DOI: 10.3217/978-3-85125-663-5-10

Flexible industrial mobile manipulation: a software perspective

Thomas Haspl[1], Benjamin Breiling[1], Bernhard Dieber[1], Marc Pichler[1], Guido Breitenhuber[1]

Abstract— **With ongoing research in robotics, some specific architectural approaches of robotic systems earn more and more interest by all kinds of industries. Mobile manipulators–robots consisting of a mobile base and a serial manipulator–provide the ability to make robotic manipulation location-independent, which will be an essential feature in future production. Such robot platforms offer a high level of flexibility and efficiency of robot applications. Especially under the aspect of modularity, mobile manipulators would provide even more flexibility by offering the possibility to exchange or extend the robot hardware for specific applications. To achieve this, modularity also has to be considered in software. In this paper, we present a software architecture for modular mobile manipulation applications. It provides mechanism for reconfigurability, easy programming, and an easy approach for adding external hardware components. Being targeted at industrial use, the architecture also considers security and software deployment aspects. These considerations will, in combination with all the other aspects, be presented by means of two modular mobile manipulation platforms and a set of representative scenarios.**

I. INTRODUCTION

Nowadays, the number of robots used in different industrial and non-industrial is steadily rising. These application areas range from manufacturing, transportation, household, up to health care, amongst others. Independent from the application area, the requirements to robotic platforms also heavily increased. Due to the demand of flexible applicability and the desire to let robots operate in unstructured environments, a significant amount of research work has been dedicated to this. One very promising type of robots for many kinds of applications are mobile manipulators. Those combinations of robot arms and mobile bases free the serial manipulator (the arm) from its location-dependence thus enabling a whole new range of applications. In industry, mobile manipulation is currently taking up speed where sensitive robots are employed working next to humans on the shop floor. Key aspects in the successful application of mobile manipulation is the flexibility in the use of the robot. This requires easy programming for end-users and system integrators, extensibility and reconfigurability of the robot hardware as well as the easy integration of the robot into the manufacturing IT infrastructure. Additionally, a higher level of modularity of robotic hard- and software inevitably requires strategies for secure communication between the robot's components in order to keep the system safe from intrusion.

While there have been many reports of mobile manipulators in research, this paper presents a software architecture that has already been proven in industrial settings by enabling a highly flexible use of robots in various scenarios. The architecture explicitly considers mobile manipulators that are modular in their nature. Thus, one key requirement for the software architecture is that it must be easily possible to integrate software components for new pieces of hardware like sensors or grippers but also to exchange core hardware elements like the robotic arm or mobile base. So, while we here mainly report on a configuration consisting of a *MiR100* mobile base and a *UR10* arm, the architecture also supports the integration of different bases and arms of other types and vendors. while the architecture focuses on the industrial use of mobile manipulators, it can also be used in other settings in- and outside of industry.

This paper is structured as follows: section II depicts the state of the art, which is relevant for our software architecture. This includes reconfigurability of hard- and software components, security in robotic applications and the issue of deploying robot software. Section III then elaborates our developed architecture. By providing necessary requirements at the beginning, this section then describes the core components and processing layers of the architecture. The topics security and deployment will finalize this section. Then, in section IV an evaluation of the architecture by means of three scenarios will be done. Before that, we present two in-house developed mobile manipulation platforms - the CHIMERA which can operate in industrial environments and a modular research platform for mobile manipulation. Both are driven by the software stack presented in this paper.

II. STATE OF THE ART

Robot software is becoming increasingly important as the focus of the development shifts from hardware capabilities to more intelligent robots. In terms of software, the *Robot Operating System (ROS)* [19] has become the predominant framework for prototyping robot applications and building intelligent robotic products. Over the years, several architectural approaches have been presented for domains with references to robotics [18], [12], [1], [14].

A. *Reconfigurability of Hard- and Software*

Modularity in robots has been proclaimed as one of the most promising approaches to making robots more flexible while at the same time decreasing integration times [15], [23], [20]. Modularity in hardware enables the reconfiguration or extension on a physical level by adding, removing or swapping hardware parts. In software, modularization aids re-usability and thus is aimed at minimizing time required to develop a solution.

