UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

# Towards realtime robot reactions –patterns for modular device driver interfaces

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

**Angaben zur Veröffentlichung / Publication details:**

Schierl, Andreas, Alwin Hoffmann, Andreas Angerer, Michael Vistein, and Wolfgang Reif. 2013. "Towards realtime robot reactions –patterns for modular device driver interfaces." In *ICRA2012: Workshop on Software Development and Integration in Robotics (SDIR-VIII), May 6, 2013, Karlsruhe, Germany*. Augsburg: Universität Augsburg.

**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/*

# Towards Realtime Robot Reactions – Patterns for Modular Device Driver Interfaces

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

## I. Introduction

Getting new robot hardware to work requires the implementation device drivers. When this is seen as tedious work, initial versions of drivers often tend to lack clean software architecture or component design. This leads to drivers that work acceptably in the given context, but exhibit little modularity or reusability for other contexts. Especially for use cases that contain real-time robot reactions (reflexes) to sensor events, these implementations are often not usable. This paper describes different device driver implementation patterns found in existing robot software, and analyses them towards advantages and disadvantages, aiming to provide advice which pattern to use in which context.

## II. Patterns

The main point about device drivers is to implement the communication with specific hardware devices and to provide the functionality through a higher-level interface. Usually, communication to the real hardware is performed through a vendor-specific interface, often over bus systems such as USB, EtherCAT or other field bus systems. The corresponding driver implementation is required to understand the communication protocol, and to respect the timing requirements imposed by the device (e.g. the Fast Research Interface for the KUKA Lightweight Robot [1] [2], or cyclic data transfer for CANopen devices [3] [4]). This can implicitly lead to real-time requirements within the device driver implementation. When using such device drivers in (non-realtime) component frameworks, they are expected to provide an interface that can be used without real-time guarantees. To achieve this goal, different implementation patterns can be found, which are explained below.

### A. Drivers with included non-realtime interfaces

A straightforward method is to implement the device driver in a monolithic way, directly providing the required non-realtime interface from within the real-time device driver implementation. A structural view on this pattern is given in Fig. 1.

This method has the advantage of being (relatively) easy to implement, as it is somewhat minimal and no overhead is required. However, such drivers are not reusable or modular with respect to real-time, i.e. when a new interface to the device is required, the device driver implementation (containing all the vendor-specific code) has to be adapted and extended. Additionally, drivers following this pattern generally do not allow real-time reaction to (external) sensor events, e.g. to guarantee stopping a motion when certain sensor events occur (unless the sensor driver is implemented within the device driver).

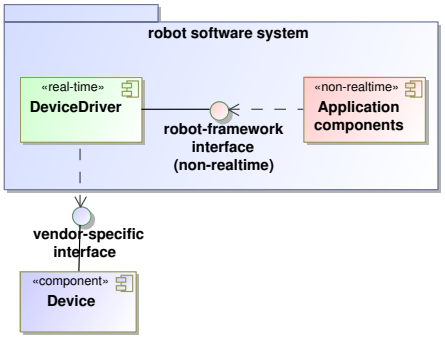


Fig. 1. Monolithic device driver

### B. Explicit interface components for devices

To cope with those shortcomings, the vendor-specific device driver implementation can be separated from the (non-realtime) interface to the component framework, resulting in two different components. Between these components, a real-time capable, device type specific interface has to be defined (cf. Fig. 2). Therefore, a real-time context is required, containing different components that can communicate with timing guarantees (e.g. by running those components within the same process on a real-time operating system [5] [6]).

