Objektorientierte Software für Industrieroboter[1]

Andreas Angerer[2]

Abstract: Industrieroboter werden heutzutage in vielen Bereichen der Fertigungsindustrie eingesetzt und entlasten dort Menschen von schweren und sich wiederholenden Tätigkeiten. Ihr breiterer Einsatz wird jedoch immer mehr durch veraltete herstellerspezifische Programmiersprachen und proprietäre Software-Ökosysteme limitiert. In dieser Arbeit wird ein objektorientiertes Software-Design vorgestellt, mit dem sich Anwendungen für Industrieroboter mit Standard-Programmiersprachen entwickeln lassen. Dabei ist es erstmals gelungen, moderne Programmiersprachen mit den für die Industrierobotik nötigen Echtzeitanforderungen zu verbinden und gleichzeitig die Flexibilität der Programmierschnittstelle zu erhöhen. Die entwickelten Softwarekonzepte wurden in einer Referenzimplementierung umgesetzt und anhand mehrerer anspruchsvoller Applikationen evaluiert. Bereits heute werden die Ergebnisse dieser Arbeit in der KUKA Sunrise Steuerung für den seit Kurzem erhältlichen Roboter LBR iiwa eingesetzt.

1 Einführung

Über 1,3 Millionen Industrieroboter sind – nach Schätzung der International Federation of Robotics (IFR) [IF14] – heute weltweit in Betrieb. Sie übernehmen vor allem in der Fertigungsindustrie schwere, sich wiederholende Tätigkeiten, die sie mit sehr hoher Geschwindigkeit und Präzision ausführen. Gerade in der Automobilindustrie, dem laut IFR nach wie vor größten Markt für Industrieroboter, wäre eine ökonomische Produktion ohne hohen Automatisierungsgrad nicht denkbar. Industrieroboter führen jedoch auch viele Tätigkeiten durch, die Menschen nicht oder nur schwer selbst übernehmen könnten. In der Fertigungsindustrie gehört dazu beispielsweise der Transport schwerer Bauteile wie Autokarosserien. In anderen Bereichen, wie der Chemie- oder Atomindustrie, werden Industrieroboter auch in gesundheitsgefährdenden Umgebungen eingesetzt, zum Beispiel beim Rückbau von Atomkraftwerken.

Industrieroboter gibt es bereits seit über 50 Jahren. Die grundsätzlichen mechanischen, mathematischen und algorithmischen Methoden, die für ihren Betrieb nötig sind, sind gut erforscht und haben sich in den letzten 20 Jahren nicht wesentlich weiterentwickelt. Zwischenzeitlich galt die industrielle Robotik in der Forschung sogar bereits als gelöstes Problem [HNP08]. Verschiedene Trends wie der verstärkte Einsatz von Sensorik, die Erweiterung von einzelnen Robotern zu Teams mehrerer Roboter sowie die engere Zusammenarbeit von Robotern und Menschen bringen die existierenden Steuerungen jedoch an ihre Grenzen. Vor diesem Hintergrund fand dieses spannende, interdisziplinäre Forschungsfeld in den letzten Jahren wieder mehr Beachtung.

[1] Englischer Titel der Dissertation: “Object-oriented Software for Industrial Robots”

[2] Institut für Software & Systems Engineering, Universität Augsburg, andreas.angerer@gmail.com

Das Feld der Softwaretechnik hat sich in den letzten Jahrzehnten substantiell weiterentwickelt. Die Entstehung von Softwareentwicklungsprozessen, neuen Modellierungsansätzen, Programmiersprachen sowie zahlreicher Entwicklungswerkzeuge, Bibliotheken und Plattformen haben die Softwareentwicklung weiter in Richtung einer Ingenieursdisziplin gebracht. Der Stand der Technik bei der Entwicklung von Software für Industrieroboter hinkt dem jedoch hinterher: Die meisten Software-Ökosysteme[3] in diesem Bereich sind stark proprietär und werden von den Roboterherstellern selbst oder von einem relativ kleinen Kreis an Systemintegratoren entwickelt. Während für klassische Einsatzgebiete von Industrierobotern – wie das Schweißen oder Lackieren – durchaus branchenspezifische Softwareunterstützung existiert, sind andere Anwenderkreise auf sich selbst gestellt. Bei der Entwicklung einer für sie passenden Softwarelösung sind sie durch die proprietären Entwicklungsplattformen limitiert. Sie können kaum von den Werkzeugen und Methoden der modernen Softwaretechnik profitieren, wie zum Beispiel objektorientiertes Design, serviceorientierte Architekturen, Test- und Debugging-Frameworks oder hochentwickelte integrierten Entwicklungsplattformen. Im Gegenteil: Werden solche Technologien eingesetzt, ergibt sich daraus ein beträchtlicher Aufwand für die Integration mit dem eingesetzten Robotersystem, wie einige dokumentierte Beispiele ([GY07], [PVA09]) zeigen.

