Software Engineering in der Industrierobotik: der SoftRobot-Ansatz

Alwin Hoffmann, Andreas Angerer, Andreas Schierl, Michael Vistein, Wolfgang Reif
Institut für Software & Systems Engineering
Universität Augsburg
Universitätsstr. 6a
86159 Augsburg
E-Mail: {*hoffmann, angerer, schierl, vistein, reif*}@informatik.uni-augsburg.de

Abstract: Die Industrierobotik ist geprägt von technisch ausgereiften mechanischen Komponenten und hoch entwickelten Regelungsalgorithmen. Der produktive Einsatz von Robotersystemen ist jedoch, insbesondere in Hinblick auf Flexibilität, Wiederverwendbarkeit und Erweiterbarkeit, stark durch existierende Programmierkonzepte eingeschränkt. Dementsprechend ist die Softwareentwicklung in der Industrierobotik mit hohen Kosten und enormem Aufwand verbunden. Diese Arbeit analysiert die Ursachen aus der Sicht des Software Engineering und skizziert einen Ansatz, der diese Probleme adressiert und eine Lösung für effiziente Softwareentwicklung in der Industrierobotik anbietet.

Stichworte: Industrierobotik, Softwareentwicklung, Robotersteuerung

1 Einleitung

Verglichen mit traditionellen Werkzeugmaschinen sind Industrieroboter durch ihre mechanische Struktur darauf ausgelegt, möglichst flexibel und anpassungsfähig zu sein. Gemäß [ISO8373] ist ein Industrieroboter definiert als ein „automatisch gesteuerter, frei programmierbarer Mehrzweck-Manipulator [...] zur Verwendung in der Automatisierungstechnik". Dementsprechend können Roboter, ausgestattet mit passenden Werkzeugen, eine Vielzahl unterschiedlichster Arbeiten ausführen. Wenn man jedoch in die Unternehmen blickt, so beschränkt sich der praktische Einsatz von Robotern meist auf Massenfertigung und einfache, sich wiederholende Aufgaben. Eine wichtige Ursache dafür liegt in deren Programmierung. Bereits 1990 identifizierten Miller und Lennox [MiLe1990] die zugrundeliegenden Probleme, die auch knapp zwei Jahrzehnte nach Erscheinen des Artikels noch immer aktuell sind. Dazu gehören die hohen Kosten der Anwendungsentwicklung, die geringe Wiederverwendbarkeit von Roboterquellcode und die Schwierigkeit, bestehende Anwendungen um neue Aufgaben und Geräte zu erweitern. Infolgedessen ist der Einsatz von Industrierobotern in der Regel mit hohem Aufwand und hohen Kosten verbunden.

Betrachtet man den aktuellen Stand der Softwareentwicklung in der Industrierobotik, muss man zwischen industrieller Praxis und Forschung unterschieden. Die Forschung konzentriert sich vor allem auf die experimentelle Robotik [Bru2007] und betrachtet die Industrierobotik kaum, da die Probleme dort als gelöst betrachtet werden [HNP2008]. Außerdem werden in der experimentellen Robotik hauptsächlich Einzellösungen entwickelt, bei denen kein großer Wert auf Wiederverwendbarkeit gelegt wird, da sich die verfügbare Technologie und damit die

möglichen Lösungswege schnell ändern [BruPra2009]. Es gibt aber auch Versuche, Ansätze der Softwaretechnik insbesondere aus dem Bereich der Modellgetriebenen Entwicklung (vgl. [ICO2009]) oder der Formalen Methoden (vgl. [BGI2009]) in die Robotik einzubringen. Laut Bruyninckx [Bru2001] sind Objektorientierung und Software-Patterns die wichtigsten Einflüsse der Softwaretechnik auf die Robotik. Jedoch haben diese Ergebnisse bisher kaum den Weg in die industrielle Robotik gefunden.

