A Plug and Produce Framework for Industrial Collaborative Robots

Schou, Casper; Madsen, Ole

Published in:
International Journal of Advanced Robotic Systems

DOI (link to publication from Publisher):
10.1177/1729881417717472

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Research Article

INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS

International Journal of Advanced Robotic Systems
July-August 2017: 1–10

DOI: 10.1177/1729881417717472
journals.sagepub.com/home/arx
SAGE

A plug and produce framework for industrial collaborative robots

Casper Schou and Ole Madsen

Abstract

Collaborative robots are today ever more interesting in response to the increasing need for agile manufacturing equipment. Contrary to traditional industrial robots, collaborative robots are intended for working in dynamic environments alongside the production staff. To cope with the dynamic environment and workflow, new configuration and control methods are needed compared to those of traditional industrial robots. The new methods should enable shop floor operators to reconfigure the robot. This article presents a plug and produce framework for industrial collaborative robots. The article focuses on the control framework enabling quick and easy exchange of hardware modules as an approach to achieving plug and produce. To solve this, an agent-based system is proposed building on top of the robot operating system. The framework enables robot operating system packages to be adapted into agents and thus supports the software sharing of the robot operating system community. A clear separation of the hardware agents and the higher level task control is achieved through standardization of the functional interface, a standardization maintaining the possibility of specialized function features. A feasibility study demonstrates the validity of the framework through a series of reconfigurations performed on a modular collaborative robot.

Keywords

Collaborative robot, multiagent system, plug and produce, human robot interaction, modular architecture, robot operating system, primitives, robot reconfiguration

Date received: 25 May 2016; accepted: 19 May 2017

Topic: Robotics Software Design and Engineering (ROSENG)
Topic Editor: Anis Koubaa
Associate Editor: Salvatore Graziani

Introduction

In response to the challenges derived from the globalization, manufacturing companies today face the need for more flexible and agile manufacturing equipment. Traditional industrial robots constitute a flexible manufacturing resource as they hold the option to be reprogrammed to perform new tasks. However, once the robot is commissioned in a manufacturing line, it is often fixed to a dedicated workstation doing a single repetitive task. As a result, the original flexibility is not utilized. Collaborative robots, on the other hand, are intended to operate in the more dynamic production environment of the human operators with lower batch sizes, greater variety, diverse tasks, and more frequent changeovers. In this context, the reconfiguration of the robot to a new task should no longer be an engineering task but should be handled by the production staff. However, this requires new approaches to configuring (installing, equipping, programming) and operating the collaborative robot as compared to a traditional industrial robot.[1]

The transitioning of a collaborative robot to a new task covers two main reconfigurations: (1) programming the robot to the new task, and (2) configuring the hardware of the robot.

Department of Materials and Production, Aalborg University, Aalborg, Denmark

Corresponding author:
Casper Schou, Aalborg University, Fibigerstraede 16, Aalborg 9220, Denmark.
Email: cs@m-tech.aau.dk

In the industry, the variation in tasks is often of such magnitude that they cannot be solved by a single hardware configuration.[2] Thus, the need for hardware reconfiguration emerges.

In previous research, we have focused on intuitive robot programming and industrial applications for collaborative robots.[3,4] Through this work it has become clear to us that as the task variety increases, simply reprogramming the robot is not sufficient; the hardware must be reconfigured as well. Thus, it is our experience from prior research that hardware reconfiguration is inevitable in industry. Consequently, our ongoing research on hardware reconfiguration for collaborative robots is rooted in these practical experiences and applications.

Recent work by Schou and Madsen[5] propose the following four research objectives as a roadmap toward an intuitive hardware management framework that enables shop floor operators to perform the hardware reconfiguration of a collaborative robot:

- modular architecture,
- module selection,
- module exchange, and
- module utilization.

If shop floor operators are to exchange hardware modules of the robot at the shop floor, the exchange must be fast and efficient, and it should not require mechanical, robotics, or programming expertise. Hence, the hardware exchange should be done in a *plug and produce* manner. Realizing this requires not only standardization of the physical interfaces between the modules but also standardization of the control interfaces. Schou and Madsen[5] present a conceptual overview of an envisioned architecture for a plug and produce framework.

In this article, we propose, design, and demonstrate a plug and produce framework for modular collaborative robots based on robot operating system (ROS).[6] We adopt the architecture outline from Schou and Madsen[5] and use this outline to propose an architecture for a hardware management framework. The framework allows ROS drivers from the ROS community to be used within the framework with only very limited adaptation. In extend to the hardware management framework, we also propose a function generalization which supports the use of both generic and specific device functionality.

The remaining part of the article is structured as follows. Related research is presented in the second section. The third section describes the concept and implementation of the hardware management framework. Results from a feasibility study of the presented framework are described in the fourth section, and the last section draws the conclusions.

Related research

The idea of plug and produce was first proposed by Arai et al.[7] as a response to the need for agile manufacturing systems. The term is derived from the "plug and play" concept of the IT world. The purpose of plug and produce is to enable quick (un)plugging of components from a manufacturing system with little to no reprogramming and reconfiguration of the remaining system. Since the manufacturing equipment domain is vast, complex, and lacks standardization of interfaces, a modular hardware architecture is often introduced to encapsulate components as modules with well-defined interfaces. Several authors have presented modular hardware architectures for robotics.[8–17] In extend to the physical structure, the control and communication architecture must also be considered. One approach is agent-based systems in which active modules become independent agents. Agents have some degree of self-contained control and can provide and request functionality to the rest of the system. Agent-based systems or multiagent systems originate from the computational domain; however, agent-based approaches have been proposed in many different aspects of manufacturing enterprises.[18] Within the domain of manufacturing equipment, multiagent systems have been proposed on several technical granularities.[19] Some are focused on the production line or system level, where each agent thus becomes individual stations or machines. Other focus on the machine level with individual devices as agents. The latter being related to the structure of a collaborative robot.

