Paper accepted for Workshop on Software Development and Integration in Robotics (SDIR VII), St. Paul, Minnesota, USA, May 14, 2012

Using Java for Real-Time Critical Industrial Robot Programming

Andreas Schierl, Andreas Angerer, Alwin Hoffmann, Michael Vistein and Wolfgang Reif

I. Vision

Industrial robotics is characterized by sophisticated mechanical components and control algorithms. However, the efficient use of robotic systems is very much limited by existing programming methods which make software development complex and time-consuming. In order to overcome these shortcomings, the vision of the research project *SoftRobot* was to facilitate robotics software development, i.e. increase reuse and reduce development time, by providing "robotics" as just another API in Java which is currently the most popular programming language with billions of supported devices, millions of skilled software developers and a broad range of available libraries and tools. Hence, robotics software can be easily combined with those libraries and can be programmed using integrated development environments (e.g. Eclipse) with their rich built-in support for build management, revision control, or unit tests. In comparison to C/C++, Java is more comfortable to use and less error-prone due to its automatic memory management.

II. Goals

Based on this vision, *SoftRobot* aimed to develop an object-oriented framework that allows to program industrial robots using Java. Like in traditional robot languages (e.g. KRL), the new framework had to provide real-time guarantees for robot control, in order to offer the expected predictability and precision of the robot. However, when developing robotics software, one should only be aware that real-time guarantees are important for certain task steps, but should not be bothered with low-level aspects of real-time programming (e.g. scheduling). This is an important goal to decrease complexity and allows to offer an API in plain Java instead of real-time capable programming languages.

Additionally, to become an acceptable replacement in industrial applications, the framework had to support easy motion programming and control flow, and also motion blending and the definition of motion trigger actions to achieve the performance and expressiveness of traditional robot programs. Beyond that, controlling multiple robots or devices from one application, as well as sensor-guided motions were in the focus of the project. From an non-functional point of view, the architecture should be extensible to allow adding new devices or robot capabilities.

The authors are with the Institute for Software and Systems Engineering, University of Augsburg, D-86135 Augsburg, Germany. E-mail of corresponding author: `schierl@informatik.uni-augsburg.de`

This work presents results of the research project *SoftRobot* which is funded by the European Union and the Bavarian government. The project is carried out together with KUKA Laboratories GmbH and MRK-Systeme GmbH and is kindly supported by VDI/VDE-IT GmbH.

III. Solution

To achieve these goals, we developed a multi-layered software architecture [1], which is depicted in Fig. 1, and provided prototypical reference implementation.

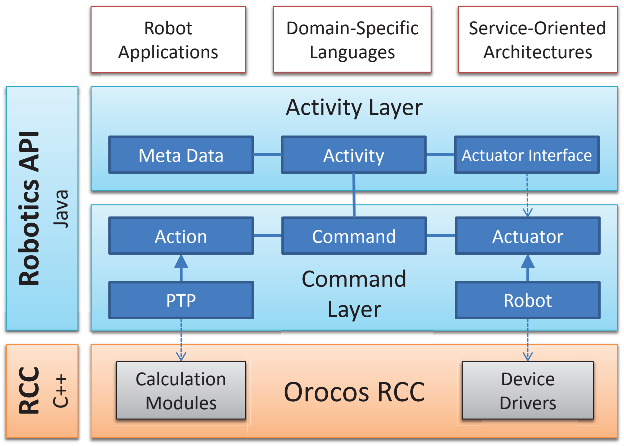

Fig. 1. The *SoftRobot* architecture

Robot control is performed in the so called *Robot Control Core* (RCC) which is implemented in C++ using Orocos [2]. It fulfills the hard real-time requirements of robot control and allows to control two KUKA Lightweight robots synchronously using the *Fast Research Interface* [3] at a rate up to 1kHz. The RCC is interfaced through an extensible data-flow language called *Realtime Primitives Interface* (RPI) which is described in [4].

On top of that, we implemented the Java-based *Robotics API*. Its lower part, the *Command Layer* [5], is used for describing real-time critical robot transactions as a composition and coordination of robot Commands. Robot Commands consist of an actuator (e.g. a Lightweight robot) commanded to execute an action (e.g. a point-to-point motion from given start to goal position). This layer includes an automatic translation of robot transactions into RPI at runtime, so that they can be submitted to and executed on the RCC.

Application programmers use the *Activity Layer* of the Robotics API that provides robot Activities through an extensible set of actuator interfaces. These Activities correspond to robot transactions with meta data about the expected result states of the system. Additionally, these Activities allow composition using typical composition patterns such as sequential or parallel execution.