Die Softwareentwicklung bei kommerziell erhältlichen Industrierobotern ist in der Regel stark durch die – je nach Hersteller unterschiedlichen – Programmierwerkzeuge getrieben. Die Basis bilden dabei überwiegend proprietäre Spezialprogrammiersprachen, die auf die Eigenschaften der Robotersteuerungen zugeschnitten sind. Während bspw. Robotersysteme von KUKA die Sprache KRL unterstützen, müssen entsprechende Systeme von ABB mit RAPID programmiert werden. Der Funktionsumfang dieser Sprachen ist im Vergleich zu Standardprogrammiersprachen limitiert. So ist zum Beispiel keine Unterstützung für die Erstellung graphischer Eingabemasken vorhanden und die Kommunikation mit anderen Systemen ist auf die Ebene von I/O-Operationen beschränkt – im Gegenzug sind sie aber bis auf wenige Ausnahmen mit Echtzeitgarantien interpretierbar. Dies ist notwendig, um die hohe Präzision, die exakte Wiederholbarkeit und die Betriebssicherheit von Industrierobotern zu gewährleisten. Auf der Grundlage dieser Sprachen bieten die Hersteller in der Regel Erweiterungen an. Diese umfassen unterschiedliche Granularitäten und reichen von der Unterstützung spezieller Werkzeuge über Makros für häufige Schritte spezieller Arbeitsprozesse bis hin zu Konstrukten zur Synchronisation mehrerer Roboter.

2 Softwaretechnische Problemanalyse

Der immer häufigere Einsatz von Robotersystemen als Teil sehr unterschiedlicher industrieller Prozesse führt dazu, dass Roboter oft zu einer einzelnen Komponente in einem Fertigungsverbund werden. Betrachtet man darüber hinaus Aspekte, wie z.B. den verstärkten Einsatz in der Kleinserienproduktion [Pir2008], bedienerfreundliche Aufgabenbeschreibungen [HNP2008] oder den Einsatz kooperierender Roboter [HSS2005], so stoßen existierende Lösungen an ihre Grenzen. Das grundlegende Modell eines einzelnen Roboters mit montiertem Werkzeug, für das viele Steuerungen ursprünglich entwickelt wurden, ist nicht mehr ausreichend. Essentiell sind die Interaktion mit anderen Systemen, Koordination und Kooperation, die Behandlung systemübergreifender Ereignisse und Fehlerfälle sowie eine einfache und intuitive Benutzerführung. Aus einer softwaretechnischen Perspektive – unter Berücksichtigung der Kriterien Functionality (Funktionalität), Usability (Benutzbarkeit), Reliability (Zuverlässigkeit), Performance (Effizienz) und Supportability (Wartbarkeit, Portabilität und Erweiterbarkeit) – zeigen sich hier signifikante Probleme und Nachteile. Die genannten Kriterien sind ein verbreitetes Qualitätsmaß für Software (vgl. [Grady1992]) und auch bekannt unter dem Akronym FURPS.

Bei der *Funktionalität* wird die Fokussierung der Robotersteuerungssprachen auf wenige, grundlegende Bewegungsanweisungen nicht mehr allen Anforderungen gerecht. Die stärkere Integration von Sensorfeedback in die Regelung von Bewegungen (vgl. [Mas1981]) ist heute nur eingeschränkt verfügbar. Wie bereits erwähnt, ist im Vergleich zu Standardprogrammiersprachen

der Funktionsumfang der Roboterprogrammiersprachen beschränkt. Dieses Defizit hat zu einer zunehmenden Entwicklung von dual programmierten Anwendungen geführt. Die Interaktion mit Benutzern, die Kommunikation mit externen Systemen oder die Entwicklung domänenspezifischer Anwendungslogik wird in Standardprogrammiersprachen bzw. mit speicherprogrammierbaren Steuerungen realisiert. Dagegen wird Robotersteuercode ausschließlich für die Bewegungsprogrammierung und die echtzeitfähige Kommunikation mit Werkzeugen und Sensoren eingesetzt. Ein Beispiel dafür ist bei [GeYin2007] zu finden, die eine Anwendung mit graphischem Frontend zur Bedienung einer Plastikspritzgussanlage beschreiben. Der Anlagenaufbau erfordert eine Kooperation zwischen einem Robotersystem von ABB, einem Förderband und der tatsächlichen Spritzgussmaschine. Der überwiegende Teil der domänenspezifischen Anwendungslogik inklusive des graphischen Frontends wurde mit der objektorientierten Programmiersprache C# entwickelt (ca. 50.000 Zeilen Programmcode). Roboterprogramme in der ABB-Sprache RAPID im Umfang von ca. 5.000 Codezeilen werden nur zur Bewegungssteuerung verwendet.

