

Eclipse als Werkzeug zur objektorientierten Roboterprogrammierung

Andreas Angerer, Andreas Schierl, Christian Böck, Alwin Hoffmann, Michael Vistein, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Angerer, Andreas, Andreas Schierl, Christian Böck, Alwin Hoffmann, Michael Vistein, and Wolfgang Reif. 2012. "Eclipse als Werkzeug zur objektorientierten Roboterprogrammierung." In *Tagungsband: Internationales Forum Mechatronik 2012, Mayrhofen, Austria, November 21-22, 2012*, 33–49. Innsbruck: Standortagentur Tirol.

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/>



Eclipse als Werkzeug zur objektorientierten Roboterprogrammierung

**ANGERER A., SCHIERL A., BÖCK C.,
HOFFMANN A., VISTEIN M., REIF W.**



M.Sc.

ANDREAS ANGERER

Studium

Universität Augsburg

- B.Sc. Informatik und Informationswirtschaft

Universität Augsburg, TU München, LMU München

- M.Sc. Software Engineering

Berufliche Karriere

KUKA Roboter GmbH, 2007-2008

- Softwareentwickler, Bereich Vorentwicklung

Universität Augsburg, 2008-heute

- Wissenschaftlicher Mitarbeiter, Mechatronik & Robotik

Derzeitiger Arbeitgeber / Institution

Institut für Software & Systems Engineering, Universität Augsburg

<http://www.isse.de>

angerer@informatik.uni-augsburg.de

Eclipse als Werkzeug zur objektorientierten Roboterprogrammierung

**ANGERER A., SCHIERL A., BÖCK C., HOFFMANN A., VISTEIN M., REIF W.
INSTITUT FÜR SOFTWARE & SYSTEMS ENGINEERING, UNIVERSITÄT AUGSBURG**

Im Rahmen des Forschungsprojekts SoftRobot wurde eine Softwarearchitektur entwickelt, mit der Industrieroboter mit der Programmiersprache Java programmiert werden können. Die Grundlage dafür stellt das objektorientierte Robotics Application Programming Interface (Robotics API) dar. Durch die Wahl der Programmiersprache Java steht ein sehr großes Ökosystem an Bibliotheken und Entwicklungswerkzeugen zur Verfügung. Mit der Eclipse-Plattform existiert eine sehr reichhaltige und leicht erweiterbare Entwicklungsumgebung. Im Rahmen des SoftRobot-Projekts wurde eine umfangreiche Erweiterung (Plugin) für Eclipse geschaffen, welche alle relevanten Abläufe bei der Erstellung von Roboterapplikationen unterstützt. Als Vorbild dienten dabei zum Teil die Funktionen, die die KUKA Roboter GmbH in den Bediengeräten für ihre Roboter anbietet. Einige Funktionen des Robotics API Eclipse-Plugins gehen jedoch darüber hinaus. Außerdem stehen durch die Verwendung von Eclipse automatisch einige Werkzeuge und Funktionalitäten zur Verfügung, die in der Softwareentwicklung Stand der Technik sind, aber in der Roboterprogrammierung oft noch vermisst werden: So sind Syntax-Highlighting, Code-Vervollständigung und das sofortige Anzeigen von Syntaxfehlern Standard. Aber auch die Entwicklung in Teams vereinfacht Eclipse durch die sehr gute Integration von Versionsverwaltungssystemen (z.B. SVN und Git) und die Unterstützung der Erstellung und automatisierten Durchführung von Unit-Tests. In diesem Papier werden die Konzepte und Funktionen des Robotics API Eclipse Plugins vorgestellt.

Inhaltsverzeichnis

- 1. Einleitung**
- 2. Die SoftRobot-Architektur**
 - 2.1. Realtime Primitives Interface
 - 2.2. Robotics API
- 3. Das Robotics API Eclipse-Plugin**
 - 3.1. Struktur der Eclipse-Plattform
 - 3.2. Erweiterungen durch das Robotics API Plugin
 - 3.2.1. Projekttyp „RoboticsAPI Project“
 - 3.2.2. Erweiterungen des Project Explorer
 - 3.2.3. Configuration-Editor, Frames-Editor und Frames View
 - 3.2.4. Handverfahren mit dem Jogging View
 - 3.2.5. Datenanzeige mit Joint Monitor und Cartesian Monitor
- 4. Zusammenfassung**
- 5. Literaturverzeichnis**

Eclipse als Werkzeug zur objektorientierten Roboterprogrammierung

ANGERER A., SCHIERL A., BÖCK C., HOFFMANN A., VISTEIN M., REIF W.
INSTITUT FÜR SOFTWARE & SYSTEMS ENGINEERING, UNIVERSITÄT AUGSBURG

1. Einleitung

Durch ihre mechanische Struktur und die daraus resultierende freie Programmierbarkeit sind Industrieroboter höchst flexible Systeme. Ausgestattet mit den passenden Werkzeugen können Roboter eine Vielzahl unterschiedlichster Arbeiten ausführen. Gerade in kleineren und mittelständischen Unternehmen, sind Roboter hingegen wenig verbreitet, obwohl gerade diese Unternehmen von einer Automatisierung in der Fertigung profitieren könnten. Oft ist jedoch die aufwändige Programmierung dieser Systeme ein Ausschlusskriterium: 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 und wenig Strukturierungsmöglichkeiten auf. Im Gegenzug wird garantiert, dass die Mechanik präzise und zuverlässig gesteuert wird. Man spricht in diesem Zusammenhang von echtzeitfähiger Programmierung. Mit diesen Programmiersystemen ist jedoch kaum Softwareentwicklung möglich, wie sie inzwischen Stand der Technik ist.