[1]The authors are with the Institute for ROBOTICS and Mechatronics, JOANNEUM RESEARCH, Klagenfurt, Austria `firstname.lastname@joanneum.at`

B. Security

Also security has been addressed in our previous work. Especially in the context of *ROS*, we have focused on securing robots i) in the applications [11], ii) in the *ROS*-core [3] or iii) in their integration in larger IT infrastructures [9]. In addition to our work, contributions to ii)[22] and iii)[16] can be found as well. In general, a multi-layer approach to security is recommended in- and outside of the robotics domain (i.e., the combination of all three aforementioned security approaches) [5], [10]. Security flaws in robot systems are especially grave since recent studies have shown a very large number of robots to be publicly accessible via the internet [8], [21].

C. Deployment

Software deployment is an activity performed for or by the customer where all the customer-centric configuration and customization is done [7]. It is a process consisting of activities related to the release, installation, activation, deactivation, update, and removal of components [6]. Different generic standards for software deployment like RPM (RPM Package Manager) or DPKG (Debian Package) on Linux or OMG's (Object Management Group) deployment and configuration specification for component based distributed applications [17] exist. In the field of robotics software deployment is a challenging problem due to the complexity and variability of robots [4], [13]. Robot manufacturers tackle the problem with custom proprietary solutions. *Universal Robots* uses *URCaps* for installing additional components or system updates. *Franka Emika* has a concept of APPs, which add new functionality to the controller. In *ROS* there is `roslaunch`, which at least covers the configuration and activation step of the deployment process. Yet we found no related research for an end-to-end approach on how to deploy a robotic software from a software developer via optional system integrators and/or customers to a robot in a secure way where each of the stakeholders should be able to add or modify parts of a deployment package.
In contrast to the works reported above, the architecture presented in this paper considers all of the aspects of security, modularity and deployment within an end-to-end concept for industrial mobile manipulators.

III. ARCHITECTURE

In this section, we first present the requirements for our architecture before going into detail on its components and their interactions.

A. Requirements

Considering the hardware-related aspects and requirements as mentioned in the introduction, we can derive a number of requirements regarding the software structure of mobile manipulators from them.

1) Hardware reconfiguration: As hardware reconfiguration is key to enabling a flexible use of a robot, it is also necessary for the underlying software to support this aspect. This requires a hardware-driver model and defined communication channels to other components. This basically means that the software has to be designed in a way that its components for controlling a mobile platform and a serial manipulator are capable of communicating over defined interfaces, but can basically operate self-contained. Furthermore, these components should meet the requirement that they are independent from the applied hardware. Providing these attributes a software structure for mobile manipulation can reach a high level of flexibility and reusability.

2) Security: It is undoubted that human safety is an essential aspect of all robotic platforms and applications. When it comes to mobile manipulation this aspect earns even more interest as the typical operation area of mobile manipulators can also include interactions with humans in any kind of way. A key requirement we actually want to address with this argument is security. Security in a robotic context relates to a secure (intrusion-proof) communication structure between hardware and software components, whereas the term safety is used for human safety. Even the most excellent safety concept for a robotic application becomes worthless if security issues are not considered. Accordingly, security is strongly connected to safety and as we already stated in our previous work [2] it is even practically not possible to guarantee safety without considering security. In this sense, security is a necessary but not sufficient precondition for safety. It must be assured, that (typically software-based) safety measures are not undermined by a lack of security.

3) Extensibility: Another requirement we want to point out is extensibility. A system integrator as one important end-user of our software does not want to struggle with issues because of a complicated way of programming when it comes to extending the software setup. The whole concept should be based on structural decisions which allow a modular behavior of the application software. With this, it should not only be easy to add custom modules such as sensors or grippers in hardware, but also in software.

4) Deployment: In addition to easy programming, a simple and scalable procedure for deployment and integration is an important point. When deploying software to a robot it must be made sure that only genuine software components are deployed and used on the robot. We dedicated a particular section to this important topic in III-E.