In the EU FP6 project "EUPASS,"[20] a multiagent system architecture was developed defining both hardware and control interfaces of assembly systems. The architecture covers automation equipment used for precision assembly in electronics manufacturing; thus, this includes robots and robot modules.[21] Extending the results of EUPASS, the EU FP7 project "IDEAS"[22] developed an integrated agent control board used as a proxy to adapt legacy components into agents.[23] The proposed framework and agent controller are tested through a series of industrial experiments which demonstrates the viability of the agent-based approach for shop floor reconfiguration of manufacturing systems in real-world settings.[23] In the EU FP7 project "PRIME,"[24] a multiagent system architecture was proposed which includes both standardized hardware and control interfaces as a means to developing highly adaptable and reconfigurable plug and produce systems. In PRIME, explicit focus was given to adapting legacy components into agents.[25] Despite their high relevance to this work, EUPASS, IDEAS, and PRIME all focus on multiagent systems on a manufacturing system level. All three projects include robotics in their architecture, but they do not present a detailed approach and decomposition for collaborative robots.

In the study by Andersen et al.,[26] a control framework specifically for a collaborative robot is presented. The framework enables reuse of both hardware and control modules and furthermore enables online exchange of hardware components. The article describes the overall concept and architecture, but only very few implementation details

are provided. Furthermore, the task control architecture of the study by Andersen et al.[26] uses a taxonomy with a less clear separation between higher level task control modules and low-level device functionalities.

In this article, we describe the design, implementation, and test of a plug and produce framework for industrial, collaborative robots. Several plug and produce and agent-based frameworks for robotics have been proposed in literature; however, only few of them present implementation details. Given that the usage and acceptance of ROS is widespread within the collaborative robotics community, we see the need for a compatible plug and produce framework. The proposed framework provides necessary adaptation for any ROS package to be used in the framework. It is important to note that ROS provides the implementation infrastructure, and it does not in itself provide a plug and produce solution. As part of the plug and produce framework, a function generalization is introduced to clearly separate the task control system from the device-level control framework. The focus of this article is the system-level design of the framework. As a result, we built the framework on well-established communication and architectural schemes.

Hardware management framework

The motivation of the hardware management framework is to introduce an agent-based management and control scheme for a modular hardware architecture based on ROS. The framework must be separated from the task control system and thus be independent from the task control's internal architecture. The goal of which is to make the task control system independent of specific hardware configurations. Thus, the task control becomes solely focused on the task related goals, while the device control system focuses on achieving the actions requested. The framework manages the connected modules and introduces a standardized control interface between the hardware modules and the task control system. The latter is done using a set of general functions called *primitives* serving as a common function interface between the task control and the hardware devices.

The hardware management framework takes it offset in an agent-based architecture. It builds on the architecture outline introduced by Schou and Madsen,[5] which in this work has been developed into further details. The resulting architecture used in the hardware management framework is presented in Figure 1.

Each physical device (e.g. gripper, camera, robot arm, etc.) is represented by an associated *device driver* acting as an agent and providing interaction with the given device. The *device manager* manages the connected devices and their respective device drivers. It keeps track of the available functionality and provides hardware independent and abstract functions to the task control layer. For clarity, the specific functions provided by the devices are referred to as

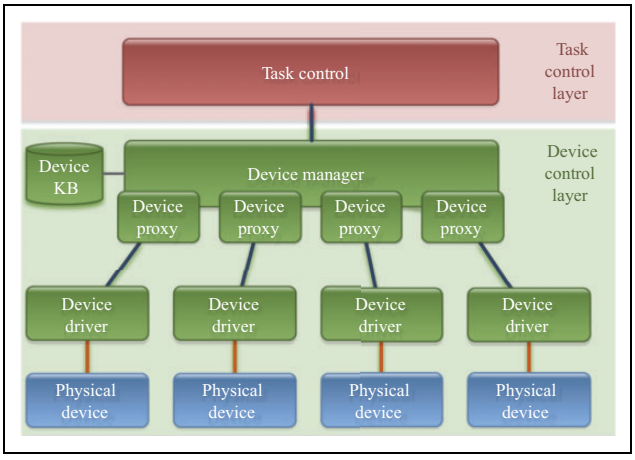

Figure 1. Architecture of the agent-based hardware management framework. The device manager with a set of device proxies is introduced as an intermediate control agent between the task control and the device drivers. It provides both agent management and a standardization of the control interface of the device drivers.

device functions and the generalized hardware functions are referred to as primitives.

To facilitate the communication between the different software nodes, ROS is used. First and foremost, the community behind ROS offers open source sharing of software, for example, device drivers. The possibility to download ROS device drivers provides a great advantage in terms of development resources; especially on a collaboraitve robot composed of commercial-of-the-shelf components.

The following sections will elaborate on each node in the architecture as shown in Figure 1.

Device driver

The device drivers constitute the agents of the system. Each active hardware module has a corresponding software driver, which in Figure 1 is referred to as the device driver. On one side, the device driver provides the low-level communication and control toward the device. Such communication is highly device specific and dependent on the physical connection to the device. On the other side, the device driver implements a ROS interface toward the device manager, through which the driver provides a set of functions. Given that drivers might be downloaded from the ROS community, the implementation of the ROS interfaces of the drivers will be diverse, that is, with diverse syntax, structure, and communication structure. Figure 2 illustrates the communication type and data flow between the nodes.

Device proxy

With the presented framework, the task control operates on abstract and generalized functionalities, , that is, primitives. However, at some point through the control pipeline (see

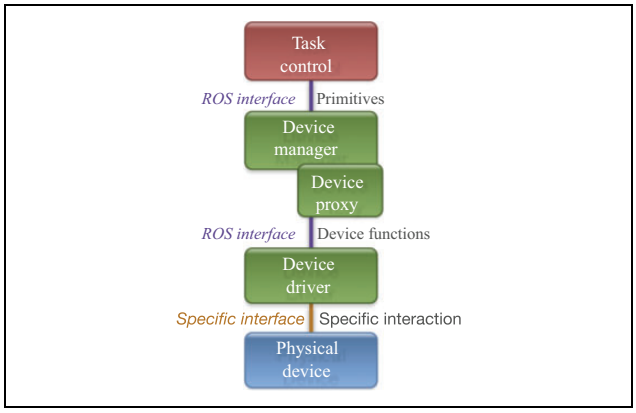

Figure 2. Subset of Figure 1 illustrating the interface types and data flow. The italic text denotes interface type. The text to the right of the connection denotes the data passed.