Daher ist das Forschungsprojekt SoftRobot angetreten, die Methoden und Werkzeuge des modernen Software Engineering, beispielsweise die objektorientierte Entwicklung, in die Robotik zu transferieren. Das Projekt wurde gemeinsam vom Institut für Software & Systems Engineering der Universität Augsburg, der KUKA Laboratories GmbH und der MRK-Systeme GmbH durchgeführt und von der Europäischen Union und der High-Tech-Offensive Bayern der Bayerischen Staatsregierung gefördert. In ihrer Funktion als Projektträger unterstützte die VDI-VDE-IT

GmbH das Projekt. Im Rahmen des Projekts wurde eine neuartige Softwarearchitektur entwickelt, deren zentraler Bestandteil, das Robotics Application Programming Interface (Robotics API), alle notwendigen Mittel zur objektorientierten Programmierung von Industrierobotern zur Verfügung stellt. Die Robotics API steht in der Programmiersprache Java zur Verfügung und bietet eine umfangreiche Klassenhierarchie, die eine Beschreibung wichtiger Bestandteile der physikalischen Welt beinhaltet und die Interaktion mit Robotern, Werkzeugen und Sensoren ermöglicht. Durch Abstützung auf eine erweiterbare, echtzeitfähige Robotersteuerung können Roboter flexibel programmiert und gleichzeitig präzise gesteuert werden.

Neben der eigentlichen Programmiersprache ist das Ökosystem von Software-Tools und Bedienfunktionen, das für die Robotersteuerungen der verschiedenen Hersteller heute existiert, ein wichtiger Faktor für die Benutzbarkeit und Integrierbarkeit eines Robotersystems. Dazu zählen zum einen die Programmierhandgeräte (z.B. KUKA smartPAD¹), die dem Benutzer etwa die folgenden Funktionalitäten anbieten:

- Kalibrieren des Roboters
- Handverfahren des Roboterarms
- Einlernen von Werkzeugen und Koordinatensystemen
- Einlernen von Prozess-Punkten für den Roboter
- Steuerung angeschlossener Geräte (z.B. Roboterwerkzeuge)
- Programmierung des Roboterarms
- Testläufe von Roboterprogrammen
- Ausführen von Roboterprogrammen in verschiedenen Betriebsmodi

Zum anderen existieren verschiedene Offline-Programmierungsumgebungen für Robotersysteme verschiedener Hersteller, teils von den Herstellern selbst (z.B. ABB RobotStudio², KUKA.WorkVisual³) oder auch von Drittanbietern (z.B. RobotMotionCenter von Blackbird Robotersysteme GmbH⁴). Diese bieten zum Beispiel:

- Definition und Verwaltung von Koordinatensystemen
- Werkstück-basierte Definition/Generierung von Roboterprogrammen
- Ansteuerung angeschlossener Geräte
- Integration des Roboters in Automatisierungszellen

Für die im Projekt SoftRobot neu entwickelte Robotersteuerungs-Architektur existiert ein solches Ökosystem zunächst nicht. Da viele Offline-Programmierungsumgebungen Robotersteuerungscode

¹ <http://www.kuka-robotics.com/germany/de/products/controllers/smartPAD/>

² <http://www.abb.de/product/seitp327/78fb236cae7e605dc1256f1e002a892c.aspx>

³ http://www.kuka-robotics.com/germany/de/products/software/engineering_environment/

⁴ http://www.blackbird-robotics.de/download/BB_OfflineSolutions.pdf#page=1

generieren, könnten sie grundsätzlich um die Unterstützung der SoftRobot-Architektur erweitert werden. Andere, grundsätzliche Funktionen wie das Kalibrieren eines Roboters oder auch das Handverfahren könnten dadurch jedoch nicht direkt realisiert werden.

Im SoftRobot-Projekt wurde jedoch ein anderer Ansatz untersucht: Es wurde basierend auf der Standard-Softwareentwicklungsumgebung Eclipse ein Prototyp eines Plugins entwickelt, das diese Entwicklungsumgebung um Robotik-spezifische Grundfunktionalitäten erweitert. Dies hat den Vorteil, dass für die Erweiterungen bereits auf sehr mächtige Schnittstellen und Software-Bibliotheken zurückgegriffen werden kann. Darüber hinaus ist Eclipse auch für die Java-Entwicklung sehr weit verbreitet und gehört oft zum Standard-Repertoire eines erfahrenen Programmierers. Durch die Anlehnung an heute übliche Benennungen und Konventionen in der Roboterprogrammierung zielt das entwickelte Plugin aber auch darauf ab, Roboterprogrammierern mit Erfahrung mit bisherigen Robotersystemen einen einfachen Einstieg zu ermöglichen.

In diesem Papier wird im folgenden Kapitel 2 die SoftRobot-Architektur überblicksweise erläutert. In Kapitel 3 wird anschließend das entwickelte Eclipse-Plugin detailliert vorgestellt. Kapitel 4 fasst die gewonnenen Erkenntnisse zusammen.