Dieser Ansatz weist jedoch enorme Probleme bei der *Wartbarkeit* bzw. *Erweiterbarkeit* auf, da beide Programmteile getrennt voneinander entwickelt und durch geeignete Mechanismen miteinander synchronisiert werden müssen. Eine anschließende Anpassung ist nur mit hohem Aufwand möglich, da Änderungen meist in beiden Programmteilen nachgezogen werden müssen. Aufgrund mangelnder Skalierbarkeit ist der Aufwand für Erweiterungen auf neue Aufgaben und Geräte kaum geringer als bei einer Neuentwicklung. Auch die *Portabilität* von bestehenden Programmen auf andere Robotersysteme ist schwierig, da die Sprachen üblicherweise nur von einem einzigen Hersteller unterstützt werden. Etwas verbessert wird diese Situation durch die teilweise angebotene Möglichkeit der automatischen Generierung von Roboterprogrammen, da es durch die zusätzliche Indirektion einfacher wird, Code wiederzuverwenden, andere Robotersprachen zu verwenden oder externe Systeme einzubinden. Nachteilig ist, dass bei Änderungen meist das ganze Programm erneut generiert und übertragen werden muss, um anschließend neu konfiguriert, getestet und eventuell nachjustiert zu werden. Eine Rückkopplung der Nachjustierungen in das generierende Modell ist meist nicht vorhanden.

Die von den Herstellern mitgelieferte Programmierschnittstelle in Form einer Roboterprogrammiersprache mit speziellem Editor ist hinsichtlich *Benutzbarkeit* für viele Endanwender schlecht geeignet. Aus Sicht der Autoren ist die mangelnde Differenzierung der Benutzergruppen die Hauptursache dieses Problems. Eine Analyse der Schulungsangebote der Hersteller ABB Robotics, Fanuc Robotics und der KUKA Roboter GmbH hat ergeben, dass zwar augenscheinlich eine Differenzierung existiert. Bei genauer Betrachtung ergibt sich jedoch eine Unterscheidung in lediglich drei Benutzergruppen: Bediener, Instandhaltungspersonal und Programmierer. Zudem ist bei jeder Benutzergruppe die Interaktion mit dem Robotersystem gleich, d.h. sie erfolgt über die Programmiersprache und deren Laufzeitumgebung. Unter diesem Aspekt betrachtet sind die angesprochenen dual programmierten Applikation, in denen die Steuerung des Robotersystems teilweise vor dem Benutzer versteckt wird, oft weiter entwickelt. Hier variiert die Güte abhängig von der Anwendungsdomäne und dem entsprechenden Softwareangebot für diese Domäne.

Programme, die rein mit den speziellen, proprietären Programmiersprachen entwickelt wurden, weisen oft eine hohe *Zuverlässigkeit* auf, gerade weil diese Sprachen eingeschränkte Funktionalität anbieten, die gut getestet werden kann. Allerdings wird diese Zuverlässigkeit meist durch geringere Flexibilität erkauft. Wird dagegen eine duale Programmierung verwendet, leidet üblicherweise die Zuverlässigkeit der Anwendung darunter. Auch bei der *Effizienz* bietet der eingeschränkte Umfang spezieller Roboterprogrammiersprachen Vorteile, da sie oft gut optimierbar ist, und zudem auch noch direkt auf der Steuerung ausgeführt werden können. Bei dual programmierten Anwendungen hingegen entsteht durch die notwendige Kommunikation zwischen der steuernden Anwendung und der Robotersteuerung ein größerer Overhead.

Zusammenfassend lässt sich feststellen, dass bei der Entwicklung von reinen und dualen Roboterprogrammen aus softwaretechnischer Sicht Probleme im Hinblick auf Funktionalität, Flexibilität und weiteren Qualitätsanforderungen auftreten. Diese Probleme sind auf die Verwendung spezieller, im Funktionsumfang eingeschränkter Roboterprogrammiersprachen und ihrer spezifischen Anforderungen wie z.B. der echtzeitfähigen Steuerung der Robotermechanik zurückzuführen. Jedoch können diese Probleme durch die Einführung einer integrierten Softwarearchitektur, die im nächsten Abschnitt vorgestellt wird, behoben werden.