Was macht es nun so schwierig, moderne Programmiersprachen und ihre mächtigen Ausführungsplattformen für Industrieroboter einzusetzen? Diese Frage wurde zu Beginn des SoftRobot-Projekts näher untersucht. In diesem Projekt[4] arbeiteten die Universität Augsburg, die KUKA Laboratories GmbH[5] und die MRK Systeme GmbH zusammen. Die wesentliche Herausforderung ergab sich unmittelbar aus den größten Stärken heutiger Industrieroboter: ihre Geschwindigkeit und Präzision. KUKA garantiert für fast alle ihrer Roboter eine Wiederholgenauigkeit von mindestens $\pm 0,1$ Millimeter. Das bedeutet, dass ein programmierter Ablauf auch bei beliebig vielen Wiederholungen immer bis auf ein Zehntel Millimeter genau gleich ausgeführt wird – und das bei Geschwindigkeiten von bis zu 3 Metern pro Sekunde. Umgekehrt heißt das, dass etwa eine zeitliche Verzögerung von einer Millisekunde bei der Ausführung eines Steuerungs-Algorithmus bereits zu einer Abweichung von 3 Millimetern führen und damit die gewünschte Wiederholgenauigkeit deutlich überschreiten würde. Auf diese Art von strikten zeitlichen Garantien – man spricht hier auch von harten Echtzeit-Anforderungen – sind moderne Ausführungsplattformen wie die Java- oder .NET-Plattform nicht ausgelegt. Durch ihre automatische Speicherverwaltung erlauben sie zwar eine wesentliche effizientere Entwicklung von Software [Ph99], diese ist allerdings gleichzeitig eine inhärente Quelle von zeitlichem Indeterminismus bei der Ausführung. Zur Erfüllung harter Echtzeitanforderungen müssten jedoch nicht nur die Ausführungsplattformen (z.B. Programminterpreter), sondern auch alle eingesetzten Bibliotheken gezielt angepasst werden. Dieser Ansatz würde das Ziel konterkarieren, die Welt der modernen Software-Ökosysteme für Industrieroboter zugänglich zu machen.

Um eine neue Generation von Industrieroboter-Steuerungen zu schaffen (vgl. Abbildung 1), wurde im SoftRobot-Projekt ein anderer Ansatz verfolgt. Nach einer Analyse

[3] damit ist hier die Gesamtheit von Programmiersprachen, Entwicklungswerkzeugen und Schnittstellen gemeint

[4] SoftRobot wurde im Rahmen des Förderprogramms „Informations- und Kommunikationstechnik Bayern“ von der Bayerischen Staatsregierung und der EU gefördert.

[5] zu Beginn des Projekts stattdessen die KUKA Roboter GmbH

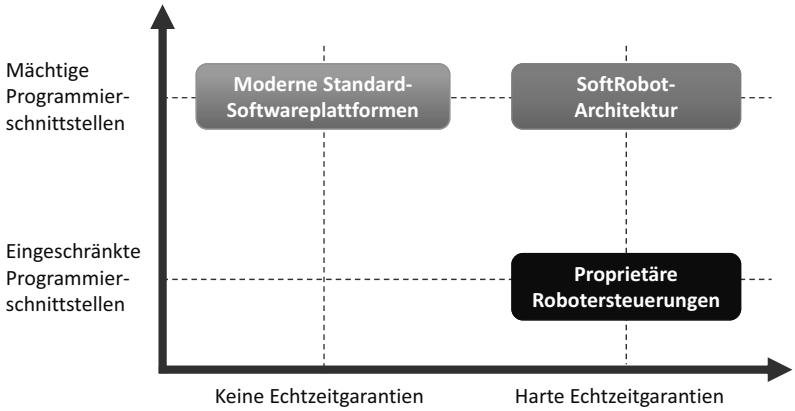

Abb. 1: Das Ziel des SoftRobot-Projekts bestand in der Harmonisierung zweier Dimensionen: Ein hoher Grad an Echtzeitfähigkeit (horizontale Achse) bei gleichzeitig hoher Mächtigkeit der Programmierschnittstelle (vertikale Achse). Existierende proprietäre Robotersteuerungen (rechts unten) bieten harte Echtzeitgarantien, weisen aber Limitierungen bei der Programmierung auf. Moderne Softwareplattformen wie die Java- oder .NET-Plattform (links oben) bieten mächtige Programmierschnittstellen, aber in der Regel keine Garantien hinsichtlich des Zeitverhaltens. Die SoftRobot-Architektur hingegen kann beides vereinen.

