TOWARDS OBJECT-ORIENTED SOFTWARE DEVELOPMENT FOR INDUSTRIAL ROBOTS*

Facilitating the Use of Industrial Robots by Modern Software Engineering

Alwin Hoffmann, Andreas Angerer, Andreas Schierl, Michael Vistein and Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, Augsburg, Germany

Keywords: Industrial robotics, Object-oriented software development, Robot programming, Software engineering.

Abstract: Industrial robotics is characterized by sophisticated mechanical components and highly-developed control algorithms. However, the efficient use of robotic systems – with regard to flexibility, reusability and extensibility – is very much limited by existing programming methods. As a consequence, software development for industrial robots is a complex and time-consuming task which generates considerable costs. This work performs an analysis of the current software development for robotics systems and identifies shortcomings from a software engineering point of view. Based on that, it outlines an architectural approach that addresses the identified problems and allows efficient software development for industrial robotic systems.

1 INTRODUCTION

According to ISO standard 8373, an industrial robot is defined as "an automatically controlled, reprogrammable, multipurpose manipulating machine [...] for use in industrial automation applications". Given an appropriate tool, industrial robots are able to perform a large variety of tasks ranging from assembly over welding to quality assurance. But when looking deeper, the use of robots in companies is mostly restricted to mass production with rather easy, recurring tasks. Although small and medium enterprises with their small production batches could benefit from flexible production systems, industrial robots are rarely found there. One major reason for the limited use is the way of adapting robots to perform a task. Today, industrial robots are still programmed with special robot programming languages which have robotic-specific data types, allow the specification of motions, and communicate with external devices and systems via fieldbus (e.g. tools, sensors, or PLCs). Programs written in these languages are executed in special runtime environments by stepwise interpretation of instructions and submitting them to the underlying robot controller.

Due to these low-level programming techniques, developing software for an industrial robot is a complex and tedious task requiring considerable technical expertise. Already in 1990, (Miller and Lennox, 1990) identified the fundamental problems of programming robots which are still relevant in industrial robotics even two decades later. The problems mentioned are e.g. high costs for application development, low reusability of robot-specific source code, and the complexity of extending existing applications with new tasks or devices. Despite these problems, research in software engineering for robotics is mostly targeting experimental robotics (Brugali, 2007), because industrial robotics has been considered an already solved problem for a long time (Hägele et al., 2008).

Hence, this paper analyzes the software development for robotics systems and explains why the current practice has shortcomings from a software engineering point of view (Sect. 2). Subsequently, we briefly present an object-oriented approach that can overcome those shortcoming in Sect. 3 and illustrate the possible benefits in Sect. 4. Finally, conclusions and future work are presented in Sect. 5.

2 PROBLEM ANALYSIS

Software development for commercially available in-

*This work presents results of the research project *SoftRobot* funded by the European Union and the Bavarian government within the *High-Tech-Offensive Bayern*. The project is carried out together with KUKA Roboter GmbH and MRK-Systeme GmbH.

Hoffmann A., Angerer A., Schierl A., Vistein M. and Reif W. (2010).
TOWARDS OBJECT-ORIENTED SOFTWARE DEVELOPMENT FOR INDUSTRIAL ROBOTS - Facilitating the Use of Industrial Robots by Modern Software Engineering.
In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics*, pages 437-440
DOI: 10.5220/0003002304370440

dustrial robots is usually performed using the development tools available from the robot manufacturers, where each manufacturer provides its own system. These systems are mostly based on proprietary, special programming languages which are tailored to the features of the underlying robot controller. Examples are the KUKA Robot Language or RAPID from ABB. These languages are rather limited compared to general-purpose programming languages. For example, there is no built-in support for the development of graphical user interfaces, and interaction with tools and external systems is based on low-level fieldbus communication. On the other hand, these languages focus on the basic features of automation systems and these limitations help ensuring safety and real-time capabilities, which are necessary to achieve precision and repeatability. Based on their basic languages, manufacturers provide programming extensions for special application domains like welding. These extensions contain macros for common application-specific tasks and allow to configure tools.

However, with increasing requirements to the applications, using robot programming languages alone is not sufficient any more. Especially user interaction or complex domain-specific application logic is often realized using a general-purpose programming language, while robot languages are only used for moving the robot and triggering exactly timed tool actions. The interplay between such an application and the robot controller depends on the use case. One approach is to generate robot source code and to transfer it to the robot controller. This approach is often used in off-line programming systems where a virtual model of the robotic workcell is used to program and simulate tasks.