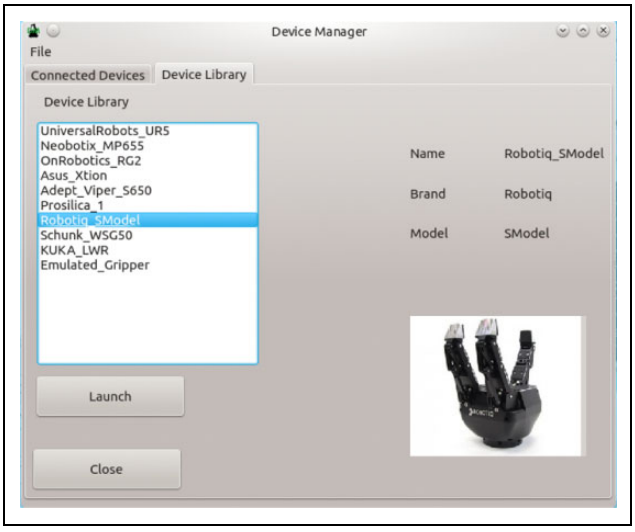

Figure 3. Screenshot of the GUI of the device manager. The image shows the library tab, where all known devices are listed. GUI: graphical user interface.

Figure 2), the primitive must be "translated" into the syntax and structure of a specific device function on a specific device. The device proxy is introduced to perform this translation, and consequently the main purpose of the device proxy is to provide a mapping from primitive to device-specific syntax. The device proxies are created as dynamically linked libraries, which are loaded during runtime by the device manager.

Device manager

The device manager is the central node in the hardware management framework. It connects the above task-level control system to the various device drivers. The two main tasks of the device manager are to (1) register and manage connected agents and (2) designate incoming primitive requests to a specific device and pass the primitive to the associated proxy. To keep track of the connected devices, the device drivers (agents) are registered on the device manager upon their launch. This will be explained in further details in "Device registration" section. An incoming primitive request from the task control system must be matched against the set of available primitives. The device manager searches for a matching available primitive and passes the request to the related proxy. This will be elaborated in "Function generalization" section. The device manager also provides a graphical user interface (GUI) (see Figure 3). Through the interface, a list of all connected and running device drivers can be monitored. The library of all known devices is available in the GUI, from which the device drivers can be manually started.

Task control

The device manager, device drivers, and device proxies together constitute the nodes of hardware management framework. Above is the *task control layer*, which is responsible for the higher level robot control and, hereby, the accomplishment of task-related goals. As we will not discuss the structure of the task control layer in this paper, it is only represented by a single node (*task control*) in Figure 1. However, as long as it complies with the interface described in "Function generalization" section, the structure of the task control is irrelevant. In other words, any task control system would apply as long as it interacts with the device manager in terms of primitives. In conjunction with this work, we have used a task control system developed for intuitive, manual task-level programming. The system is build on the concept of skills, which serve as task-related control modules that can be concatenated and parameterized to form a sequence of operations leading to the accomplishment of a task. Details on this work can be found in Schou and Madsen, 2016; Pedersen et al., 2016.[5,29]

Device knowledge base

In order to both successfully register a device and subsequently utilize the functions provided by the device, the device manager needs information about the device in question. This information is stored in the Device Knowledge Base (see Figure 1). The device knowledge base is an ontology containing semantic descriptions of both the robot equipment domain and of specific device variants. The semantic knowledge base is formatted using the Ontology Web Language 2.[27] By using an ontology-based knowledge base, the device information could potentially come from external sources or be shared via the "Semantic Web"[28] between robots. In order for the device manager to recognize, register, and utilize a device, sufficient semantic information about the device must be available in the device knowledge base. At the very least, the following information must be available:

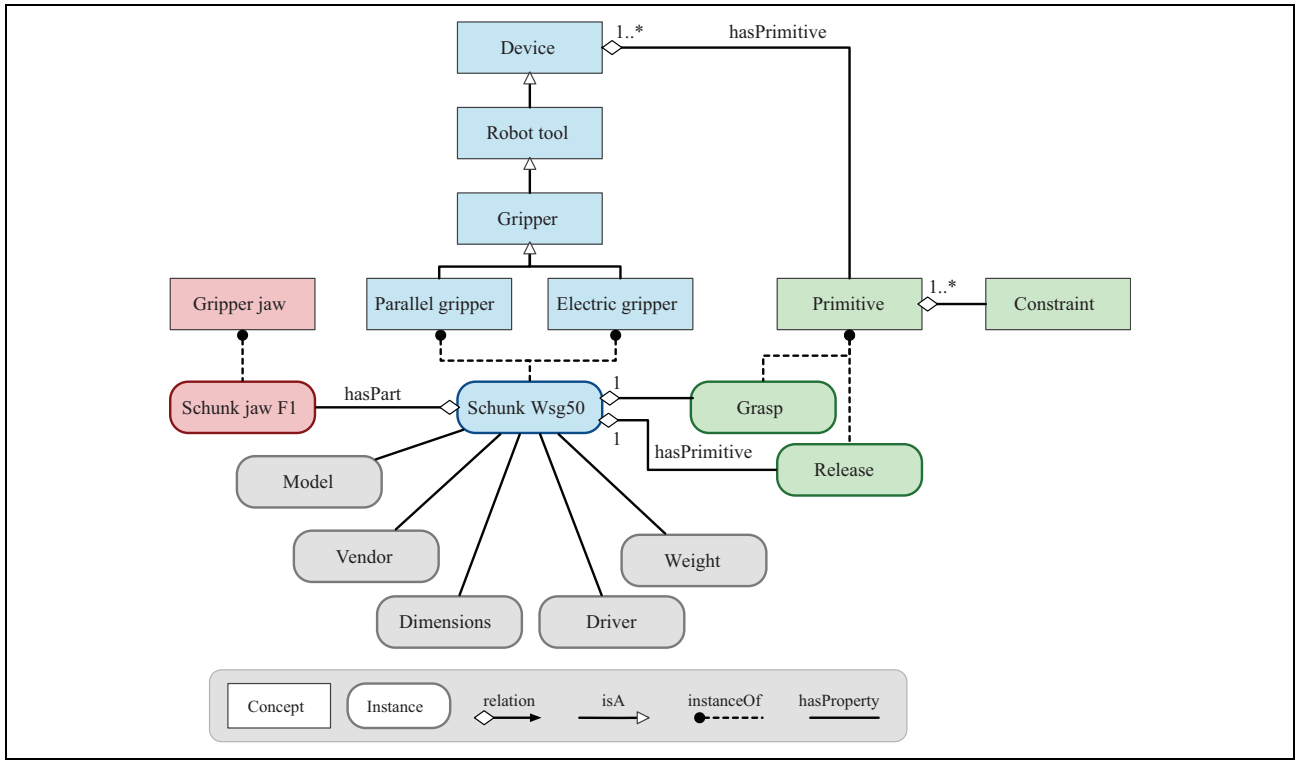

Figure 4. Excerpt of knowledge from both T-box and A-box of the device knowledge base illustrating the relations between a specific gripper (Schunk WSG50) and two primitives.

- device identification,
- device classification information,
- spatial information,
- driver information, and
- device functionalities.

The device identification denotes brand, model-name, and other information used to uniquely identify a given device variant. This is accompanied by spatial information such as geometry and weight. The driver information is used by the device manager to launch the associated ROS driver for the particular device. Lastly, a list of functionalities for the particular device is necessary in order to determine which primitives can be associated with the given device.

An excerpt of the ontology illustrating the relations between a specific gripper and primitives is depicted in Figure 4.

Device registration

In this section, the registration of newly connected devices is described. The approach is inspired from service oriented architectures and is which in this work used as a well-established method for handling the registration.

When a new device is connected and the associated device driver is launched, the driver should automatically be registered by the device manager and included as an active agent in the system. However, downloaded ROS drivers do not include this functionality, and consequently device drivers must be adapted to conform to the agent-based system. The solution is a precompiled shared object providing the agent interaction protocol, which is then augmented to the device driver. This shared object provides the methods for the agent advertisement, registration, and continuous interaction with the device manager. Figure 5 presents a sequence diagram of the device driver registration process.

In the registration process, the driver will include the device name and device type. The device name is the brand and model combined into a single string. The device type emerges from a classification of each module according to a taxonomy created on hardware modules for industrial collaborative robots in Hvilshøj, 2012.[30] A few example types would be gripper, robot arm, camera, and pan-tilt unit. Based on the device name and device type, the device manager will query the device knowledge base for information about the given device. If found, the device is assigned a unique ID which is valid for the given session, and thus until the driver or device manager is shut down. Once successfully registered, the device driver will start publishing a "heart beat" to a designated ROS topic at a specified rate. This is used by the device manager to keep track of the device driver's state and check if it unintentionally terminates. Should information about the device not be available in the knowledge base, the device is

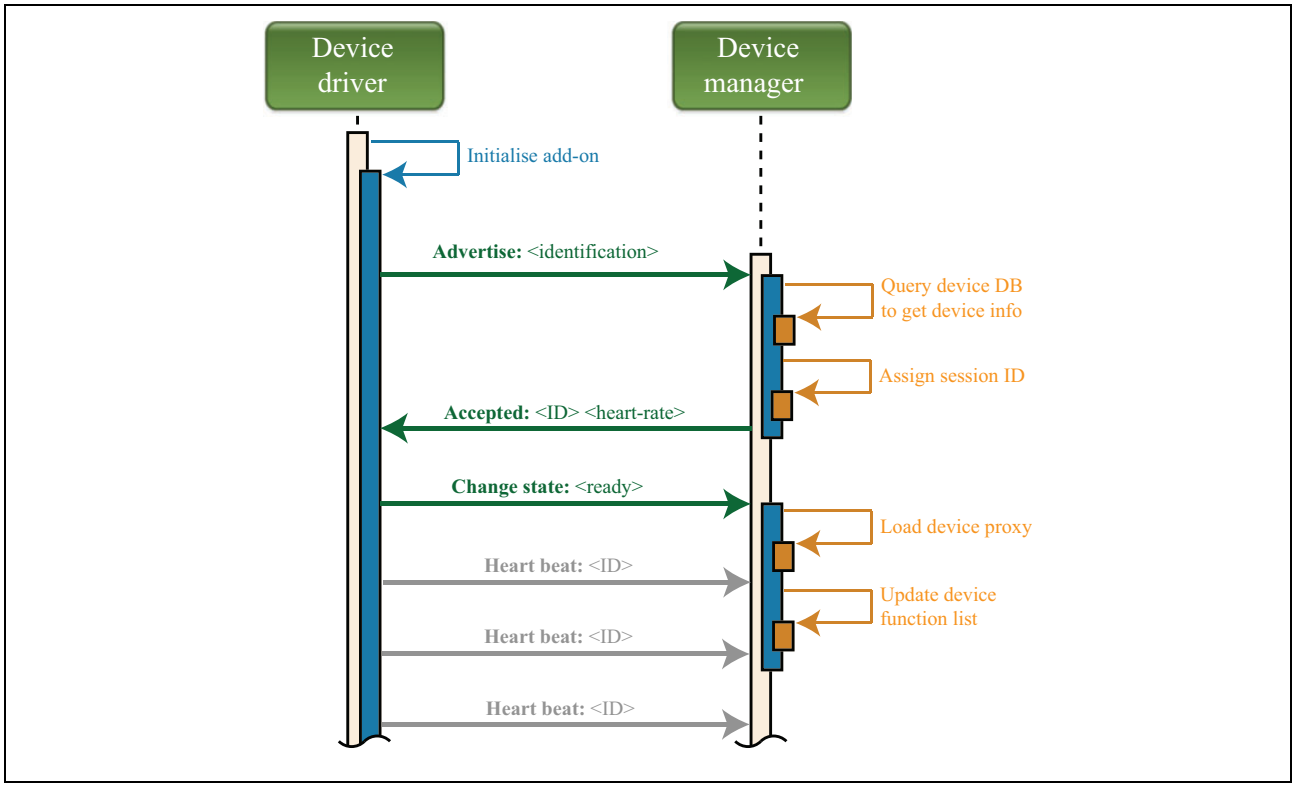

Figure 5. Sequence diagram illustrating a successful device registration process. The diagram shows the overall communication between the device driver and the device manager.

