

Objektorientierte Programmierung von Industrierobotern

Andreas Angerer, Alwin Hoffmann, Andreas Schierl, Michael Vistein,
Wolfgang Reif, Markus Bischof, Alexander Chekler, Christian Tarragona

Angaben zur Veröffentlichung / Publication details:

Angerer, Andreas, Alwin Hoffmann, Andreas Schierl, Michael Vistein, Wolfgang Reif, Markus Bischof, Alexander Chekler, and Christian Tarragona. 2010. "Objektorientierte Programmierung von Industrierobotern." In *Internationales Forum Mechatronik 2010, Winterthur, Switzerland, November 3-4, 2010*, 1–16. Linz: Clusterland Oberösterreich.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/>



Information zum Referenten:

Dipl.-Inf.

Alwin Hoffmann



Studium

Universität Augsburg, Deutschland

- Informatik (Diplom) mit Schwerpunkt Softwareentwicklung

Technische Universität München, Deutschland

- Finance & Information Management (M.Sc.)

Berufliche Karriere

Universität Augsburg, Deutschland

- Wissenschaftlicher Mitarbeiter am Institut für Software & Systems Engineering
- Koordination des Forschungsprojekts *SoftRobot*

Derzeitiger Arbeitgeber / Institution

Institut für Software & Systems Engineering
Universität Augsburg

<http://www.isse.de>

hoffmann@informatik.uni-augsburg.de

Objektorientierte Programmierung von Industrierobotern

Andreas Angerer, Alwin Hoffmann, Andreas Schierl, Michael Vistein und Wolfgang Reif, Institut für Software & Systems Engineering, Universität Augsburg, Augsburg

Markus Bischof, Alexander Chekler und Christian Tarragona, KUKA Roboter GmbH, Augsburg

Zusammenfassung

Im Laufe der Zeit haben viele Robotik-Unternehmen proprietäre Sprachen zur Programmierung ihrer Robotersysteme entwickelt. Diese Sprachen bieten einerseits die Möglichkeit, den Programmfluss zu definieren, andererseits werden somit die Roboterfähigkeiten dem Anwender über eine einfache Programmierschnittstelle zur Verfügung gestellt. Zudem abstrahieren die Sprachen von den steuerungsspezifischen Details wie Echtzeitanforderungen oder Bahnplanung.

Ein Problem dieses Ansatzes ist, dass die Roboterhersteller der Aufgabe gegenüberstehen, die steigende Komplexität der Anwendungen in ihrer Programmiersprache abzubilden. Allerdings ist dies aufgrund der schnell wachsenden Anforderungen eine kostenintensive Angelegenheit. Daher ist eine interessante Alternative, eine moderne, state-of-the-art Programmiersprache inkl. der entsprechenden Entwicklungsumgebung zu verwenden, die durch eine große Entwicklergemeinschaft genutzt und vorangetrieben wird. Roboterspezifika können durch den Roboterhersteller über eine geeignete Programmierschnittstelle, ein so genanntes Application Programming Interface (API), zur Verfügung gestellt werden.

Eine solche objektorientierte Programmierschnittstelle ist Gegenstand dieser Arbeit. Sie wurde in der Programmiersprache C# entwickelt und beinhaltet ein Domänenmodell zur Definition von physikalischen Objekten, Bewegungen oder Schaltaktionen innerhalb einer Roboteranwendung. Die echtzeitfähige Robotersteuerung wird durch datenflussorientiert beschriebene, komplexe Einzelaktionen angesprochen. Durch die Verwendung der modernen Sprache C# können Roboterprogramme fließend in andere Anwendungen integriert werden, die weit mehr als nur ein Roboterprogramm umsetzen. Zudem stehen moderne Sprachkonstrukte und Werkzeuge zur Verfügung, um den Ablauf der Anwendung zu modellieren.

1 Einleitung

In den letzten Jahrzehnten hat der Einsatz von Industrierobotern in der Fertigung massiv zugenommen. Vor allem in der Automobilindustrie ist eine Produktion ohne die Hilfe von Robotern inzwischen kaum noch vorstellbar, da nur durch die schnellen, hochpräzisen und exakt reproduzierbaren Verarbeitungsschritte eines Roboters die erforderliche Qualität und Produktionsgeschwindigkeit erreicht werden kann. In anderen Produktionszweigen, gerade in kleineren und mittelständischen Unternehmen, sind Roboter hingegen weit weniger verbreitet, obwohl auch diese Unternehmen von einer Automatisierung in der Fertigung profitieren könnten. In diesen Branchen sind jedoch die Anforderungen völlig anders als in der industriellen Massenfertigung (vgl. Pires, 2008). Während in der Automobilfertigung Großserien dominieren, die über Jahre hinweg unverändert und in sehr hoher Stückzahl produziert werden, sind für kleine Unternehmen flexible Systeme nötig, die sich in kürzester Zeit leicht an neue Anforderungen, wie zum Beispiel andere Werkstücke, anpassen lassen.

Ein großes Problem ist jedoch häufig die Programmierung dieser Systeme. Industrieroboter werden traditionell mit den Programmiersystemen und -sprachen der jeweiligen Hersteller programmiert. Diese weisen – gerade im Vergleich zu den heute üblichen Programmiersprachen – einen eingeschränkten Funktionsumfang auf, was nicht zuletzt der Notwendigkeit geschuldet ist, die Mechanik präzise und zuverlässig zu steuern. Man spricht in diesem Zusammenhang von echtzeitfähiger Programmierung. Mit diesen Programmiersystemen ist jedoch kaum eine Softwareentwicklung möglich, wie sie inzwischen Stand der Technik ist.

Um diese Lücke zu schließen, hat das Forschungsprojekt *SoftRobot* zum Ziel, Methoden und Werkzeuge des modernen Software Engineering in die Robotik zu transferieren. Vor allem objektorientierte Analyse, Design und Entwicklung sind hier zentrale Aspekte, um Roboterprogramme änderbar, wartbar und vielfältig einsetzbar umzusetzen. Das Projekt wird vom Institut für Software & Systems Engineering der Universität Augsburg koordiniert und gemeinsam mit der KUKA Roboter GmbH, Augsburg und der MRK-Systeme GmbH, Augsburg, durchgeführt. Gefördert wird das Projekt von der Europäischen Union und der High-Tech-Offensive Bayern der Bayerischen Staatsregierung.

Im Folgenden wird in Abschnitt 2 der aktuelle Stand der Softwareentwicklung in der Industrie sowie in der Forschung untersucht. Anschließend wird in Abschnitt 3 das Vorgehen beschrieben, welches zum Einsatz kam, um ein objektorientiertes Robotik-Domänenmodell und eine passende Software-Architektur zu entwerfen. Abschnitt 4 stellt die Ergebnisse hierzu im Detail vor. Ein Anwendungsbeispiel in Abschnitt 5 zeigt, wie weitere Konzepte und Funktionen in das Domänenmodell bzw. die Architektur integriert werden können. Schließlich wird in Abschnitt 6 ein Fazit über die bisherige Arbeit gezogen und ein Ausblick auf ausstehende Arbeiten gegeben.