aller für die KUKA-Steuerung verfügbarer Programmpakete stellte sich heraus, dass alle eine gemeinsame Charakteristik aufweisen: Sie lassen sich unterteilen in eine abgeschlossene Menge von *Echtzeit-Patterns*, also wiederkehrende Muster von Operationen mit harten Echtzeitanforderungen, und eine Ablauflogik, die diese Echtzeit-Operationen „orchestriert“, selbst jedoch keine harten zeitlichen Schranken einhalten muss [Ho09]. Dieser Aufteilung folgt die *SoftRobot-Architektur*. Echtzeit-Operationen werden vom *Robot Control Core (RCC)* ausgeführt, während die Ablauflogik eines Roboterprogramms sowie die flexible Definition und Parametrisierung der Echtzeit-Operationen mit der objektorientierten *Robotics API* beschrieben werden können. Applikationsentwickler müssen sich dadurch insbesondere keine Sorgen machen, dass ihre Programme durch zeitaufwändige Berechnungen die Performanz der Robotersteuerung beeinträchtigen.

Diese Arbeit beschäftigt sich mit dem Softwaredesign der Robotics API. Es gelang, dieses Design *Programmiersprachen-unabhängig* zu gestalten. Das bedeutet, dass eine Implementierung in jeder modernen, objektorientierten Programmiersprache möglich ist und keine besonderen Anforderungen an die Echtzeitfähigkeit der Ausführungsumgebung stellt. Das Design ist darüber hinaus *Hersteller-unabhängig*, so dass mit der Referenzimplementierung inzwischen Industrieroboter vier unterschiedlicher Hersteller unterstützt werden können. Durch gezielte Erweiterungspunkte, eine explizite Modellierung von Sensoren sowie ein feingranulares Modell von Roboter-Operationen kann die Robotics API die Schwächen heutiger Roboterprogrammiersprachen beseitigen und bleibt gleichzeitig *erweiterbar*. Schließlich ermöglicht es insbesondere das angesprochene Kommando-

Modell, die nötigen Echtzeit-Anforderungen bei der Ausführung von Roboter-Operationen im RCC zu gewährleisten.

2 Objektorientierte Modelle für Industrieroboter

Die Robotics API verknüpft wie in Abbildung 2 dargestellt fünf Modelle miteinander, die zusammen eine flexible und mächtige Programmierschnittstelle für Industrieroboter bilden. Existierende Forschungsansätze beschäftigen sich nur mit Teilaspekten dieser Modelle und liegen teilweise auch dem Design der Robotics API zugrunde. Eine Verknüpfung aller wichtigen Teile – wie in der Robotics API – gelang bisher jedoch nicht.

Das *Device Model* erlaubt eine Modellierung aller Arten steuerbarer Geräte. Es ist auf der einen Seite generisch genug, um eine große Vielzahl unterschiedlicher Geräte wie Roboterarme, Linearachsen, Roboterwerkzeuge und mobile Roboterplattformen abbilden zu können. Auf der anderen Seite ermöglicht es eine ausreichende Detaillierung, um auch feine Unterschiede zwischen verschiedenen Geräten auszudrücken.

