchronous communication.

- `[st]=lin(hnd,x,y,z,a,b,c)`
  Command for moving the TCP along a straight line (linear-interpolation) from current position to a given position in cartesian coords: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=linrel(hnd,x,y,z,a,b,c)`
  Command for a relative movement of the TCP along a straight line (linear-interpolation) from current position ABOUT a given position in relative cartesian coords: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=linpos(hnd,x,y,z,a,b,c,s,t)`
  Command for moving the TCP along a straight line (linear-interpolation) from current position to a given position in cartesian coords: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linrelpos(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative movement of the TCP along a straight line (linear-interpolation) from current position ABOUT a given position in relative cartesian coords: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=lincdis(hnd,x,y,z,a,b,c)`
  Command for a linear continuous-path motion with approximate positioning according to the cdis-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=lincori(hnd,x,y,z,a,b,c)`
  Command for a linear continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=lincvel(hnd,x,y,z,a,b,c)`
  Command for a linear continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z) and 3 rotations (a,b,c).

- `[st]=linrelcdis(hnd,x,y,z,a,b,c)`
  Command for a relative linear continuous-path with approximate positioning motion according to the cdis-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=linrelcori(hnd,x,y,z,a,b,c)`
  Command for a relative linear continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=linrelcvel(hnd,x,y,z,a,b,c)`
  Command for a relative linear continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a straight line (line-interpolation) from

current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z) and 3 changes of rotations (a,b,c).

- `[st]=linposcdis(hnd,x,y,z,a,b,c,s,t)`
  Command for a linear continuous-path motion with approximate positioning according to the cdis-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linposcori(hnd,x,y,z,a,b,c,s,t)`
  Command for a linear continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linposcvel(hnd,x,y,z,a,b,c,s,t)`
  Command for a linear continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in cartesian coords approximately: 3 translational positions (x,y,z), 3 rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linrelposcdis(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative linear continuous-path motion with approximate positioning according to the cdis-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linrelposcori(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative linear continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=linrelposcvel(hnd,x,y,z,a,b,c,s,t)`
  Command for a relative linear continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a straight line (line-interpolation) from current position via a given intermediate position in relative cartesian coords approximately: 3 changes of translational position (x,y,z), 3 changes of rotations (a,b,c) and Statusbit (s) plus Turnbit (t) to define an unambiguous robot position.

- `[st]=lin_async(hnd,x,y,z,a,b,c)`
  Command is identical to `lin`-function however it enables an asynchronous communication.

- `[st]=linrel_async(hnd,x,y,z,a,b,c)`
  Command is identical to `linrel`-function however it enables an asynchronous communication.

- `[st]=linpos_async(hnd,x,y,z,a,b,c,s,t)`
  Command is identical to `linpos`-function however it enables an asynchronous communication.

- `[st]=linrelpos_async(hnd,x,y,z,a,b,c,s,t)`
  Command is identical to `linrelpos`-function however it enables an asynchronous communication.

- `[st]=circ(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for moving the TCP along a circular path (circular-interpolation) from current position via a given intermediate position in cartesian coords (x,y,z,a,b,c) to a given target position in cartesian coords (x2,y2,z2,a2,b2,c2).

- `[st]=circrel(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a relative movement of the TCP along a circular path (circular-interpolation) from current position ABOUT a given intermediate position in relative cartesian coords (x,y,z,a,b,c) to a given target position in relative cartesian coords (x2,y2,z2,a2,b2,c2).

- `[st]=circpos(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command for moving the TCP along a circular path (circular-interpolation) from current position via a given intermediate position in cartesian coords (x,y,z,a,b,c,s,t) to a given target position in cartesian coords (x2,y2,z2,a2,b2,c2,s2,t2) while each position is extended by Statusbit (s,s2) and Turnbit (t,t2) to define an unambiguous robot position.

- `[st]=circrelpos(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command for a relative movement of the TCP along a circular path (circular-interpolation) from current position ABOUT a given intermediate position in relative cartesian coords (x,y,z,a,b,c,s,t) to a given target position in relative cartesian coords (x2,y2,z2,a2,b2,c2,s2,t2) while each position is extended by Statusbit (s,s2) and Turnbit (t,t2) to define an unambiguous robot position.

- `[st]=circangle(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circ`-function however with an angle as additional parameter `angle`. Now the target position of the circular path is determined by the value of this angle.

- `[st]=circrelangle(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circrel`-function however with an angle as additional parameter `angle`. Now the target position of the circular path is determined by the value of this angle.

- `[st]=circcdis(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a circular continuous-path motion with approximate positioning according to the cdis-criterion. The TCP moves along a circular path (circular-interpolation) from current position via a given intermediate position in cartesian coords (x,y,z,a,b,c) to a given target position in cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circcori(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a circular continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a circular path (circular-interpolation) from current position via a given intermediate position in cartesian coords (x,y,z,a,b,c) to a given target position in cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circcvel(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a circular continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a circular path (circular-interpolation) from current position via a given intermediate position in cartesian coords (x,y,z,a,b,c) to a given target position in cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circrelcdis(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a relative circular continuous-path motion with approximate positioning according to the cdis-criterion. The TCP moves along a circular path (circular-interpolation) from current position ABOUT a given intermediate position in relative cartesian coords (x,y,z,a,b,c) to a given target position in relative cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circrelcori(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a relative circular continuous-path motion with approximate positioning according to the cori-criterion. The TCP moves along a circular path (circular-interpolation) from current position ABOUT a given intermediate position in relative cartesian coords (x,y,z,a,b,c) to a given target position in relative cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circrelcvel(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command for a relative circular continuous-path motion with approximate positioning according to the cvel-criterion. The TCP moves along a circular path (circular-interpolation) from current position ABOUT a given intermediate position in relative cartesian coords (x,y,z,a,b,c) to a given target position in relative cartesian coords (x2,y2,z2,a2,b2,c2) approximately.

- `[st]=circanglecdis(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.

- `[st]=circanglecori(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.

- `[st]=circanglecvel(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.

- `[st]=circrelanglecdis(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circrelangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.

- `[st]=circrelanglecori(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circrelangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.
- `[st]=circrelanglecvel(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2,angle)`
  Command is identical to `circrelangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.
- `[st]=circposcdis(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.
- `[st]=circposcori(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.
- `[st]=circposcvel(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.
- `[st]=circrelposcdis(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circrelpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.
- `[st]=circrelposcori(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circrelpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.
- `[st]=circrelposcvel(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circrelpos`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.
- `[st]=circposanglecdis(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.
- `[st]=circposanglecori(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.
- `[st]=circposanglecvel(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.
- `[st]=circrelposanglecdis(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circrelposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cdis-criterion.
- `[st]=circrelposanglecori(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circrelposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cori-criterion.

- `[st]=circrelposanglecvel(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circrelposangle`-function however it enables a circular continuous-path motion with approximate positioning according to the cvel-criterion.

- `[st]=circposangle(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circpos`-function however with an angle as additional parameter `angle`. Now the target position of the circular path is determined by the value of this angle.

- `[st]=circrelposangle(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2,angle)`
  Command is identical to `circrelpos`-function however with an angle as additional parameter `angle`. Now the target position of the circular path is determined by the value of this angle.

- `[st]=circ_async(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command is identical to `circ`-function however it enables an asynchronous communication.

- `[st]=circrel_async(hnd,x,y,z,a,b,c,x2,y2,z2,a2,b2,c2)`
  Command is identical to `circrel`-function however it enables an asynchronous communication.

- `[st]=circpos_async(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circpos`-function however it enables an asynchronous communication.

- `[st]=circrelpos_async(hnd,x,y,z,a,b,c,s,t,x2,y2,z2,a2,b2,c2,s2,t2)`
  Command is identical to `circrelpos`-function however it enables an asynchronous communication.

3. Functions for setting parameters of robot:

- `[st]=setacceleration(hnd,a1,a2,a3,a4,a5,a6)`
  Command sets the maximum of axis-specific accelerations for each robot axis (a1...a6) given as percentages of the maximum value defined in the machine data. These accelerations relate to ptp-motions.

- `[st]=setaccelerationcp(hnd,value)`
  Command sets the path acceleration in m/s$\hat{2}$. The path acceleration relates to the motion of the TCP during a continuous-path motion.

- `[st]=setaccelerationori1(hnd,value)`
  Command sets the swivel acceleration in $^\circ/s^2$. The swivel acceleration relates to the motion of the TCP during a continuous-path motion

- `[st]=setaccelerationori2(hnd,value)`
  Command sets the rotational acceleration in $^\circ/s^2$. The rotational acceleration relates to the motion of the TCP during a continuous-path motion.

- `[st]=setbase(hnd,nr)`
  Command sets the BASE coordinate system (CS). If `nr=0`, then BASE-CS is equal to WORLD-CS. A predefined BASE-CS can be selected by parameter `nr`.

- `[st]=setbaseparam(hnd,nr,x,y,z,a,b,c)`
  Command defines/changes the BASE coordinate system (CS) in cartesian coords of the WORLD-CS. Parameter: IDnumber of BASE-CS (nr), point of origin in cartesian coords (x,y,z,a,b,c).

- `[st]=setipomode(hnd,mode)`
  Command sets the mode of interpolation of a path motion. `mode=1` enables a gripper-related interpolation in the TOOL coordinate system. `mode=0` enables a base-related interpolation in the BASE coordinate system.

- `[st]=setoritype(hnd,mode)`
  Command serves for changing the orientation of the tool during a path motion. `mode=1` enables a constant tool orientation during a path motion according to the initial orientation. `mode=0` enables a continuous orientation change from the initial orientation to the end orientation (variable orientation).

- `[st]=setcirctype(hnd,mode)`
  Command sets either a space-related orientation control (`mode=1`) or a path-related orientation control (`mode=0`) during the circular motion.

- `[st]=setcdis(hnd,value)`
  Command sets the translational distance criterion (cdis) in mm for continuous-path motions with approximate positioning.

- `[st]=setcori(hnd,value)`
  Command sets the orientation distance criterion (cori) in $^\circ$ for continuous-path motions with approximate positioning.

- `[st]=setcvel(hnd,value)`
  Command sets the velocity criterion (cvel) in $^\circ/s$ for continuous-path motions with approximate positioning.

- `[st]=setcptp(hnd,value)`
  Command sets the beginning of ptp-motions with approximate positioning. `value` is given in percent of the angles defined by `setapodisptp(...)`. The higher `value`, the higher is the approximate positioning.

- `[st]=setapodisptp(hnd,a1,a2,a3,a4,a5,a6)`
  Command sets the angles (a1...a6) for each robot axis that control the beginning of ptp-motions with approximate positioning. If a robot axis is below this angle the approximate positioning starts.

- `[st]=setspeed(hnd,va1,va2,va3,va4,va5,va6)`
  Command sets the maximum of axis-specific velocities for each robot axis (va1...va6) given as percentages of the maximum value defined in the machine data. These velocities relate to ptp-motions.

- `[st]=setspeedcp(hnd,value)`
  Command sets the path velocity in m/s. The path velocity relates to the motion of the TCP during a continuous-path motion.

- `[st]=setspeedori1(hnd,value)`
  Command sets the swivel velocity in °/s. The swivel velocity relates to the motion of the TCP during a continuous-path motion.

- `[st]=setspeedori2(hnd,value)`
  Command sets the rotational velocity in °/s. The rotational velocity relates to the motion of the TCP during a continuous-path motion.

- `[st]=settool(hnd,nr)`
  Command sets the TOOL coordinate system (CS). If `nr=0`, then TOOL-CS is equal to point of origin of the TCP. A predefined TOOL-CS can be selected by parameter `nr`.

- `[st]=settoolparam(hnd,nr,x,y,z,a,b,c)`
  Command defines/changes the TOOL coordinate system (CS) in cartesian coords of the robot CS. Parameter: IDnumber of TOOL-CS (nr), point of origin of tool in cartesian coords (x,y,z,a,b,c).

- `[st]=setoutput(hnd,nr)`
  Command sets the selected binary output (nr) to TRUE.

- `[st]=clearoutput(hnd,nr)`
  Command sets the selected binary output (nr) to FALSE.