2 Aktueller Stand in der Industrie und Forschung

Betrachtet man den aktuellen Stand der Programmierung und der Softwareentwicklung in der Industrierobotik, muss man zwischen industrieller Praxis und Forschung unterscheiden, vor allem weil nur selten ein Transfer zwischen beiden Welten stattfindet.

2.1 Industrie

Robotersteuerungen sind auf einen speziellen technischen Kontext – die Steuerung mechanischer Manipulatoren – zugeschnitten und haben die Aufgabe, eine hochgenaue und betriebssichere Regelung der sich bewegenden mechanischen Komponenten zu erreichen. Aus diesem Grund hat eine

Softwaresteuerung für Roboter die Charakteristika eines Echtzeitsystems. Die exakte Einhaltung harter Zeitschranken ist unabdingbar, um sowohl die wichtigsten funktionalen Merkmale eines Roboters, d.h. hohe Präzision und Wiedergenauigkeit, als auch die Betriebssicherheit eines Robotersystems zu gewährleisten. Dieser Umstand wirkt sich auf die Art der Programmierung aus, die dem Endanwender zur Anpassung eines Robotersystems an eine bestimmte Aufgabe zur Verfügung steht. So ist der Einsatz von Standardprogrammiersprachen im Hinblick auf die breit gefächerte Zielgruppe der Endanwender von Robotersystemen problematisch. Diese Sprachen bieten zwar eine weitreichende Kontrolle über die Ressourcen des Systems, aber im Falle eines Robotersystems wäre der Programmierer dafür verantwortlich, dass das von ihm erstellte Programm die wesentlichen Zeitschranken einhält. Dieser Umstand ist selbst für erfahrene Programmierer eine enorme Herausforderung und würde den Lernaufwand für unerfahrene Roboterprogrammierer drastisch erhöhen.

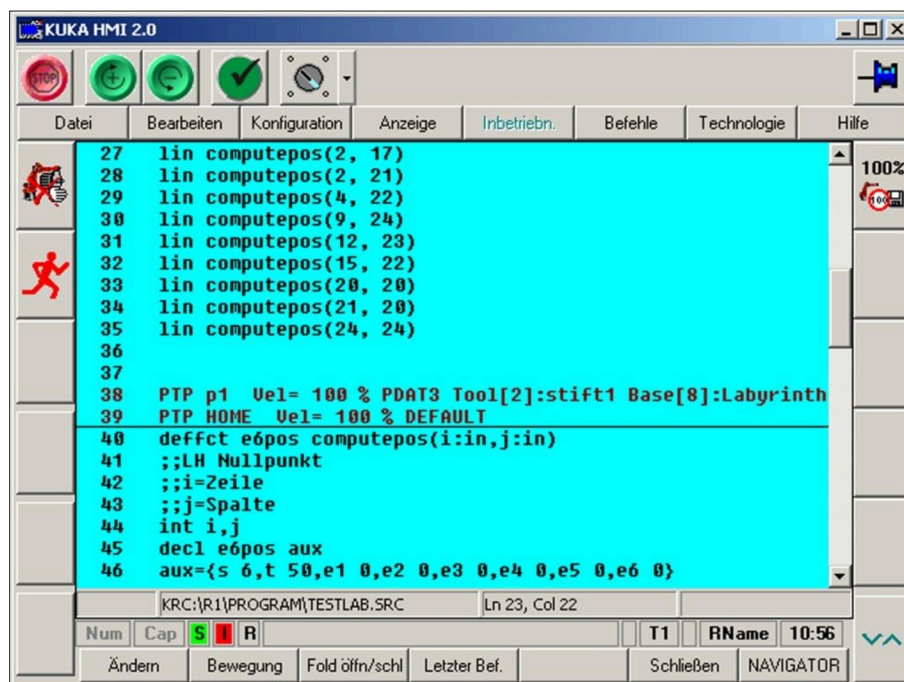


Abbildung 1: Zur Programmierung von Industrierobotern bietet jeder Hersteller eine eigene Programmiersprache. Das obige Beispiel zeigt die KUKA Robot Language im Kontext der Oberfläche des KUKA Control Panel.

Daher haben die Roboterhersteller eigene Sprachen entwickelt, die speziell auf die Bedürfnisse von Industrierobotern zugeschnitten sind. Während beispielsweise Robotersysteme der KUKA Roboter GmbH die KUKA Robot Language (KRL) unterstützen (siehe Abbildung 1), müssen entsprechende Systeme von ABB oder FANUC mit RAPID bzw. KAREL programmiert werden. Diese Sprachen werden von herstellerspezifischen Programmier- und Laufzeitumgebungen interpretiert, welche den Programmablauf schrittweise abarbeiten und die entsprechenden Befehle zur echtzeitfähigen Steuerung übertragen. Außerdem sind diese Sprachen stark roboterzentriert und abstrahieren von der dem System zugrunde liegenden Komplexität. Dadurch stellen sie eine *einfache* Programmierschnittstelle dar, um grundlegende Bewegungen und Schaltaktionen zu spezifizieren. Zwar ist dadurch der Funktionsumfang 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. Allerdings stehen Endanwender, die über keine Programmiererfahrung verfügen, dennoch vor einer beträchtlichen Einstiegshürde.

Auf der Grundlage dieser Steuerungssprachen 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. Darüber hinaus existieren Produkte der Roboterhersteller selbst oder auch von Drittanbietern zur graphischen (Offline-)Programmierung¹ von Robotersystemen. Diese Entwicklungswerkzeuge unterstützen Anwender vor allem bei der Erstellung von Bewegungsbahnen und der Spezifikation von bahn- oder zeitspezifischen Schaltaktionen. Beispiele solcher Produkte sind KUKA.Sim Pro oder das Robot Motion Center der Firma Blackbird Robotersysteme.

Allerdings führt der immer häufigere Einsatz von Robotersystemen als Teil sehr unterschiedlicher industrieller Prozesse dazu, dass Roboter oft zu einer einzelnen Komponente in einem größeren Fertigungsverbund werden. Das grundlegende Modell eines einzelnen Roboters mit einem anmontierten Werkzeug, für das viele Steuerungen ursprünglich entwickelt wurden, ist daher nicht mehr ausreichend. Essentiell ist die Interaktion mit anderen Systemen eines Verbunds, Koordination und Kooperation, die Behandlung zahlreicher systemübergreifender Ereignisse und Fehlerfälle und eine reibungslose, benutzergeführte Steuerung des Gesamtsystems.

Daher wird immer mehr dazu übergegangen, die Robotersprachen ausschließlich für die Bewegungsprogrammierung und die echtzeitfähige Kommunikation mit Werkzeugen und Sensoren einzusetzen. Die Interaktion mit Benutzern, die Kommunikation mit externen Systemen oder die Entwicklung domänenspezifischer Anwendungslogik wird dagegen in Standardprogrammiersprachen realisiert (vgl. Hoffmann et al., 2010b). Die Wechselwirkung mit dem Robotersystem ist abhängig vom Anwendungsfall: Entweder es wird Roboter Quellcode erzeugt und auf die Steuerung übertragen oder es werden bereits existierende Roboterprogramme durch das Setzen globaler Variablen parametrisiert und anschließend ferngesteuert.