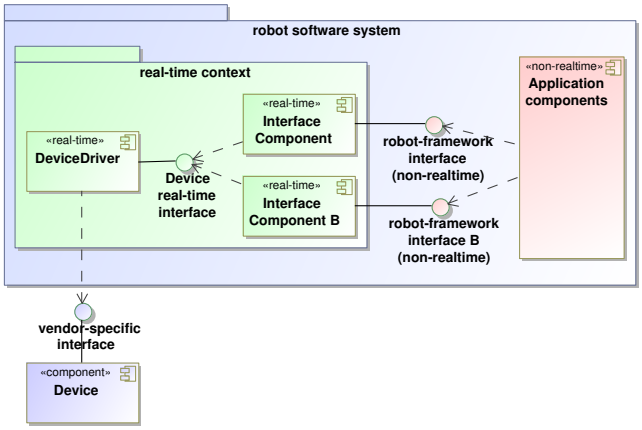


Fig. 2. Explicit interface components

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

In the robot case, using a cyclic position interface allows to abstract from hardware details while still providing enough flexibility to implement different framework interfaces. One interface component could provide a velocity interface to the robot, while the second can provide a trajectory following interface.

Using this pattern decouples the driver components from the interface components, making both of them reusable (the driver can be used with various interfaces, and one interface component can be used for all drivers that implement the same device type specific interface). However, this still does not yet allow real-time reaction to (external) sensor events, unless multiple device drivers can be placed within the same real-time context, and interface components communicate with and synchronize multiple devices (cf. Fig. 3).

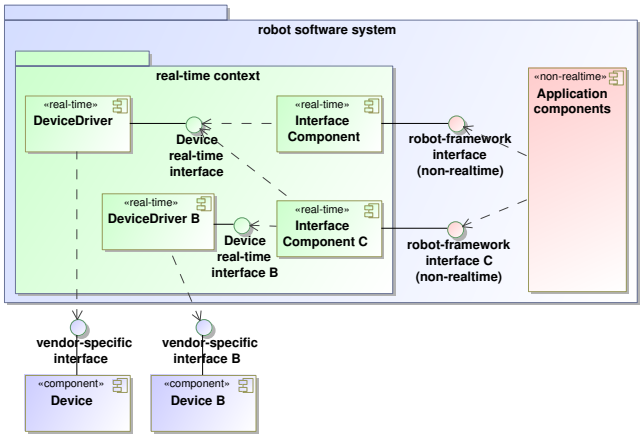


Fig. 3. Explicit interface components for multiple devices

One use of this pattern is to provide sensor-guarded robot motions. For example, once tactile sensors at the robot finger tips observe contact, the robot immediately reacts by following the observed contact so that the desired object can be grasped [7]. The single interface component responsible for this behaviour therefore uses a real-time connection to the sensor and the arm to handle the real-time reaction (reflex), while providing a non-realtime interface that allows to start or trigger this behaviour.

Despite the advantages of reusable components and the possibility of real-time reaction, there are still drawbacks: The interface components just provide application specific behaviour and have to be implemented with respect to real-time requirements. Additionally, as multiple interface components control the same device, synchronization and access control for the interface components is required.

### C. Using a generic interface component

To solve these problems, a generic interface component can be used (cf. Fig. 4). This component receives a task description through a non-realtime interface and executes it using the real-time interfaces to the device drivers. Tasks can e.g. be described through state machines [8], data-flow graphs [9] or constraints [10]. The non-realtime interface of the generic interface component can then be used either directly by applications, or through task-specific concrete interface components that abstract from the task details and make it easier to use the provided functionality.

