# Using Object-Oriented Development for Planning and Controlling Industrial Robot Systems

Alwin Hoffmann, Ludwig Nägele, Andreas Angerer, Andreas Schierl, Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, 86159 Augsburg, Germany
Email: {hoffmann, naegele, angerer, schierl, reif}@isse.de

## I. Introduction

As physical devices or even items are getting more and more embedded with electronics, software and sensors, they are able to build a network of interconnected devices – the *Internet of Things (IoT)* – Hence, these computational entities are on the one hand in intensive connection with their physical environment and on the other hand able to collect, process and exchange data with other entities. By giving such a cyber-physical systems self-awareness and intelligence through software, they enable smart buildings, smart electric grid, or smart manufacturing. In Industry 4.0 [6], the latter is based on *cyber-physical production systems* which are comprised of a network of intelligent automation components [10].

From our point of view, robotic system will play an important role in future cyber-physical production systems due to both their mechanical flexibility and their ability to be freely programmable. However, robotics systems need to be self-aware in order to adapt to new tasks at runtime or autonomously plan collision-free motions or even whole assembly tasks. Hence, the robotic system needs a virtual model for each of its component (i. e., its manipulators, end-effectors, mobile platforms and sensors). Together they form the virtual model of the overall robotics systems which includes, for example, important spatial features, 3D models, physical or mechanical properties. To inherently integrate these virtual models into robot programming, we propose to use object-oriented software development for robotics, where real-world entities are directly mapped to software objects.

The presented approach is based on the software architecture (cf. Fig. 1) which was developed together with KUKA in the research project *SoftRobot*. The approach is aimed at realizing manipulation tasks of single robots or small teams of robots. The main idea [4] is that robotics software is developed against the modular *Robotics API* [1] and robot operations are executed with hard real-time guarantees on the underlying *Robot Control Core (RCC)*. Thus, the RCC is responsible for all real-time critical parts of the robotic systems – especially for controlling robotic devices [14]. This separation is possible, because robot applications do not inherently have real-time requirements for the application logic, but only for parts of the program like motions or tools actions.

KUKA was applying this approach to its novel *LBR iiwa* which is a lightweight robot designed as an intelligent industrial work assistant. The LBR iiwa and its Java-based Sunrise
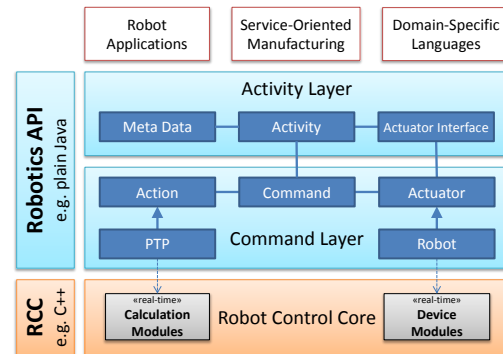


Fig. 1: The proposed software architecture for industrial robotics: While the RCC is responsible for real-time device control, the Robotics API allows object-oriented development.

controller form the core of Industry 4.0 applications such as human-robot collaboration and smart platforms [3]. In parallel, we develop this approach further towards an open framework for the object-oriented software engineering of industrial robotics cell and cyber-physical production systems [6]. This paper highlights its key features for (i) modeling robot cells, for (ii) planing tasks and collision-free motions as well as for (iii) controlling industrial robots with real-time guarantees.

## II. Object-oriented modeling in robotics

The Robotics API is a modular framework to model, program and control industrial robotics applications in an object-oriented manner. Its scope comprises both the modeling of robotics cells (cf. Sect. II-A) and, based on that, the execution of (real-time) robot commands (cf. Sect. II-B).