Eine derartige Strukturierung, d.h. die strikte Aufteilung der Anwendungslogik in roboternahe Bewegungsprogramme und aufgabenspezifische, den Arbeitsablauf modellierende Programmteile, ist typisch für viele heute existierende Applikationslösungen in der Industrierobotik. Dementsprechend werden für viele Anwendungsszenarien in der Regel mit hohem Aufwand Speziallösungen entwickelt, die darüber hinaus softwaretechnische Defizite in Hinblick auf Flexibilität, Wiederverwendbarkeit und Erweiterbarkeit aufweisen (vgl. Hoffmann et al., 2010a). Mit steigendem Grad der Komplexität der Anwendungsdomäne, in die ein Robotersystem integriert werden soll, stoßen bestehende Lösungen häufig an ihre Grenzen.

2.2 Forschung

In der Forschung wird die Softwareentwicklung für Robotersysteme ebenfalls untersucht. Dabei konzentriert sich die Forschung aktuell jedoch weniger auf Anwendungen der Industrierobotik, da die Probleme in diesem Bereich oft als gelöst betrachtet werden (vgl. Hägele et al., 2008), sondern vielmehr auf die experimentelle Robotik, d.h. autonome Robotersysteme und Servicerobotik. Gemäß Brugali (2007) ist dieser Bereich jedoch zu umfangreich und zu inhomogen, um ihn mit einer universellen Software-Architektur abzudecken, da hier unterschiedliche Formen von Robotern in vielen An-

¹ Offlineprogrammierung bezeichnet die Erstellung und Simulation von Programmen ohne angeschlossenes Robotersystem.

wendungsbereichen für die verschiedensten Funktionen eingesetzt werden. Dennoch wurden verschiedene Gemeinsamkeiten zwischen den einzelnen Robotikanwendungen gefunden. So werden *Intelligence* (der Roboter zeigt nützliches Verhalten), *Situatedness* (er befindet sich in einer Umgebung) und *Embodiment* (er hat eine physikalische Repräsentation) als stabile Anforderungen der Robotikdomäne, auch der Industrierobotik, beschrieben (vgl. Brugali, 2007).

Aufgrund der hohen Heterogenität in der Robotik hinsichtlich der verwendeten Hardware und der zu lösenden Aufgaben wurden verschiedene, meist komponentenbasierte Frameworks vorgeschlagen und entwickelt (vgl. Brugali et al., 2007). Diese Frameworks ermöglichen die Kommunikation zwischen verschiedenen Komponenten, oder durch gemeinsame Schnittstellen sogar die Integration von Komponenten aus verschiedenen Komponentenframeworks. Allerdings vernachlässigen sie in der Regel die (häufig nötigen) Echtzeitanforderungen von Robotersystemen oder ermöglichen Echtzeitfähigkeit nur innerhalb einzelner Komponenten (z.B. Vaughan, 2007; Brooks, 2007). Andere Frameworks ermöglichen zwar eine echtzeitfähige Implementierung der gesamten Anwendung (vgl. Bruyninckx, 2001), erwarten dafür aber vom Programmierer genaue Kenntnisse über die Echtzeitprogrammierung.

Neben komponentenbasierten Ansätzen existieren auch objektorientierte Frameworks für die (industrielle) Robotik. Diese beinhalten bspw. eine Klassenhierarchie für Geräte und Werkstücke (vgl. Miller/Lennox, 1990) oder eine Einteilung in robotikspezifische Module (z.B. Sensoren, Effektoren und Algorithmen) mit der Möglichkeit zur Komposition und Vererbung (vgl. McKee et al., 2001). Eine ähnliche Aufteilung findet sich auch in Löffler et al. (2004), wo die Robotikdomäne in physikalische Objekte wie Manipulatoren und Sensoren, sowie funktionale Objekte wie Regelung und Trajektorien-Generatoren aufgeteilt wird. Weitere objektorientierte Ansätze sind ZERO++ (Pelich, 1997) und MRROC++ (Zielinski, 1997). Die Domänenmodelle dieser Ansätze beschränken sich jedoch auf den Aufbau der Roboter und lassen kaum eine komplexe Modellierung von Aufgaben bzw. der Arbeitsumgebung des Roboters zu.

3 Softwaretechnisches Vorgehen

Um die Domäne der Industrierobotik und speziell die vielfältigen Applikationen vom Palettieren über das Schutzgasschweißen bis hin zur Bauteillokalisierung zu verstehen, wurden ca. 20 der momentan vertriebenen Softwarepakete der KUKA Roboter GmbH – sogenannte *Technologiepakete* – mit Methoden der Softwaretechnik analysiert. Die Technologiepakete lassen sich in zwei Kategorien unterteilen. Durch Technologiepakete der ersten Kategorie werden konkrete industrielle Applikationen realisiert. Dazu zählen u.a. folgende Applikationen, die im Rahmen der Analyse untersucht worden sind:

Industrielle Applikationen

Palettieren

Schutzgas-, Punkt- und Laserschweißen

Laserschneiden

Kleben

Bauteillokalisierung mit Hilfe eines Kamerasystems

Bauteilvermessung

Be- und Entladen von Werkzeugmaschinen (z.B. in der Blechbearbeitung)

Durchführung von Langzeittests

Außerdem gibt es Technologiepakete, die Funktionen mit *Querschnittscharakter* bereitstellen, d.h. Funktionen die in unterschiedlichen Applikationen auftreten bzw. benötigt werden:

Querschnittsfunktionen

Werkzeugansteuerung (Greifer, Schweißzangen, etc.)

Echtzeitfähige Sensorintegration (z.B. für kraftgeführte Bewegungen)

Anlagensteuerung & -integration

Synchronisierung mit einem Förderband bzw. einer mobilen Plattform

Echtzeitsynchrone Kooperation mehrerer Roboter

Arbeitsraumüberwachung und virtuelle Wände

Bei der durchgeführten softwaretechnischen Analyse wurden sowohl funktionale als auch nicht-funktionale Anforderungen untersucht. Um die funktionalen Anforderungen zu identifizieren, wurden die Technologiepakete nach wiederkehrenden Grundfunktionen und gemeinsamen Mustern untersucht. Typische Muster sind bspw. „Bewegung mit Werkzeugschaltaktionen“, „Bewegung mit Abbruchbedingung“ oder „Bewegung mit kartesischer Sensorkorrektur“. Bei den nicht-funktionalen Anforderungen wurde ein besonderes Augenmerk auf notwendige Echtzeitanforderungen und einen deterministischen bzw. zuverlässigen Programmablauf gelegt. Außerdem wurden mehrere Akteure identifiziert und deren Interaktion mit dem Robotersystem untersucht und dokumentiert.