2. Die SoftRobot-Architektur

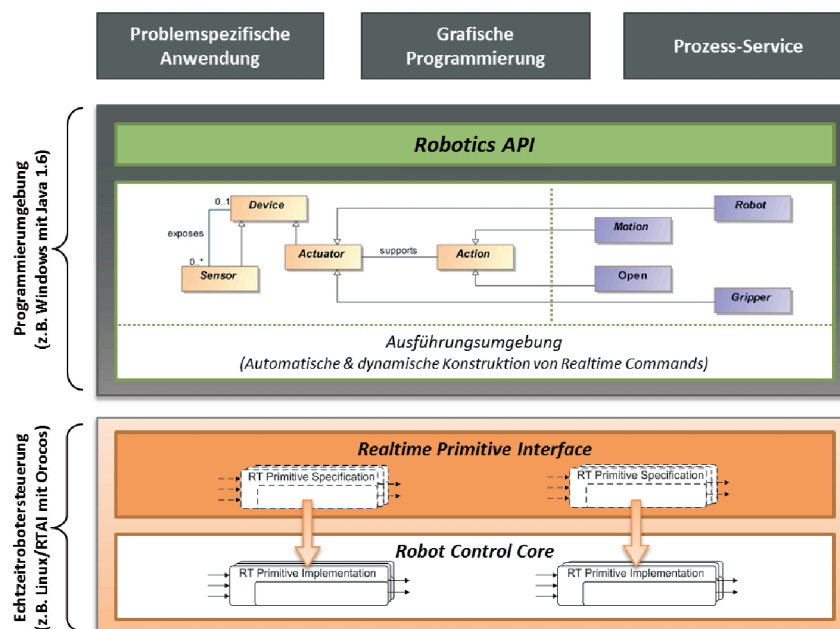


Abbildung 1: Die im Forschungsprojekt „SoftRobot“ entwickelte Architektur zur Trennung von echtzeitkritischer Robotersteuerung und objektorientierter Programmierung.

Im Forschungsprojekt *SoftRobot* wurde eine von Grund auf neue Softwarearchitektur für Roboterapplikationen entwickelt (Hoffmann et al., 2009; Schierl et al., 2012). Die Architektur ist in Ab-

bildung 1 dargestellt. Zentraler Bestandteil ist das *Robotics Application Programming Interface* (Robotics API), eine objektorientierte Programmierschnittstelle, mit der Industrieroboter aber auch Werkzeuge, Sensoren und mobile Plattformen programmiert werden können. Roboterbefehle, die mit der Robotics API definiert wurden, werden automatisch zur echtzeitfähigen Robotersteuerung übertragen, wo die Ansteuerung und Regelung der Geräte stattfindet. Die Robotersteuerung verfügt über eine flexible und erweiterbare Schnittstelle, das *Realtime Primitives Interface* (RPI). Damit ist es möglich, echtzeitkritische Aufgaben für einen oder mehrere Roboter (z.B. Bewegungen, Interaktion mit Werkzeugen und Sensoren) zu spezifizieren und auf der Robotersteuerung auszuführen.

2.1. Realtime Primitives Interface

Das *Realtime Primitives Interface* (Vistein et al., 2010) erlaubt die Definition von echtzeitkritischen Aufgaben mittels einer Datenflusssprache. Dort werden diese Aufgaben als Ganzes unter Echtzeitbedingungen ausgeführt. Dazu existieren *Realtime Primitives* und *Links*. 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. Primitives und deren Links bilden einen Graph, der – sofern er keine offenen Ein- und Ausgänge beinhaltet – als sogenanntes *Realtime Command* ausgeführt werden kann. Die Ausführung erfolgt zyklisch⁵ in einer kompatiblen Robotersteuerung, d.h. in jedem Takt wird der ganze Graph einmal ausgewertet. Für die Erweiterung der unterstützten Geräte und Steuerungskonzepte erlaubt RPI die Definition und Integration neuer Primitives.

Ein *Realtime Command* ist in sich abgeschlossen, d.h. das System befindet sich vor und nach dessen Ausführung in einem stabilen und konsistenten Zustand. In der Regel bedeutet dies, dass der Roboter beim Abschluss des Kommandos stehen muss. Darüber hinaus ist es aber auch möglich, dass Kommandos von neuen Kommandos übernommen werden können (vgl. Vistein et al., 2012). Dabei kann das Vorgängerkommando terminieren, während der Roboter noch in Bewegung ist. Allerdings muss das nachfolgende Kommando das System nahtlos in diesem Zustand weiter kontrollieren. Dieses Konzept ist bspw. für das Überschleifen von Bewegungen oder für einen kontrollierten Kraftaufbau notwendig.

Eine zu RPI kompatible Robotersteuerung, ein sogenannter *Robot Control Core*, stellt einen Interpreter für Realtime Commands und Implementierungen der benötigten Primitives bereit. Die Robotersteuerung ist für die Steuerung der Robotermechanik und der weiteren Geräte zuständig,

⁵ in einer konfigurierbaren Frequenz von 100 Hz bis 1 kHz

d.h. sie muss 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 auf einem Echtzeitbetriebssystem implementiert sein.