registered as unknown and cannot be utilized through the hardware management framework.

If the device is registered successfully, the device manager loads and initializes the device proxy. Upon loading the device proxy successfully, the device manager uses information from the device knowledge base to update the set of available primitives.

Function generalization

In a system with a static hardware configuration, the task control will often invoke functions directly on the specific hardware devices. However, in our multiagent system with exchangeable agents, the abstraction of primitives is needed. The main benefit of introducing a set of primitives is to keep the task control independent and separated from specific hardware. Hence, the task control implementation can be reused regardless of the hardware composition of the system. The primary downside to introducing primitives is that a primitive must represent all the specific device function implementations, and thus, the primitive often becomes the "simplest" version of a given functionality. For instance, all grippers have a device function resolvable to the primitive grasp. Some grippers provide a force-controlled grasp, but since not all grippers have this option, the primitive cannot require the ability of force control. Thus, the introduction of primitives will often lead to reduced utilization of the specialized functionality. The challenge is how to create a flexible interaction between the device-specific layer and the task control layer. An interaction that on one hand provides simple primitives which the task control system can invoke to achieve simple actions. On the other hand, the interaction must also allow the task control to request more specialized and complex functionality when needed. In our framework, we propose a method of amending the primitives with any number of parameters serving as constraints. For instance, requesting grasp, but amending the parameter of a specific force. Then only force-controlled grasps will apply. The parameters are implemented as key–value pairs. The key is a simple string defining the parameter name, which is used when finding matching primitives. The value consists of the actual value and a string defining the data type. By using simple key–value pairs, primitives can provide parameters with any given name. Likewise, task control can request primitives with parameters of any arbitrary names. Of course, a successful primitive request depends on the match between the parameter name of the primitive provided and the primitive requested. However, this rather relaxed approach to the implementation of parameters provides a flexible and scalable method for amending data to primitives. It allows the addition of a new primitive with any new parameter without the need to update a set of explicitly defined parameters.

Resolving primitives to specific functions on specific devices is handled by the device manager and device

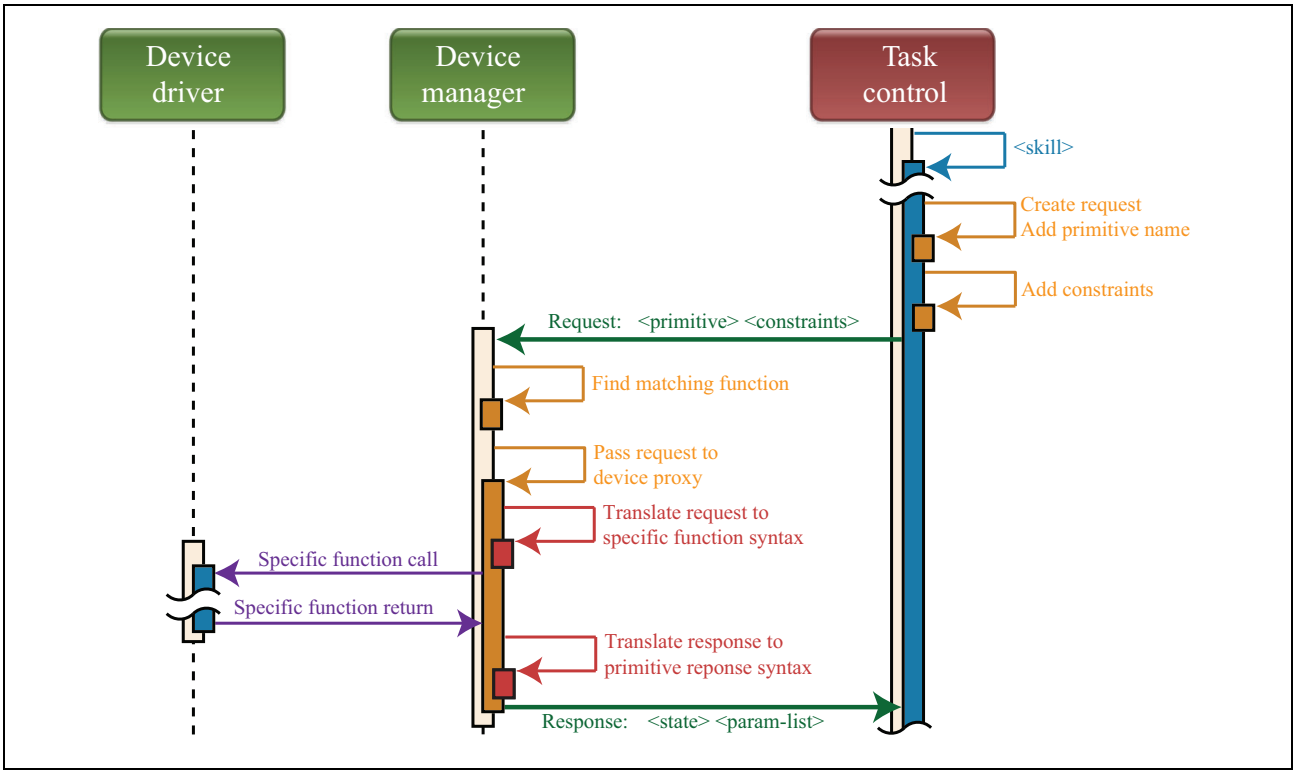

Figure 6. Sequence diagram illustrating the interaction between the task control and the device manager during a successful primitive request. Only the main aspects of the interaction are shown.