### A. Modeling of robot cells

Robotic applications always need to describe some part of the physical world or environment the robot is situated in. In standard robot programming languages (e. g., the KUKA Robot Language), one has to define at least points in space that are necessary for the specification of robot motions. Moving towards cyber-physical production systems [6], more information about both the robotics system and the environment is desired. For example, in assembly applications, the notion of workpieces that have to be assembled can be helpful. Moreover, a virtual model of the robot cell will facilitate task and path planing. Therefore, the Robotics API allows the definition of spatial points, geometric relations and physical
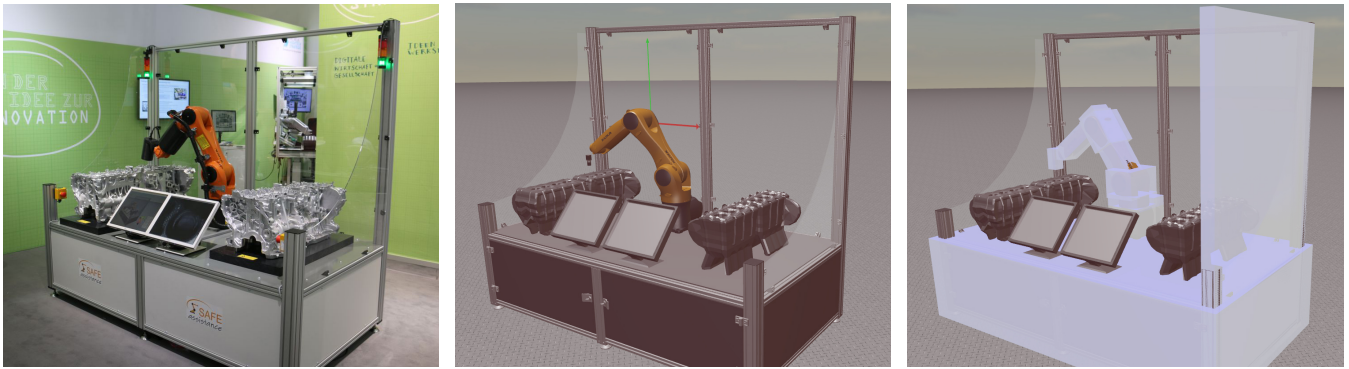
(a) Real robot cell for safe human-robot-interaction at Hannover Messe 2015

(b) Virtual model of the robot cell based on the object-oriented description

(c) Virtual robot cell partly augmented with collision hulls for path planning

Fig. 2: The Robotics API allows to create a virtual model of an industrial robotic cell with its physical objects.

objects as presented in Hoffmann et al. [5]. These entities are represented as software object which can be connected to form a *scene graph* and, thus, a virtual model of the industrial robotics cell. Fig. 2a shows, e. g., a real robot cell for safe human-robot-interaction at Hannover Messe 2015, whereas Fig. 2b shows its virtual model based on the object-oriented scene graph. As the Robotics API is extensible, this model can be customized to fulfill the particular requirements.

In contrast to existing approaches (cf. [13]), our scene graph is based on a graph of inter-connected frames. Frames are (named) positions in space which can optionally belong to (the software representation of) a device or in general a physical object. Two frame can be connected by a relation with a given semantics that defines the geometric transformation (cf. [1, 5]). While a *static* relation defines a fixed transformation (e. g., a given displacement between two objects), the transformation of a *dynamic* relation can change over time. For example, a revolute joint consists of two frames which are rotated against each other according to the joint's current position. The transformation of dynamic relations will be automatically provided by the framework. Besides static and dynamic relations, we distinguish between connections and placements. Where placements can be changed or removed at any time (e. g., to indicate that some object is currently placed on a table), connections are long-lasting relations between frames, e. g., a (mechanical) connection between two joints.

Physical objects are software objects that have a counterpart in the robot cell. They can be either *passive* objects such as work-pieces or *active* objects (e. g., robots or tools). Physical objects can consist of multiple parts (e. g., a robot arm consists of links and joints) whereas their spatial relation is defined through frames that belong to them. Moreover, different physical objects can be arranged to build up a robotics cell: a robot arm is placed on a workbench and a tool adapter and a gripper are mounted on its flange. By having semantically enriched relations between frames, conclusions about the scene graph can be drawn. For example, before grasping an object it can be inspected whether it is placed on a table (using a placement) or fixed to it (using a static connection).

### B. Modeling of capabilities

The Robotics API has two mechanisms for making the capabilities of robots, sensors, or tool available (cf. Fig. 1). At the *Command Layer*, there is a very abstract model of robotic tasks, which consists of actions which can be performed by actuators. Assigning an action to an actuator yields a command, which can be submitted to a RCC and executed there. Actions can be for example point-to-point or linear motions, and an actuator can be a specific type of robot arm. Actuators need a corresponding driver and primitives in the RCC, whereas actions usually map to a set of calculation modules which together form the requested action [12]. Furthermore, multiple commands can be combined using an event mechanism, e. g., to trigger tool operations based on sensor data.

In contrast, the *Activity Layer* provides an intuitive and more convenient access to the capabilities of robotic devices. On this level, actuators offer a set of interfaces, each providing specific activities that the actuator may execute. For instance, robots offer a *MotionInterface*, providing activities for different kinds of motions (e. g., to absolute goals in space or to goals specified relative to the current position). An activity is defined as a real-time critical operation, affecting one or more actuators that supplies meta data about the state of each actuator during or after the execution of the operation. However, the real-time critical execution logic is completely implemented using the above mentioned commands. Activities are characterized by a particularly designed asynchronous execution semantics. This semantics allows for easily specifying continuous execution of real-time operations on a programming language level. In addition, there exist some predefined ways of combining activities (e. g., for parallel or conditional execution).