- `[st]=setworkspace(hnd,nr,x,y,z,a,b,c,x1,y1,z1,x2,y2,z2,mode)`
  Commands sets a workspace for supervision by 2 points in cartesian coords.

- `[st]=setworkspacemode(hnd,nr,mode)`
  Command sets the mode of workspace supervision (mode) for a predefined workspace with IDnumber (nr).

4. Functions for getting parameters of robot:

- `[aa1,aa2,aa3,aa4,aa5,aa6]=infoacceleration(hnd)`
  Command returns the current axis-specific accelerations of each robot axis given as percentages of the maximum value defined in the machine data.

- `[acc]=infoaccelerationcp(hnd)`
  Command returns the current path acceleration in m/s^2. The path acceleration relates to the motion of the TCP during a continuous-path motion.

- `[acc]=infoaccelerationori1(hnd)`
  Command returns the current swivel acceleration in $°/s^2$. The swivel acceleration relates to the motion of the TCP during a continuous-path motion.

- `[acc]=infoaccelerationori2(hnd)`
  Command returns the current rotational acceleration in $°/s^2$. The rotational acceleration relates to the motion of the TCP during a continuous-path motion.

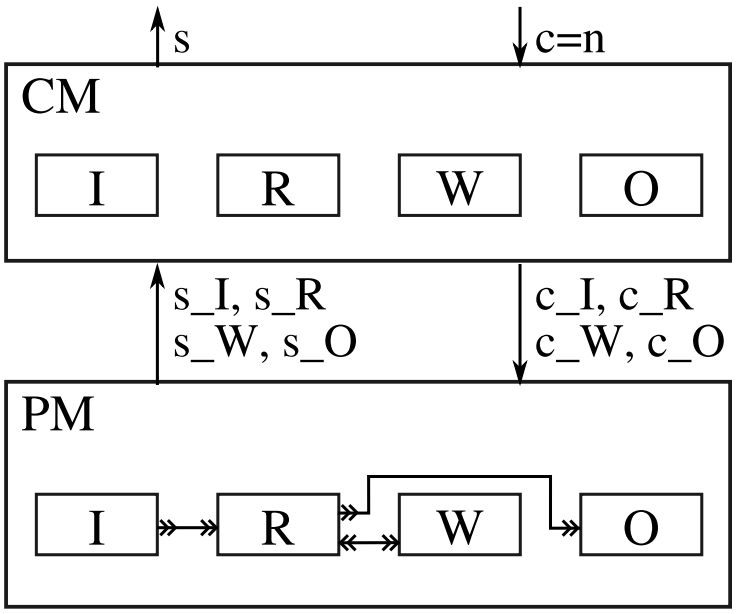
- `[x,y,z,a,b,c]=infobaseact(hnd)`
  Command returns the point of origin in cartesian coords of the current BASE coordinate system.

- `[x,y,z,a,b,c]=infobasenr(hnd,nr)`
  Command returns the point of origin in cartesian coords of a predefined BASE coordinate system with IDnumber (nr).
- `[x,y,z,a,b,c]=infoposition(hnd)`
  Command returns the current position of the TCP in cartesian coords of the BASE coordinate system.
- `[a1,a2,a3,a4,a5,a6]=infopositionaxis(hnd)`
  Command returns the joint angles of each robot axis.
- `[va1,va2,va3,va4,va5,va6]=infospeed(hnd)`
  Command returns the current axis-specific velocities of each robot axis given as percentages of the maximum value defined in the machine data.
- `[vel]=infospeedcp(hnd)`
  Command returns the current path velocity in m/s. The path velocity relates to the motion of the TCP during a continuous-path motion.
- `[vel]=infospeedori1(hnd)`
  Command returns the current swivel velocity in $^\circ/s$. The swivel velocity relates to the motion of the TCP during a continuous-path motion.
- `[vel]=infospeedori2(hnd)`
  Command returns the current rotational velocity in $^\circ/s$. The rotational velocity relates to the motion of the TCP during a continuous-path motion.
- `[x,y,z,a,b,c]=infotoolact(hnd)`
  Command returns the point of origin in cartesian coords of the current TOOL coordinate system.
- `[x,y,z,a,b,c]=infotoolnr(hnd,nr)`
  Command returns the point of origin in cartesian coords of a predefined TOOL coordinate system with IDnumber (nr).
- `[x,y,z,a,b,c,x1,y1,z1,x2,y2,z2,mode]=infoworkspacenr(hnd,nr)`
  Command returns the workspace-data of a predefined workspace with IDnumber (nr).
- `[mode]=infoworkspacenrstate(hnd,nr)`
  Command returns the supervision mode of a predefined workspace with IDnumber (nr).
- `[x,y,z,a,b,c]=infoposition_async(hnd)`
  Command is identical to `infoposition`-function however it enables an asynchronous communication. The command returns the current position of the TCP in cartesian coords of the BASE coordinate system during the motion is executed.

A.2 Prozesselementeorientierter Steuerungsentwurf für das materialtechnische Bearbeitungsproblem

A.2.1 Erster Schritt der Automatisierungsphase

Nach den Ergebnissen der Planungsphase (s. Abschnitt 5.1) basiert der Entwurf auf dem nachfolgend dargestellten Konzeptmodell.



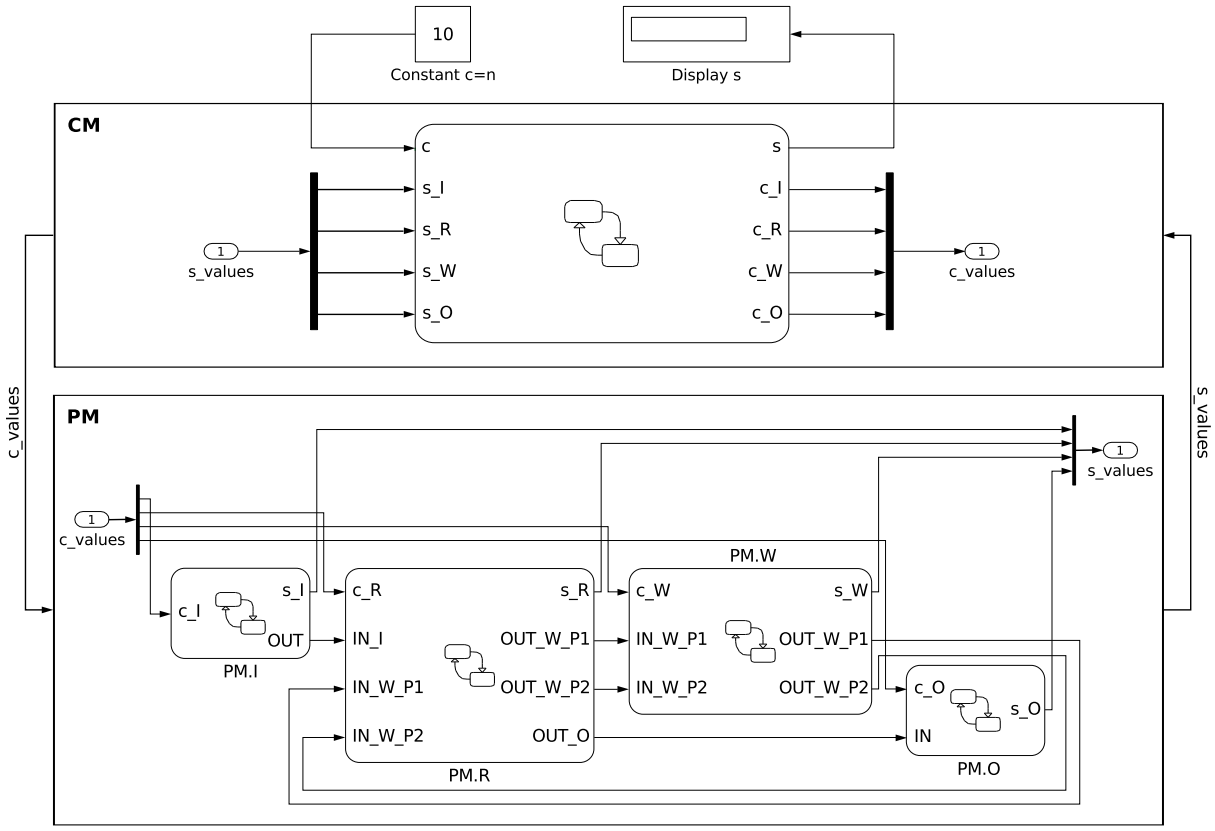
Die im Konzeptmodell vorgesehenen Steuergrößen sind in der nachfolgenden Tabelle aufgeführt.

Steuergröße	**Bedeutung**
c=n	• Ankunftssignal für eine neue Palette mit n Werstücken in I
c_I, mit:	• Anweisung an Eingangspuffer, mit:
c_I[1] = n	- Initialisierungssignal an I (Anzahl der Werkstücke in I wird auf n gesetzt)
c_I[2] = 1	- Signal für die Freigabe eines Werkstückes zur Entnahme
c_R, mit:	• Anweisung an Roboter, mit:
c_R = 12	- Signal für die Bestückung von P1 mit Werkstück aus I
c_R = 13	- Signal für die Bestückung von P2 mit Werkstück aus I
c_R = 24	- Signal für die Entstückung von P1 nach O
c_R = 34	- Signal für die Entstückung von P2 nach O
c_W, mit:	• Anweisung an Bearbeitungsstation, mit:
c_W[1] = 1	- Bearbeitungssignal für ein unbearbeitetes Werkstück auf P1
c_W[2] = 1	- Bearbeitungssignal für ein unbearbeitetes Werkstück auf P2
c_W[1] = 2	- Signal für die Freigabe eines bearbeiteten Werkstückes auf P1 zur Entnahme
c_W[2] = 2	- Signal für die Freigabe eines bearbeiteten Werkstückes auf P2 zur Entnahme
c_O	• Initialisierungsanweisung an Ausgangspuffer (Anzahl der Werkstücke in O wird auf 0 gesetzt)

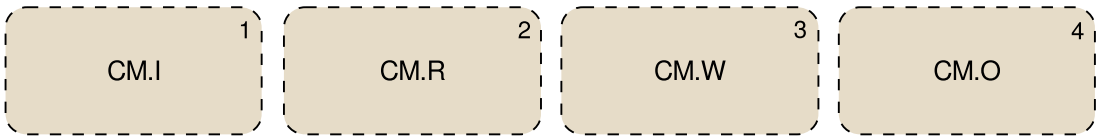
Die im Konzeptmodell vorgesehenen Zustandsgrößen sind in der nachfolgenden Tabelle aufgeführt.