3 Der SoftRobot-Ansatz

Das Forschungsprojekt *SoftRobot* adressiert die erkannten Probleme, indem mit Methoden der Softwaretechnik die Domäne der Industrierobotik analysiert und eine modulare und erweiterbare Softwarearchitektur für die Programmierung und Steuerung von Industrierobotern entworfen wurde. Um die Industrierobotik und speziell die vielfältigen Einsatzdomänen zu verstehen, wurden ca. 20 der momentan vertriebenen Softwarepakete der KUKA Roboter GmbH auf Funktionalität, Echtzeitanforderungen und gemeinsame Muster untersucht. Diese Pakete decken das industrielle Einsatzspektrum von üblichen Palettieranwendungen über Schutzgasschweißen bis hin zu Spezialanwendungen wie Bauteillokalisierung mit Hilfe eines Kamerasystems ab. Darauf basierend wurde ein Domänenmodell mit den zentralen Konzepten in UML entwickelt. Aus den durch die Anforderungs- und Domänenanalyse gewonnen Erkenntnisse wurde eine Softwarearchitektur entwickelt, die detailliert bei Hoffmann et al. [HAO2009] beschrieben und in Abb. 1 dargestellt ist. In Zusammenarbeit mit der KUKA Roboter GmbH wurde das Domänenmodell in ein Designmodell überführt und in die Architektur integriert.

Den harten Echtzeitanforderungen, die bei der Steuerung von Robotersystemen eine große Rolle spielen, trägt die Abstützung der gesamten Architektur auf einen sogenannten *Robot Control Core* (RCC) Rechnung. Dieser ist für die performante und deterministische Ansteuerung von Hardwarekomponenten wie Sensoren und Aktuatoren verantwortlich und damit die Basis für die *Effizienz* und *Zuverlässigkeit* der gesamten Architektur. Im Gegensatz zu den existierenden kommerziellen, monolithischen Robotersteuerungen, welche im Wesentlichen einen Satz einfacher Bewegungsbefehle sowie I/O-Operationen unterstützen, wird vom RCC eine Unterstützung des *Realtime Primitives Interface (RPI)* gefordert. Diese Schnittstelle ermöglicht es, atomare und echtzeitkritische Aufgaben für einen oder mehrere Roboter (z.B. Bewegungen, Interaktion mit Werkzeugen und Sensoren) zu spezifizieren und auf der Robotersteuerung

auszuführen. Die Befehle werden dazu aus einer Menge von Modulen zusammengesetzt, deren Interaktion in einer Datenflusssprache beschrieben ist.

Der modulare und feingranulare Aufbau von RPI impliziert eine hohe Ausdrucksmächtigkeit, deren volles Spektrum nur durch eine mächtige Programmierschnittstelle genutzt werden kann. Diese Aufgabe übernehmen die *Robotics Class Libraries (RCL)*, welche alle wesentlichen Konzepte der industriellen Robotikdomäne wie Aktuatoren, Sensoren, (Bewegungs-)Instruktionen oder physikalische Gegenstände in Form einer objektorientierten Bibliothek als Erweiterung einer Standardprogrammiersprache zur Verfügung stellen. Programmierer können – deren öffentliche Schnittstelle – das *Robotics Application Programming Interface (RAPI)* – sowie die Kontrollstrukturen der Wirtsprogrammiersprache zur Formulierung vollständig ablauffähiger Roboterprogramme benutzen. Dabei können echtzeitkritische Operationen wie Bewegungs- und Werkzeugbefehle definiert und miteinander synchronisiert werden. Vor der Ausführung werden diese Befehle dynamisch in RPI-Kommandos übersetzt und anschließend vom Robot Control Core ausgeführt. Die Robotics Class Libraries stellen somit im Zusammenspiel mit RPI eine breite *Funktionalität* zur Verfügung.