2.2. Robotics API

Das objektorientierte *Robotics Application Programming Interface* (Angerer et al. 2010a) 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 um neue Robotertypen und Werkzeuge bzw. Bewegungen und Aktionen erweiterbar ist. Die Klassenhierarchie ist teilweise in Abbildung 1 dargestellt und besteht aus drei verschiedenen Bereichen, die jeweils unterschiedliche Aspekte der Domäne repräsentieren: das Gerätemodell, das Weltmodell und das Kommandomodell.

Jedes verfügbare Gerät wird durch die Klasse *Device* repräsentiert. Eine Spezialisierung davon ist die Klasse *Actuator*, die eine steuerbare Einheit repräsentiert und von der weiter abgeleitet werden kann, um konkrete Ausprägungen (z.B. Roboter oder Werkzeuge) zu definieren. Außerdem ist es möglich, die physikalischen Eigenschaften von Geräten und weiteren, real vorhandenen Bestandteilen der Arbeitsumgebung (z.B. Werkstücke) explizit in Software darzustellen und dadurch die für die Aufgabe wichtigen, applikationsspezifischen Informationen zu modellieren.

Aktionen (*Actions*) sind abstrakte Beschreibungen einer Fähigkeit (z.B. eine Punkt-zu-Punkt-Bewegung eines Roboters) und werden über ein Kommando (*Command*) an ein konkretes steuerbares Gerät gebunden. Durch die Modellierung der Fähigkeiten als eigene Objekte ist es möglich, aus einzelnen, elementaren Fähigkeiten komplexere Fähigkeiten zusammenzustellen, die als *Realtime Command* in der Robotersteuerung ausgeführt werden. Da jede Aktion bzw. jedes Gerät ihm spezifische Zustände (*States*) ausweist, deren Betreten bzw. Verlassen jeweils ein Ereignis darstellt, können durch *EventHandler* diese Ereignisse mit einer Reaktion (*EventEffect*) verknüpft werden. Beispiele dafür sind das Starten eines neuen bzw. das Abbrechen eines laufenden Kommandos.

Die Robotics API wurde im Projekt *SoftRobot* prototypisch in Java implementiert. Durch die in der Objektorientierung übliche Vererbung können Klassen bzw. Schnittstellen für neue steuerbare Geräte (inkl. physikalisches Modell und Aktionen) von geeigneten Oberklassen abgeleitet werden. Die Definition von Werkzeugen (z.B. ein elektrischer Greifer) ist ebenso Teil einer solchen Erweiterung wie die Definition der Klasse *Robot* und der dazugehörigen Bewegungen. Der KUKA Leichtbauroboter ist beispielsweise wiederum eine Spezialisierung eines Roboters. Diese Erweiterungen der Applikationsschnittstelle sind ebenfalls in Abbildung 1 dunkel dargestellt.

Durch diese Struktur ist die Robotics API ein erweiterbares Framework für die High-Level-Programmierung von Anwendungen für Industrieroboter. Mechanismen zur detaillierten Modellierung der physikalischen Umgebung des Roboters sowie erweiterbare Schnittstellen für verschiedene Roboteraktionen erlauben die Programmierung auch sehr komplexer Roboteranwendungen. Jedoch sind in der Praxis für die Roboterprogrammierung noch Funktionen wünschenswert, die über die Programmiersprachenebene hinausgehen. Solche Funktionen (wie die Verwaltung von Koordinatensystemen oder Werkzeugen) sind, wie in Kapitel 1 erläutert, in der Regel Bestandteil der proprietären Toolketten, die die Hersteller für ihre Robotersysteme anbieten. Für die Soft-Robot-Architektur, die auf Allzweck-Programmiersprachen und in der Softwareentwicklung etablierte Werkzeuge setzt, ist eine Erweiterung dieser Werkzeuge um spezielle Funktionen für die Anwendungsentwicklung für Robotersysteme sinnvoll. Als Plattform für die Realisierung solcher Funktionen wurde Eclipse aufgrund seiner hohen Verbreitung als IDE in der Softwareentwicklung und der guten Erweiterbarkeit ausgewählt. Im folgenden Kapitel wird die Eclipse-Plattform und das Robotics API Eclipse-Plugin, das im Rahmen von SoftRobot entwickelt wurde, vorgestellt.

3. Das Robotics API Eclipse-Plugin

Die Eclipse-Plattform wird von einer großen Open-Source-Community mit etwa 1.000 aktiven Entwicklern⁶ seit 2001 ständig weiterentwickelt. Eclipse wird zum einen als IDE für zahlreiche Programmiersprachen (Java, C/++, Python, ...) eingesetzt, zum anderen auch als Plattform für eine Reihe von Desktop-Applikationen⁷. Ein beträchtlicher Teil der Beliebtheit von Eclipse dürfte sicherlich in seinem modularen Aufbau und der vielfältigen Anpassbarkeit und Erweiterbarkeit gründen. Sowohl die graphische Oberfläche lässt sich nahezu beliebig anpassen, als auch die Funktionalitäten ändern oder erweitern.

Durch diese Voraussetzungen eignet sich Eclipse gut als Basis für robotik-spezifische Erweiterungen. In der hier vorgestellten Arbeit wurde die *Eclipse IDE for Java Developers* als Grundlage gewählt und um eine Reihe von sogenannten *Plugins* erweitert. Die Summe dieser Plugins wird im Folgenden vereinfacht als das *Robotics API Eclipse Plugin* bezeichnet.