A different approach is to use a pre-defined set of robot programs which can be remotely parameterized and executed by an application. Today, such a mixed application structure is typical for many existing solutions in industrial robotics. For example, (Ge and Yin, 2007) describe the implementation of a plastic injection molding application using the general-purpose programming language C# for the domain logic and the graphical user interface whereas RAPID was only used for commanding the robot. Another example is described by (Pires et al., 2009) where a programming-by-demonstration application is presented. The main application as well as a speech interface were again realized with C#. Although the force-controlled guiding system was directly implemented on the robot controller, it is forwarding its data to the main application for further processing. After having taught a new task by demonstration, robot code is generated and submitted to the controller. As these examples show, it is possible to implement complex domain-specific applications and customized user interfaces on top of existing robot programming environments. However, the result is always an individual solution, requiring great effort and expertise for development and, consequently, causing high costs. Evaluating these solutions from a software engineering point of view shows shortcomings in software quality. While functionality and usability is often increased using such approaches, other quality attributes (i.e. reliability, performance, maintainability, extensibility or portability) are influenced negatively. For example, robot motions that are triggered inside an application must always be backed with appropriate programs on the robot controller. Consequently, both application parts have to be maintained or extended carefully with mutual dependencies in mind. Approaches that rely on automatic generation of robot control code provide a higher level of extensibility (by modifying code generators) and increase portability (with multiple code generators). However, maintainability problems arise when generated code has to be adjusted: In most cases, these adjustments cannot be transferred back to the generating model. Further issues concern performance (especially for code generation) and reliability, as concurrent programs are difficult to predict, test and debug.

Standard business applications are nowadays developed on top of modern frameworks like Microsoft .NET or Java Enterprise Edition. These frameworks offer solutions for many standard problems like communication in distributed systems, persistency and security. Our goal is to show that with a novel approach robotic applications can be developed in the same way as business applications. By providing a robotic framework that already incorporates the required infrastructure solutions, such an approach can increase productivity, reduce costs and weaken the trade-off between functionality and software quality.

3 APPROACH

We have analyzed a broad variety of typical tasks for industrial robots (e.g. welding or palletizing). This analysis has shown that robot programs usually consist of a defined set of real-time critical control actions which are embedded into a high-level work flow. For example, welding a single seam on a workpiece is one real-time critical action consisting of operations of the welding torch during the robot's movement. The work flow of a welding application only coordinates the proper execution of these actions in order to weld

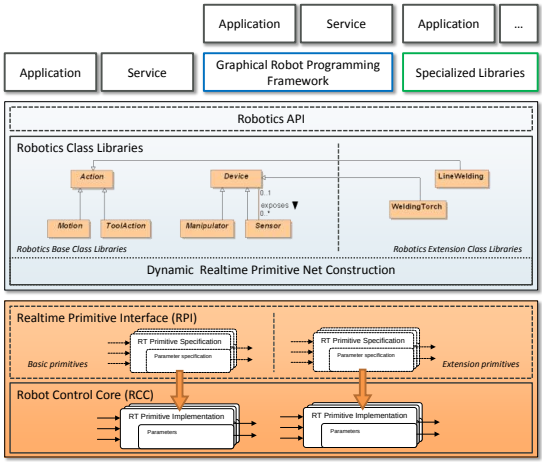

Figure 1: The components of the proposed multi-tier software architecture.

a series of seams. Hence, if it is possible to create an adequate abstraction layer for encapsulating such real-time critical actions, robotic applications can be programmed in any programming language or environment regardless of real-time issues.

Based on this observation, we have developed a multi-tiered software architecture (Hoffmann et al., 2009) which is shown in Figure 1. Core part of this architecture are the *Robotics Base Class Libraries (RBCL)*. They represent an object-oriented framework for robotic applications and provide an abstract class model for the industrial robotics domain as well as the aforementioned abstraction layer for specifying real-time critical actions. The domain model provides objects for common concepts like robots, tools, frames or instructions (e.g. motions or tool actions). They can be extended in order to encapsulate task-specific functionality (e.g. for arc welding) or to add new robots, tools and sensors. All functionality exposed by the RBCL and their extensions (*Robotics Extension Class Libraries*) forms the *Robotics Application Programming Interface (Robotics API)*. This interface is intended to be a broad platform for developers of robot programs. At the moment, there exists an implementations of the RBCL in C# which is able to control KUKA lightweight robots.

Instructions of the Robotics API can either be executed independently of each other, or be combined to larger instruction units by certain operators with defined semantics (e.g. temporal delay, superposition). When executed independently, no hard guarantees can be given concerning timing relations across multiple instructions. When executing complex instructions that have been pre-combined using one of the aforementioned operators, real-time deterministic execution is guaranteed. To achieve this, all instructions are dynamically translated into commands for the *Realtime Primitives Interface (RPI)* and executed by a *Robot Control Core (RCC)*. Such an RPI command consists of a combination of certain, predefined (yet extendable) calculation modules, and a specification of the data flow among them. Implementing only the RCC with real-time aspects in mind is sufficient to allow the atomic execution of Robotics API instructions under real-time conditions.

4 BENEFITS

Using this approach, applications for industrial robots can be mainly developed using standard technologies and environments. For example, a modern general-purpose programming language such as C# and a development environment such as Microsoft Visual Studio can be used. Common functionality of industrial robots is provided through a well-defined and extensible application programming interface (the Robotics API), and can be easily used by application developers. In consequence, they are able to focus on solving domain-specific problems and, as far as possible, do not have to deal with issues like communication and synchronization with the robot control or real-time programming.