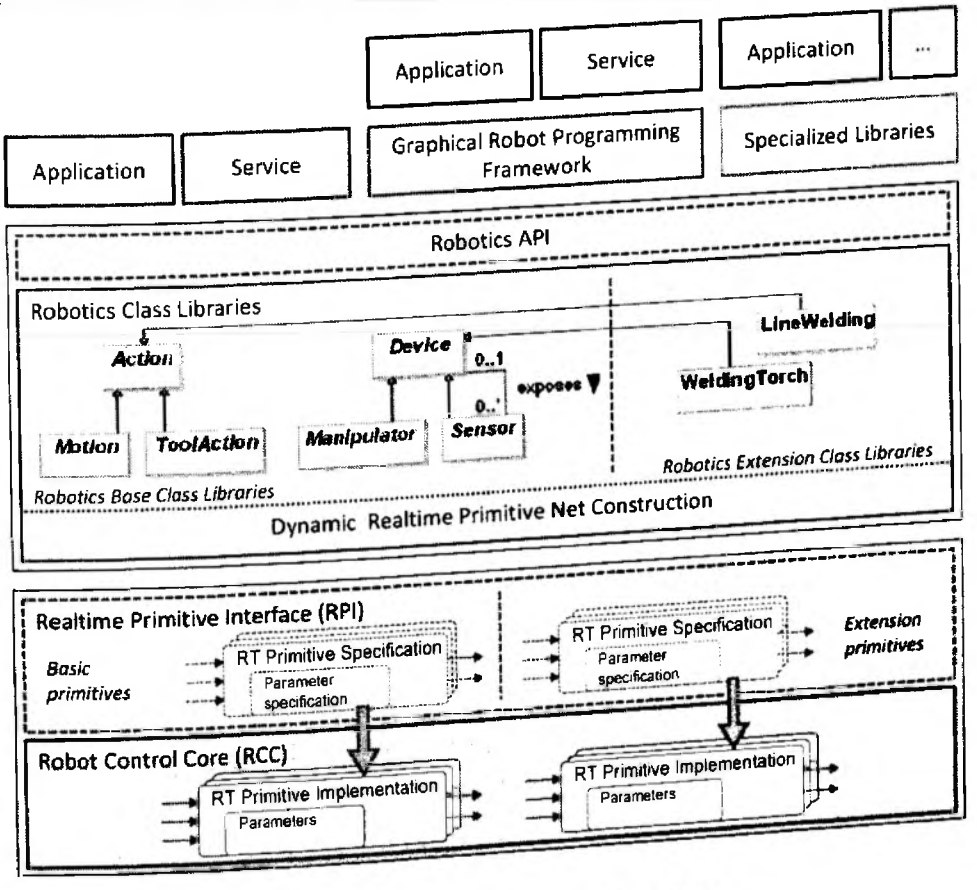

Abbildung 1: Die SoftRobot-Architektur

Die *Robotics Base Class Libraries (RBCL)* definieren den grundlegenden Aufbau der Robotics API durch eine geeignete Klassenstruktur, welche alle im Rahmen der Domänenanalyse ermittelten Konzepte vereint. Analog definiert RPI eine Menge an grundlegenden Modulen (*Basic Primitives*), die jeder konforme RCC bereitstellen muss. Dazu gehören insbesondere Module, die generische Funktionalität als Grundlage für die dynamische Übersetzung komplexer RCL-Befehle abbilden (z.B. Trigger oder Koordinatentransformationen). Diese Voraussetzungen sind kritisch für die *Wartbarkeit*, da Änderungen oder Erweiterungen der RCL weitgehend

generisch in RPI-Kommandos übersetzbar sind. Zusätzlich sind die entstehenden Roboterbefehle bei RPI sehr modular aufgebaut und enthalten nur sehr wenig Logik, wodurch das Testen bzw. die Änderung einzelner Schritte deutlich einfacher ist als bei vollständig generierten Programmen. Zudem treten im Gegensatz zu traditionellen Roboterprogrammiersprachen die Nachteile der dualen Programmierung nicht auf, da die Bewegungsbefehle und Werkzeugaktionen direkt in einer Standardprogrammiersprache realisierbar sind.