### III. PLANNING BASED ON AN OBJECT-ORIENTED MODEL

This section shows with several examples how an object-oriented model of an industrial robot system can be utilized for different kinds of planing – either off-line for programming and commissioning the system or at runtime to adapt to new or changed tasks.

## A. Process planning

There are robot-based production processes such as the manufacturing of carbon-fiber reinforced polymers (CFRP) which consist of many *similar* production steps that may slightly differ in robot movements or end-effector control depending on various parameters. However, some of these parameters can be *computed* based on geometric information included in a virtual model of the robot system. Other parameters are based on *domain expert knowledge* and, thus, depend on user input (e. g., the maximum speed during handling fragile carbon-fiber textiles). Hence, we have developed an offline programming platform [11] tailored to the requirements of CFRP manufacturing and integrated an approach [9] for semi-automatically programming such manufacturing processes with industrial robots. This approach uses the concept of *task contribution units* which provide a flexible and extensible ecosystem of planing modules. Each module can solve different task based on the object-oriented model of the robot system and the manufacturing process, e. g., to plan robot motions, to compute end-effector poses, or query expert user input if automatic planing is not feasible.

## B. Path planning

The object-oriented description and, thus, the virtual model of an industrial robot system can also be used for both on-line and off-line path planning. For this purpose, the virtual model can be augmented with approximate collision hulls (cf. Fig. 2c) or if required with detailed collision models. These models are used in a physics simulation which can be employed for collision detection or even further physical computations. Based on the collision detection, we implemented an efficient, single-query, collision-free path planner. The planner incrementally creates two rapidly-exploring random trees [8] rooted at the start and the goal configurations. Moreover, the implementation uses path optimizers that improve the paths constructed by the planner in terms of size and time. Experimental results have shown that the path planner performs fast enough to be used on-line. Hence, it can conveniently be accessed through an *CollisionFreeMotionInterface* (see Sect. II-B).

Moreover, we have shown in Angerer et al. [2] that a CFRP manufacturing process (i. e., the automated production of an aircraft fuselage) involving cooperating robots can be planned and simulated offline and, subsequently, can be deployed to the real cell for execution without changes. The system contains a collision-free path planner for cooperating robots which also uses a simulation environment for collision detection. The execution of cooperating robot motions is performed using the Robotics API.

## C. Grasp planning and optimization

In robot-based CFRP manufacturing, huge end-effectors are used to handle, transform and drape carbon-fiber textiles [11]. Similar to classical grasp planning, a common problem is to compute a draping position and end-effector deformation which optimally fits the transformed gripper onto the target position of the textile cut-piece. This is shown in Fig. 3a where



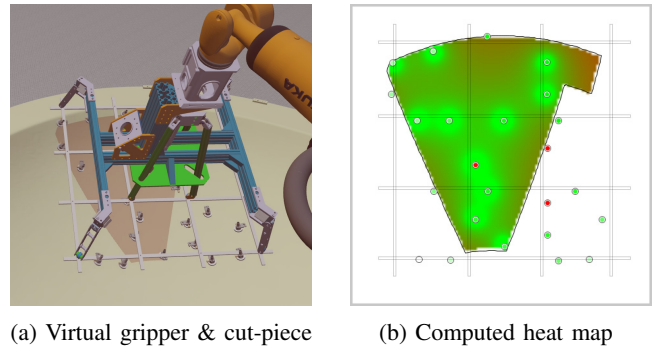(a) Virtual gripper & cut-piece     (b) Computed heat map

Fig. 3: Geometrical and physical grasp analysis and optimization by heat map computation

the vacuum modules of the end-effector must be fitted onto the cut-piece by either transforming the gripper or moving the robot. Fig. 3b shows a heat map with the coverage of gripping vacuum modules above the cutpiece. To compute this heat map, a detailed collision model is extracted from the object-oriented description of the robot manufacturing system. Distances between vacuum modules and the cut-piece are calculated by a physics engine. Subsequently, the coverage of vacuum modules on the textile is determined which indicates the quality of a draping position with different colors in the heat map. Furthermore, the heat map was utilized to automatically compute an optimal draping position with an evolutionary optimization algorithm. This algorithm retrieves genetic variations of possible draping positions of the gripper along with their collision information from the physics engine. The results are evaluated relating to their respective fitness, i. e., the coverage of vacuum modules on the textile and their collision distances to the form.

## IV. REAL-TIME CONTROL OF ROBOTIC DEVICES

While the Robotics API allows to model the robot cell, the *Robot Control Core (RCC)* takes care of all real-time critical parts (cf. Fig. 1). The reference implementation – the SoftRobot RCC – is implemented in C++ and is targeting Linux Xenomai for robot control as well as Windows for simulation purposes. It provides a flexible, generic interface: the *Realtime Primitives Interface (RPI)* is a dataflow based language, consisting of basic calculation modules, which can be combined to form complex commands. Besides these basic calculation modules, there are also device modules for sending data to or retrieving data from sensors and actuators. As mentioned in Sect. II-B, the Robotics API automatically combines these calculation and device modules to create real-time commands and submits them to the RCC for execution.