Ein zentrales Ergebnis dieser Analyse war die Feststellung, dass Roboterprogramme üblicherweise nicht während der gesamten Laufzeit unter echtzeitkritischen Bedingungen ausgeführt werden müssen. Zwar wird eine echtzeitfähige Steuerung benötigt, um jede einzelne Bewegung (durch die Regelung der Achsen) präzise und deterministisch zu steuern und Werkzeugaktionen exakt zu schalten, jedoch beinhalten typische Roboterprogramme einen großen Anteil an nicht-echtzeitkritischer Ablauflogik. Daher können Bewegungen und Werkzeugaktionen in eine Art *echtzeitkritische Transaktion* zusammengefasst werden, die auf einem echtzeitfähigen Betriebssystem deterministisch, zuverlässig und präzise ausgeführt wird. Die eigentliche Anwendungslogik und die Ablaufsteuerung kann unter Standardbetriebssystemen (z.B. unter Windows oder Linux) ausgeführt werden. Evtl. auftretende Verzögerungen zwischen einzelnen Transaktionen sind unproblematisch, da sie die Semantik des Roboterprogramms nicht verändern.

Diese Feststellung war die Grundlage für den Entwurf einer Softwarearchitektur, die eine klare Trennung zwischen echtzeitkritischer Steuerung der Mechanik und der Programmierung der Anwendungslogik vorsieht. Dadurch ergeben sich folgende Auswirkungen:

- Roboter können in modernen Standardprogrammiersprachen (z.B. Java, C#) programmiert werden, die durch Merkmale wie eine automatische Speicherverwaltung nur sehr eingeschränkt echtzeitfähig sind und bisher nicht zur Roboterprogrammierung verwendet werden konnten. Außerdem können Anwendungen auf Standardbetriebssystemen (z.B. Windows oder Linux) und unter Verwendung integrierter Entwicklungsumgebungen (z.B. Visual Studio, Eclipse) implementiert werden.
- Durch das Transaktionskonzept kann von der Regelungs- bzw. und Steuerungsebene, die eine echtzeitfähige Programmierung voraussetzt, abstrahiert werden. Stattdessen kann sowohl grundlegende Funktionalität eines industriellen Robotersystems als auch applikationsspezifische Funktionalität über eine wohldefinierte Programmierschnittstelle zur Verfügung gestellt werden, die von einem Anwendungsentwickler benutzt werden kann.

Diese beiden Aspekte ermöglichen es, die Bewegungssteuerung von Industrierobotern direkt in die Anwendungsprogrammierung mit einzubeziehen und so zusätzlichen Aufwand und Kosten zu vermeiden (vgl. Abschnitt 2.1).

Die Programmierschnittstelle wurde objektorientiert konzipiert, da sich die Industrierobotik sehr gut eignet, um durch ein (abstraktes) Domänenmodell dargestellt zu werden. Das heißt, dass die (Begriffs-)Welt der Industrierobotik und seiner Applikationen in ein Modell aus (Software-)Objekten mit gegenseitigen Verknüpfungen, so genannten Assoziationen, abgebildet werden kann. Diese Objekte aus der Domäne der Industrierobotik können verwendet werden, um die zu programmierende Aufgabe zu beschreiben und anschließend zu implementieren. Abstrakt bedeutet, dass das Modell erst durch die jeweilige Applikationen (z.B. Schweißen oder Palettieren) bzw. die verwendeten Geräte konkretisiert wird. Dazu werden spezialisierte Objekte von bereits vorhandenen (abstrakten) Domänenobjekten abgeleitet und mit der entsprechenden Funktionalität für die konkrete Applikation oder das konkrete Gerät ausgestattet. Während ein abstraktes Domänenobjekt bspw. ein Roboter ist, der über mehrere Achsen verfügt, werden konkrete Objekte für unterschiedliche Robotertypen (mit der richtigen Anzahl an Achsen) abgeleitet und implementiert (für Details siehe Abschnitt 5).

Um die objektorientierte Programmierschnittstelle zu entwerfen, wurden – basierend auf der bestehenden Analyse der Technologiepakete – Szenarien erarbeitet. Ein Szenario ähnelt stark einem aus der Softwaretechnik bekannten Anwendungsfall bzw. *Use Case* (vgl. Cockburn, 2001), ist aber bzgl. seiner Granularität gröber, d.h. er ist etwas allgemeiner gehalten und abstrahiert von vielen Details. Die Aufgabe eines Szenarios ist die Beschreibung einer applikationsspezifischen Interaktion mit dem System unter Berücksichtigung der bereits identifizierten Grundfunktionen und Muster, um zentrale und wichtige Konzepte der Industrierobotik herauszustellen. Darüber hinaus wurde bei der Beschreibung der Szenarien unterschieden, wie die identifizierten Akteure bei der konkreten Applikation mit dem Robotersystem interagieren.

Die durch die Szenarien identifizierten Konzepte wurden zuerst grob kategorisiert, bevor sie anschließend iterativ in ein Konzeptmodell (vgl. Larman, 2004) überführt wurden. Das Konzeptmodell besteht aus drei Bereichen, die wiederum über Konzepte und Assoziationen zusammenhängen:

- Modellierung der physikalischen Welt und der Arbeitsumgebung des Robotersystems
- Modellierung der steuerbaren Geräte, d.h. der vorhandenen Aktuatoren (z.B. Roboter, Werkzeuge) und Sensoren (z.B. Kraft-Momenten-Sensoren)
- Modellierung der verfügbaren Aktionen (z.B. Bewegungen) der jeweiligen Geräte

Bei jeder Iteration wurden weitere Konzepte aufgenommen und bereits bestehende Konzepte verfeinert bzw. angepasst. Zur Modellierung wurden dabei UML-Klassendiagramme (vgl. Fowler, 2003) verwendet. Anschließend wurde das Konzeptmodell schrittweise in ein Entwurfsmodell transferiert und prototypisch implementiert. Die dabei entstandene Plattform ist derzeit in der Lage, KUKA Leichtbauroboter (vgl. Bischoff, 2010) zu programmieren und zu steuern. Eine detaillierte Beschreibung des entwickelten, abstrakten Domänenmodells und dessen Implementierung kann in Abschnitt 4 gefunden werden.

4 Architektur

Aus den durch die Anforderungsanalyse gewonnen Erkenntnissen (vgl. Abschnitt 3) wurde eine Softwarearchitektur entwickelt, die detailliert bei Hoffmann et al. (2009) beschrieben und in Abbildung 2 dargestellt ist. Zentraler Bestandteil der Architektur ist die Programmbibliothek *Robotics Base Class Libraries (RBCL)*, welche dem Anwendungsprogrammierer über das objektorientierte *Robotics Application Programming Interface (Robotics API)* alle notwendigen Mittel zur Programmierung von Indust-

rierobotern zur Verfügung stellt. Roboterbefehle, die mit den Mitteln der Robotics API definiert wurden, werden automatisch von der RBCL zur Robotersteuerung übertragen, wo die Ansteuerung und Regelung der Hardware stattfindet. Damit Garantien bezüglich der Genauigkeit und Zuverlässigkeit möglich sind, muss dieser Teil der Architektur echtzeitfähig implementiert sein.