Dennoch wird sowohl auf der Ebene des RCC als auch der RCL *Erweiterbarkeit* berücksichtigt. Mit Hilfe einer *Robotics Extension Class Library (RECL)* können sowohl neue Geräte als auch neue Steuerungsbefehle in die Klassenhierarchie eingefügt werden und vom Applikationsentwickler über die Robotics API benutzt werden. Auf diese Weise können applikationsspezifische Erweiterungen programmiert werden. Sofern die Erweiterung der unterstützten Hardware und Regelungsalgorithmen nötig ist, ist die Definition und Integration neuer Module in RPI als *Extension Primitives* möglich.

Die Robotics API eignet sich durch ihre Mächtigkeit und durch die Abstützung auf eine Standardprogrammiersprache für die Entwicklung komplexer Applikationen. Für die Gruppe der Endbenutzer ist diese Schnittstelle jedoch ähnlich schlecht geeignet wie traditionelle Roboterprogrammiersprachen. Zur Steigerung der *Benutzbarkeit* ist deswegen die Entwicklung (möglicherweise domänenspezifischer) Applikationen nötig, die Benutzer bei der Beschreibung von Aufgaben des Robotersystems unterstützt. Die SoftRobot-Architektur vereinfacht die Entwicklung solcher Applikationen bereits direkt durch die mächtige Robotics API. Zusätzlich kann ein *Graphical Robot Programming Framework* als Grundlage für die Entwicklung von Applikationen zur graphischen Aufgabenbeschreibung von Robotersystemen dienen. Spezielle Bibliotheken auf Grundlage der Robotics API können häufig benutzte Funktionalität einer Klasse von Applikationen oder bestimmter Software-Produktlinien kapseln.

4 Experimenteller Vergleich

Zur Validierung der Umsetzbarkeit und als Vergleichsbasis gegenüber heutigen Steuerungsarchitekturen wurden RBCL und RCC im Umfang eines Prototyps implementiert. Darauf aufbauend wurde eine Applikation für einen Versuchsaufbau bestehend aus zwei KUKA Leichtbaurobotern umgesetzt. Die zu lösende Aufgabe bestand aus einem einfachen Pick&Place-Ablauf, jedoch wurde zwischen zwei unterschiedlichen Kategorien von zu transportierenden Gegenständen unterschieden: Kleine Gegenstände konnten von einem Roboter alleine transportiert werden, während größere Gegenstände eine Kooperation der beiden Roboter bei gleichzeitiger Echtzeit-Lastverteilung erforderten. Zusätzlich erfolgte die Identifikation der Art des Gegenstands mittels eines Kamerasystems. Dazu wurde im Versuchsaufbau eine Standard-Webcam verwendet.

Die Entwicklung dieser Applikation profitierte in mehrfacher Hinsicht von der SoftRobot-Plattform: Gerade die Programmierung kooperierender Roboter ist mit der klassischen KUKA-Steuerungsarchitektur eine sehr zeitaufwändige und fehlerträchtige Aufgabe, da getrennte Programme für beide Roboter erstellt und mit Synchronisationsmarken versehen werden müssen. Dies führt in der Regel zu Codeduplizierung und einem unübersichtlichen Programmfluss. Im

Gegensatz zur KUKA-Robotersprache KRL ist die Programmierung beliebig vieler Roboter und auch anderer Geräte innerhalb eines Programms mit der Robotics API problemlos möglich. Dadurch konnte ein kompaktes, leicht zu verstehendes Programm zur integrierten Steuerung beider Roboter erstellt werden. Für die Realisierung der Lastverteilung konnte vorhandene Basisfunktionalität zur Steuerung der Roboter sowohl auf Ebene der RCL als auch des RCC mit wenig Zusatzaufwand erweitert werden.

Durch die Abstützung auf Standardtechnologien (Programmiersprachen bzw. Laufzeitumgebungen) konnte außerdem für die Ansteuerung der Kamera und die Verarbeitung des Kamerabildes auf eine bestehende Anwendungsbibliothek (Microsoft Touchless SDK) zurückgegriffen werden. Die Verwendung von Standardtechnologien bringt viele weitere Vorteile mit sich, so existieren beispielsweise mächtige Entwicklungs- und Testumgebungen, die letztlich die Benutzbarkeit auch eines Robotersystems steigern.