Zustandsgröße	**Bedeutung**
s	• momentane Anzahl der bearbeiteten Werkstücke im Ausgangspuffer
s_I	• momentane Anzahl von Werkstücken im Eingangspuffer
s_R, mit: s_R = 0 s_R = 1	• Zustand des Roboters, mit: - R ist frei - R ist belegt
s_W, mit: s_W[1] = 0 s_W[2] = 0 s_W[1] = 1 s_W[2] = 1 s_W[1] = 2 s_W[2] = 2	• Zustand der Bearbeitungsstation, mit: - P1 auf W ist frei - P2 auf W ist frei - P1 auf W ist mit unbearbeitetem Werkstück belegt - P2 auf W ist mit unbearbeitetem Werkstück belegt - P1 auf W ist mit bearbeitetem Werkstück belegt - P2 auf W ist mit bearbeitetem Werkstück belegt
s_O	• Signal für die Ankunft eines Werkstückes im Ausgangspuffer

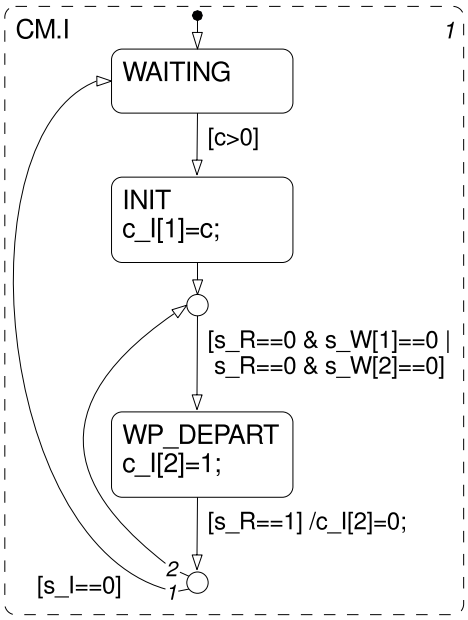
Oberste Ebene der Umsetzung des Konzeptmodells mit Simulink/Stateflow:



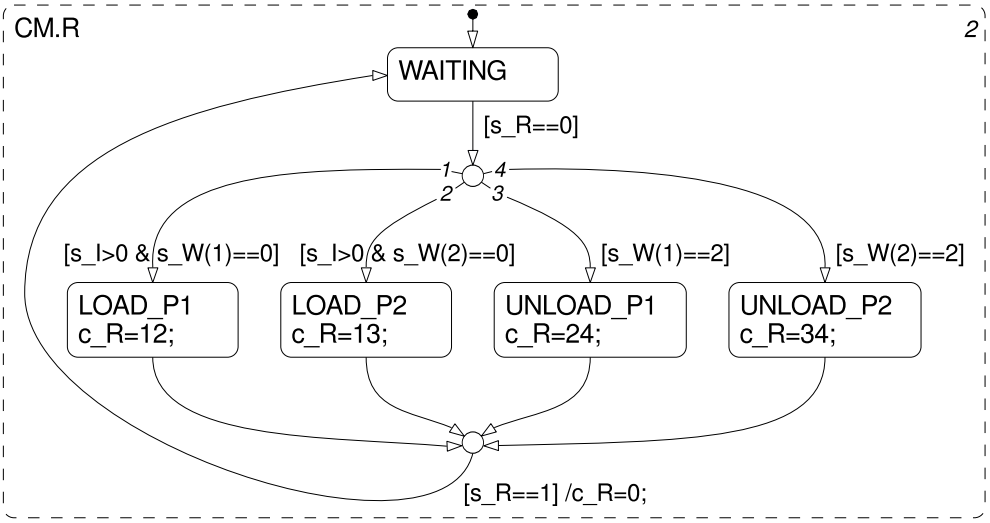
Umsetzung des prozesselementeorientierten Steuerungsentwurfs mittels paralleler Stateflow-Zustände:



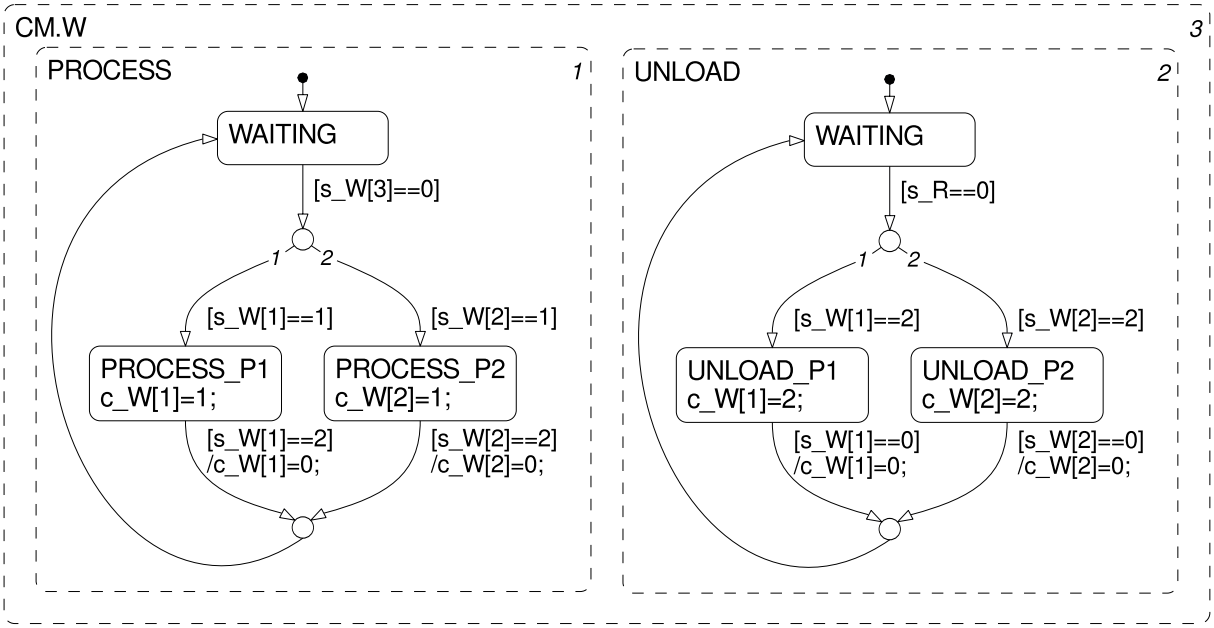
Paralleler Stateflow-Elternzustand CM.I mit Implementierung der Steuerungslogik des Eingangspuffers:



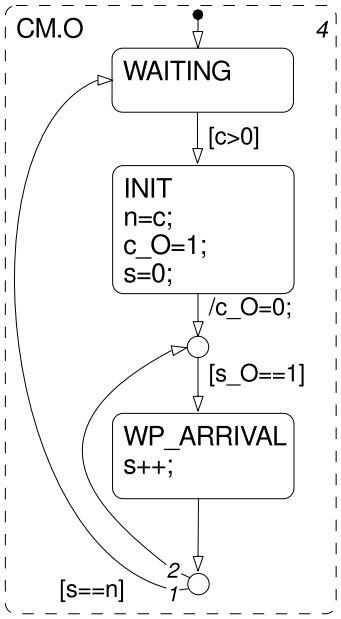
Paralleler Stateflow-Elternzustand CM.R mit Implementierung der Steuerungslogik des Roboters:



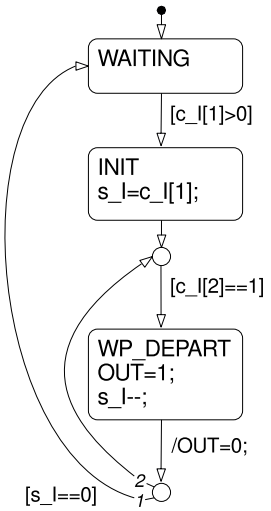
Paralleler Stateflow-Elternzustand CM.W mit Implementierung der Steuerungslogik der Bearbeitungsstation:



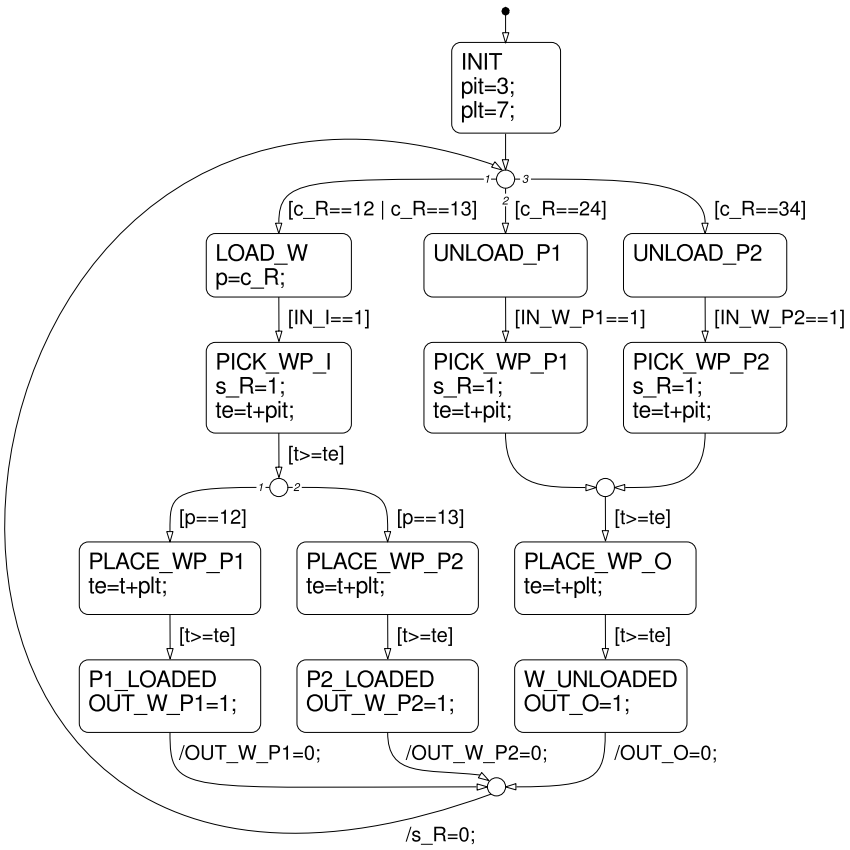
Paralleler Stateflow-Elternzustand CM.O mit Implementierung der Steuerungslogik des Ausgangspuffers:



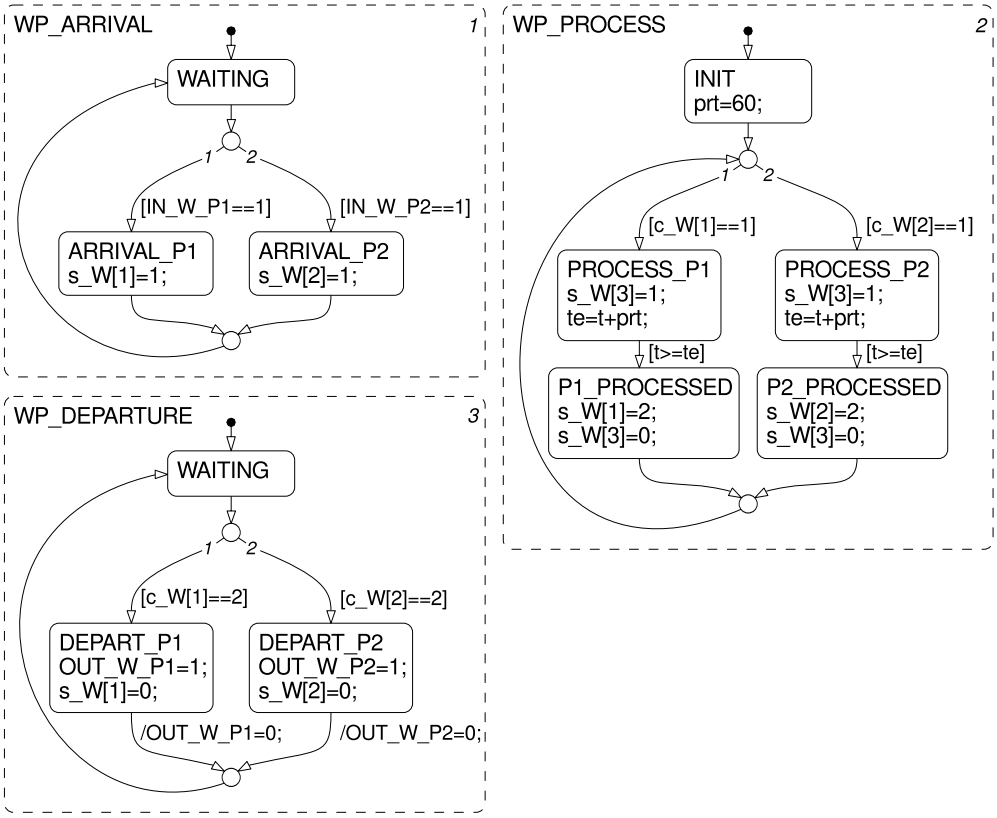
Stateflow-Diagramm PM.I zur Modellierung der Prozesskomponente Eingangspuffer:



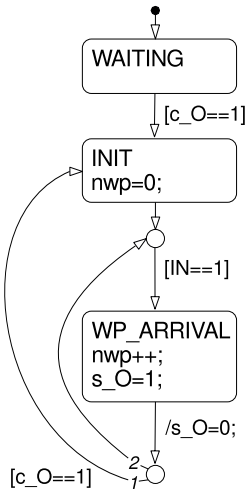
Stateflow-Diagramm PM.R zur Modellierung der Prozesskomponente Roboter:



Stateflow-Diagramm PM.W zur Modellierung der Prozesskomponente Bearbeitungsstation:

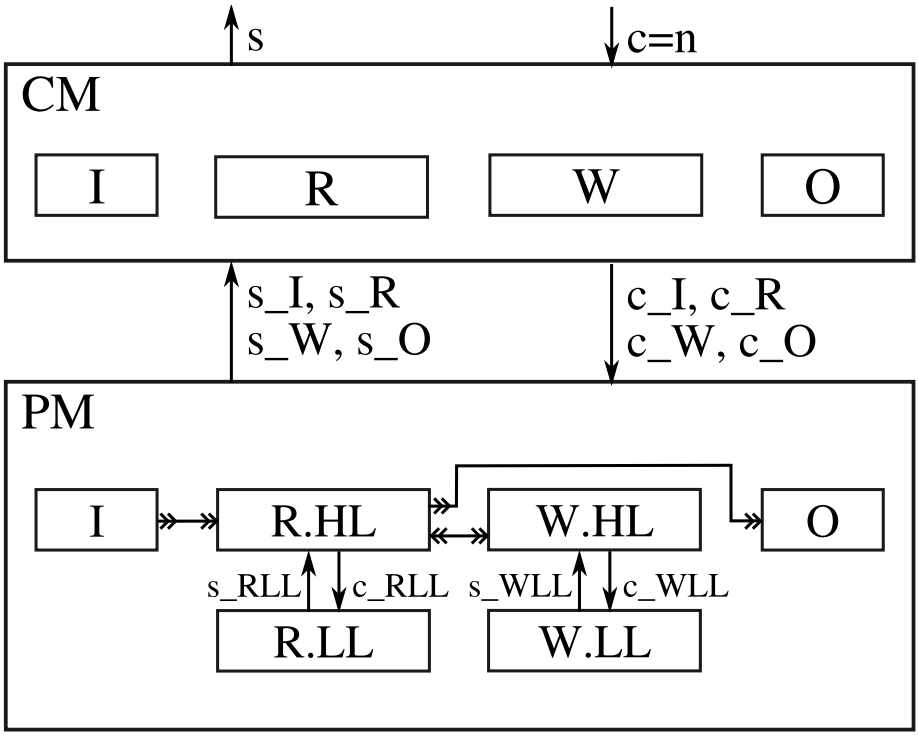


Stateflow-Diagramm PM.O zur Modellierung der Prozesskomponente Ausgangspuffer:



A.2.2 Zweiter Schritt der Automatisierungsphase

Gemäß dem SBC-Ansatz ist das generische Entwurfsmodell des ersten Schrittes der Automatisierungsphase mittels einer Low-Level-Komponente zur Beschreibung der aktiven herstellerspezifischen Prozesselemente bis auf das Niveau der konkreten Sensor-/Aktorebene zu detaillieren:

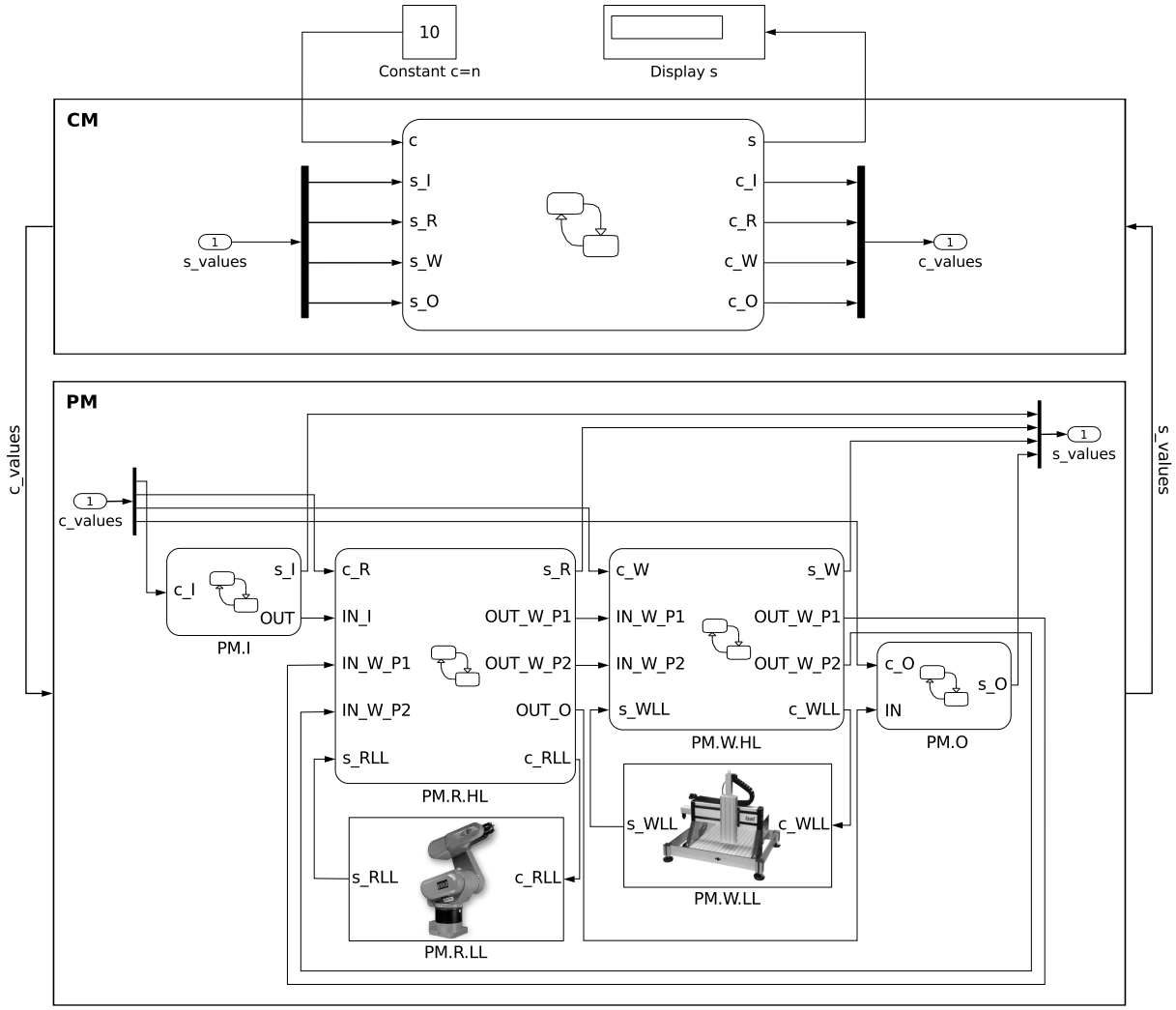


Zusätzliche Steuer- und Zustandsgrößen des detaillierten Konzeptmodells:

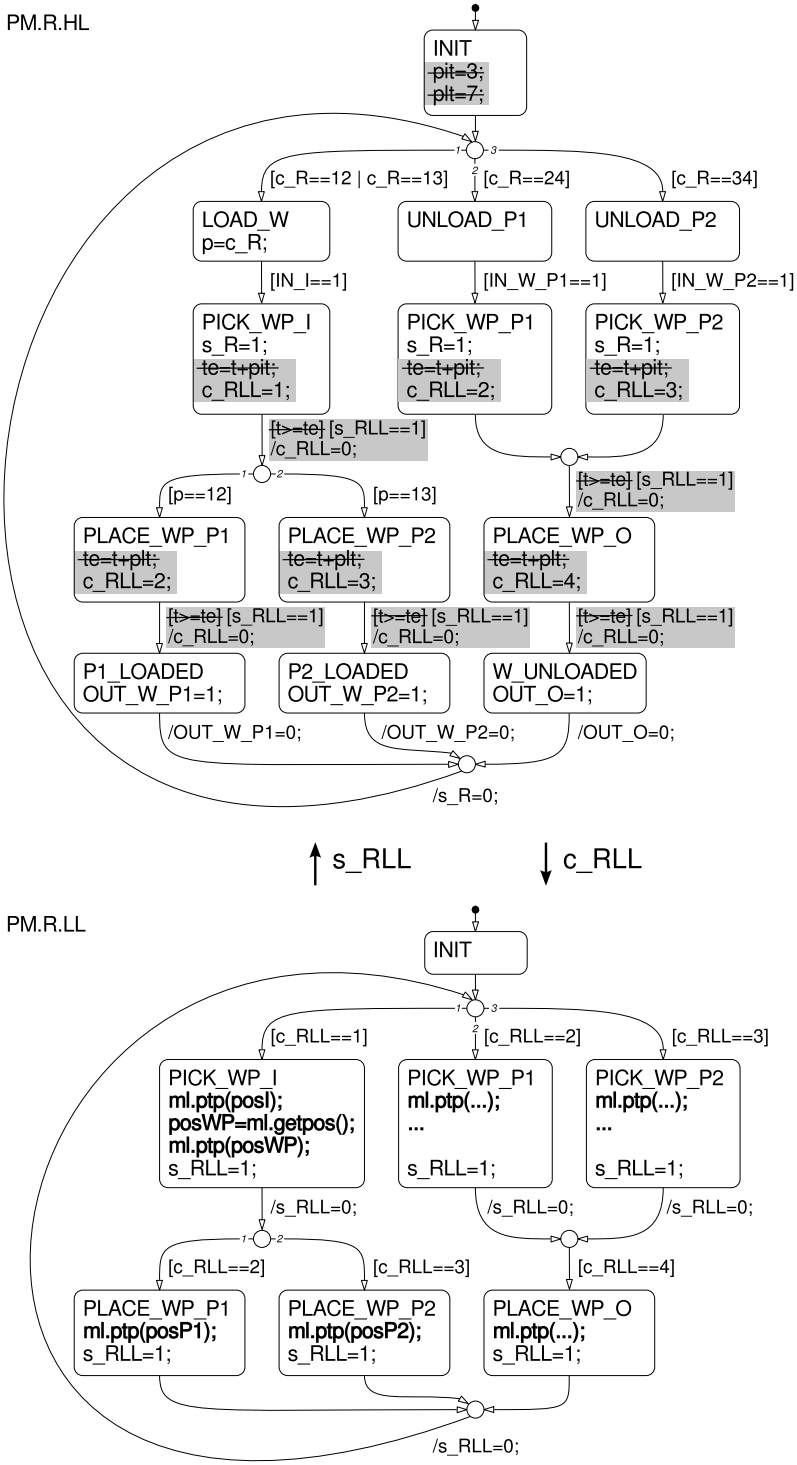
Steuergröße	**Bedeutung**
c_RLL, mit:	• Schnittstellengröße an Roboter KUKA KR3, mit:
c_RLL = 1	- Signal für die Entnahme eines Werkstückes aus I
c_RLL = 2	- Signal für die Bestückung/Entstückung von P1 in W
c_RLL = 3	- Signal für die Bestückung/Entstückung von P2 in W
c_RLL = 4	- Signal für die Bestückung von O
c_WLL, mit:	• Schnittstellengröße an Bearbeitungsstation isel C116 - 4, mit:
c_WLL[1] = 1	- Bearbeitungssignal für ein Werkstück auf P1
c_WLL[2] = 1	- Bearbeitungssignal für ein Werkstück auf P2

Zustandsgröße	**Bedeutung**
s_RLL, mit:	• Schnittstellengröße vom Roboter KUKA KR3, mit:
s_RLL = 1	- Zustandssignal über die erfolgreiche Ausführung einer Roboteraktion
s_WLL, mit:	• Schnittstellengröße der Bearbeitungsstation isel C116 - 4, mit:
s_WLL = 1	- Zustandssignal über die erfolgreiche Ausführung einer Bearbeitungsaktion