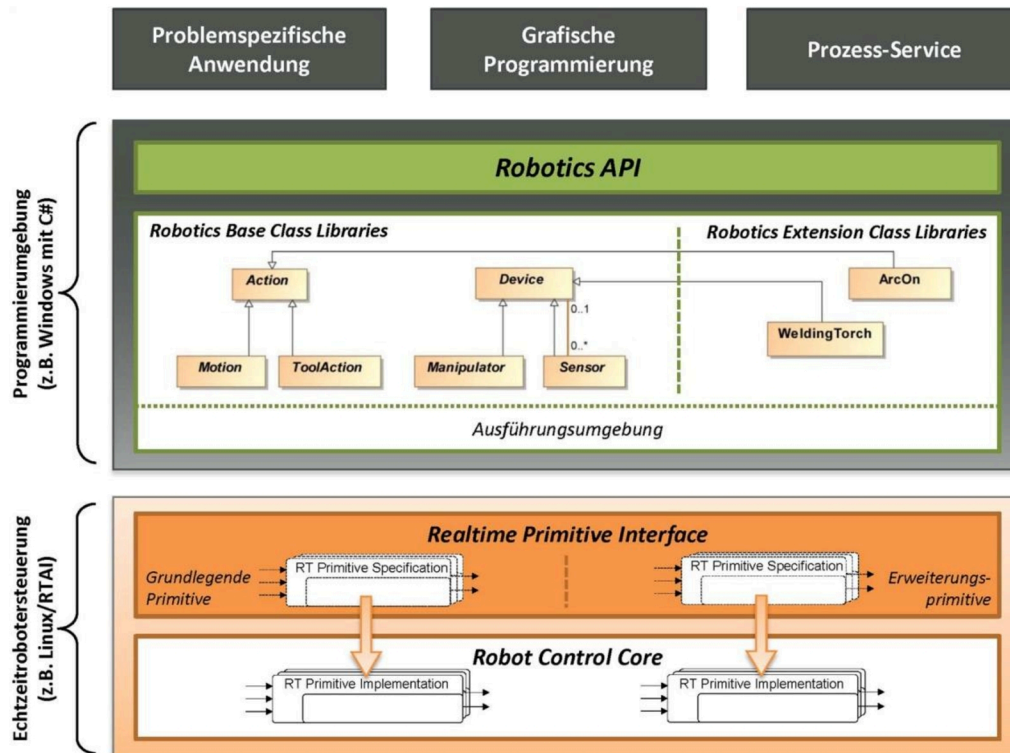


Abbildung 2: Die im Forschungsprojekt „SoftRobot“ entwickelte Architektur mit drei Schichten zur Trennung von echtzeitkritischer Robotersteuerung, objektorientierter Programmierung und problem-spezifischer Anwendung

Um eine generische und erweiterbare Programmierschnittstelle zu gewährleisten, ist eine ausdrucks-mächtige und ebenso erweiterbare Schnittstelle zur Robotersteuerung notwendig. Diese Schnittstelle, das *Realtime Primitive Interface* (RPI), ermöglicht es, atomare und echtzeitkritische Aufgaben für einen oder mehrere Roboter (z.B. Bewegungen, Interaktion mit Werkzeugen und Sensoren) zu spezifi-zieren und auf der Robotersteuerung auszuführen. Die Befehle werden dazu aus einer Menge von Modulen zusammengesetzt, deren Interaktion in einer Datenflusssprache beschrieben ist. In den nächsten beiden Abschnitten werden die beiden wichtigsten Komponenten der Architektur – die Robotics API und das Realtime Primitive Interface – detailliert beschrieben.

4.1 Realtime Primitive Interface

Das *Realtime Primitive Interface* (RPI) erlaubt die Spezifikation von echtzeitkritischen Aufgaben mit-tels einer Datenflusssprache und definiert eine Schnittstelle, um solche Aufgaben an eine kompatible Robotersteuerung zu übertragen. Dort werden diese Aufgaben als Transaktionen unter Echtzeitbe-dingungen ausgeführt.

In RPI existieren zur Beschreibung von Aufgaben *Realtime Primitives* (Primitive) und *Links* (Verbin-dungen). Während ein Realtime Primitive ein atomarer Funktionsblock mit Ein- und Ausgängen ist,

der eine in der echtzeitfähigen Robotersteuerung implementierte Funktionalität repräsentiert, stellt ein Link eine Verbindung zwischen Ein- und Ausgängen von Instanzen von Realtime Primitives dar. Beide Konzepte sind wie folgt definiert:

- Ein *Realtime Primitive* ist eine Abbildung einer Menge von n typisierten Eingabewerten auf eine Menge von m typisierten Ausgabewerten mit $n, m \in \mathbb{N}_0$. Im Besonderen sind Primitive ohne Eingabewerte oder ohne Ausgabewerte zulässig. Typische Beispiele sind Primitive für die achsspezifische Steuerung eines Roboters, Primitive für die Transformation von Koordinatensystemen oder Primitive für die Berechnung von Trajektorien.
- Ein *Link* verbindet einen Ausgang eines Realtime Primitives mit einem Eingang eines anderen. Ein Ausgang kann über mehrere Links mit mehreren Eingängen verbunden werden, während ein Eingang nur mit einem Link verbunden werden kann. Die Ein- und Ausgänge sind statisch typisiert. Daher können nur Links zwischen Ein- und Ausgängen gleichen Typs definiert werden. Neben den typischen Datentypen (Boolean, Integer, Double und String) existieren auch komplexere, robotikspezifische Datentypen (z.B. homogene Transformationsmatrizen).

Primitive und deren Links bilden einen Graph. Beinhaltet dieser keine offenen Ein- und Ausgänge, kann er als sogenanntes *Realtime Command* ausgeführt werden. Dem Transaktionskonzept folgend ist ein *Realtime Command* in sich abgeschlossen, d.h. das System befindet sich vor und nach dessen Ausführung in einem stabilen und konsistenten Zustand. Die Ausführung erfolgt zyklisch in einer kompatiblen Robotersteuerung, d.h. in jedem Takt wird der ganze Graph einmal ausgewertet: Jedes Primitive liest seine Eingänge, führt die entsprechenden internen Berechnungen/Aktionen aus und schreibt anschließend seine Ausgänge. Da die Graphen im Wesentlichen azyklisch sind, kann die Auswertung linearisiert werden, so dass eine optimale sequentielle Ausführungsordnung der Primitive bestimmt werden kann. Außerdem kann so die maximale Ausführungszeit (*Worst Case Execution Time*) eines Kommandos berechnet werden. Für die Erweiterung der unterstützten Hardware und Steuerungskonzepte erlaubt RPI die Definition und Integration neuer Primitive.

Eine kompatible Robotersteuerung, ein sogenannter *Robot Control Core*, muss einen Interpreter für Realtime Commands und Implementierungen der benötigten Primitive bereitstellen. Die Robotersteuerung ist für die Steuerung der Robotermechanik und der weiteren Geräte zuständig, d.h. sie muss bspw. mit dem Roboter kommunizieren, um diesem neue Sollwerte für seine Achsen zu übermitteln. Sie stellt die echtzeitkritische Funktionalität des gesamten Systems zur Verfügung und muss daher ausreichend performant auf einem Echtzeitbetriebssystem implementiert sein.