Furthermore, the Robotics API provides a comprehensive, abstract model of the robotics domain – including physical objects, interesting points on those objects and ways to define specialized devices (e.g. a welding robot). This framework can be used to formulate certain tasks that are specific to the applications of interest (e.g. welding a line on a workpiece). Basic functionality can be reused, in particular existing mechanisms for encapsulating real-time critical work flows (e.g. moving along a welding line with exactly timed activation of a welding torch). With this abstraction, existing development processes, methods and tools of modern software engineering can be used to construct intuitive domain specific applications, regardless of real-time issues. As a consequence, complex application logic and high usability can be achieved, whereas maintainability, performance and reliability are taken care of by the automatic generation and execution of RPI commands.

Besides basic motion commands, our architecture also supports the integration of new motion types. The fine grained structure of RPI allows a flexible specification of motion control algorithms as e.g. required for sensor integration and compliant motions. The tight integration of high-level motion specification and low-level motion execution, which is achieved by the automatic generation and execution

of RPI commands, leads to good maintainability of resulting applications, and real-time constraints can be satisfied by an appropriate implementation of an RPI interpreter, ensuring performance and reliability for the critical system parts.

Traditionally, the communication with external devices and systems is performed using a fieldbus that is connected to a port with analog and digital I/Os. In our approach, the physical I/Os can be mapped to logical units, e.g. the necessary I/Os to drive a gripper can be controlled using a single gripper object. This object encapsulates the properties, i.e. the configuration, as well as the behavior of the tool and can be used for programing. Besides, complex devices can be controlled with real-time performance by introducing specialized RPI modules and corresponding objects for the Robotics API. Furthermore, instructions usually incorporate a two-stage error handling. If an error occurs during the execution of the instruction, basic error handling causes the robot system to reach a stable and secure state. Subsequently, an exception is thrown and a high-level recovery strategy can be executed. Thus, it is possible to guarantee reliable error handling and to provide high-level error mechanisms that developers can use.

There are some tasks which require the use of multiple robots, e.g. mobile manipulation scenarios or lifting large and heavy workpieces. Traditionally, each robot has its own controller and must be programmed individually, using fieldbus communication or special markers to handle synchronization with the other robots. Using our approach, it is possible to develop a single program that controls multiple robots. The robots can be synchronized with real-time performance using special operators offered by the Robotics API. It is even possible to define a logical unit, that consists of multiple robots, but can be programed as a single robot (e.g. for load sharing, or for a mobile manipulator).

5 CONCLUSIONS

This paper presents a new approach for the software development for industrial robots – an approach that is rather radical. Its focus does not originate from mechanical engineering or control theory, but from software engineering. Furthermore, it is not directly compatible with current robot controllers and even requires a new generation of robot control software. However, it shows a way to overcome the intrinsic problems and limitations of current programming environments for industrial robots. The development of this approach is performed in tight cooperation with industrial partners: KUKA Roboter GmbH, a specialist for robotics internals and Europe's leading robot manufacturer, and MRK Systeme GmbH, a system integrator working with KUKA robot and future user of our approach. This cooperation and the joint development of prototypical systems helped greatly in validating the feasibility and industrial meaningfulness of our research.

The presented approach promises advantages on the feature side – as it was designed with extensibility in mind – but the main benefits can be found coping with non-functional requirements. Most notably, usability and maintainability benefit from the advances achieved in the field of software engineering during the last 15 years, which can directly be applied to the robotics domain using our approach. Although the benefits and first results are promising, the approach must demonstrate its advantages in practice. Therefore, future work focuses on creating challenging application examples on top of our approach. This includes the development of a reusable graphical robot programming framework for SMEs and the introducing of service-oriented architectures for robot-based automation processes.

REFERENCES

Brugali, D., editor (2007). *Software Engineering for Experimental Robotics*. Springer Tracts in Advanced Robotics. Springer.

Ge, J. G. and Yin, X. G. (2007). An object oriented robot programming approach in robot served plastic injection molding application. In *Robotic Welding, Intelligence & Automation*, volume 362 of *Lect. Notes in Control & Information Sciences*, pages 91–97. Springer.

Hägele, M., Nilsson, K., and Pires, J. N. (2008). Industrial robotics. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, chapter 42, pages 963–986. Springer, Berlin, Heidelberg.

Hoffmann, A., Angerer, A., Ortmeier, F., Vistein, M., and Reif, W. (2009). 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, USA.

Miller, D. J. and Lennox, R. C. (1990). An object-oriented environment for robot system architectures. In *Proc. 1990 IEEE Intl. Conf. on Robotics and Automation*, pages 352–361, Cincinnati, Ohio, USA.

Pires, J. N., Veiga, G., and Araújo, R. (2009). Programming by demonstration in the coworker scenario for SMEs. *Industrial Robot*, 36(1):73–83.