proxies. Thereby, the task control layer only interacts directly with the device manager (see Figure 1). In the implementation, a simple algorithm is used to match a primitive request to the available primitives in the device layer. The algorithm will consider the constraints given by the task control in order to find the first match fulfilling the constraints. A simple algorithm is chosen as the solution space, for example, the number of available modules is relatively small. Thus, the simple algorithm is used to demonstrate the purpose of the device manager; however, any suitable matchmaking algorithm could in principle be used. The interaction between the task control system and the device manager starts with the creation of a primitive request on the task control side. Such request contains the following information:

- device name
- device type
- primitive name
- parameter list
    - parameter name
    - parameter value

The device name and device type are optional, but they can be used to specialize the primitive request. Explicitly stating the device type of the primitive requested can be necessary in cases where multiple device types might offer the same primitive; for example, *move joint* would both be available on a pan/tilt unit and a robot arm. The device name entry enables the request of a specific device through the generalized function interface. The only mandatory content of the request is the primitive name. In addition, the given primitive might require some parameters to be specified. In such case, if the parameters are not specified, the request will return an error. For example, the primitive *move joint* clearly needs to know the desired joint values. The joint values are then defined as required parameters for the primitive. If not defined as required, the parameters are optional but can be used to amend "constraints." If a device function takes in a parameter, that is not required in the primitive, the parameter will be set to a default value during the mapping from primitive to device function. For instance, if the task control requests a simple *grasp*, but the device function available is a *grasp with force control*, then the force parameter will be defaulted by the proxy. Figure 6 shows the sequence in successfully resolving a primitive request.

Should no device function match the requested primitive, an error will be returned to the task control. Data returned by the specific device function undergoes a translation into a generalized format defined for the given primitive. That is, like the primitive requests are translated into a device-specific syntax by the device proxy, the return state and parameters are translated back into the generalized syntax of the primitive. Again, the primitives are

Table 1. Device modules used in the feasibility study.[a]

Brand/model	Type	Sub-type
Schunk WSG50	Gripper	Parallel
Robotiq 3-finger	Gripper	Dexterous
KUKA LWR	Robot arm	Articulated
Universal Robots UR5	Robot arm	Articulated

[a]The study is scoped to devices of type robot arm and gripper.

defined as only requiring the most essential return values, but allowing for additional parameters to be attached.

Feasibility study

In order to examine the presented hardware management framework, a feasibility study is conducted. This study demonstrates how the task control layer can be kept independent of the hardware devices and thus reused across multiple hardware configurations. Furthermore, the study demonstrates that hardware modules can indeed be exchanged online, hence, without resetting the device manager or the task control system. In the study, two different instances of *Robot Arm* and two different instances of *Gripper* are used (see Table 1 for brand and model).

Modules are realized by adapting each device to comply with the same mechanical interfaces. This makes the physical module exchange quick and intuitive. Between robot arm and gripper, a set of adapter plates are used. These consist of a robot arm part and a gripper part; hence, one plate is permanently fixed to each module and the standardized interface hereby becomes the mechanical join between the plates. To mount the robot arms, a robot cell with four slots for modular pallets on its top surface is used. Pallets to mount the UR5 and the KUKA LWR are already available. Thus, the standardized interface from the robot arm to the cell is the mechanical interface of the pallets. The hardware setup is shown in Figure 7.

The mechanical interfaces are designed to provide a mechanical alignment reducing the need for subsequent software calibration of tool center point. In this study, no software calibration is being made subsequent to the exchange of modules.

In terms of software, each device has a corresponding device driver readily available. Some are downloaded from the ROS community, and others have been developed locally. For each of them, a device proxy has been developed since all the drivers have different ROS interfaces, syntaxes, and command structures. Table 2 illustrates the implemented primitives used in this study.

The initial robot configuration consists of the UR5 robot arm and the Robotiq gripper. Using this setup, a simple pick and place task is instructed in the task control, where the center part of the Cranfield benchmark[31] is picked from one of the main plates and placed on the table surface. Afterward, a repeating execution of the task is started. For