B. Components

The core of our software consists of four components which provide the basic functionality for a mobile manipulator consisting of a mobile platform, a serial manipulator and a gripper. Additionally, the architecture provides interfaces for extending the basic application. An exemplary composition of an application using our software is visualized in figure 1. Basically, the architecture defines three layers, one for components working close to hardware, a task layer, which abstracts specific hardware functionality and exposes atomic

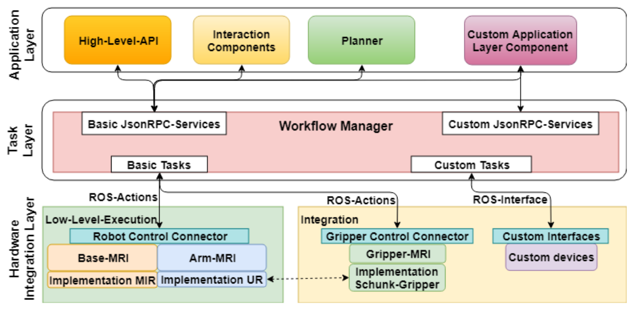

Fig. 1. Architecture of the software running on the core NUC

functions to the application layer where complex applications and the integration to outside infrastructure are realized.

C. Architecture layers

In the application layer, the implementation of a specific use case takes place. Various options are available for this, ranging from code-based implementation to teaching by demonstration. The result is always the same, namely a workflow with already parameterized tasks. A high-level *API* is maybe one of the flexible ways to create applications. Here, a system integrator can easily implement new processes for the robot platform. Basically, client libraries for any language can be implemented as long as they support *JSON-RPC* communication. Another option for creating executable workflows is a planner. Planners are basically used for more complex actions, where for instance tasks have to be dynamically reordered.
The task layer is responsible for processing a workflow task by task and to manage the status feedback to the application layer. From a workflow definition, which can be an ordered sequence or complexer structures with branches, one task after another is triggered by the *Workflow Manager*. This happens by calling the appropriate *ROS*-action server from the underlying robot control connector (or custom interfaces). A workflow also contains strategies if one or more tasks fail. As this information is required by the workflow templates, it can be assured that a workflow always terminates in a defined state, even in case of hardware execution errors.
On the hardware integration layer, the actuation of hardware components, thus, the actual execution of tasks takes place. The responsible components, namely the *Robot Control Connector* and the *Gripper Control Connector*, serve as an interface between the single tasks maintained by the *Workflow Manager* from the task layer and the robot commands used by the *Multi Robot Interface*. In particular, the control connectors wait for an *ROS*-action goal, extract the task parameters and utilize the *Multi Robot Interface* to generate and execute base- and arm commands (*Robot Control Connector*) as well as gripper commands (*Gripper Control Connector*) respectively. For integrating custom hardware, a system integrator can define and implement his own task definitions and execution components. After completion of a command, successful or not, the result is sent back to the *Workflow Manager*. In the current version of our software every task is executed in a blocking manner. Hence, parallel task execution is not yet supported.

1) Workflow Manager: The *Workflow Manager* (WFM) is used to trigger individual atomic actions (e.g. navigating the mobile base to a specific pose, move the TCP of the serial manipulator to a specific pose, move the serial manipulator to a specific joint configuration), but also supports composition of such atomic actions to complex workflows. The mobile manipulator then executes these sequences autonomously. The dynamic JSON-RPC interface of the *Workflow Manager* is extended if new hardware is integrated in the integration layer. This means that higher layer actions of this hardware are also available. On task level, the *Workflow Manager* runs ROS-action clients for sending basic operations to the *Robot Control Connector* and *Gripper Control Connector* respectively. If additional hardware needs to be integrated, the WFM can be dynamically extended by new interfaces to the hardware integration layer. For that, the source code of the interfacing components has to be stored in a predefined directory. Then, the *Workflow Manager* loads the interfacing components automatically at the next startup, while its source code remains untouched.

2) Control connectors: These components, namely the *Robot Control Connector* and the *Gripper Control Connector*, serve as an interface between the single tasks maintained by the *Workflow Manager* and the robot commands used by the *Multi Robot Interface* (see below). In particular, the control connectors maintain ROS-action servers, receiving task actions, in order to execute commands on the robot's hardware via the *Multi Robot Interface*. Like the *Workflow Manager* the source code of the control connectors is independent from the current hardware configuration. However, they must be initialized with hardware specific implementations of the related *Multi Robot Interface* components at runtime.