4.2 Robotics API

Das objektorientierte *Robotics Application Programming Interface (Robotics API)* stellt die Programmierschnittstelle für die Entwicklung von Robotikanwendungen dar. Für die Programmierung von Industrierobotern und deren Werkzeugen bietet die Robotics API eine umfangreiche Klassenhierarchie, die grundlegende Funktionalität bereitstellt und gleichzeitig für eigene Applikationen und Einsatzbereiche um neue Robotertypen und Werkzeuge bzw. Bewegungen und Aktionen erweiterbar ist.

Im Rahmen der objektorientierten Gestaltung der Robotics API wurde die Domäne der Industrierobotik so detailliert wie möglich durch (Software-)Objekte und Assoziationen abgebildet. Dieses Objektmodell ist in Abbildung 3 dargestellt und besteht analog zu Abschnitt 3 aus drei größeren Bestandteilen, die die Betrachtungsweise widerspiegeln, dass ein Robotersystem (d.h. Roboter, Werkzeuge und Sensoren) mit seiner (Arbeits-)Umgebung interagiert. Das Gerätemodell (*Device Model*) findet sich unten rechts in der Abbildung und repräsentiert die in der Robotersteuerung verfügbaren Geräte.

Das Weltmodell (*World Model*) ist unten links dargestellt und bietet Klassen an, um sowohl die steuerbaren Geräte als auch deren Umwelt bzw. deren Arbeitsumgebung zu modellieren. Das Aktionsmodell (*Command Model*) wird in der oberen Hälfte der Abbildung gezeigt und repräsentiert die potentiellen Fähigkeiten der steuerbaren Geräte, mit denen sie mit ihrer Umwelt interagieren können.

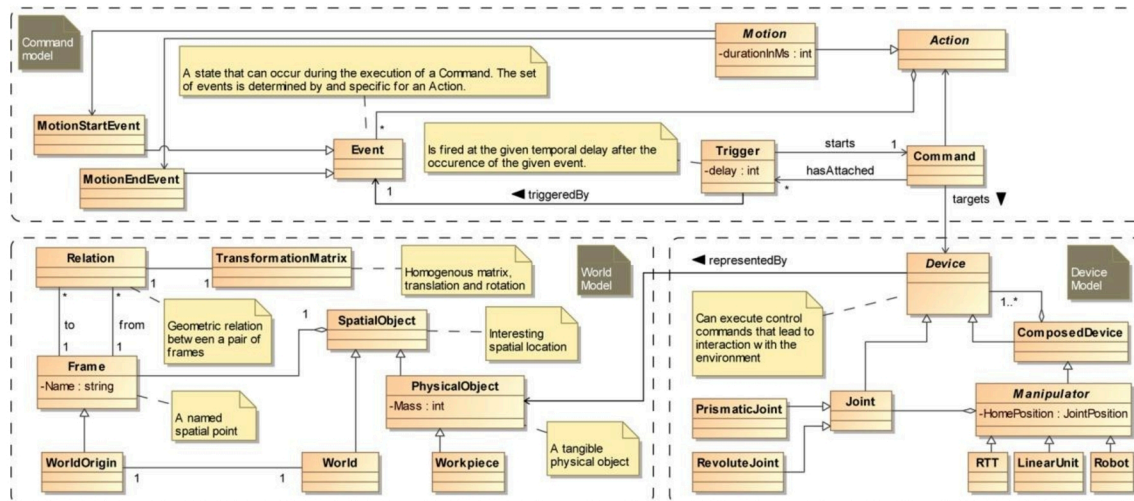


Abbildung 3: Die grundlegende Klassenhierarchie der Robotics API

Diese drei Modelle stellen alle Bestandteile (d.h. Objekte) bereit, um Roboteranwendung objektorientiert zu programmieren:

- Das Gerätemodell (*Device Model*) repräsentiert die in der Robotersteuerung verfügbaren und dadurch steuerbaren Einheiten. Dazu zählen vor allem Roboter und Werkzeuge, aber auch Sensoren, Lineareinheiten oder mobile Plattformen. Eine steuerbare Einheit wird durch die Klasse *Device* repräsentiert, von der weiter abgeleitet werden kann, um konkrete Ausprägungen zu definieren. Eine Besonderheit ist das *ComposedDevice*, das aus mehreren steuerbaren Einheiten besteht. Einfachsten Beispiel dafür ist ein Roboter, der aus einzelnen Achsen (*Joints*) besteht.
- Die physikalischen Eigenschaften von steuerbaren Geräten sind neben der Modellierung weiterer Elemente der Arbeitsumgebung Teil des Weltmodells (*World Model*). Dadurch ist es möglich, real vorhandene Bestandteile (*PhysicalObjects*) der Arbeitsumgebung (z.B. Werkstücke) explizit in Software darzustellen und die für die Aufgabe wichtigen, applikationsspezifischen Informationen zu modellieren. Ein weiterer Teil des Weltmodells umfasst die Modellierung geometrischer Informationen, die zur Definition von Roboterbewegung unabdingbar sind. Dazu gehören Koordinatensysteme (*Frame*), Beziehungen zwischen Koordinatensystemen (*Relation*), Transformationen (*TransformationMatrix*) und ein eindeutiger, geometrischer Ursprung (*WorldOrigin*) der modellierten Welt (*World*).
- In der objektorientierten Programmierung führt jedes Objekt seine Fähigkeiten in Form von Methoden mit sich. In der Robotics API sind diese Fähigkeiten allerdings als eigene Objekte modelliert (vgl. das „Command Pattern“ in Gamma et al., 1994). Aktionen (*Actions*) sind abstrakte Beschreibungen einer Fähigkeit und werden über ein Kommando (*Command*) an ein konkretes steuerbares Gerät gebunden. Durch die Modellierung der Fähigkeiten als einzelne Objekte ist es möglich, aus einzelnen Fähigkeiten größere und komplexere Fähigkeiten zu komponieren, die dem Transaktionskonzept folgend atomar als *Realtime Command* in der Robotersteuerung ausgeführt werden. Diese Verknüpfung findet durch Ereignisse (*Events*)

statt. Jede Aktion weist spezifische Ereignisse auf. Bei einer Bewegung tritt bspw. am Start und am Ende der Bewegung jeweils ein Ereignis (*MotionStartEvent* bzw. *MotionEndEvent*) auf. Durch so genannte *Trigger* können neue Aktionen bzw. Kommandos an ein bestehendes Kommando angefügt werden.

Im Prototyp ist die Robotics API durch die *Robotics Base Class Libraries (RBCL)* in der Programmiersprache C# implementiert und kann bis zu zwei KUKA Leichtbauroboter synchron oder unabhängig voneinander steuern. Eine zentrale Aufgabe der RBCL ist die Bereitstellung von entsprechenden Objekten für die in der Robotersteuerung verfügbaren Roboter, Werkzeuge und Sensoren. Zudem bildet sie diese Domänenobjekte und ihre Fähigkeiten (Actions) zur Ausführung auf echtzeitkritische Transaktionen ab, die oben beschriebenen Realtime Commands. Applikations-spezifische Erweiterungen der Robotics API werden durch sogenannte *Robotics Extension Class Libraries (RECL)* implementiert. Solche Erweiterungsbibliotheken verwenden dabei die (abstrakten) Klassen der RBCL und erweitern bzw. spezialisieren sie entsprechend den Anforderungen. Ein Beispiel für eine solche Erweiterung wird in Abschnitt 5 vorgestellt.