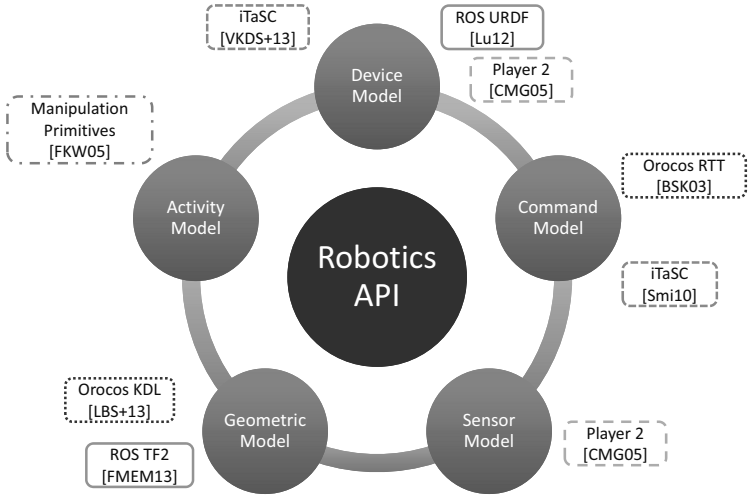

Abb. 2: Die Robotics API ist aus fünf zusammenhängenden Modellen aufgebaut. Im Zusammenspiel ermöglicht sie die Modellierung steuerbarer Geräte, die Beschreibung echtzeitkritischer Operationen dieser Geräte, den Zugriff auf alle Arten von Echtzeit-Daten, die Beschreibung geometrischer Beziehungen und schließlich die kontextabhängige Planung von Roboteraktionen. Existierende Forschungsansätze wie das populäre ROS, Orocos, das Player-Projekt oder auch industrienähere Forschung zu Manipulation Primitives decken bis heute jeweils nur Teile der hier beschriebenen Aspekte ab.

Zur echtzeitkritischen Steuerung dieser Geräte enthält die Robotics API das *Command Model*. Damit ist es möglich, elementare Operationen der Geräte, wie zum Beispiel Bewegungen oder Werkzeugaktionen, zu definieren und diese zu komplexen Abläufen zu kombinieren. Die Abläufe müssen dabei nicht starr vorgegeben sein, sondern können etwa auf Gegebenheiten der Umgebung oder Fehler während der Ausführung feingranular

reagieren – vom Umplanen einer Operation über Warnmeldungen bis hin zum sicheren Abbruch der Tätigkeit. Das besondere Softwaredesign des Command Model erlaubt es, all dies auf die datenfluss-basierte Schnittstelle des Robot Control Core abzubilden und dadurch mit harten Echtzeit-Garantien auszuführen.

Eine wichtige Voraussetzung für flexiblere Roboterapplikationen ist der Einsatz von Sensoren und die geeignete Verarbeitung der gelieferten Daten zur Laufzeit. Dafür stellt das *Sensor Model* die passenden Konzepte bereit. Daten des Roboters selbst, seiner Werkzeuge oder dedizierter Sensorik aller Art können fusioniert und so der Zustand des Robotersystems und der Umgebung bestimmt werden. Dieser Zustand und der Ausführungsfortschritt von Roboteroperationen können die Grundlage der oben erwähnten, feingranularen Reaktionen bilden. So ist es zum Beispiel beim Bahnschweißen entscheidend, den Schweißbrenner so schnell wie möglich vom Werkstück weg zu bewegen, wenn während des Schweißvorgangs durch einen Sensor ein Fehler erkannt wird. Andernfalls kann das zu schweißende Teil irreparabel beschädigt werden. Die Robotics API unterstützt auch die Verarbeitung und Fusion von Sensordaten in harter Echtzeit.

In Roboteranwendungen ist es nötig, räumliche Zusammenhänge zu definieren – sowohl zur Beschreibung struktureller Eigenschaften („Wo ist der Roboter montiert?“) als auch zur Parametrisierung von Operationen („Wohin soll das Werkstück transportiert werden?“). Das *Geometric Model* der Robotics API stützt sich auf etablierte mathematische Formalismen ab und verknüpft diese mit dem Command Model und Sensor Model. Es können nicht nur statische, sondern auch dynamische Aspekte beschrieben werden. Komplexe geometrische Bedingungen lassen sich so zur Laufzeit überwachen und es kann entsprechend reagiert werden.

Für eine einfache Benutzbarkeit der Robotics API durch Anwendungsentwickler kommt dem *Activity Model* eine entscheidende Bedeutung zu. Aufbauend auf das Command Model stellt es ein einfach zu benutzendes und elegantes Modell typischer Aktivitäten von Industrierobotern zur Verfügung. Die Ausdrucksstärke des Command Model wird dabei mit den Konzepten des Device Model, Sensor Model und Geometric Model kombiniert. Das so entstehende Ausführungsmodell beinhaltet Meta-Daten über die ausgeführten Operationen: Wo lag das Ziel einer Bewegung? Hat sich dieses Ziel selbst bewegt, weil es zum Beispiel auf einem Förderband liegt? Welche Kraft wurde vom Roboter auf seine Umgebung aufgebaut? Auf Basis dieser Daten können Aktionen kontextabhängig geplant und ausgeführt werden und dem Entwickler hilfreiches Feedback gegeben werden.