3) Multi Robot Interface: The *Multi Robot Interface* (MRI) is a software interface to access common functionalities of different robot classes. It provides generic command factories for serial manipulators, mobile bases as well as grippers and so abstracts the underlying hardware. This especially means that task- and application layer components are independent from the current hardware configuration and do not need to be modified when the hardware configuration and their software drivers change. As depicted in figure 1, the MRI is split into three parts; i) Base-MRI providing basic commands for controlling a mobile base, ii) Arm-MRI providing basic commands for controlling a serial manipulator, iii) Gripper-MRI providing basic commands for controlling a gripper. These interfaces can then be utilized by the *Robot Control Connector* and *Gripper Control Connector* to trigger commands. As a consequence, the source code of both control connectors is independent from the hardware configuration. As already described in III-C.2 the low level execution process is initialized with hardware specific implementations of the MRI components at runtime.

D. Security

As already indicated in the introduction, there is a gap between security requirements for the internal communication and the need of interaction with external entities. Consequently, our proposed security architecture has to work on multiple levels, in particular on network, operating system and application level. The overall network architecture of our mobile manipulator, shown in figure 3, is separated into i) an internal network for data processing and low level task execution, ii) an interaction network for the communication to external components. Particularly, the second provides secure access to an administrative interface for maintenance and configuration as well as secured interfaces to a superordinate task planning component and external interaction devices. The most important point in this setup is that there is no direct network connection between the interaction network on the right hand side and the robotic hardware on the left hand side of the figure. As a consequence, an attacker will not be able to directly aim at potentially vulnerable devices (with regards to cybersecurity), like the base or the serial manipulator. Further, we pass the information between the two networks via inter-process communication techniques, in order to prevent unauthorized access to the passed information according to the security requirements we already proposed in [9].

E. Secure Deployment

In order to deploy new software and updates to the mobile manipulation platform, the architecture has to allow for both, secure deployment as well as secure execution of the deployed code. In our work, secure deployment is achieved by introducing a signed package format, which allows configuration data to be modified by a system integrator, while still ensuring integrity of the software contained in the package. Secure execution on the other hand is accomplished by extensive sandboxing of applications, as well as by restricting access to system resources such as network or file access.

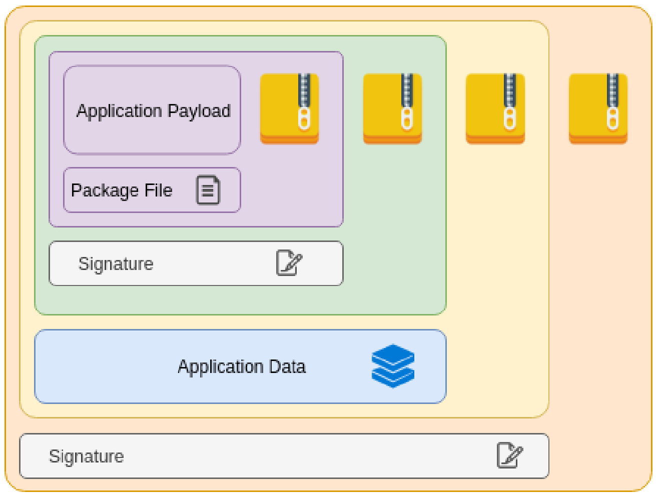

Fig. 2. Application Package

1) Package format: As seen in Figure 2, an application package consists of multiple parts. This allows for flexibility in modifications to parts of the package.

The application payload is the actual application or plug-in that is deployed to a part of the system. For actual stand-alone software, this is a *Docker* container, for plug-ins this might be Python modules, dynamically linked libraries, or any other payload authored by a developer. The application payload is digitally signed by the developer, and this signature is used on deployment to verify the package's integrity and origin.

The package file is a text file that contains metadata about the package itself such as name, version information and intended target container.

The application data part contains configuration files and other modifiable resources. Splitting application data and application payload in two allows for some flexibility in configuration of packages. For instance if a system integrator wants to reconfigure a software package for a specific use-case. This can be accomplished by unpacking the outer package, changing the application data, and finally re-signing the outer package with their own private-key. Since the application payload is left untouched, it is not re-signed and can be assumed to still work as intended by its developers.

2) Package broker: The package broker is an intermediary component between the interaction network and the internal network (as seen in Figure 3). Due to the necessity of deploying packages to any component of the system as well as due to security requirements, it exists as a component that receives packages from the interaction network and distributes them to either the interaction network or the internal network. The package broker is responsible for package-verification and deployment to the intended target package container within the system.