Dadurch ist die Robotics API ein erweiterbares Framework für die High-Level-Programmierung von Anwendungen für Industrieroboter. Durch die Bereitstellung von grundlegenden Konzepten für die Modellierung der Umgebung, der steuerbaren Geräte und ihrer Fähigkeiten, unterstützt die Robotics API die Wiederverwendung von Funktionalität und domänenspezifischer Logik. Im Gegensatz zu der heutigen Roboter-zentrierten Programmierung unterstützt die Robotics API das Modellieren der Arbeitsräume und der zu bearbeitenden Werkstücke durch Softwareobjekte. Außerdem ist durch die Verwendung von Standardtechnologien ein Anstieg der Softwarequalität bei Roboteranwendungen zu erwarten (vgl. Hoffmann, 2010a).

5 Anwendungsbeispiel

Um die Vorteile der Robotics API bei der Programmierung industrieller Anwendungen zu evaluieren, wurde eine exemplarische Implementierung einer Schweißanwendung entworfen. Diese Anwendung basiert auf KUKA CR.ArcTech Digital (vgl. KUKA, 2008), dem Technologiepaket für Schutzgasschweißen der KUKA Roboter GmbH. Für die Evaluierung wurde Schweißen gewählt, da es eine typische industrielle Applikation von Robotersystemen darstellt und einige anspruchsvolle Aspekte beinhaltet:

- Konfiguration und Steuerung eines Industrieroboters und einer Schweißzange
- Definition von Schweißpunkten und -bahnen, die vom zu bearbeiteten Werkstück abhängen
- Definition des Arbeitsablaufs inkl. der Bewegungen und der damit synchronisierten Werkzeugaktionen

Während der Implementierung wurde die Robotics API um Klassen erweitert, die für die Programmierung einer Schutzgasschweißanwendung notwendig sind. Diese Erweiterungsklassen sind in Abbildung 4 dunkel dargestellt, die Basisklassen der Robotics API weiterhin hell. Die Klassenhierarchie ist analog zu Abbildung 3 strukturiert.

Für diese Anwendung wurde das Gerätemodell der Robotics API um einen Schweißbrenner (*WeldingTorch*) und einen speziellen Schweißroboter (*WeldingRobot*) erweitert. Der Schweißbrenner ist als *ComposedDevice* modelliert, d.h. er ist aus mehreren digitalen bzw. analogen Ein- und Ausgängen (vgl. *IODevice*) zusammengesetzt. Die Klasse *WeldingTorch* repräsentiert so das Domänenobjekt „Schweißbrenner“ und stellt eine Schnittstelle zur Konfiguration und zur Steuerung des Schweißbrenners bereit. Die Klasse *WeldingRobot* ist ebenfalls als *ComposedDevice* modelliert und aggregiert Instanzen eines Robot und eines *WeldingTorch*, die den am Roboterflansch montierten

Schweißbrenner repräsentiert. Der Schweißroboter kann wie ein einfacher Roboter seinen Arm bewegen, hat darüber hinaus aber die Fähigkeit, eine komplexe Schweißoperation auf einer definierten Schweißnaht (*WeldingLine*) und mit domänenspezifischen Schweißparametern (*WeldingParameters*) auszuführen.

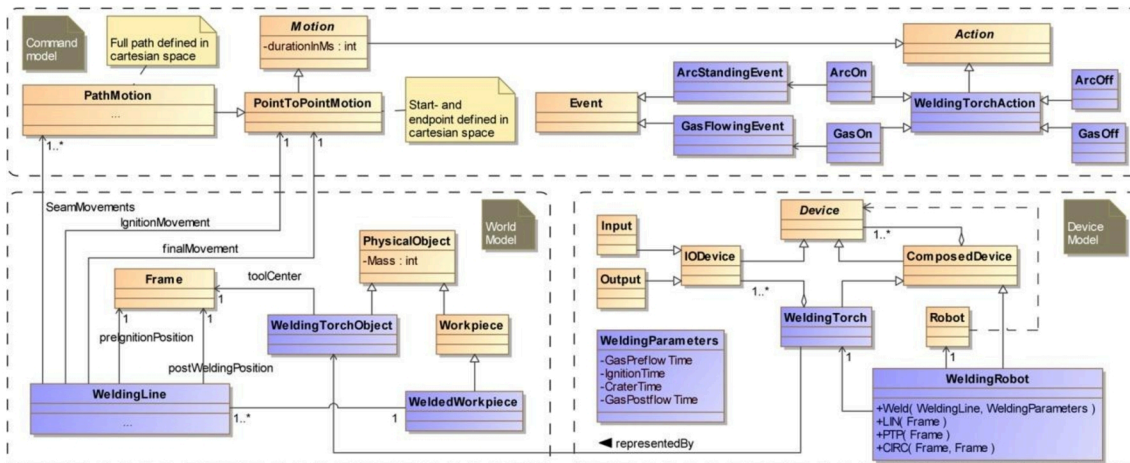


Abbildung 4: Die Klassenhierarchie einer Schweißapplikation basierend auf der Robotics API

Diese Schweißoperation ist in der Methode *Weld()* der Klasse *WeldingRobot* implementiert und bildet den kompletten Schweißvorgang einer Naht ab, d.h.:

1. Bewegung mit dem Schweißbrenner an den Startpunkt der Naht bei gleichzeitigem Einschalten des Schutzgases
2. Einschalten des Lichtbogens, nachdem eine vordefinierte Gasvorströmzeit vergangen ist
3. Beginn der Bewegung entlang der Schweißnaht, sobald der Lichtbogen steht
4. Kontinuierliche Durchführung aller weiteren notwendigen Bewegungen, um die Schweißnaht komplett abzufahren
5. Sofortiges Ausschalten des Lichtbogens bei der Endkraterposition der Naht
6. Ausschalten des Schutzgases, nachdem eine vordefinierte Gasnachströmzeit vergangen ist
7. Bewegung mit dem Schweißbrenner zu einer Position über dem Endpunkt der Naht

Die informelle Beschreibung dieser Operation zeigt, dass die Ausführung der einzelnen Schritte echtzeitkritisch ist, d.h. die Abfolge der Bewegungen, des Ein- und Ausschaltens des Lichtbogens bzw. des Schutzgases muss ohne unerwünschte Verzögerung erfolgen, um eine qualitativ hochwertige Schweißnaht zu erhalten. Daher wird diese komplexe Operation als einzelne, echtzeitkritische Transaktion abgebildet und als ein *Realtime Command* auf der Robotersteuerung ausgeführt. Um eine solche Transaktion in der Robotics API zu spezifizieren, können die elementaren Fähigkeiten bzw. Aktionen (*Actions*) eines Roboters (Bewegungen) und des Schweißbrenners (Ein- und Ausschaltens des Lichtbogens bzw. des Schutzgases) miteinander durch Ereignisse (*Events*) synchronisiert werden. Diese Aktionen (z.B. *ArcOn* oder *GasOff*) und Ereignisse (z.B. *GasFlowingEvent*) sind in der oberen Hälfte von Abbildung 4 dargestellt. Desweiteren ist es möglich, komplexe Fehlerbehandlungsstrategien (z.B. bei Zündfehlern) zu integrieren.