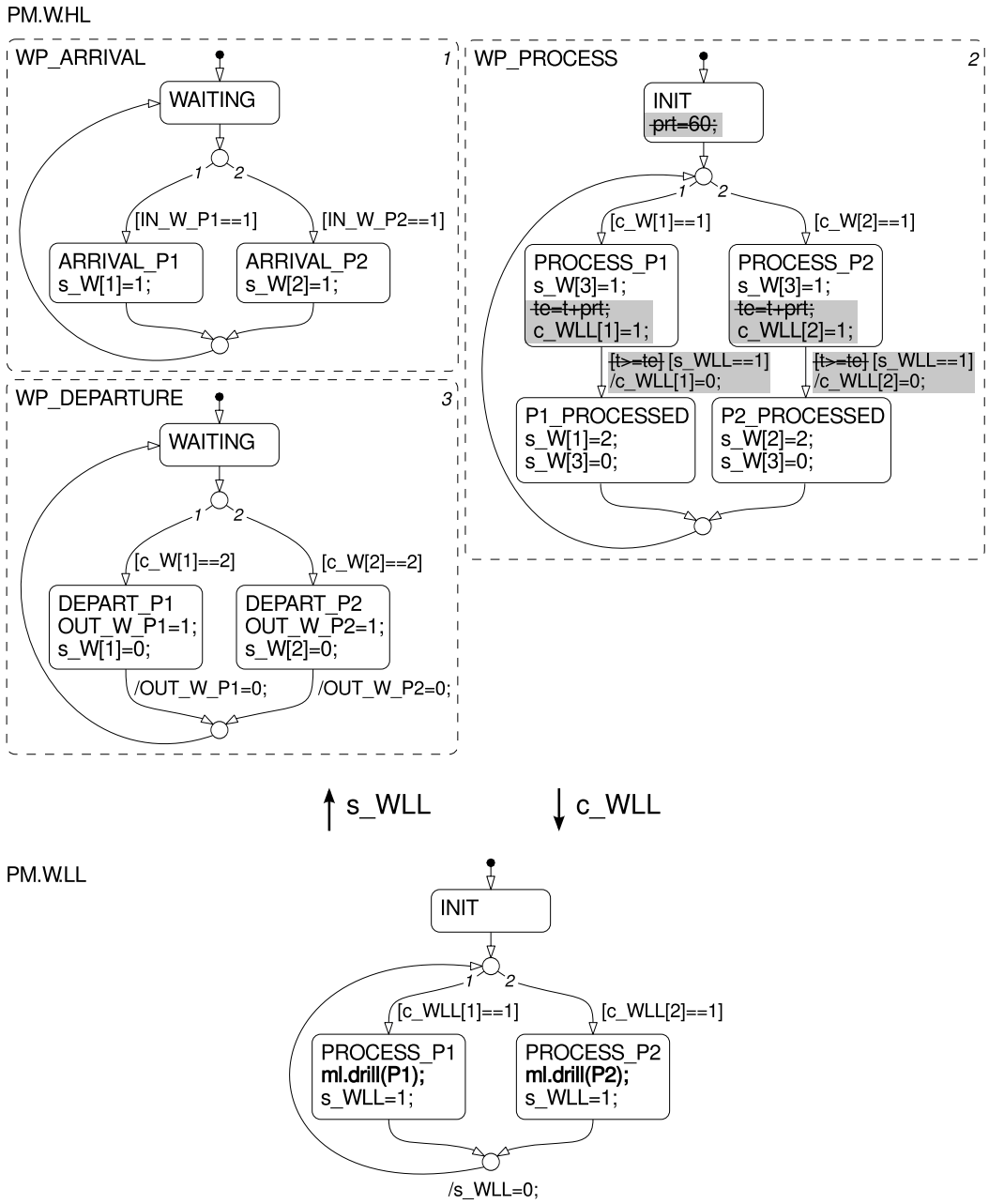
Oberste Ebene der Umsetzung des Konzeptmodells im zweiten Schritt der Automatisierungsphase mit Simulink/Stateflow:



Modifikation von PM.R zu PM.R.HL und Detaillierung um PM.R.LL:



Modifikation von PM.W zu PM.W.HL und Detaillierung um PM.W.LL:



A.2.3 Konkrete Abbildung auf das Prozess-Interface

In Abschnitt 5.3 als auch in A.2.2 wurden die Aufrufe an das Prozess-Interface von R und W nur in einem nicht-blockierenden Pseudocode dargestellt. Aufgrund der nicht vorhandenen Multi-Thread-Unterstützung in den derzeit verfügbaren Matlab-Versionen müssen die Prozess-Interface-Aufrufe für lauffähige Stateflow-Implementierungen aber nicht-blockierend erfolgen. Die detaillierten Substitutionen der Pseudocode-Passagen in den zuvor dargestellten Zuständen PM.R.LL und PM.W.LL sind nachfolgend aufgeführt.

PICK_WP_I
```
ml.ptp_async(ml.hndR,350,260,350,90,180,0);
posWP = ml.getpos();
ml.ptp_async(ml.hndR,posWP);
ml.closeGripper(ml.hndR);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PICK_WP_P1
```
ml.ptp_async(ml.hndR,-130,600,240,0,180,0);
ml.lin_async(ml.hndR,-130,600,215,0,180,0);
ml.closeGripper(ml.hndR);
ml.lin_async(ml.hndR,-130,600,240,0,180,0);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PICK_WP_P2
```
ml.ptp_async(ml.hndR,150,600,240,0,180,0);
ml.lin_async(ml.hndR,150,600,215,0,180,0);
ml.closeGripper(ml.hndR);
ml.lin_async(ml.hndR,150,600,240,0,180,0);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PLACE_WP_P1
```
ml.ptp_async(ml.hndR,-130,600,240,0,180,0);
ml.lin_async(ml.hndR,-130,600,215,0,180,0);
ml.openGripper(ml.hndR);
ml.lin_async(ml.hndR,-130,600,240,0,180,0);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PLACE_WP_P2
```
ml.ptp_async(ml.hndR,150,600,240,0,180,0);
ml.lin_async(ml.hndR,150,600,215,0,180,0);
ml.openGripper(ml.hndR);
ml.lin_async(ml.hndR,150,600,240,0,180,0);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PLACE_WP_O
```
ml.ptp_async(ml.hndR,470,-350,200,0,180,0);
ml.lin_async(ml.hndR,470,-350,150,0,180,0);
ml.openGripper(ml.hndR);
ml.lin_async(ml.hndR,470,-350,200,0,180,0);
ml.ptp_async(ml.hndR,350,260,380,90,180,0);
```

```
[ml.CheckPos_R(ml.hndR,350,260,380,90,180,0)==1]
/s_RLL=1;
```

PROCESS_P1
```
ml.move_async(ml.hndW,5,200,0);
ml.move_async(ml.hndW,5,200,60);
ml.move_async(ml.hndW,5,200,0);
ml.move_async(ml.hndW,5,0,0);
```

```
[ml.CheckPos_W(ml.hndW,5,0,0)==1]
/s_WLL=1;
```

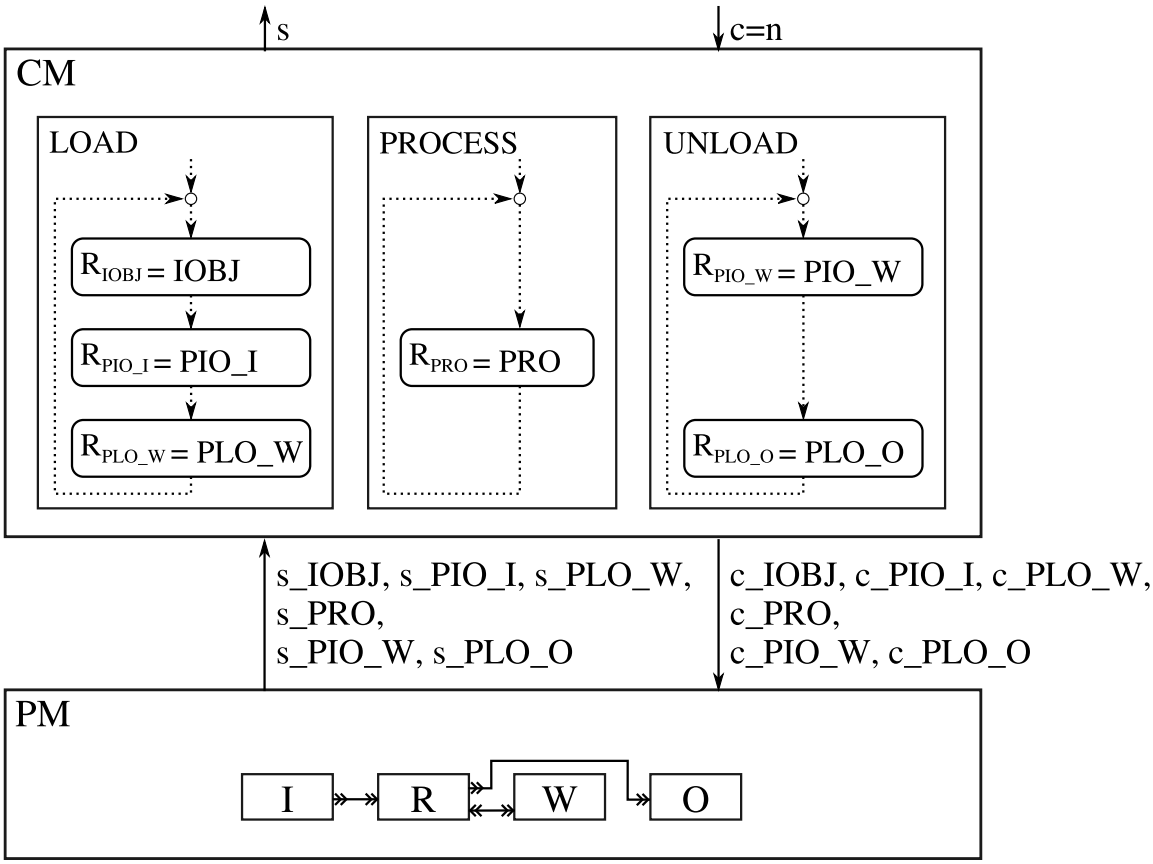
PROCESS_P2
```
ml.move_async(ml.hndW,290,200,0);
ml.move_async(ml.hndW,290,200,60);
ml.move_async(ml.hndW,290,200,0);
ml.move_async(ml.hndW,290,0,0);
```