Once the RCC has fully received such a command, it periodically executes all modules with real-time guarantees. During execution, sensor drivers provide (raw or processed) sensor data at a high frequency. Calculation modules can further process such sensor data, calculate set-points for trajectories, or trigger tool actions. Finally, actuator drivers expect to

(a) Cooperation of two KUKA LBR 4     (b) Cooperation between a Staubli TX90L and a KUKA KR16     (c) Cooperation between KUKA industrial robots at the DLR-ZLP in Augsburg [7]

Fig. 4: Examples for cooperating industrial robots which are currently supported by the Robotics API.

receive set-points (e. g., position, velocity, acceleration) at a high frequency (usually 1 kHz) from a running command.

Currently, it is possible to control a wide range of different manipulators and end-effectors with the presented approach. For example, the Robotics API supports the programming of several light-weight robots (e. g., the KUKA LBR 4 and its successor LBR iiwa), low payload industrial robots such as a Staubli TX90L or a KUKA KR16 and even high payload robots (e. g., KUKA KR 270). Moreover, different mobile platforms and manipulators are included (e. g. Segway RMP, KUKA youBot). Due to the flexible approach, it is even possible to coordinate different kinds of industrial robots with real-time guarantees. Fig. 4 shows several examples of robot cooperation performed with the Robotics API[1].

## V. CONCLUSION

In this paper, we have shown how an object-oriented software framework – the Robotics API – can be useful for modeling industrial robot system. Based on that virtual software model of the robot system, several planing algorithms can be applied. Moreover, it is possible to directly control the devices with real-time guarantees. From our point of view, using such an inherently integrated approach can facilitate the software development of cyber-physical production systems.

## REFERENCES

[1] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif. Robotics API: Object-oriented software development for industrial robots. *J. of Softw. Eng. for Robotics*, 4(1), 2013.

[2] A. Angerer, A. Hoffmann, L. Larsen, M. Vistein, J. Kim, M. Kupke, and W. Reif. Planning and execution of collision-free multi-robot trajectories in industrial applications. In *Proc. 47th Intl. Symp. on Robotics, Munich*. VDE, 2016.

[3] M. Haag. Kollaboratives Arbeiten mit Robotern – Vision und realistische Perspektive. In *Zukunft der Arbeit in Industrie 4.0*. Springer, 2015.

[4] A. Hoffmann, A. Angerer, Frank 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 Intell. Robots and Systems, St. Louis, USA*, 2009.

[5] A. Hoffmann, A. Angerer, A. Schierl, M. Vistein, and W. Reif. Service-oriented robotics manufacturing by reasoning about the scene graph of a robotics cell. In *Proc. 45th Intl. Symp. on Robotics, Munich*. VDE, 2014.

[6] H. Kagermann, W. Wahlster, and J. Helbig, editors. *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. acatech, 2013.

[7] F. Krebs, L. L.en, G. Braun, and W. Dudenhausen. Design of a multifunctional cell for aerospace CFRP production. In *Advances in Sustainable and Competitive Manufacturing Systems*, LNME. Springer, 2013.

[8] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. 2000 IEEE Intl. Conf. on Rob. and Autom., San Francisco, USA*. IEEE, 2000.

[9] M. Macho, L. Nägele, A. Hoffmann, A. Angerer, and W. Reif. A flexible architecture for automatically generating robot applications based on expert knowledge. In *Proc. 47th Intl. Symp. on Robotics, Munich*. VDE, 2016.

[10] Laszlo Monostori. Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia CIRP*, 17:9–13, 2014.

[11] L. Nägele, M. Macho, A. Angerer, A. Hoffmann, M. Vistein, M. Schönheits, and W. Reif. A backward-oriented approach for offline programming of complex manufacturing tasks. In *Proc. 6th Intl. Conf. on Autom., Robotics and Applications, Queenstown, N. Zeeland*. IEEE, 2015.

[12] A. Schierl, A. Angerer, A. Hoffmann, M. Vistein, and W. Reif. From robot commands to real-time robot control. In *Proc. 9th Intl. Conf. on Inform. in Cont., Autom. & Robot., Rome, Italy*, 2012.

[13] R. Smits. *Robot Skills: Design of a constraint-based methodology and software support*. PhD thesis, KU Leuven, 2010.

[14] M. Vistein, A. Angerer, A. Hoffmann, A. Schierl, and W. Reif. Flexible and continuous execution of real-time critical robotic tasks. *Intl. J. Mechatronics & Autom.*, 4 (1), 2014.

---

[1]See also the video at: http://video.isse.de/spaghetti/