Abbildung 3 zeigt die objektorientierte Struktur des Command Model. Es basiert auf dem in der Softwaretechnik etablierten *Command Pattern* von Gamma et al. [Ga94]. Dieses behandelt jedoch nicht den Aspekt der Ausführung von Kommandos mit der Garantie harter Echtzeit. Wir haben die Struktur des Command Model deswegen im Vergleich zum Command Pattern angepasst, so dass es sich zur Beschreibung echtzeitkritischer Operationen in der Industrierobotik eignet. Eine besondere Bedeutung kommt dem internen Design des *TransactionCommand* zu: Es ermöglicht eine flexible Orchestrierung interner Commands abhängig von Sensorwerten und dem Ausführungsfortschritt, wie oben beschrieben. Instanzen des Command Models beschreiben somit echtzeitfähige Abläufe, die von einer deterministischen Steuerung ausgeführt werden können.

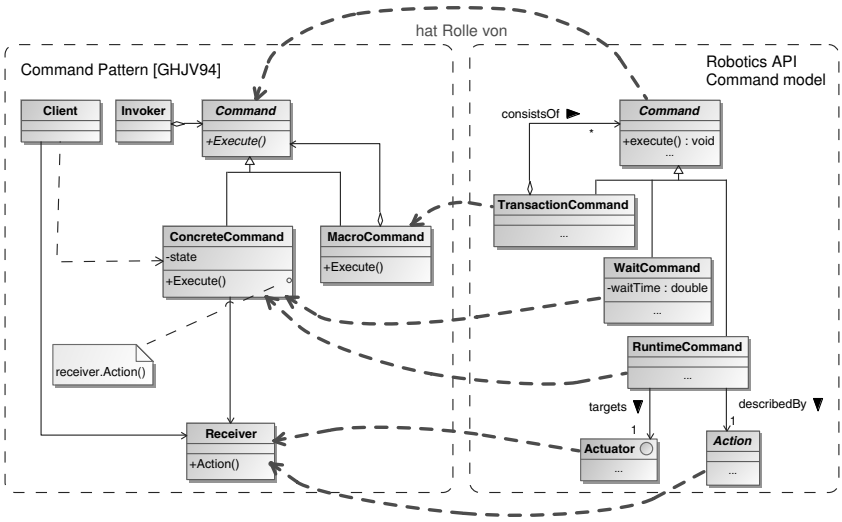

Abb. 3: [An14] Ein zentraler Bestandteil der Robotics API ist das Command Model, das eine Abwandlung des Command Pattern darstellt. Die konkreten Kommandos RuntimeCommand und WaitCommand im Command Model verfügen daher über keine spezifische Implementierung einer Ausführungsmethode. Stattdessen wird bei einem RuntimeCommand die auszuführende Aktion durch eine *Action* („Was soll ausgeführt werden?“) und einen *Actuator* („Wer soll es ausführen?“) beschrieben. Ein *WaitCommand* definiert hingegen nur eine Warteaktion für eine gegebene Zeit.

Mit dem Design der Robotics API ist es gelungen, die Grundlage für den Einsatz moderner Programmiersprachen und Softwareplattformen zur Steuerung von Industrierobotern zu legen. Sie beinhaltet nicht nur ein umfassenderes Gesamtmodell als der bisherige Stand der Forschung und Technik, sondern berücksichtigt im Gegensatz zu den meisten existierenden Ansätzen auch harte Echtzeit-Anforderungen. Im Vergleich mit heutigen industriellen Robotersteuerungen ist die Robotics API eine wesentlich flexiblere und besser erweiterbare Plattform auf dem Stand aktueller Softwaretechnik.

3 Evaluation

Um die Frage zu beantworten, wie erweiterbar diese Plattform wirklich ist, untersucht die Arbeit die Wiederverwendbarkeit unterschiedlicher Komponenten der Referenzimplementierung der SoftRobot-Architektur. Als Indikator für Wiederverwendbarkeit dient die Verteilung der Programmcode-Zeilen. Zwar ist dies kein präzises Maß für Wiederverwendbarkeit, die Deutlichkeit der Ergebnisse spricht jedoch für sich. Weiterhin stellt die Arbeit mehrere Applikationen mit ganz unterschiedlichen Anforderungen vor, denen die Robotics API durchwegs gerecht werden kann.

