Using Java for Real-Time Critical Industrial Robot Programming

Andreas Schierl, Andreas Angerer, Alwin Hoffmann, Michael Vistein, Wolfgang Reif
Software Development for Industrial Robots

Current situation

• Sophisticated mechanical components and control algorithms

• High precision, reliability and repeatability

• Specialized, proprietary programming languages and outdated software concepts (e.g. KUKA KRL)

Vision

• Apply modern software engineering to industrial robots

• Facilitate robotics software development by providing “robotics” as just another API in a popular programming language (the “Robotics API”)

• Thus increase reuse and reduce development time
Programming Robots in KRL

LIN P1 C_DIS
TRIGGER WHEN PATH=70 DELAY=0 DO $OUT[2] = TRUE
LIN P2 C_DIS
LIN P3

KUKA KRC
Programming Robots in Java

```java
lwr.ptp(p1).execute();
lwr.lin(p2).execute();
gripper.open().execute();
```
Real-time Robot Control

• KUKA Lightweight Robot requests position set points every $n$ milliseconds (configured, hard real-time) through the Fast Research Interface (FRI)

• Requires a real-time system
Robot Control Core (RCC)

- Implemented in C++ using Orocos
- Linux with real-time extension (Xenomai / RTAI)
RCC Structure and Interfaces

Realtime Primitives Interface (RPI)

Data flow graph
Calculation module
Device module
Device driver
Example: Cartesian Motion Data-flow Graph

- Linear Trajectory
  - from, to

- Transformation
  - matrix

- Inverse Kinematics
  - device

- Robot
  - device

- Termination

\[ T_{F}^{MCP} \]

\[ T_{Base}^{Flange} \]

\[ \theta_1 \ldots \theta_7 \]
Still missing: Programming in Java

Java-representation of the concepts in robotics

OrocosRCC (C++)
Robotics API: Command Layer

Commands in the Robotics API:

- Action
- Command
- Actuator

Command Layer (Java)

- PTP
- Robot
- LWR

Calculation module

OrcosRCC

Device driver
Actions and actuators

- **Commands** describe who (*Actuator*) shall do what (*Action*)

- **Actuators** are proxy objects for the devices on the RCC

- **Actions** describe what to do, independent from the actuator

- **Commands** can be combined if real-time reaction or precise timing is required
From Commands to Data Flow Graphs

Automatic transformation of commands at runtime

Diagram showing the flow from Action, Command, Actuator, PTP, Robot, LWR, Calculation module, OrocospRCC, and Device driver.
What’s still missing?

```
lwr.ptp(p1).execute();
lwr.lin(p2).execute();
gripper.open().execute();
```

**Command Layer**

- **Action**
- **Command**
- **Actuator**

**PTP**

- from: double[7]
- to: double[7]

**Robot**
Robotics API: Activity Layer

• Some commands need to know the initial device state
  – e.g. start position for a point-to-point motion

• Activities store a command and meta data about its final state(s), which can be used in the following Activities

• Actuators contain a set of ActuatorInterfaces as factories for their supported Activities
SoftRobot Architecture Overview

Robot Applications

Domain-Specific Languages

Service-Oriented Architectures

Activity Layer

Meta Data

Activity

Actuator Interface

Command Layer

Action

Command

Actuator

Robot

Orocos RCC

Calculation Modules

Device Drivers

Robotics API

Java

RCC

C++
Object-Oriented API Using Java

Development Tools
- Eclipse IDE
- extended by Robotics plugin

Existing Libraries
- Microsoft Surface SDK
- OpenCV

Platform Independence
- Windows
- Linux
- Android

Service Oriented Architectures
- using OSGi
- for Automation

Domain Modeling
- Encapsulating domain knowledge for later reuse
Motion Programming & Tool Integration

• Motion programming:
  – Similar to KRL (PTP, LIN, no explicit start position for motions)
  – Supports motion blending

```java
// initialize actuator interfaces
MotionInterface lwr = robot.use(MotionInterface.class);
GripperInterface g = gripper.use(GripperInterface.class);

// move robot to frame p1, allow blending
lwr.ptp(p1).beginExecute();

// move robot linearly to frame p2, wait for completion
lwr.lin(p2).execute();

// open gripper
g.open().beginExecute();
```
Motion Programming & Tool Integration

• Motion programming:
  – Similar to KRL
    \( PTP, LIN \), no explicit start position for motions
  – Supports motion blending

• Robot tool integration:
  – Common interface for similar tools
  – Abstracts from hardware details (field bus outputs, etc.)

// initialize actuator interfaces
MotionInterface lwr = robot.use(MotionInterface.class);
GripperInterface g = gripper.use(GripperInterface.class);

// move robot to frame p1, allow blending
lwr.ptp(p1).beginExecute();

// move robot linearly to frame p2, wait for completion
lwr.lin(p2).execute();

// open gripper
g.open().beginExecute();
Real-time Awareness & Error Handling

• Real-time awareness:
  – Java control flow where no real-time timing is required
  – Real-time guarantees where required (ptpGrip)
  – No need to care about scheduling/memory allocation

• Error handling:
  – Real-time reaction to command error states
  – Non-real-time exception handling in Java

// move robot to frame p1, allow blending
robot.ptp(p1).beginExecute();

// move to frame p2, open the gripper at 30%
MotionProgressActivity ptp = robot.ptp(p2);
RtActivityWithSubactivities ptpGrip =
  new RtActivityWithSubactivities(ptp);
ptpGrip.addSubActivity(
  lin.getMotionTimePercent(30), g.open());
ptpGrip.beginExecute();
Extensibility & Sensor Integration

- **Extensibility:**
  - *LwrMotionInterface* extends normal motion interface with special LWR functions
  
  ```
  // initialize actuator interfaces
  LwrMotionInterface robot = lwr.use(LwrMotionInterface.class);
  LinInterface twoArm = twoArmRobot.use(LinInterface.class);
  ```

  ```
  // move to frame p1, stop if force of 5N is measured
  robot.linToContact(p1, 5).execute();
  ```

  ```
  // move both arms synchronously to p2, allow blending
  twoArm.lin(p2).beginExecute();
  ```

  ```
  // move both arms synchronously to p3, wait for completion
  twoArm.lin(p3).execute();
  ```

- **Sensor integration:**
  - *linToContact* as sensor guarded motion
Extensibility & Sensor Integration

• Extensibility:
  – *LwrMotionInterface* extends normal motion interface with special LWR functions
  – *TwoArmRobot* as a new device controlling two robots synchronously
  – Multiple robots can be controlled from one application

• Sensor integration:
  – *linToContact* as sensor guarded motion
  – Access to sensors, handling in Java or real-time

```java
// initialize actuator interfaces
LwrMotionInterface robot = lwr.use(LwrMotionInterface.class);
LinInterface twoArm = twoArmRobot.use(LinInterface.class);

// move to frame p1, stop if force of 5N // is measured
robot.linToContact(p1, 5).execute();

// move both arms synchronously to p2, // allow blending
twoArm.lin(p2).beginExecute();

// move both arms synchronously to p3, // wait for completion
twoArm.lin(p3).execute();
```
This work presents results of the research project SoftRobot which was funded by the European Union and the Bavarian government. The project was carried out together with KUKA Laboratories GmbH and MRK-Systeme GmbH and was kindly supported by VDI/VDE-IT GmbH.

Thank you for your attention!