```
[ml.CheckPos_W(ml.hndW,290,0,0)==1]
/s_WLL=1;
```

A.3 Aufgabenorientierter Steuerungsentwurf für das materialtechnische Bearbeitungsproblem

A.3.1 Erster Schritt der Automatisierungsphase

Konzeptmodell des aufgabenorientierten Steuerungsentwurfs:

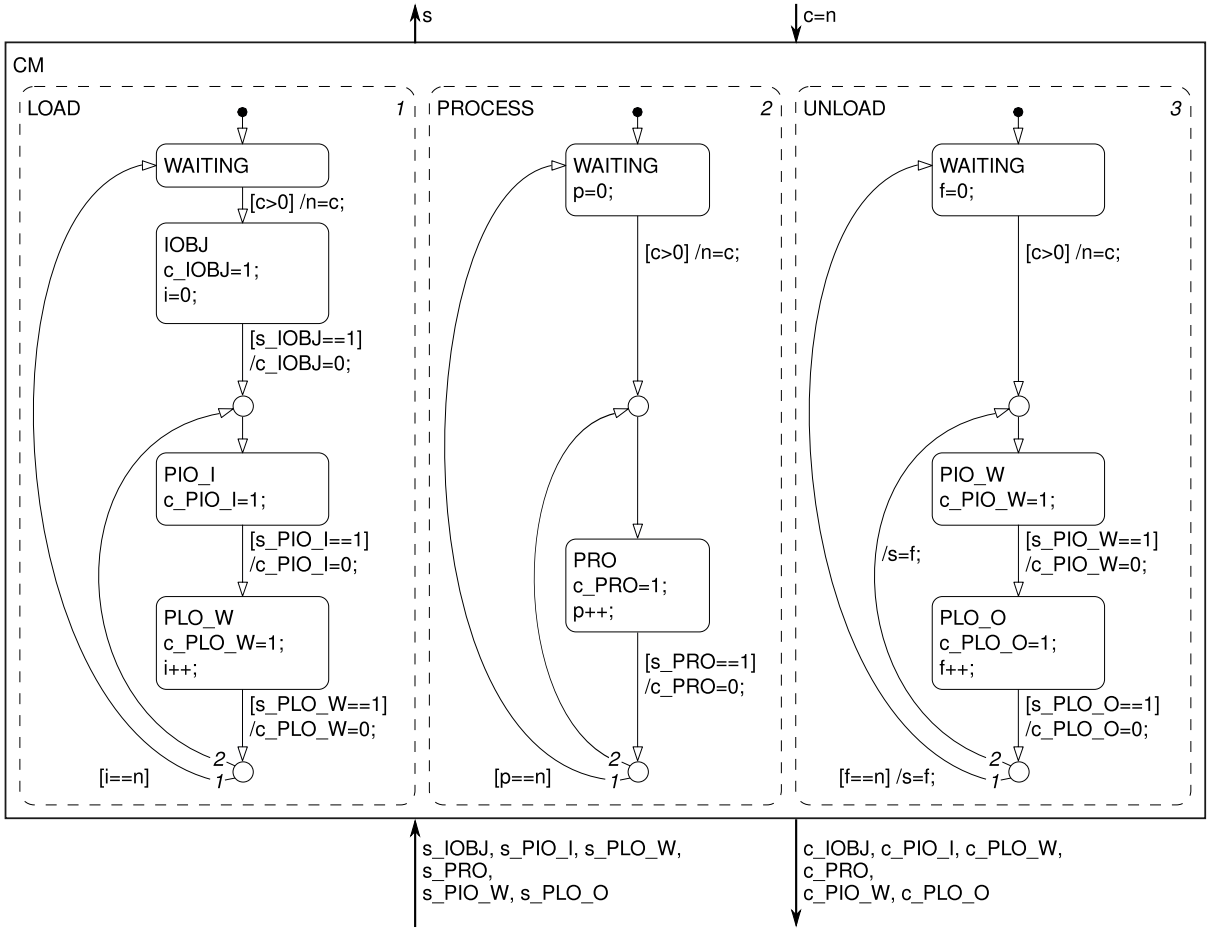


Die im Konzeptmodell vorgesehenen aufgabenorientierten Steuer- und Zustandsgrößen:

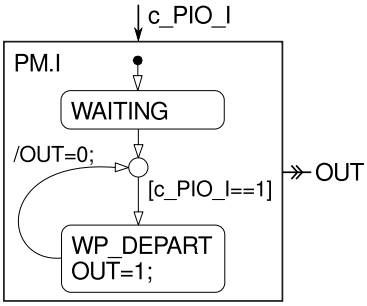
Steuergröße	**Bedeutung**
c=n	• Ankunftssignal für eine neue Palette mit n Werkstücken in I
c_IOBJ = 1	• Startsignal für die Ausführung der IOBJ - Aufgabe zur Identifikation und Positionsbestimmung eines Werkstückes in I
c_PIO_I = 1	• Startsignal für die Ausführung der PIO_I - Aufgabe zur Entnahme eines Werkstückes aus I
c_PLO_W = 1	• Startsignal für die Ausführung der PLO_W - Aufgabe zur Abgabe eines Werkstückes in W
c_PRO = 1	• Startsignal für die Ausführung der PRO - Aufgabe zur Bearbeitung eines Werkstückes in W
c_PIO_W = 1	• Startsignal für die Ausführung der PIO_W - Aufgabe zur Entnahme eines Werkstückes aus W
c_PLO_O = 1	• Startsignal für die Ausführung der PLO_O - Aufgabe zur Abgabe eines Werkstückes in O

Zustandsgröße	**Bedeutung**
s	• momentane Anzahl der bearbeiteten Werkstücke in O
s_IOBJ = 1	• Signal für die erfolgreiche Beendigung der IOBJ - Aufgabe
s_PIO_I = 1	• Signal für die erfolgreiche Beendigung der PIO_I - Aufgabe
s_PLO_W = 1	• Signal für die erfolgreiche Beendigung der PLO_W - Aufgabe
s_PRO = 1	• Signal für die erfolgreiche Beendigung der PRO - Aufgabe
s_PIO_W = 1	• Signal für die erfolgreiche Beendigung der PIO_W - Aufgabe
s_PLO_O = 1	• Signal für die erfolgreiche Beendigung der PLO_O - Aufgabe

Umsetzung des Steuerungsmodells als Zustandsdiagramm:



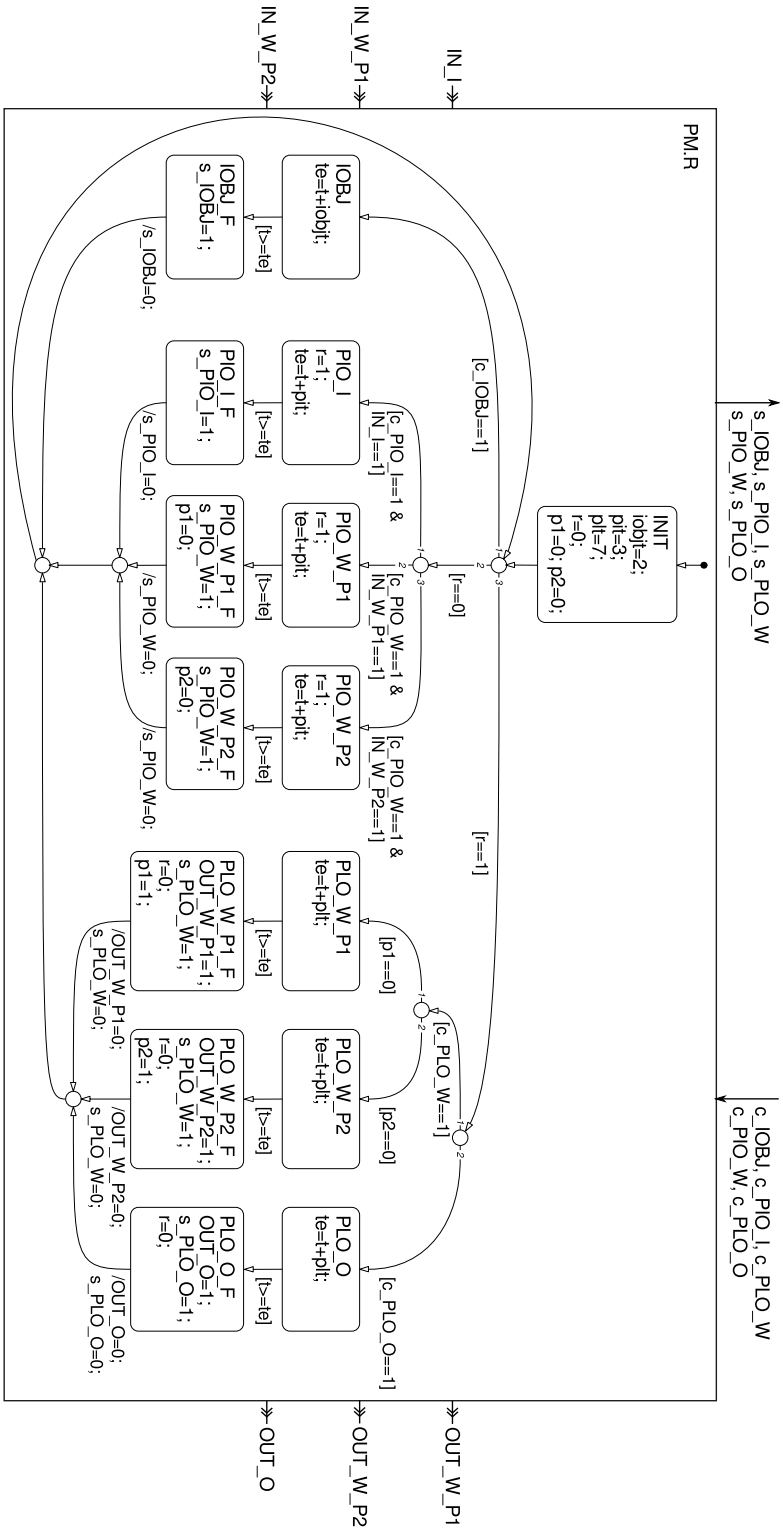
Zustandsdiagramm PM.I zur Modellierung der Prozesskomponente Eingangspuffer:



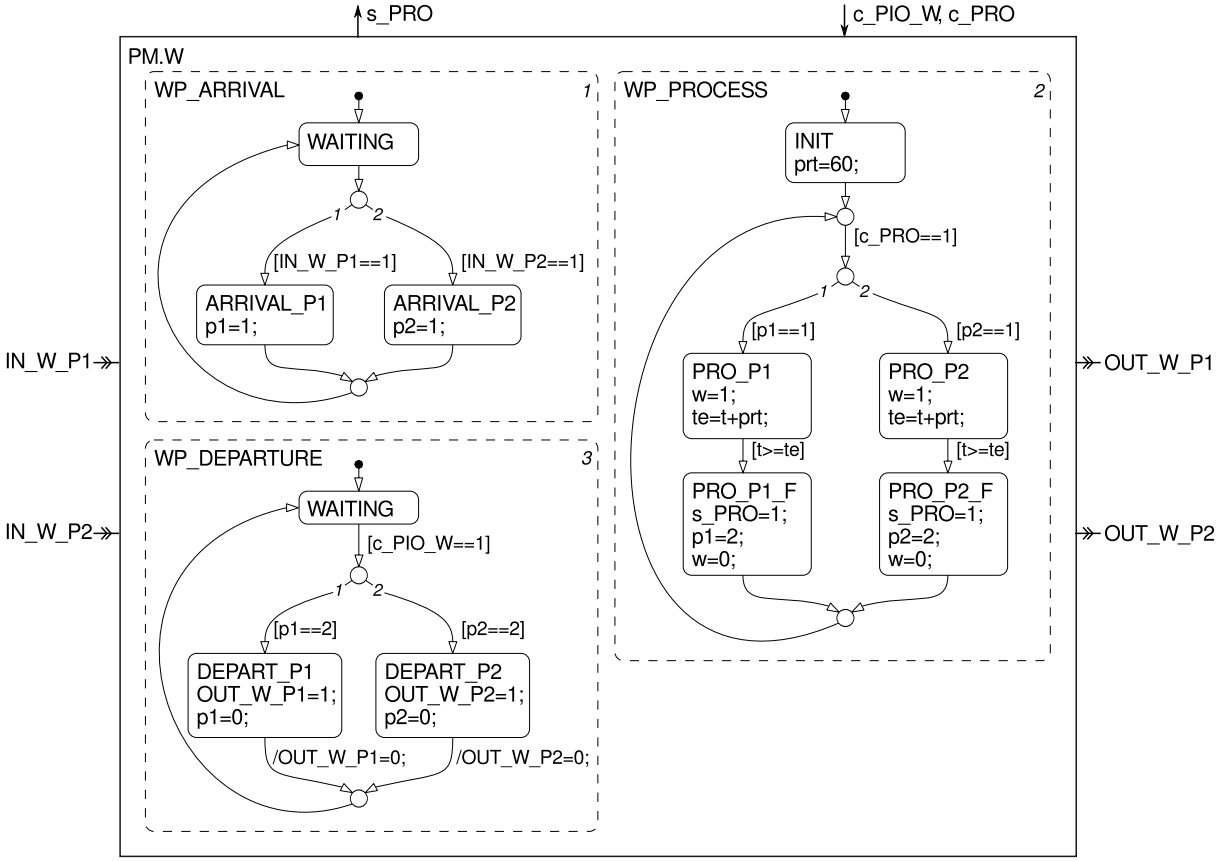
Die im Prozessmodell in den Zustandsdiagrammen PM.R und PM.W vorgesehenen Größen zur Weltmodellierung:

Größe	**Bedeutung**
r, mit:	• Belegungszustand des Roboters, mit:
r = 0	- R ist frei
r = 1	- R ist belegt
p1, mit:	• Belegungszustand P1 auf W, mit:
p1 = 0	- P1 auf W ist frei
p1 = 1	- P1 auf W ist mit unbearbeitetem Werkstück belegt
p1 = 2	- P1 auf W ist mit bearbeitetem Werkstück belegt
p2, mit:	• Belegungszustand P2 auf W, mit:
p2 = 0	- P2 auf W ist frei
p2 = 1	- P2 auf W ist mit unbearbeitetem Werkstück belegt
p2 = 2	- P2 auf W ist mit bearbeitetem Werkstück belegt

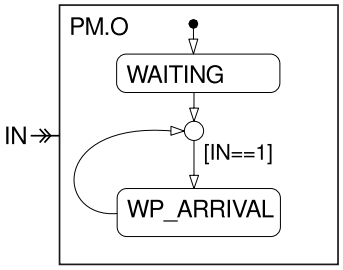
Zustandsdiagramm PM.R zur Modellierung der Prozesskomponente Roboter:



Zustandsdiagramm PM.W zur Modellierung der Prozesskomponente Bearbeitungsstation:

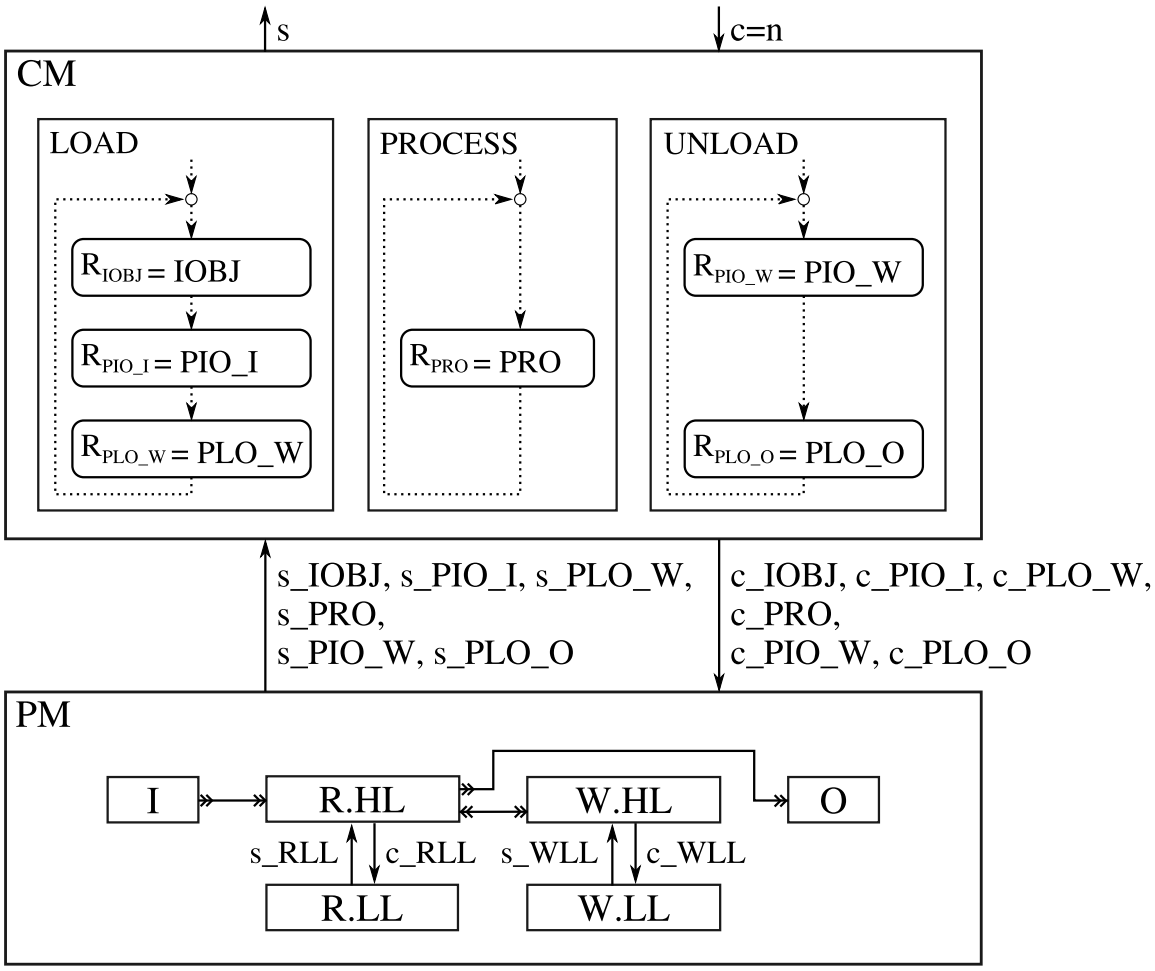


Zustandsdiagramm PM.O zur Modellierung der Prozesskomponente Ausgangspuffer:



A.3.2 Zweiter Schritt der Automatisierungsphase