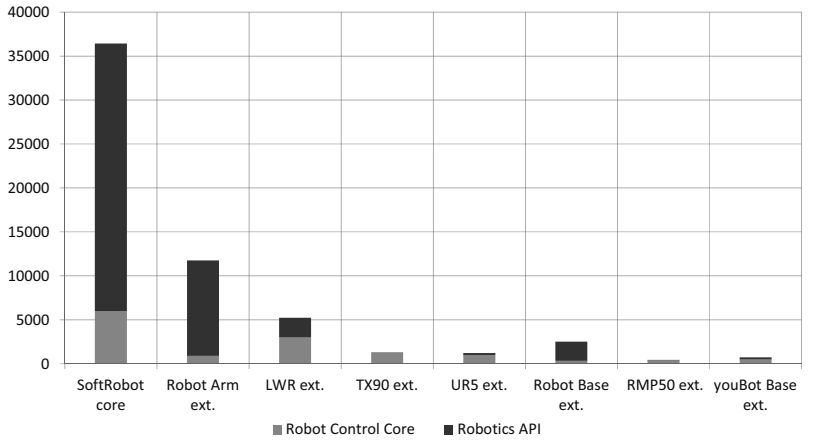

Abb. 4: Die Verteilung des Programmcodes innerhalb einzelner Teile der SoftRobot-Architektur. Die horizontale Achse zeigt acht verschiedene Teile der SoftRobot-Referenzimplementierung: Die Kernkomponente (*SoftRobot core*), die *Robot Arm extension*, darauf aufbauend die Erweiterungen für die drei Roboterarme KUKA LBR, Stäubli TX90 und Universal Robots UR5, schließlich die *Robot Base extension* für mobile Roboter und darauf aufbauend die Erweiterungen für die Geräte Segway RMP50 und die KUKA youBot-Plattform. Die vertikale Achse zeigt die Anzahl der Programmcode-Zeilen (Lines of Code), aufgeteilt nach Robot Control Core und Robotics API.

Abbildung 4 zeigt die Verteilung der Programmcode-Zeilen (Lines of Code, LOC) und liefert zwei wichtige Erkenntnisse: Zum einen erlaubt es das Softwaredesign der Robotics API, sehr viel Funktionalität in abstrakten, generischen Komponenten zu implementieren. Insbesondere die Einbindung neuer Geräte ist dadurch sehr einfach möglich und erfordert zum Beispiel für einen Stäubli TX90 Roboterarm nur wenige Dutzend Zeilen Programmcode. Zum anderen gelang es, einen Großteil der Funktionalität der gesamten SoftRobot-Plattform innerhalb der Robotics API zu implementieren und den im RCC nötigen Code bei Erweiterungen auf ein Minimum zu reduzieren. Bei der Betrachtung der umfangreichen Kernkomponente der Architektur fällt auf, dass über 80% des Programmcodes auf die Robotics API entfällt. Dieser Teil der Robotics API enthält überwiegend abstrakte Implementierungen der schon vorgestellten fünf Modelle, die dann in sogenannten *extensions* weiter verfeinert werden. Die Kernkomponente der Architektur ist so generisch gehalten, dass sogar das Konzept eines Roboterarms erst durch die *Robot Arm extension* eingeführt wird. Das Ziel dieser extremen Reduktion der Kernkomponente war es, so weit wie möglich versteckte Annahmen bei der Implementierung generischer Konzepte zu eliminieren.

Im Rahmen der Arbeit wurden verschiedene herausfordernde Applikationen mit hoch entwickelten Leichtbaurobotern von KUKA umgesetzt. Diese Roboter zeichnen sich unter anderem durch eine integrierte Kraft-Momenten-Sensorik aus. Sie erlaubt neuartige, feinfühlige Manipulationsaufgaben, erhöht aber gleichzeitig die Anforderungen an die Mächtigkeit der Programmierschnittstelle.

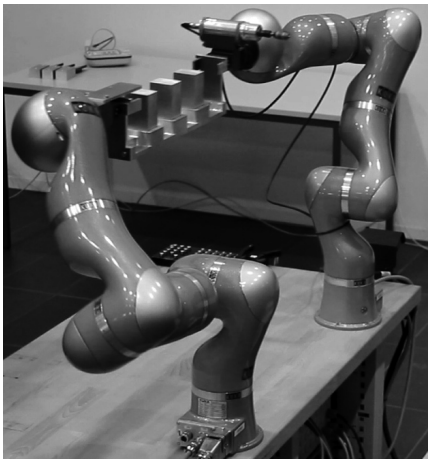

Abb. 5: Gemeinsamer Transport in der Intelligenten Fertigungszelle.