Nachdem die grundlegenden Klassen für das Schutzgasschweißen einmal (im Idealfall unter Hinzunahme eines Domänenexperten) entworfen und implementiert worden sind, kann eine Schweißanwendung als Abfolge von Schweißoperation (mit jeweils unterschiedlicher Parametrisierung) und geeigneter Transferbewegungen zwischen den einzelnen Nähten programmiert werden.

Der Anwendungsentwickler muss dementsprechend nicht mehr die elementaren Operationen unter Berücksichtigung von Echtzeitkriterien programmieren, da die benötigte Funktionalität bereits vollständig abgebildet und in einer RECL mit entsprechenden Domänenklassen gekapselt ist.

6 Fazit

Die derzeit existierenden, proprietären Roboterprogrammiersprachen müssen sich den gestiegenen Anforderungen heutiger Roboteranwendungen stellen. Ihnen fehlt allerdings oft die notwendige Flexibilität und Funktionalität, um den Anforderungen gerecht zu werden. Im Forschungsprojekt *SoftRobot* wird diesem Aspekt mit Hilfe einer modernen, objektorientierten Architektur begegnet, welche in einer Standardprogrammiersprache umgesetzt wurde. Dadurch kann die Entwicklungseffizienz gesteigert werden, da die Roboter-Anwendungen dieselben Techniken und Werkzeuge verwenden, welche sich auch bei Standard-Anwendungen bewährt haben. So lassen sich bspw. verschiedene Standardtechnologien für die Entwicklung von Benutzerschnittstellen oder die Datenhaltung direkt verwenden. Außerdem ist durch das objektorientierte Domänenmodell eine größere Wiederverwendbarkeit von domänenspezifischen Programmierlösungen zu erwarten.

Mit Hilfe der entwickelten Architektur mit Robotics API, RBCL und RPI, wurde eine Basis geschaffen, um qualitativ hochwertige Roboteranwendungen zu erstellen, welche sich auf die generischen Schnittstellen der Schichten stützen. Die Flexibilität wird durch das präsentierte Anwendungsbeispiel sichtbar, welches aufzeigt, wie die Architektur um spezifische Funktionen und Domänenobjekte erweitert werden kann. Die entwickelte Architektur wurde prototypisch implementiert und ermöglicht eine Steuerung von KUKA Leichtbaurobotern. Die Erkenntnisse aus dieser Implementierung zeigten, dass der entwickelte Ansatz realisierbar und effektiv ist, und fließen in die weitere Entwicklung der Architektur und der prototypischen Implementierung ein.

Um den verschiedenen Anforderungen von Roboterprogrammierern und -anwendern Rechnung zu tragen, wird im Forschungsprojekt *SoftRobot* darüber hinaus die Entwicklung von Anwendungen und Komponenten betrachtet, die auf die Robotics API aufbauen und exemplarische, speziell zugeschnittene Branchenlösungen darstellen. Außerdem ist geplant, eine grafische Benutzerschnittstelle zu entwickeln, die einfach an die jeweilige (industrielle) Applikation anpassbar ist und mit Hilfe derer auch Menschen ohne Programmierkenntnisse einfach und schnell Roboterprogramme umsetzen können. Die Entwicklung solcher Lösungen sollte dabei wesentlich einfacher, schneller und damit kostengünstiger als bisher sein, da komplett auf Standardtechnologien gesetzt werden kann.

Literatur

- Bischoff, R., Kurth, J., Schreiber, G., Koeppel, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G., Hirzinger, G.: *The KUKA-DLR Lightweight Robot arm – A new reference platform for robotics research and manufacturing*; Proc. Intl. Symposium on Robotics, München, 2010
- Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: *ORCA: A component model and repository*; In Brugali, D. (Hrsg.): *Software Engineering for Experimental Robotics*, Springer Tracts in Advanced Robotics, Springer, Berlin Heidelberg: 2007, S. 231–251
- Brugali, D. (Hrsg.): *Software Engineering for Experimental Robotics*; Springer Tracts in Advanced Robotics, Springer, Berlin Heidelberg: 2007

- Bruyninckx, H.: *Open Robot Control Software: The OROCOS Project*; Proc. 2001 IEEE Intl. Conference on Robotics and Automation, Seoul, Korea: Mai 2001, S. 2523-2528
- Cockburn, A.: *Writing effective use cases*; Addison-Wesley, Upper Saddle River, NJ, USA: 2001
- Fowler, M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*; Addison-Wesley, Upper Saddle River, NJ, USA: 2003
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley, Upper Saddle River, NJ, USA: 1994
- Hägele, M., Nilsson, K., Pires, J. N.: *Industrial Robotics*; In: Siciliano, B., Khatib, O. (Hrsg.): *Springer Handbook of Robotics*, Springer, Berlin Heidelberg: 2008, Kapitel 42, S. 963-986
- 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 Intl. Conference on Intelligent Robots and Systems, St. Louis, MO, USA: Oktober 2009, S. 2108-2113
- Hoffmann, A., Angerer, A., Schierl, A., Vistein, M., Reif, W.: *Software Engineering in der Industrierobotik: der SoftRobot-Ansatz*; Entwurf komplexer Automatisierungssysteme 2010, Magdeburg: Mai 2010
- Hoffmann, A., Angerer, A., Schierl, A., Vistein, M., Reif, W.: *Towards Object-Oriented Software Development for Industrial Robots*; Proc. 7th Intl. Conference on Informatics in Control, Automation and Robotics, Funchal, Portugal: Juni 2010
- KUKA Roboter GmbH (Hrsg.): *KUKA CR.ArchTech Digital 2.0*; Augsburg: 2008
- Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*; Prentice Hall, Upper Saddle River, NJ, USA: 2004
- Loffler, M., Chitrakaran, V., Dawson, D.: *Design and Implementation of the Robotic Platform*; Journal of Intelligent and Robotic System, Jgg 39, 2004, S. 105-129
- McKee, G., Fryer, J. A., Schenker, P.: *Object-Oriented Concepts for Modular Robotics Systems*; Proc. 39th Intl. Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, USA: Juli 2001, S. 229-238
- Miller, D., Lennox, R. C.: *An Object-Oriented Environment for Robot System Architectures*; Proc. 1990 IEEE Intl. Conference on Robotics and Automation, Cincinnati, USA: 1990, S. 352-361
- Pelich, C., Wahl, F.: *ZERO++: An OOP environment for multiprocessor robot control*; Intl. Journal of Robotics and Automation, Jgg. 12 (2), 1997, S. 49-57

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

Vaughan, R., Gerkey, B.: *Reusable Robot Software and the Player/Stage Project*; In Brugali, D. (Hrsg.): *Software Engineering for Experimental Robotics*, Springer Tracts in Advanced Robotics, Springer, Berlin Heidelberg: 2007, S. 267-289

Zielinski, C.: *Object-oriented robot programming*; Robotica, Jgg. 15 (1), 1997, S. 41-48

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.