Gemäß dem SBC-Ansatz ist das generische Entwurfsmodell des ersten Schrittes der Automatisierungsphase mittels einer Low-Level-Komponente zur Beschreibung der aktiven herstellerspezifischen Prozesselemente bis auf das Niveau der konkreten Sensor-/Aktorebene zu detaillieren:

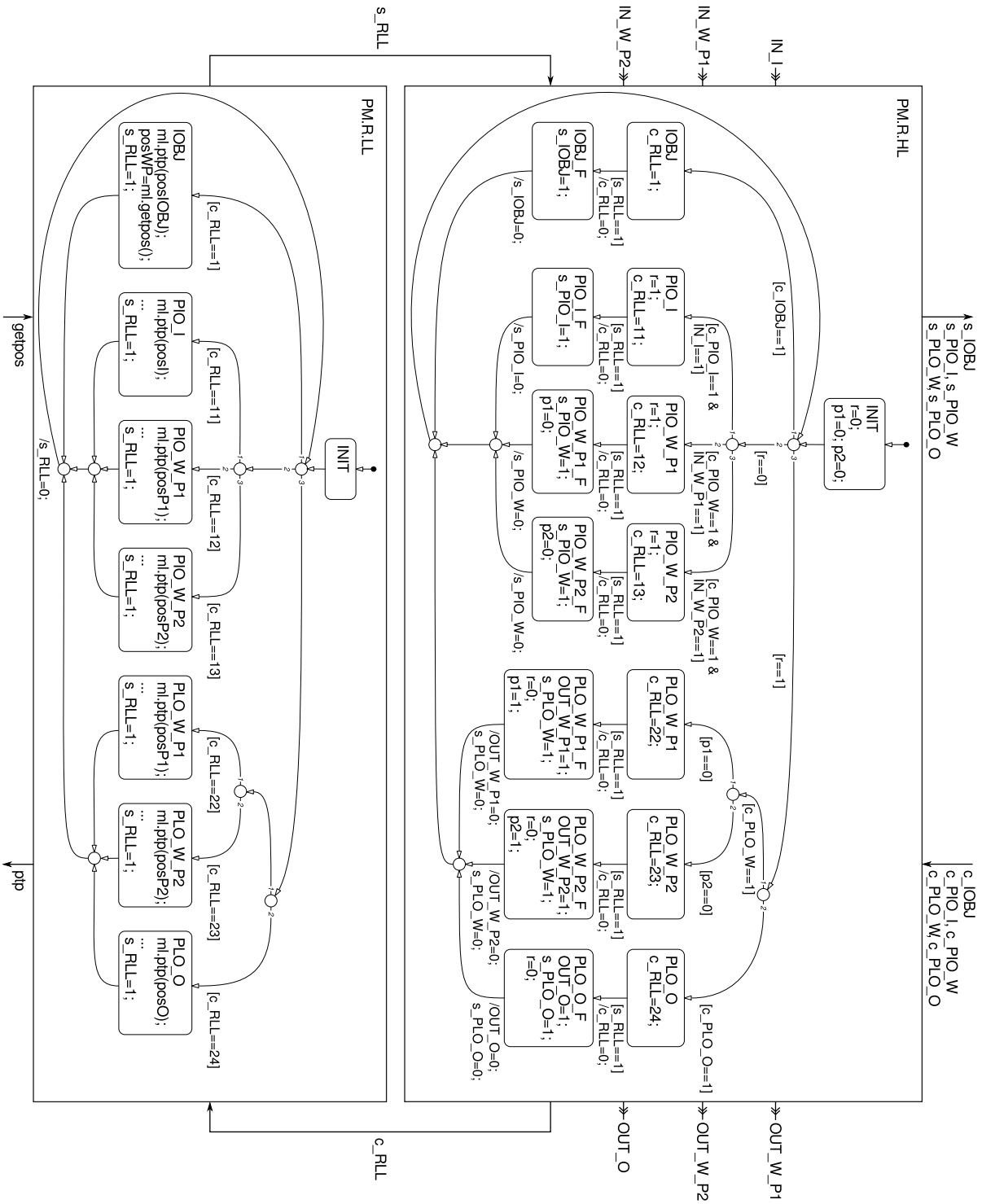


Zusätzliche Steuer- und Zustandsgrößen des detaillierten Konzeptmodells:

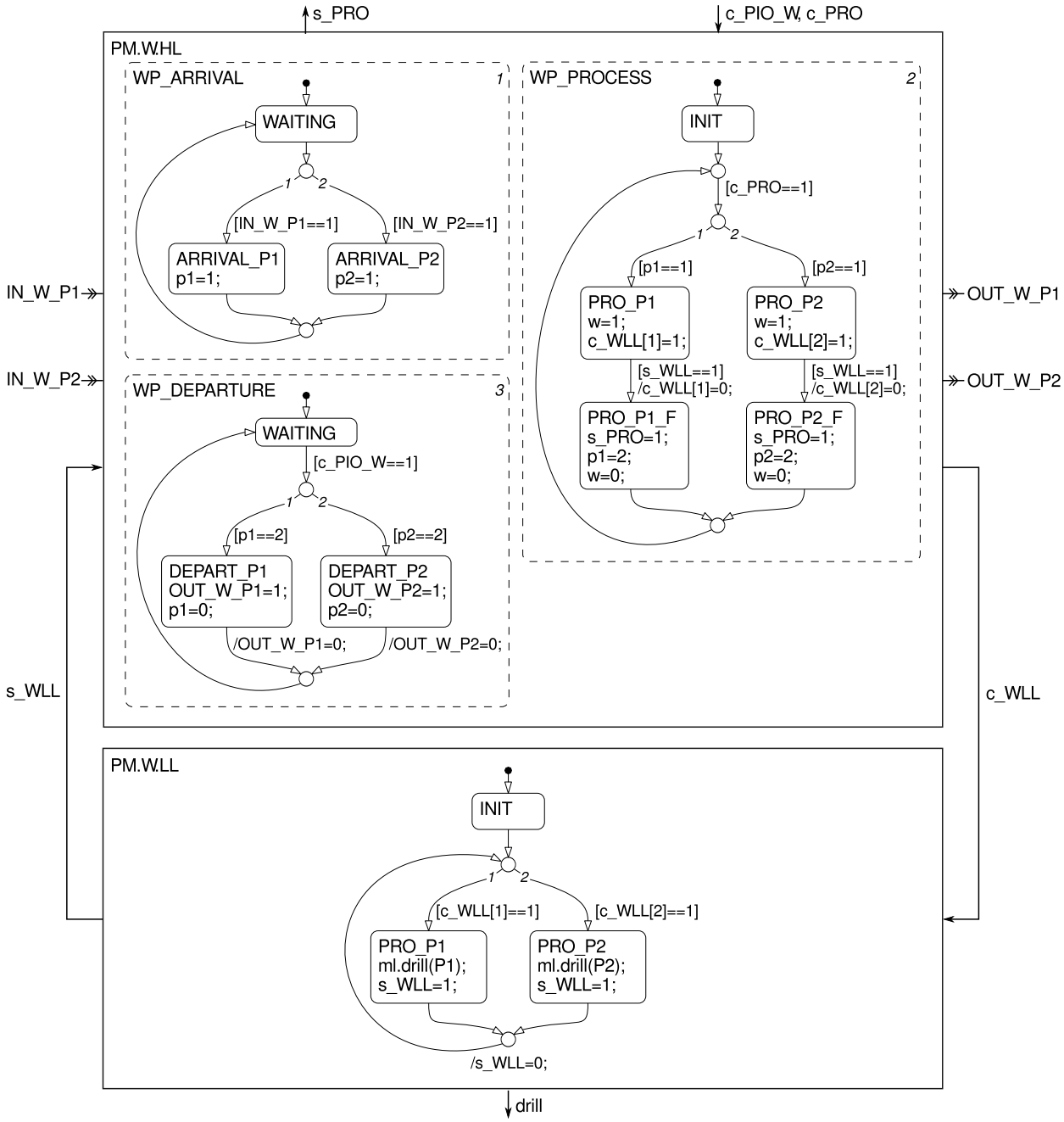
Steuergröße	**Bedeutung**
c_RLL, mit:	• Schnittstellengröße an Roboter KUKA KR3, mit:
c_RLL = 1	- Signal für die Identifikation und Positionsbestimmung eines Werkstückes in I
c_RLL = 11	- Signal für die Entnahme eines Werkstückes aus I
c_RLL = 12	- Signal für die Entstückung von P1 in W
c_RLL = 13	- Signal für die Entstückung von P2 in W
c_RLL = 22	- Signal für die Bestückung von P1 in W
c_RLL = 23	- Signal für die Bestückung von P2 in W
c_RLL = 24	- Signal für die Bestückung von O
c_WLL, mit:	• Schnittstellengröße an Bearbeitungsstation isel C116 - 4, mit:
c_WLL[1] = 1	- Bearbeitungssignal für ein Werkstück auf P1
c_WLL[2] = 1	- Bearbeitungssignal für ein Werkstück auf P2

Zustandsgröße	**Bedeutung**
s_RLL, mit:	• Schnittstellengröße vom Roboter KUKA KR3, mit:
s_RLL = 1	- Zustandssignal über die erfolgreiche Ausführung einer Roboteraktion
s_WLL, mit:	• Schnittstellengröße der Bearbeitungsstation isel C116 - 4, mit:
s_WLL = 1	- Zustandssignal über die erfolgreiche Ausführung einer Bearbeitungsaktion

Detaillierung von PM.R in PM.R.HL und PM.R.LL:

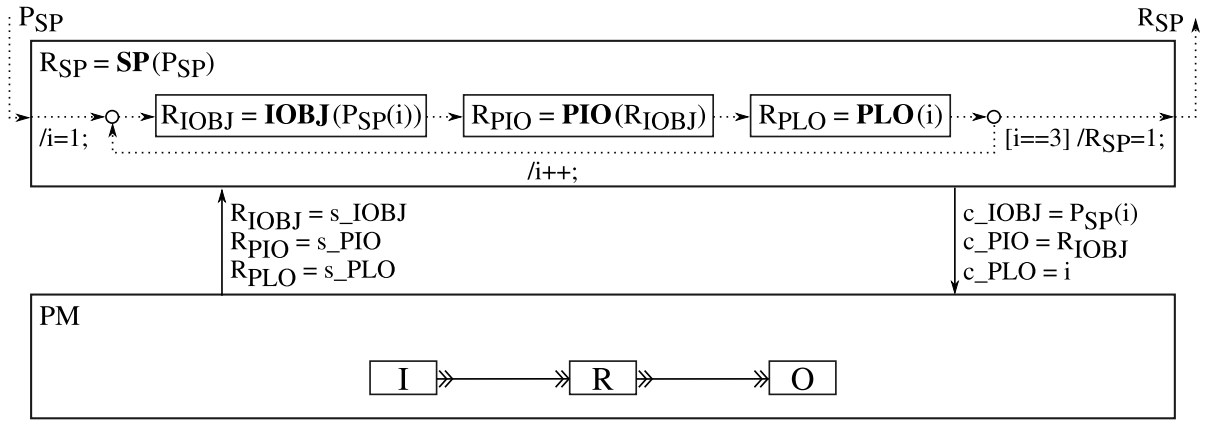


Detaillierung von PM.W in PM.W.HL und PM.W.LL:



A.4 Aufgabenorientierter Steuerungsentwurf für das Bauteilsortierungsproblem

Konzeptmodell des aufgabenorientierten Steuerungsentwurfs mit Aufgabenparametrierung:

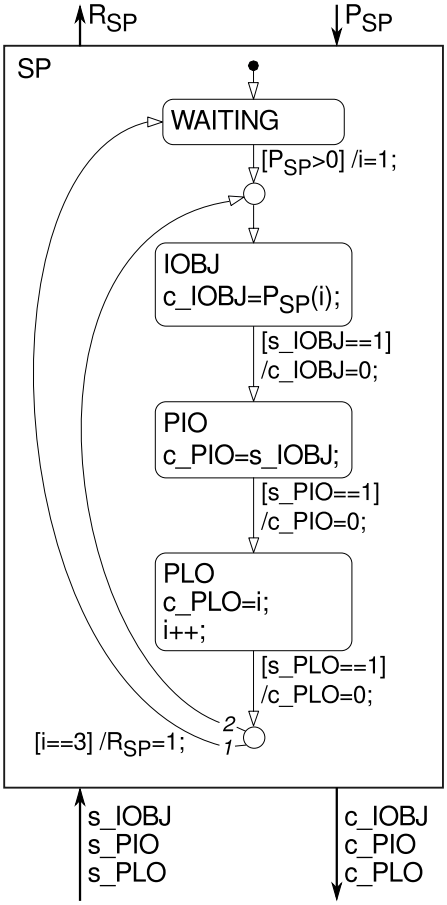


Die im Konzeptmodell vorgesehenen aufgabenorientierten Steuer- und Zustandsgrößen:

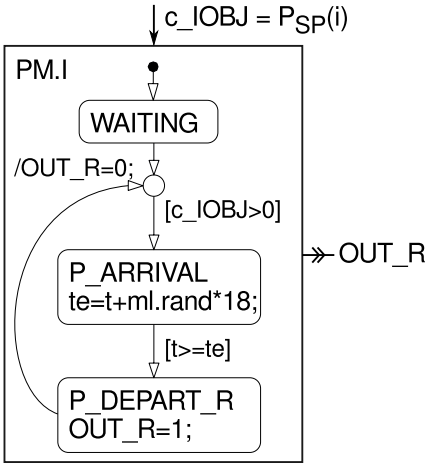
Steuergröße	**Bedeutung**
P_{SP}	• Einzelteildefinitionen des Sortierauftrages
c_IOBJ = $P_{SP}(i)$	• Startsignal für die Ausführung der IOBJ - Aufgabe zur Identifikation und Positionsbestimmung eines durch P_{SP} vorgegebenen Bauteiltyps in I
c_PIO = R_{IOBJ}	• Startsignal für die Ausführung der PIO - Aufgabe zur Entnahme eines Bauteiles aus I; die notwendigen Detailinformationen zur Bauteilaufnahme liefert das Ergebnis R_{IOBJ} einer zuvor ausgeführten IOBJ - Aufgabe
c_PLO = i	• Startsignal für die Ausführung der PLO - Aufgabe zur Abgabe eines Bauteiles in den Ausgangspufferbereich i

Zustandsgröße	**Bedeutung**
R_{SP}	• Signal für die erfolgreiche Beendigung der Sortieraufgabe
s_IOBJ = R_{IOBJ}	• Signal für die erfolgreiche Beendigung der IOBJ - Aufgabe mit zusätzlichen Detailinformationen zum identifizierten Bauteil
s_PIO = R_{PIO}	• Signal für die erfolgreiche Beendigung der PIO - Aufgabe