5 Fazit

Die Entwicklung komplexer und flexibler Anwendungen in der Industrierobotik bereitet mit existierenden Roboterprogrammierkonzepten große Schwierigkeiten. Der *SoftRobot*-Ansatz verspricht sowohl hinsichtlich des Entwicklungsaufwands als auch bei der Qualität der resultierenden Software große Verbesserungen. Durch den Einsatz von Standardprogrammiersprachen und Technologien für Benutzerschnittstellen, Datenhaltung und Kommunikation können komplexe Anwendungen effizient entwickelt werden. Die zentralen Bestandteile der SoftRobot-Architektur, d.h. RCC und RCL, wurden in einem Prototyp implementiert, der eine gleichzeitige Steuerung zweier Roboter ermöglicht. Die Erkenntnisse aus dieser Implementierung zeigten, dass der entwickelte Ansatz realisierbar und effektiv ist und fließen in die weitere Entwicklung des Modells ein.

Danksagung

Diese Arbeit entstand im Rahmen des Forschungsprojektes *SoftRobot*, welches von der Europäischen Union und der High-Tech-Offensive Bayern der Bayerischen Staatsregierung gefördert wird. Das Projekt wird gemeinsam vom Institut für Software & Systems Engineering der Universität Augsburg, der KUKA Roboter GmbH und der MRK-Systeme GmbH durchgeführt.

Literatur

[BGI2009] Bensalem, S., Gallien, M., Ingrand, F., Kahloul, I., Thanh-Hung, N.: *Designing Autonomous Robots*. IEEE Robotics & Automation Magazine, 16 (1), Seiten 67–77, 2009.

[Bru2001] Bruyninckx, H.: *Open Robot Control Software: The OROCOS Project*. Proc. 2001 IEEE International Conference on Robotics and Automation, Seiten 2523–2528, Seoul, Korea, 2001.

[Bru2007] Brugali, D. (Hrsg.): *Software Engineering for Experimental Robotics.* Springer Tracts in Advanced Robotics. Springer-Verlag Berlin Heidelberg, 2007.

[BruPra2009] Brugali, D., und Prassler, E.: *Software Engineering for Robotics.* IEEE Robotics & Automation Magazine, 16 (1), Seiten 9–15, 2009.

[GeYin2007] Ge, J. G., Yin, X. G.: *An Object Oriented Robot Programming Approach in Robot Served Plastic Injection Molding Application.* Lecture Notes in Control and Information Sciences: Robotic Welding, Intelligence and Automation, 362, Seiten 91–97, 2007.

[Grady1992] Grady, R.: *Practical Software Metrics for Project Management and Process Improvement.* Prentice-Hall Englewood Cliffs, NJ, 1992.

[HAO2009] Hoffmann, A., Angerer, A., Ortmeier, F., Vistein, M., Reif, W.: *Hiding Real-Time: A new Approach for the Software Development of Industrial Robots.* Proc. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, USA, Oktober 2009.

[HNP2008] Hägele, M., Nilsson, K., Pires, J. N.: *Industrial Robotics.* In: Siciliano B., Khatib, O. (Hrsg.): Springer Handbook of Robotics, Springer-Verlag Berlin Heidelberg, 2008, Kapitel 42, Seiten 963–986.

[HSS2005] Hägele, M., Skordas, T., Sagert, S., Bischoff, R., Brogårdh, T., Dresselhaus, M.:. *Industrial Robot Automation.* White paper, European Robotics Network, 2005.

[ICO2009] Iborra, A., Caceres, D., Ortiz, F., Franco, J., Palma, P., Alvarez, B.: *Design of Service Robots.* IEEE Robotics & Automation Magazine, 16 (1), Seiten 24–33, 2009.

[ISO8373] DIN EN ISO 8373:1996-08: Industrieroboter – Wörterbuch. 1996.

[Mas1981] Mason, M.: *Compliance and Force Control for Computer-Controlled Manipulators.* IEEE Transactions on Systems, Man, and Cybernetics, 11 (6), Seiten 418–432, 1981.

[MiLe1990] Miller, D., Lennox, R. C.: *An Object-Oriented Environment for Robot System Architectures.* Proc. 1990 IEEE International Conference on Robotics and Automation, Seiten 352–361, Cincinnati, USA, 1990.

[Pir2008] Pires, J. N.: *New challenges for industrial robotic cell programming.* Industrial Robot, 36 (1), 2008.