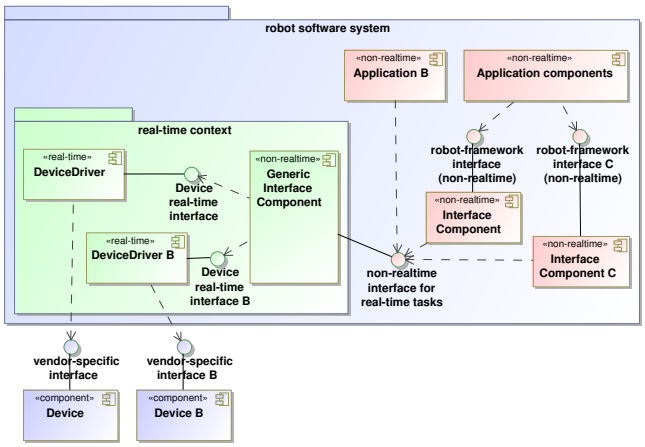


Fig. 4. Generic interface component

Using this pattern has the advantage that concrete interface components as well as applications can be implemented without real-time considerations while still being able to provide real-time reaction. Additionally, the generic interface component is flexible and reusable for different applications, as it does not contain task-specific implementations, and can handle real-time synchronization of devices and device access.

However, implementing a generic reusable interface component is complex and sometimes not justified for simple tasks, and its use causes some overhead introduced by the generic task description language and the additional level indirection.

## III. Conclusion

Usually, it is advisable to separate the device driver component from the non-realtime interface, as seen in pattern B and C, to improve reusability of device driver components. Implementing device driver components with a real-time interface allows them to be used with any kind of interface components, thus from their point of view the concrete use according to pattern B or C is just a matter of deployment. When an application requires real-time robot reactions, multiple device drivers have to be combined within one real-time context. This poses constraints on the deployment (runtime architecture), as these components have to be deployed correctly (e.g. within one process, or using a field bus with real-time guarantees). Apart from that, components can freely be deployed according to outside needs (computation power, availability). Using a generic interface component (pattern C) additionally helps to abstract from real-time concerns [11] and allows to specify robot behaviour including real-time reactions without the need for application-specific code in the real-time context.

## REFERENCES


[1] 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 Robot. & Autom.*, Anchorage, Alaska, USA, May 2010.

[2] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The KUKA-DLR lightweight robot arm - a new reference platform for robotics research and manufacturing," in *Proc. IFR Int. Symposium on Robotics (ISR 2010)*, 2010.

[3] IEC 61800-7-201, *Adjustable speed electrical power drive systems - Part 7-201: Generic interface and use of profiles for power drive systems - Profile type 1 specification*. ISO, Geneva, Switzerland, 2007.

[4] IEC 61800-7-301, *Adjustable speed electrical power drive systems - Part 7-301: Generic interface and use of profiles for power drive systems - Mapping of profile type 1 to network technologies*. ISO, Geneva, Switzerland, 2007.

[5] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Proc. 2001 IEEE Intl. Conf. on Robot. & Autom., Seoul, Korea*. IEEE, May 2001, pp. 2523–2528.

[6] K. Buys, S. Bellens, N. Vanthienen, W. Decre, M. Klotzbücher, T. De Laet, R. Smits, H. Bruyninckx, and J. De Schutter, "Haptic coupling with the PR2 as a demo of the OROCOS-ROS-Blender integration," in *The PR2 Workshop, IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, San Francisco, USA, Sep. 2011.

[7] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones, "Contact-reactive grasping of objects with partial shape information," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct., pp. 1228–1235.

[8] M. Klotzbücher and H. Bruyninckx, "Coordinating robotic tasks and systems with rFSM statecharts," *J. of Software Engineering for Robotics*, vol. 3, no. 1, pp. 28–/56, 2012. [Online]. Available: http://joser.unibg.it/index.php?journal=joser&page=article&op=view&path%5B%5D=52

[9] M. Vistein, A. Angerer, A. Hoffmann, A. Schierl, and W. Reif, "Interfacing industrial robots using realtime primitives," in *Proc. 2010 IEEE Intl. Conf. on Autom. and Logistics, Hong Kong, China*. IEEE, Aug. 2010, pp. 468–473.

[10] R. Smits, T. D. Laet, K. Claes, H. Bruyninckx, and J. D. Schutter, "iTASC: a tool for multi-sensor integration in robot manipulation," in *Proc. IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2008.

[11] 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 Intell. Robots and Systems, St. Louis, Missouri, USA*. IEEE, Oct. 2009, pp. 2108–2113.