Die aufwändigste mit der Robotics API umgesetzte Applikation ist die *Intelligente Fertigungszelle* [An14]. Zwei Leichtbauroboter arbeiten gemeinsam daran, Werkstücke aus Einzelteilen zu montieren. Dabei transportieren sie zunächst gemeinsam Werkstückträger mit den Einzelteilen auf eine Werkbank (siehe Abbildung 5), um diese anschließend koordiniert und unter Einsatz der Kraftsensorik zusammenzufügen und zu verschrauben. Beim gemeinsamen Transport teilen sich die Roboter die Last, müssen jedoch präzise synchronisiert werden, um Verspannungen und dadurch Beschädigungen zu vermeiden. Beim Fügen und Verschrauben werden Prozessunsicherheiten durch die „Feinfühligkeit“ der Kraftsensoren ausgeglichen. Die Prozessschritte orientieren sich am menschlichen Vorgehen: So wird zum Beispiel beim Schraubvorgang mit einem Elektroschrauber zunächst Kraft auf die Schraube aufgebaut, um den Schrauber dann zu starten und kraftgesteuert der Schraubenbewegung nachzuführen.

Zwei weitere Beispiele veranschaulichen die Reduzierung des Aufwands bei der Integration eines Industrieroboters mit zusätzlichen Geräten. Die *PortraitBot*-Applikation macht aus einem Roboterarm einen Porträt-Maler. Dazu werden eine Standard-Webcam und frei verfügbare Java-Bibliotheken zur Bildverarbeitung eingesetzt. Die Roboterbewegung selbst wird mit der Robotics API mit wenigen Programmzeilen in der Java-Umgebung realisiert, ohne den Umweg über die Generierung von proprietärem Code und dessen aufwändige Übertragung an eine herkömmliche Robotersteuerung. Die Applikation *Tangible Teleoperation* setzt ein sogenanntes *tangible user interface*, oder auch *Gegenständliche Benutzerschnittstelle*, zur Fernsteuerung von Robotern ein. In dieser Applikation wird die heute mögliche Interoperabilität zwischen unterschiedlichen Softwareplattformen ausgenutzt: Die Tangible Teleoperation Applikation wurde vollständig in C# auf der .NET-Plattform entwickelt, um Microsoft PixelSense[6] als Eingabegerät verwenden zu können. Auch in diesem Fall ist es durch die Robotics API gelungen, den Systemintegrationsaufwand mit einer Robotersteuerung zu vermeiden. Wie hoch dieser Aufwand sein kann, zeigte ein Experiment: Das *Fernbediente Manipulator-Steuersystem* von Projektpartner MRK Systeme, ein industrielles Fernsteuerungssystem für Industrieroboter, wurde von einer Person in etwa 2 Wochen auf die Robotics API portiert und konnte ohne Einschränkungen betrieben werden. Bei der Portierung wurde lediglich eine von MRK selbst entwickelte Schnittstelle zum verwendeten Robotersystem ersetzt. Diese Schnittstelle war ursprünglich mit einem Aufwand von 2 Personenmonaten entwickelt worden. Die Robotics API hätte diesen Aufwand also auf etwa ein Viertel reduziert.

[6] `http://www.microsoft.com/en-us/pixelsense/default.aspx`

4 Zusammenfassung und Ausblick

Der breite Einsatz von Industrierobotern wurde bis heute zunehmend durch ihre proprietären Software-Ökosysteme begrenzt. Das Forschungsprojekt SoftRobot hat diese Limitierung aufgehoben, indem die Softwareentwicklung für Industrieroboter auf das Niveau des modernen Software Engineering gehoben wurde. Durch ein Programmiersprachen- und Hersteller-unabhängiges objektorientiertes Softwaredesign trägt diese Arbeit wesentlich dazu bei, dass das Ziel erreicht wurde. Mit der Robotics API kann in Zukunft wesentlich schneller und effizienter Software für Industrieroboter entwickelt werden, da ein riesiges Ökosystem an Bibliotheken und Entwicklungswerkzeugen zur Verfügung steht, das sich ständig vergrößert. Dabei ist die umfangreiche Kernkomponente der Referenzimplementierung so allgemein gehalten, dass eine Erweiterung auf viele weitere steuerungstechnische Bereiche machbar erscheint. Die Ergebnisse dieser Arbeit werden bereits jetzt in der Praxis eingesetzt: Die KUKA Laboratories GmbH präsentierte auf der Hannover Messe 2013 den Roboter LBR iiwa, dessen „Steuerungssystem der Zukunft“[7] mit seiner Java-basierten Programmierschnittstelle die in dieser Arbeit entwickelten Konzepte in die Industrie bringt.

Literaturverzeichnis

[An14] Angerer, Andreas: Object-oriented Software for Industrial Robots. Dissertation, University of Augsburg, Mar 2014.