IV. Results

Using this software architecture, the given goals could be achieved. Motion programming similar to the KUKA Robot

```
; move linearly to point P1, allow blending
LIN P1 C_DIS

; move linearly to point P2, set a digital output at 70mm
TRIGGER WHEN PATH=70 DELAY=0 DO $OUT[2]=TRUE
LIN P2 C_DIS

; move to point P3, no blending
LIN P3
```

Listing 1. Robot program in KRL

```
// initialize device interfaces
MotionInterface r = lbr.use(MotionInterface.class);
GripperInterface g = gripper.use(GripperInterface.class);

// move linearly to frame p1, allow blending
r.lin(p1).beginExecute();

// move linearly to frame p2, open the gripper at 30%
MotionProgressActivity lin = r.lin(p2);
RtActivities.addSubActivity(lin,
     lin.getMotionTimePercent(30),
     g.open()).beginExecute();

// move to frame p3, wait for completion
r.lin(p3).execute();
```

Listing 2. Similar robot program in Java

Language (cf. Lst. 1) can be achieved by using Activities, even including the advanced capabilities of the Lightweight robot such as force-based motions. Motion planning requires the start position of the robot, but this can be retrieved from the meta data of the previous activity and does not have to be given explicitly by the programmer. Using Activity meta data is also more flexible than using the current robot position, because it allows for pre-planning a sequence of activities before execution.

Additionally, it is possible to blend between subsequent motions on a Java control flow level, as Activity meta data also contains information about possible start positions (and velocities) for blending. Thus, a following Activity can plan a motion from the given start position and velocity for blending. However, the Activity must also be able to cope with the case that the blending state has already passed when the command is issued, and the robot will thus stop at the defined end position and has to continue from there. This approach can be also applied for compliant motions, where an Activity may establish contact and subsequent Activities have to cope with this case (e.g. by first releasing the force).

Furthermore, the Activity composition patterns allow to specify trigger actions (cf. Lst. 2) that are to be executed when a certain state occurs, and also to compose a sequence of multiple Activities into one transaction (if required) to eliminate unwanted delays between the single Activities. Lst. 3 shows some advanced Activities that have no direct equivalent in KRL. The two arm robot used here provides a specialized interface that implements synchronized two arm linear motions by a quite simple composition of Activities for the single arms.

As the robot application is programmed and executed in standard Java, this architecture reduces the complexity

```
// initialize device interfaces
LwrMotionInterface r = lbr.use(LwrMotionInterface.class);
LinInterface twoArm = twoArmRobot.use(LinInterface.class);

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

Listing 3. Advanced Robotics API instructions

of integrating standard hardware (e.g. Microsoft Surface, cameras) into robot applications, and also simplifies the coordination of multiple robots, as all robots can be controlled from a single application. Although the direct code-level comparison shows that Robotics API code does not achieve the same syntactical compactness as code written in KRL (which can be considered a DSL), we believe that the above mentioned advantages clearly make the Robotics API the better overall solution.

V. Outlook

The proposed software architecture was evaluated successfully in a range of applications (cf. Factory 2020 [6]), and received a lot of interest at the final project demonstration in March 2012. Allowing to program robots in standard programming languages promises to open up a whole range of new opportunities. We started to extend Eclipse as development environment with plugins supporting the development of robot applications, but also using Service-oriented Architectures to control greater automation solutions becomes feasible now. Additionally, the Robotics API is a helpful foundation for domain specific robot languages [7], but also for graphical programming of robots (e.g. using state charts).

References

[1] A. Hoffmann, A. Angerer, F. Ortmeier, M. Vistein, and W. Reif, "Hiding real-time: A new approach for the software development of industrial robots," in *Proc. 2009 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, St. Louis, MO, USA*, 2009.

[2] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Proc. 2001 IEEE Intl. Conf. on Robotics and Automation, Seoul, Korea*, 2001.

[3] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the KUKA lightweight robot," in *Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications. IEEE Intl. Conf. on Robotics and Automation, Anchorage, AK, USA*, 2010.

[4] M. Vistein, A. Angerer, A. Hoffmann, A. Schierl, and W. Reif, "Interfacing industrial robots using realtime primitives," in *Proc. IEEE Intl. Conf. on Automation and Logistics, Hong Kong*, 2010.

[5] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif, "The Robotics API: An object-oriented framework for modeling industrial robotics applications," in *Proc. 2010 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Taipeh, Taiwan*, 2010.

[6] Factory 2020. Institute for Software and Systems Engineering. [Online]. Available: http://video.isse.de/factory

[7] H. Mühe, A. Angerer, A. Hoffmann, and W. Reif, "On reverse-engineering the KUKA Robot Language," *Workshop on Domain-Specific Languages and models for ROBotic systems, 2010 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Taipeh, Taiwan*, 2010.