In den folgenden Abschnitten wird zunächst die Struktur der Eclipse-Plattform überblicksweise dargestellt. Anschließend werden die einzelnen Erweiterungen des Robotics API Plugins und ihre jeweilige Funktionalität vorgestellt.

⁶ http://www.eclipse.org/org/community_survey/Eclipse_Survey_2010_Report.pdf

⁷ <http://www.eclipse.org/community/rcp.php>

3.1. Struktur der Eclipse-Plattform

Der Kern von Eclipse basiert auf einem modularen Komponentensystem nach der OSGi-Spezifikation (OSGi Alliance, 2003), realisiert durch die Implementierung Equinox (McAffer et al., 2010). Jeder Teil von Eclipse ist somit als OSGi-Komponente umgesetzt, die Erweiterungsmechanismen sind dadurch bereits auf unterster Ebene verankert. Dieser Komponentenansatz spiegelt sich auch in der Benutzeroberfläche wieder: Sie ist aus einer Reihe sogenannter Views zusammengesetzt, deren Implementierungen von einer oder mehreren Eclipse-Komponenten bereitgestellt werden. Die Anordnung dieser Views innerhalb der Eclipse-Oberfläche ist beliebig änderbar. Sogenannte *Perspectives* definieren Anordnungen spezifischer Views auf eine Weise, die sich für einen bestimmten Anwendungsfall bewährt hat. So ordnet beispielsweise die *Java*-Perspective links den sogenannten *Package Explorer* an, in der Mitte den gerade aktiven *Editor* (zum Beispiel für Java-Code-Dateien) und rechts den *Outline*-View, der einen Überblick über den Inhalt der gerade geöffneten Datei gibt. Abbildung 2 zeigt beispielhaft den Aufbau der Eclipse-Oberfläche aus Views, die gemäß einer bestimmten Perspective angeordnet sind.

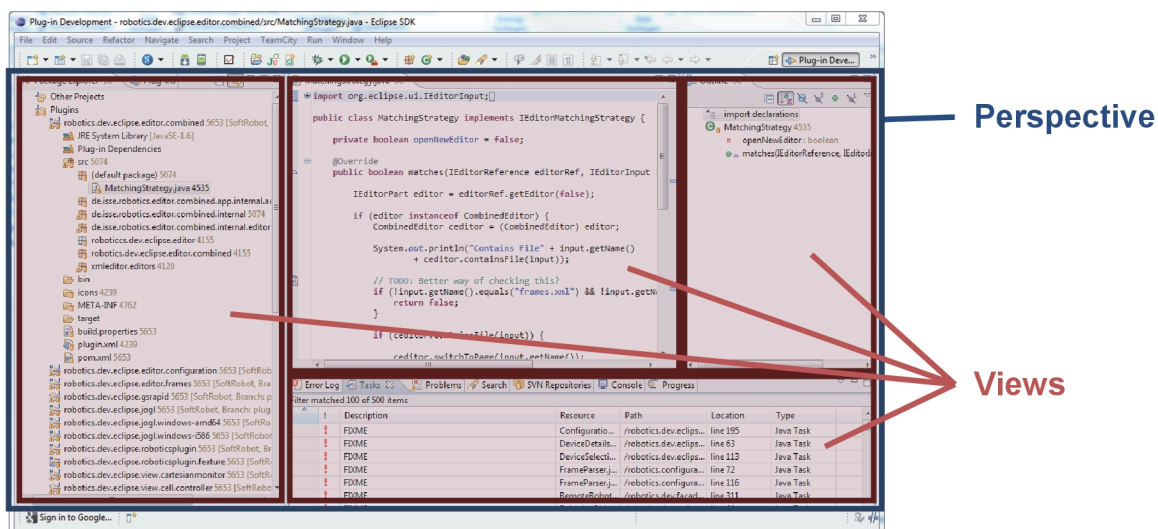


Abbildung 2: Struktur der Eclipse-Oberfläche

Als Mechanismus zur Integration von Erweiterungen bietet Eclipse sogenannte *Extension Points* an, für die jeweils *Extensions* registriert werden können. Der Extension Point definiert dabei die Art und die möglichen Parameter der Extension (z.B. Anzeigenname einer Perspective, implementierende Klasse, ...). Über diese Extension Points können neue Komponenten in die Eclipse-Umgebung integriert werden, die unabhängig von anderen Komponenten beim Eclipse-Start mit geladen werden, andere Komponenten um neue Funktionalität ergänzen oder sogar bestehende Funktionalität neu definieren. Auf diese Mechanismen baut auch das Robotics API Plugin auf. Details zu den Eclipse-Erweiterungsschnittstellen finden sich bei Clayberg et al., 2008.

3.2. Erweiterungen durch das Robotics API Plugin

Das Robotics API Plugin erweitert Eclipse an unterschiedlichen Stellen um robotikspezifische Funktionalität. Folgende Funktionen werden neu eingeführt:

- ein Projekttyp *Robotics API Project* mit spezieller Projektstruktur
- Erweiterung des Project Explorer für die Anzeige von Robotics API Projekten
- Editoren für die Verwaltung von Konfigurationen und Koordinatensystemen
- die Views *Joint Monitor*, *Cartesian Monitor*, *Frame View* und *Jogging*
- eine *Robotics*-Perspective

In den nächsten Abschnitten werden die einzelnen Erweiterungen im Detail vorgestellt.