3) Package containers: A package container registers with the broker to receive packages since the broker acts as the central instance for package distribution inside the system. Once a package arrives at the broker, the broker inspects the signature and extracts the intended target package container from the package file. It then verifies that the creator of the package and source files has the permissions to deploy to the package container in question. After performing these tasks, the broker hands the package off to the package container for further processing.

Once the package arrives at the package container, the container starts unpacking the package, does some additional verification, and finally runs the package container-specific update procedure. When done, it reports the package as installed to the broker, providing feedback to the user, and giving the broker the authority to start, stop or uninstall the application in question.

4) Additional considerations: In order to make the packaging procedure easier, a separate application for packaging and signing is provided. In addition to this, a user interface connected to the package broker allows for starting and stopping running applications on package containers, as well as for installing and uninstalling packages, hiding the inner workings of the deployment and maintenance procedures.

IV. Evaluation

In this section we evaluate our software architecture against the requirements stated in III-A. For that, we first introduce two mobile manipulators able of running our software. Then, we describe different scenarios, how our software architecture supports i) the exchange of the hardware modules, ii) the integration of custom hardware and iii) the deployment of new software packages.

A. Platforms

As target platforms for our software architecture we aim at two mobile manipulation platforms with certain characteristics. One manipulator is used in industrial settings and has been proven in real factories already. The second model is a research platform composed of exchangeable modules. The cross-cutting assumption for this in the software architecture is to have a robot with a mobile base and a serial manipulator attached.

1) CHIMERA: The CHIMERA consists of a *MiR100* including its internal hardware parts, the network with router, PC (Intel® NUC) and PLC, as well as an UR controller for an *UR10* robot arm and an additional PC (Intel® NUC) for integrating additional hardware and software. Figure 3 shows the overall network architecture of the CHIMERA, which is shown in figure 4.

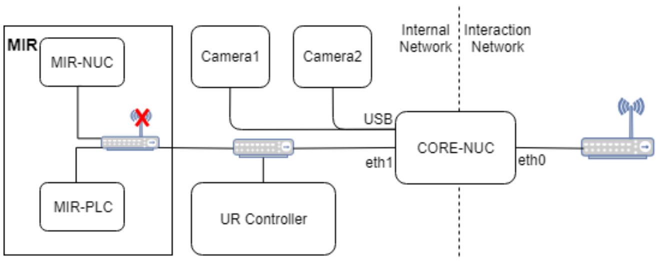

Fig. 3. Network architecture

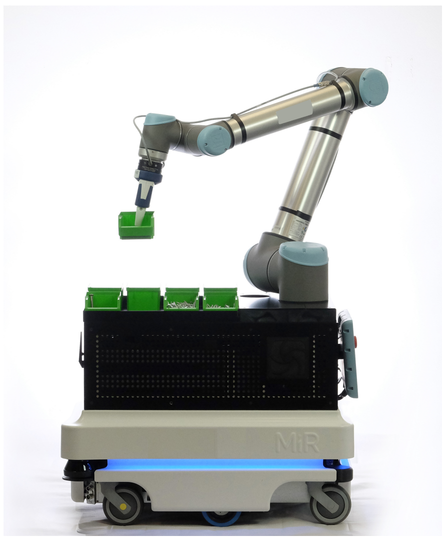

Fig. 4. CHIMERA

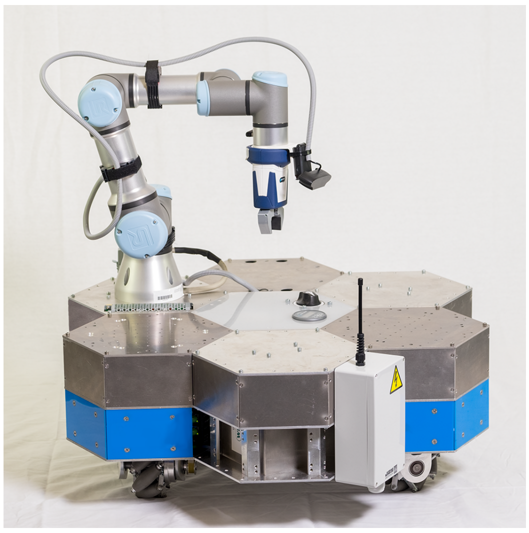

Fig. 5. Research Platform for modular mobile Manipulation

From the software point of view, our goal is to combine the functionalities of the *MiR100* and the *UR10* in order to execute customizable mobile manipulation workflows. At the same time we keep our software as generic as possible, which means that we can exchange the base, the arm or both without changing the major part of our software.