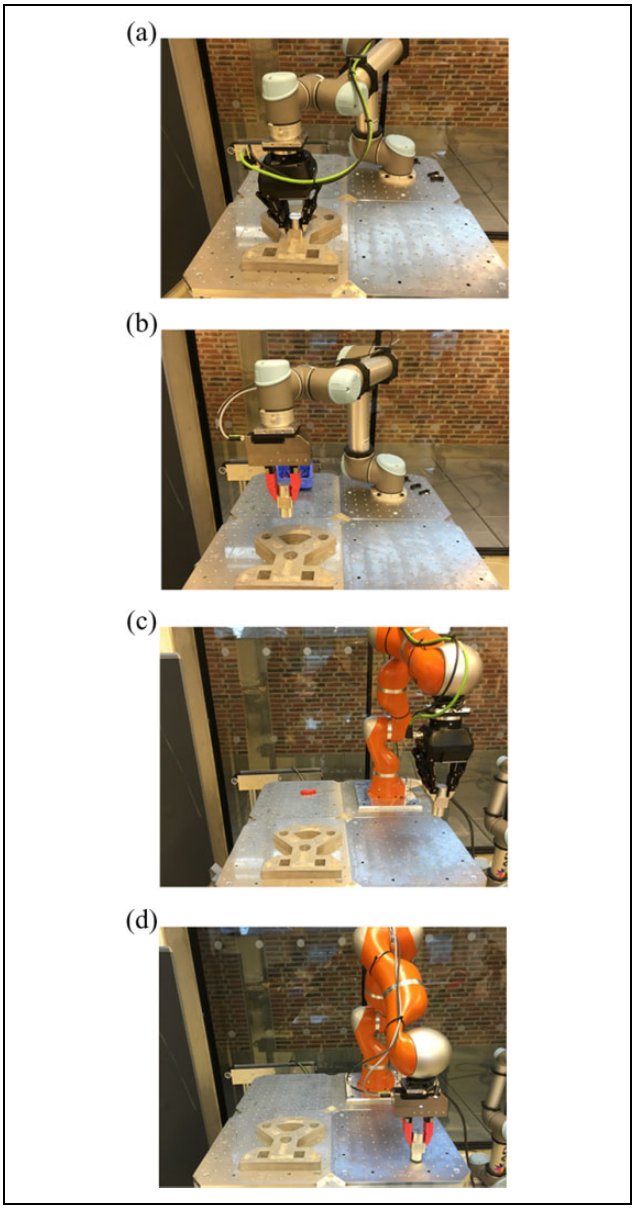

Figure 7. The four configurations validated in the feasibility study. The images were captured during execution of the pick and place task. Combined, the images show the sequence of the task. (a) Configuration 1: Universal Robots UR5 and Robotiq 3-finger. (b) Configuration 2: Universal Robots UR5 and Schunk WSG50. (c) Configuration 3: KUKA LWR and Robotiq 3-finger. (d) Configuration 4: KUKA LWR and Schunk WSG50.

Table 2. Primitives implemented and used in this study.

Type	Primitive
Gripper	Grasp
	Release
	MoveFingers
	GetTCP
Robot arm	MoveCartesian
	MoveJoint
	SetTool

Table 3. Order of the hardware configurations obtained and tested in the study.

Order	Robot arm	Gripper
1	Universal Robots UR5	Robotiq 3-finger
2	Universal Robots UR5	Schunk WSG50
3	KUKA LWR	Robotiq 3-finger
4	KUKA LWR	Schunk WSG50

each hardware reconfiguration, the execution is paused after a full cycle (a pick and place has been performed), hardware module(s) are exchanged, and the task execution is resumed. Hence, the task is not reinstructed or reset in between the modules exchanges. The hardware modules are exchanged in the order illustrated in Table 3.

Figure 7 shows the robot cell, the task used for the feasibility study, and the four hardware configurations during the execution. During the four hardware reconfigurations, the task control was not shut down or reset. The alterations to the configuration were updated through the device manager GUI, which then stopped or launched the affected drivers.

Conclusion

This article has presented a plug and produce hardware management framework for controlling the device layer of a modular collaborative industrial robot using ROS-enabled devices. A key purpose of the framework is to separate the task control level from the device control level and thereby allows for the task control system to become independent of specific hardware components and syntax. The framework is built upon an agent-based architecture and uses well-established communication and architectural schemes. To achieve separation between task control and device levels, a concept of *primitives* is introduced. Primitives generalize device functionalities of various devices of same type and thereby constitute a standardized function interface between the task control and the device levels. A device manager serves as an agent manager keeping track of the connected devices and resolving primitive requests to specific devices. A device proxy implemented as a dynamically linked library to the device manager performs a translation from the primitive syntax to the syntax of the specific device driver.

A key challenge in introducing primitives is to limit the loss of specialized features and behaviors. As part of the framework, an interaction scheme between the task control layer and the device control layer is proposed, which enables the task control layer to extend primitives with parameters. The parameters can either be optional, required by the device layer or required by the task layer. In the latter two cases, the parameters serve as constraints specializing the primitive request. Thus, the interaction scheme associated with the primitives allows the task control to use both basic and complex functions depending on the task requirements.

With the proposed architecture, an intermediate step in the communication structure is introduced in the form of the device manager. As a result, an increased delay will be present in the primitive execution. Although the delay is not noticeably in the conducted experiment, further analysis is necessary in order to determine the effect on processes with high continuity demands.

A feasibility study has demonstrated that the presented hardware management framework allows a plug and produce approach to exchange of hardware modules. This makes hardware exchange both expedite and less complicated, since the entire robot cell does not have to be reinstructed or reset. The framework also allows us to separate the task-level operation from the device-level control. This makes the task control layer independent of the specific hardware composition of the robot as demonstrated in the feasibility study. In the study, the framework was examined on a modular robot cell using two robot arms and two grippers as exchangeable modules. While executing the very same task, a series of hardware exchanges were successfully made without intervention in the task control. In summary, all four configurations obtained were able to successfully continue the task execution. In conclusion of the feasibility study, the proposed framework is considered beneficial in performing quick and online hardware module exchanges. Furthermore, it streamlines the implementations and developments of task control modules in the task control layer, and thus increasing the clarity of the behaviors intended from the task control.

With the proposed hardware management framework, a robot control architecture supporting online exchange of active modules can be realized using ROS drivers. To fully enable plug and produce of hardware modules, our future work will focus on standardizing the physical interfaces between modules including a method enabling quick physical exchange of modules. In addition, a procedure is needed to verify the correctness of the obtained configuration. We will in future work address the task of module selection because finding a valid set of modules for a given task is not trivial.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The financial support from Innovation Fund Denmark through the project CARMEN is gratefully acknowledged.

References

1. SPARC. Robotics 2020 multi-annual roadmap for robotics in Europe. 2015, https://eu-robotics.net/cms/upload/press/2015/files/H2020_Robotics_Multi-Annual_Roadmap_ICT-2016.pdf.
2. El Maraghy H A. Flexible and reconfigurable manufacturing systems paradigms. *Int J Flexible Manuf Syst* 2006; 17(4): 261–276.