3.2.1. Projekttyp „RoboticsAPI Project“

Das Robotics API Plugin führt einen neuen Projekttyp ein, der beim Anlegen eines neuen Projekts als Template zur Verfügung steht. Ein solches Robotics API Project erhält eine besondere *Robotics-Nature*, anhand derer es von den anderen Erweiterungen identifiziert und entsprechend behandelt wird. Um die Funktionalität der Robotics API bei der Programmierung verwenden zu können, werden die nötigen Bibliotheken in das Projekt kopiert und in den Java-Bibliothekspfad eingebunden. Darüber hinaus werden in das Projekt Konfigurationsdateien eingebunden, die für die Initialisierung der Robotics API verwendet werden.

Die spezielle Struktur von Robotics API Projekten ist besonders gut im erweiterten Project Explorer zu erkennen, der nachfolgend vorgestellt wird.

3.2.2. Erweiterungen des Project Explorer

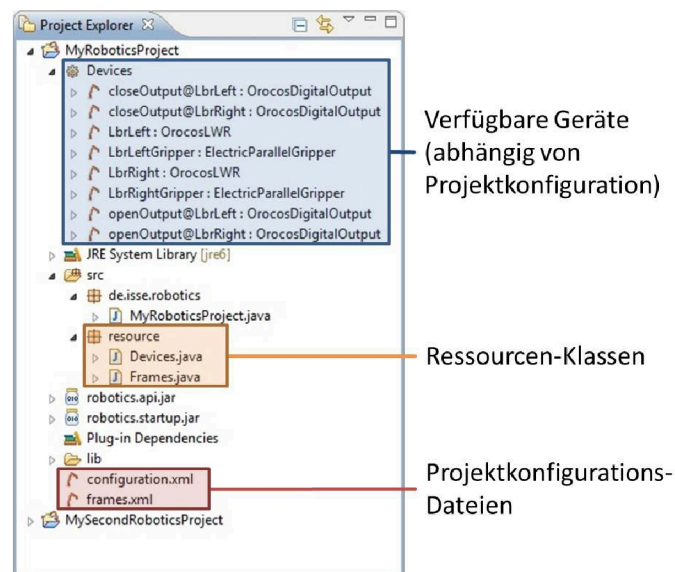


Abbildung 3: Erweiterter Project Explorer

Der Eclipse Project Explorer wurde um einige Mechanismen erweitert, um der besonderen Struktur von Robotics API Projekten Rechnung zu tragen und diese dem Benutzer zu visualisieren. Abbildung 3 zeigt ein Beispiel dieser Erweiterungen. Zunächst erhält die Ansicht des Projektbaums einen neuen Knoten Devices. In diesem Knoten werden alle Geräte angezeigt, die aufgrund der Projektkonfiguration innerhalb dieses Projekts verfügbar sind.

Die Projektkonfiguration selbst ist in den Konfigurations-Dateien *configuration.xml* und *frames.xml* gespeichert. Sie werden im erweiterten Project Explorer mit einem speziellen Symbol markiert, um ihre Bedeutung hervorzuheben. Bei Öffnen dieser beiden Dateien durch Doppelklick werden spezielle Editoren gestartet (vgl. Abschnitt 3.2.3). Nach jeder Veränderung an der Projektkonfiguration werden außerdem sogenannte *Ressourcen-Klassen* automatisch generiert. Dabei handelt es sich um Java-Klassen mit einer Reihe von statischen Feldern, die im Quellcode für den Zugriff auf alle Geräte und Koordinatensysteme verwendet werden können. Dadurch wird dem Programmierer dieser Zugriff leicht gemacht, und darüber hinaus können Änderungen an der Konfiguration (z.B. Umbenennung eines Geräts) automatisch in bestehenden Quellcode propagiert werden.

3.2.3. Configuration-Editor, Frames-Editor und Frames View

Für die Bearbeitung der Projektkonfigurations-Dateien stellt das RoboticsPlugin zwei neue Editoren in Eclipse bereit: *Configuration Editor* und *Frames Editor*. Sie werden direkt mit den beiden Konfigurationsdateien assoziiert und dadurch automatisch zum Bearbeiten der Dateien geöffnet (z.B. bei Doppelklick auf eine der Dateien).

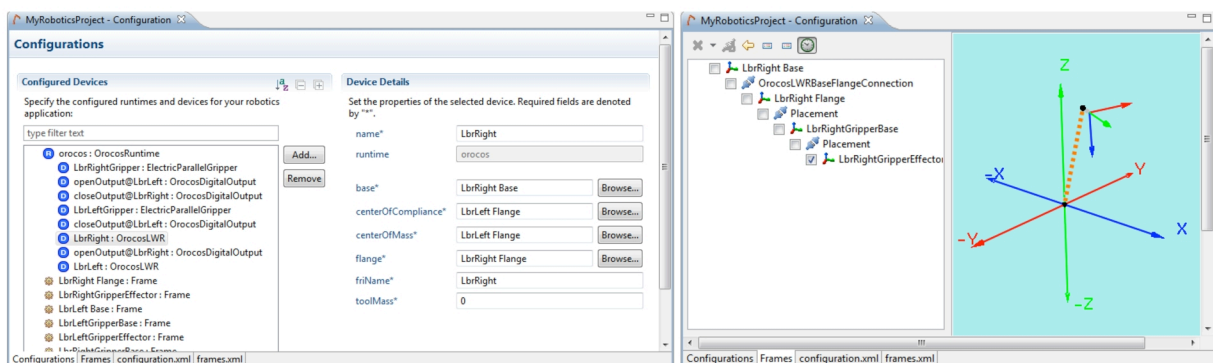


Abbildung 4: Configuration Editor (links) und Frame Editor (rechts)