[BSK03] Bruyninckx, Herman; Soetens, Peter; Koninckx, Bob: The Real-Time Motion Control Core of the Orocos Project. In: Proc. 2003 IEEE Intl. Conf. on Robot. & Autom. Seoul, Korea, S. 2766–2771, Mai 2003.

[CMG05] Collett, Toby; MacDonald, Bruce; Gerkey, Brian: Player 2.0: Toward a Practical Robot Programming Framework. In: Proc. 2005 Australasian Conf. on Robotics and Automation. Sydney, Australia, Dezember 2005.

[FKW05] Finkemeyer, Bernd; Kröger, Torsten; Wahl, Friedrich M.: Executing Assembly Tasks Specified by Manipulation Primitive Nets. Advanced Robotics, 19(5):591–611, 2005.

[FMEM] Foote, Tully; Marder-Eppstein, Eitan; Meeussen, Wim: , tf2 - ROS Wiki. http://wiki.ros.org/tf2 (accessed Jul. 2013).

[Ga94] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design patterns: Elements of reusable object-oriented software. Addison Wesley, 1994.

[GY07] Ge, Jing Guo; Yin, Xing Guo: An Object Oriented Robot Programming Approach in Robot Served Plastic Injection Molding Application. In: Robotic Welding, Intelligence & Automation. Jgg. 362 in Lect. Notes in Control & Information Sciences. Springer, S. 91–97, 2007.

[HNP08] Hägele, Martin; Nilsson, Klas; Pires, J. Norberto: Industrial Robotics. In (Siciliano, Bruno; Khatib, Oussama, Hrsg.): Springer Handbook of Robotics, Kapitel 42, S. 963–986. Springer, Berlin, Heidelberg, 2008.

[7] siehe `http://www.kuka-labs.com/de/service_robotics/robot_control_system/`

[Ho09] Hoffmann, Alwin; Angerer, Andreas; Ortmeier, Frank; Vistein, Michael; Reif, Wolfgang: Hiding Real-Time: A new Approach for the Software Development of Industrial Robots. In: Proc. 2009 IEEE/RSJ Intl. Conf. on Intell. Robots and Systems, St. Louis, MO, USA. IEEE, 2009.

[IF14] IFR Statistical Department: , World Robotics 2014 - Executive Summary. http://www.worldrobotics.org/index.php?id=downloads, 2014. (accessed Jan. 2015).

[La13] Laet, Tinne De; Bellens, Steven; Smits, Ruben; Aertbelien, Erwin; Bruyninckx, Herman; Schutter, Joris De: Geometric Relations between Rigid Bodies (Part 1) – Semantics for Standardization. IEEE Robot. & Autom. Mag., S. 84–93, Jun 2013.

[Lu12] Lu, David: , URDF and You. Talk at ROSCon 2012, May 2012. http://www.cse.wustl.edu/ dvl1/?p=93 (accessed Feb. 2014).

[Ph99] Phipps, Geoffrey: Comparing Observed Bug and Productivity Rates for Java and C++. Softw. Pract. Exper., 29(4):345–358, April 1999.

[PVA09] Pires, J. Norberto; Veiga, Germano; Araújo, Ricardo: Programming by demonstration in the coworker scenario for SMEs. Industrial Robot, 36(1):73–83, 2009.

[Sm10] Smits, Ruben: Robot Skills: Design of a constraint-based methodology and software support. Dissertation, Katholieke Universiteit Leuven - Faculty of Engineering, Kasteelpark Arenberg 1, B-3001 Leuven (Belgium), Mai 2010.

[Va13] Vanthienen, D.; Klotzbuucher, M.; De Schutter, J.; De Laet, T.; Bruyninckx, H.: Rapid application development of constrained-based task modelling and execution using domain specific languages. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. S. 1860–1866, Nov 2013.

Andreas Angerer, geboren am 12. Juli 1984, studierte von 2003 bis 2006 Angewandte Informatik in Augsburg, bevor er 2006 in den Elite-Masterstudiengang „Software Engineering“ der Universität Augsburg, TU München und LMU München wechselte. Seit seinem Abschluss im Jahr 2008 arbeitet er als wissenschaftlicher Mitarbeiter am Lehrstuhl für Softwaretechnik der Universität Augsburg und promovierte dort im Mai 2014. Als Postdoktorand forscht er seitdem in verschiedenen Kooperationen mit Unternehmen und Forschungseinrichtungen wie dem Deutschen Zentrum für Luft- und Raumfahrt weiter im Bereich moderne Softwarearchitekturen für Industrieroboter.