3. Schou C, Damgaard JS, Bøgh S, et al. Human-robot interface for instructing industrial tasks using kinesthetic teaching. In: *2013 44th international symposium on robotics (ISR)*, 2013, pp. 1–6. IEEE. DOI: 10.1109/ISR.2013.6695599.
4. Madsen O, Bøgh S, Schou C, et al. Integration of mobile manipulators in an industrial production. *Industrial Robot* 2015; 42(1): 11–18. DOI: 10.1108/IR-09-2014-0390.
5. Schou C and Madsen O. Towards shop floor hardware reconfiguration for industrial collaborative robots. In: *Advances in cooperative robotics: proceedings of the 19th international conference on Clawar*, London, UK, 12–14 September 2016, pp.158–168. Singapore: World Scientific Publishing Company.
6. Quigley M, Conley K, Gerkey B, et al. ROS: An open-source robot operating system. In: *ICRA workshop on open source software*, 2009, Vol. 3.
7. Arai T, Aiyama Y, Maeda Y, et al. Agile assembly system by "plug and produce". *CIRP Ann Manuf Technol* 2000; 49(1): 1–4.
8. Hvilshøj M and Bøgh S. "Little Helper"—an autonomous industrial mobile manipulator concept. *Int J Adv Robotic Syst* 2011; 8(2): 1–11.
9. Park E, Kobayashi L, and Lee SY. Extensible hardware architecture for mobile robots. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*, Barcelona, Spain, 18–22 April 2005, pp. 3084–3089. IEEE. DOI: 10.1109/ROBOT.2005.1570584.
10. Hermann A, Xue Z, Rühl SW, et al. Hardware and software architecture of a bimanual mobile manipulator for industrial application. In: *Proceedings of the 2011 IEEE international conference on robotics and biomimetics*, Karon Beach, Phuket, Thailand, 7–11 December 2011, pp.2282–2288. IEEE. DOI: 10.1109/ROBIO.2011.6181638.
11. Datta S, Ray R, and Banerji D. Development of autonomous mobile robot with manipulator for manufacturing environment. *Int J Adv Manufact Technol* 2008; 38(5–6): 536–542. DOI: 10.1109/ROBIO.2011.6181638.
12. Merten M and Gross HM. Highly adaptable hardware architecture for scientific and industrial mobile robots. In: *Proceedings of the 2008 IEEE international conference on robotics, automation and mechatronics*, Chengdu, China, 21–24 September 2008, pp.1130–1135. IEEE. DOI: 10.1109/RAMECH.2008.4681459.
13. Helms E, Schraft RD, and Häegle M. Rob@work: robot assistant in industrial environments. In: *Proceedings of ROMAN-2002*, Berlin, Germany, 27–27 September 2002, pp.399–404. IEEE. DOI: 10.1109/ROMAN.2002.1045655
14. Katz D, Horrell E, Yang Y, et al. The UMass mobile manipulator UMan: an experimental platform for autonomous mobile manipulation. Paper presented at *Workshop on Manipulation in Human Environments at Robotics: Science and Systems*, 2006.
15. Xuemei L and Liangzhong J. Study on control system architecture of modular robot. In: *Proceedings of the 2007 IEEE international conference on robotics and biomimetics*, Sanya, China, 15–18 December 2007, pp. 508–512. IEEE DOI: 10.1109/ROBIO.2007.4522214.
16. Hentout A, Bouzouia B, Akli I, et al. Mobile manipulation: a case study. In: Lazinica A and Kawai H (eds) *Robot manipulators new achievements*. Croatia: InTech, 2010.
17. Hammer B, Koterba S, Shi J, et al. An autonomous mobile manipulator for assembly tasks. *Auton Robots* 2010; 28(1): 131–149.
18. Monostori L, Váncza J, and Kumara SRT. Agent-based systems for manufacturing. *CIRP Ann Manuf Technol* 2006; 55(2): 697–720.
19. Mauro Onori and José Barata. *Evolvable production systems: mechatronic production equipment with process-based distributed control*. The Netherlands: Elsevier Ltd, 2009.
20. EUPASS. Evolvable ultra-precision assembly systems. EU project funded under the European Community's Sixth Framework Programme, 2004.
21. Ferreira P, Lohse N, and Ratchev S. *Multi-agent architecture for reconfiguration of precision modular assembly systems*. In: Proceedings on precision assembly technologies and systems: 5th IFIP WG 5.5 international precision assembly seminar, IPAS 2010, Chamonix, France, 14–17 February 2010, pp.247–254. Berlin, Heidelberg: Springer.
22. IDEAS. Instantly deployable evolvable assembly systems. EU project funded under the European Community's Seventh Framework Programme, 2010.
23. Onori M, Lohse N, Barata J, et al. The ideas project: plug and produce at shop floor level. *Assemb Autom* 2012; 32(2): 124–134.
24. PRIME. Plug and PRoduce intelligent multi agent environment based on standard technology. EU project funded under the European Community's Seventh Framework Programme, 2012.
25. Rocha A, di Orio G, Barata J, et al. An agent based framework to support plug and produce. In: *2014 12th IEEE International Conference on Industrial informatics (INDIN),* Porto Alegre, Brazil, 27–30 July 2014, pp. 504–510. IEEE. DOI: 10.1109/INDIN.2014.6945565.
26. Andersen RH, Dalgaard L, Beck AB, et al. An architecture for efficient reuse in flexible production scenarios. In: *2015 IEEE International conference on automation science and engineering (CASE)*, 2015, pp.151–157.
27. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. http://www.w3.org/TR/owl2-overview/ (2009, accessed 2 May 2016).
28. Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. 2nd ed. San Francisco: Morgan Kaufmann Publishers Inc., 2011.
29. Pedersen MR, Nalpantidis L, Andersen RS, et al. Robot skills for manufacturing: From concept to industrial deployment. *Robot Comput Integr Manuf* 2016; 37:282–291. DOI: 10.1016/j.rcim.2015.04.002.
30. Hvilshøj M. *Autonomous Industrial Mobile Manipulation (AIMM) - maturation, exploitation and implementation: Developing modular and (re)configurable AIMM families based on architectures*. PhD thesis, Department of Mechanical and Manufacturing Engineering, Aalborg University, 2012.
31. Collins K, Palmer AJ, and Rathmill K. Robot technology and applications: In: *Proceedings of the 1st robotics Europe conference Brussels*, Berlin, Heidelberg, 27–28 June 1984, pp. 187–199. Springer Berlin Heidelberg, 1985.