Der Configuration Editor (Abbildung 4, links) unterstützt sowohl beim Anpassen existierender Konfigurationen (z.B. Parameteränderungen wie Masse eines Roboterwerkzeugs) als auch beim Hinzufügen neuer Konfigurationen (z.B. eines neu angeschlossenen Roboters). Der Benutzer wird zum Beispiel durch das Anzeigen fehler- oder lückenhafter Konfigurationen oder auch durch Vorschläge für Werte von Konfigurationsparametern unterstützt.

Mit dem Frames Editor (Abbildung 4, rechts) können alle räumlichen Punkte und Koordinatensysteme verwaltet werden, die für eine Roboterapplikation benötigt werden. Die Zusammenhänge zwischen den Koordinatensystemen werden sowohl in einer Baumstruktur als auch in einer dreidimensionalen, frei dreh- und vergrößerbaren Ansicht dargestellt. Auch Veränderungen von Beziehungen zwischen Koordinatensystemen (z.B. durch Bewegung eines Roboters) werden kontinuierlich angezeigt. Der Editor erlaubt die Änderung sowie das Löschen oder Neuanlegen solcher Beziehungen. Das robotergestützte Einlernen von Punkten und Basiskoordinatensystemen findet ebenfalls mit dem Frames Editor statt. Die reine Bewegungssteuerung kann dabei parallel zum geöffneten Editor mit dem Jogging View erfolgen, der im nächsten Abschnitt vorgestellt wird.

Unabhängig vom Frames Editor steht mit dem Frames View jederzeit eine Baum-Ansicht der vorhandenen Koordinatensysteme zur Verfügung, die sich an dem Layout des Frames Editor orientiert.

3.2.4. Handverfahren mit dem Jogging View

Während der Entwicklung einer Roboterapplikation ist es manchmal notwendig, Roboter manuell zu verfahren, um zum Beispiel einen Teach-In-Vorgang durchzuführen. Diese Funktionalität stellt im Robotics API Plugin der *Jogging View* zur Verfügung (siehe Abbildung 5). Damit können sowohl einzelne Achsen eines Roboters verfahren werden als auch ein frei wählbares Roboterwerkzeug in einem beliebigen kartesischen Koordinatensystem. Auch spezielle Roboterfähigkeiten werden unterstützt: Bei Verwendung eines KUKA Leichtbauroboters wird beispielsweise die Aktivierung des Gravitationskompensationsmodus angeboten, in dem der Roboter sich durch Berührung führen lässt.

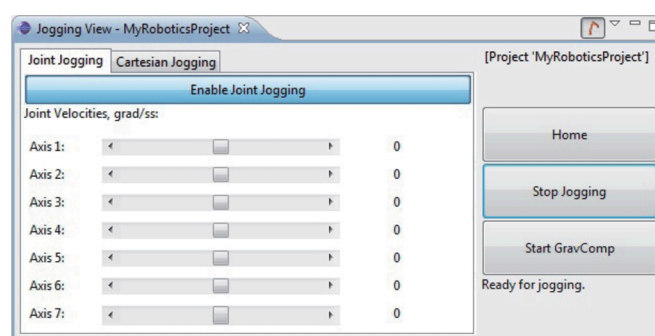


Abbildung 5: Jogging View

Die Auswahl des zu steuernden Roboters kann auf zwei Arten geschehen: Die erste Möglichkeit ist eine Auswahl des Roboters über ein Pull-Down-Menü direkt innerhalb des Jogging View. Die zweite Möglichkeit nutzt einen Mechanismus aus, den der erweiterte Project Explorer (siehe Abschnitt 3.2.2) bietet: Unter den verfügbaren Geräten eines Projekts kann eines per Rechtsklick als *aktives Gerät* definiert werden. In diesem Fall reagieren alle anderen Views auf diese Auswahl –

auch der Jogging View, der automatisch das Handverfahren des aktiven Geräts erlaubt (falls es sich dabei um einen Roboter handelt).

Der Jogging View zeigt Daten über die Achsstellungen und die kartesische Roboterposition während des Handverfahrens an. Eine umfassendere Möglichkeit zur Überwachung dieser Parameter bieten jedoch die im folgenden Abschnitt vorgestellten Views.