s_PLO = R_{PLO}	• Signal für die erfolgreiche Beendigung der PLO - Aufgabe

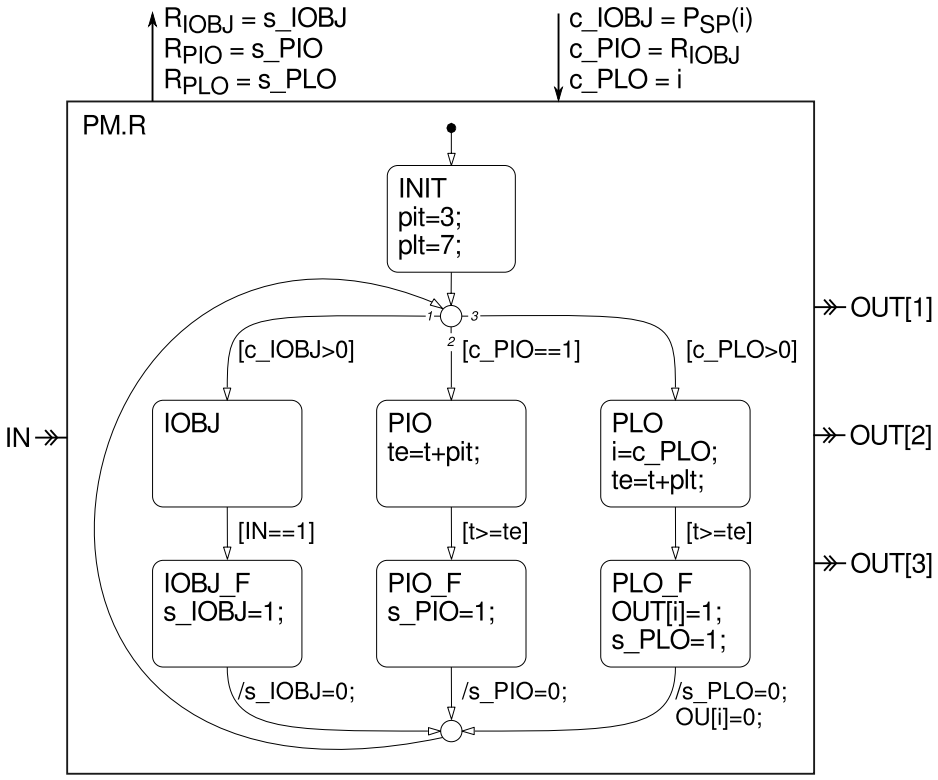
Umsetzung des Steuerungsmodells $R_{SP} = SP(P_{SP})$ als Zustandsdiagramm:



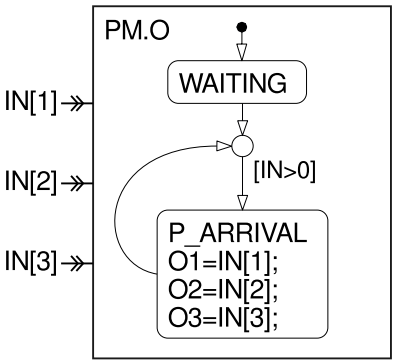
Zustandsdiagramm PM.I zur Modellierung der Prozesskomponente Eingangspuffer:



Zustandsdiagramm PM.R zur Modellierung der Prozesskomponente Roboter:



Zustandsdiagramm PM.O zur Modellierung der Prozesskomponente Ausgangspuffer:



Thesen

1. Aus steuerungstechnischer Sicht ist es sinnvoll, industrielle Roboteranwendungen in die Kategorien: *einfach*, *komplex* und *flexibel* einzuteilen.

2. Für die Steuerungsentwicklung einfacher Roboteranwendungen ist die Online-Programmierung dominierend. Für die Entwicklung komplexer und flexibler Roboteranwendungen wird dagegen die Offline-Programmierung eingesetzt.

3. Komplexe Roboteranwendungen stellen anspruchsvolle Entwurfs- und Inbetriebnahmeprobleme dar, die einen systematischen Entwicklungsprozess erfordern.

   Aus Effizienzgründen sollte dieser den Forderungen des *Rapid Control Prototypings (RCP)* genügen.

   Die Steuerungsentwicklung in der industriellen Robotik erfüllt diese Forderungen derzeit nicht.

4. Der Ansatz des *Simulation Based Control (SBC)* stellt einen RCP-Entwicklungsprozess für Industrieroboteranwendungen bereit.

5. Ein Entwicklungsprozess nach dem SBC-Ansatz ist auf der Softwareplattform Matlab realisierbar.

6. Im Gegensatz zum prozesselementeorientierten Entwurf ist die Aufgabenorientierung ein abstrakter Ansatz der Steuerungsentwicklung.

7. Eine Alternative zum Top-Down-Entwurf stellt eine nach dem Bottom-Up-Prinzip entworfene aufgabenorientierte Steuerung dar.

8. Für die Spezifikation von flexiblen Steuerungen ist als Ausgangsbasis das Konzept der Aufgabenorientierung geeignet.

9. Zur Realisierung flexibler Steuerungen werden in der Literatur vielfältige Vorschläge gemacht. Diese basieren auf einer gemeinsamen Grundstruktur und können unter dem Begriff der *adaptiven Steuerungen* zusammengefasst werden.

10. Bei adaptiven Steuerungen ist die *Entscheidungseinheit (EE)* von zentraler Bedeutung.

11. Das aus der Simulationstechnik bekannte *System Entity Structure and Model Base Framework (SES/MB)* nach Zeigler ist ein geeignetes Mittel zur Realisierung wissensbasierter Entscheidungseinheiten.

12. Zur Verringerung des Programmieraufwandes ist es vorteilhaft, alle prinzipiell möglichen Steuerungsvarianten einer flexiblen Steuerung in einer SES-ähnlichen Baumstruktur, der sogenannten *Control Specification (CS)*, abzubilden.

13. Die Generierung einer ausführbaren Steuerung setzt voraus, dass aus allen möglichen Steuerungsvarianten gezielt eine vorübergehend gültige Steuerung ausgewählt werden kann. Die dafür erforderlichen Auswahlbedingungen sind in der CS zu definieren.