3.2.5. Datenanzeige mit Joint Monitor und Cartesian Monitor

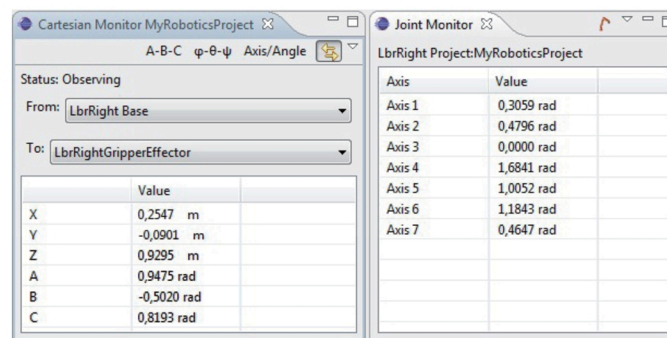


Abbildung 6: Cartesian Monitor und Joint Monitor

Zur Anzeige und Überwachung von Achspositionen eines Roboters dient der *Joint Monitor* (Abbildung 6, links). Mit dem *Cartesian Monitor* (Abbildung 6, rechts) hingegen kann die kartesische Position eines beliebigen Punkts in einem beliebigen Koordinatensystem angezeigt werden – ganz gleich ob es sich dabei um dynamisch veränderliche Daten handelt (wie beispielsweise die Position und Orientierung eines Werkzeugs relativ zur Roboterbasis) oder um statische Informationen (zum Beispiel die Position einer Roboterbasis relativ zum Koordinatensystem einer Werkbank). Diese Daten werden in jedem Fall laufend aktuell gehalten.

4. Zusammenfassung

In der vorliegenden Arbeit wurde das Robotics API Eclipse-Plugin vorgestellt. Es ist als Erweiterung der verbreiteten Entwicklungsumgebung Eclipse konzipiert und soll Softwareentwicklern Tools und Funktionen zur einfachen Roboterprogrammierung bieten. Mit Configuration Editor und Frames Editor stehen Werkzeuge für das Konfigurationsmanagement von Roboteranwendungen zur Verfügung. Darüber hinaus werden in Kombination mit dem Jogging View Verfahren zum Teach-In von Roboterpositionen unterstützt. Zur Anzeige von Daten der Zelle wie der Position des Roboters oder der geometrischen Zusammenhänge zwischen verschiedenen Bestandteilen des Aufbaus stehen Joint Monitor und Cartesian Monitor zur Verfügung. Eine Übersicht über die

Konfiguration einer Roboterapplikation liefert der erweiterte Eclipse Project Explorer, der gleichzeitig als Einstiegspunkt für weitergehende Funktionen wie die Bearbeitung von Geräte- und Koordinatensystem-Konfiguration dient.

Vorbilder bei der Entwicklung des Robotics API Plugins sind Toolchains, die heute für die Steuerungen verschiedener Industrieroboter existieren. Dabei wurden zum einen Funktionen der jeweiligen Bedienhandgeräte (z.B. KUKA smartPAD) wie Teach-In-Verfahren oder Roboterdaten-Anzeige betrachtet, als auch Funktionalitäten von Offline-Programmierungsumgebungen (z.B. ABB RobotStudio) wie die Verwaltung und dreidimensionale Anzeige von Koordinatensystemen und gespeicherten Roboterpositionen. Obwohl einige Funktionalitäten über die der heute existierenden, proprietären Tools hinaus gehen, erreicht das Robotics API Plugin in seiner jetzigen Form noch nicht sein gesamtes Potenzial. Jedoch kann bereits jetzt das Fazit gezogen werden, dass heute verfügbare Standard-Entwicklungsumgebungen mit entsprechenden Erweiterungen eine gute Basis für die Softwareentwicklung für Robotersysteme bilden können. Im Falle von Eclipse ist durch die Flexibilität der Erweiterungsschnittstellen und die zahlreichen verfügbaren Software-Bibliotheken ein solches Vorhaben mit verhältnismäßig geringem Aufwand umsetzbar.

5. Literaturverzeichnis

- OSGi Alliance: *OSGi service platform: release 3, March 2003*; IOS Press, Ohmsha: 2003
- McAffer, J., VanderLei, P., Archer, S.: *OSGi and Equinox: Creating Highly Modular Java Systems*; 1 edition (February 24, 2010); Addison-Wesley Professional, Boston, USA: 2010
- Clayberg, E., Rubel, D.: *Eclipse: Building Commercial-Quality Plug-ins*; 3rd Revised edition (REV). (22. Dezember 2008); Addison-Wesley Longman, Amsterdam, NL: 2008
- Angerer, A., Hoffmann, A., Schierl, A., Vistein, M., Reif, W.: *The Robotics API: An Object-Oriented Framework for Modeling Industrial Robotics Applications*; Proc. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), Taipeh, Taiwan: October 2010
- Schierl, A., Angerer, A., Hoffmann, A., Vistein, M., Reif, W.: *Using Java for Real-Time Critical Industrial Robot Programming*, Workshop on Software Development and Integration in Robotics (SDIR VII), 2012 IEEE International Conference on Robotics and Automation (ICRA 2012), St. Paul, Minnesota, USA, Mai 2012
- 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
- Vistein, M., Angerer, A., Hoffmann, A., Schierl, A., Reif, W.: *Interfacing Industrial Robots using Real-*

time Primitives; Proc. 2010 IEEE International Conference on Automation and Logistics (ICAL), Hong Kong: August 2010

Vistein, M., Angerer, A., Hoffmann, A., Schierl, A., Reif, W.: *Instantaneous Switching between Real-Time Commands for Continuous Execution of Complex Robotic Tasks*; Proc. 2012 IEEE International Conference on Mechatronics and Automation (ICMA), Chengdu, China: August 2012