2) Research platform for modular mobile manipulation: The specific characteristic of our wheeled mobile manipulation research platform is its modular mobile base. It can be assembled with an arbitrary number of hexagonally shaped modules in order to optimize it for specific applications. Thereby it is possible to create the most suitable robot for a given problem. The hexagonal modules can be equipped with different wheel types, other modules can be used for power supply, the robot arm or other hardware components. A computing module contains an *Intel® NUC* PC, which is responsible for running all software parts. At startup, a configuration file, that contains information about the physical configuration of the mobile base, is parsed. The gripping module includes an *UR3* as well as its controller. A composition of such a wheeled mobile manipulator can be seen in figure 5. The mobile base in this figure consists of three driving modules with omni-directional wheels, two battery modules, a gripping module with an *UR3* and a computing module.

B. Scenarios

In this section, we iterate through various situations in the lifecycle of our two platforms and explain how the software architecture will support this.

1) Integration of custom hardware: In the following scenario, we describe how a system integrator (SI) can integrate a new sensor and utilize it on the task layer in order to include it into a custom workflow. First, the SI needs to deploy a driver software for the sensor using the secure deployment mechanism. The driver can then be started in an own process. In order to use the sensor within a customized workflow, the *Workflow Manager* then needs to be extended with a sensor specific task definition and an interface to the newly integrated sensor. To make the sensor specific workflow available for an application layer component the system integrator also has to provided a related JSON-RPC service.

2) Deployment of a new software package: In this example, we outline how the SI deploys the necessary sensor driver required for the previous scenario using the secure deployment mechanism. In a first step, the sensor driver has to be implemented using a programming language of the SI's choice. Furthermore, an application setup-script to install necessary dependencies has to be created. After implementation, the driver is packaged using the packaging application which generates a *Docker* container based on the setup-script, the docker container is then packaged with the package file and is subsequently signed by the SI. The SI then logs in and pushes the package to the *package broker*, which will then perform the deployment as described in Section III-E.

3) Exchange of basic hardware: Let's suppose, we successfully executed an use-case on the research platform and want to run it on the CHIMERA. To do so, we need a *MiR100* implementation of the *Base-MRI* interface, replacing the research platform implementation. Further we have to change the configuration of the UR implementation in order to use the *UR10* instead of an *UR3*. The *Robot Control Connector* is then initialized with the *MiR100* implementation at startup. The software architecture shown in figure 1 is unmodified, including the workflows which need to be executed. Only the driver implementations for hardware components must be exchanged. Of course transferring applications to different hardware does not guarantee that it can be executed successfully out-of-the box due to physical differences like reachability of the robotic arm or maneuverability of the mobile base.

V. CONCLUSIONS

In this paper we have described a flexible software architecture for modular mobile manipulators that is suitable (and has been tested) in industrial scenarios. We have shown that it can be used on both, platforms for industrial and research use. Thus, the architecture supports the required flexibility very well.

In future work, we plan to do more experiments with different kinds of hardware in order to strengthen the robustness of the architecture. We also plan to advance tool support for developers and system integrators to simplify development and deployment of software components.

ACKNOWLEDGMENT

This work has been supported by the Austrian Research Promotion Agency in the program "ICT of the Future" funded project FlexIFF (grant no. 861264)

REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," The International Journal of Robotics Research, vol. 17, no. 4, pp. 315–337, 1998.

[2] B. Breiling, B. Dieber, B. Reiterer, A. Schlotzhauer, and S. Taurer, "Safety nicht ohne security in der kollaborativen robotik."

[3] B. Breiling, B. Dieber, and P. Schartner, "Secure communication for the robot operating system," in 11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings, 2017.

[4] D. Brugali, A. Agah, B. MacDonald, I. A. Nesnas, and W. D. Smart, "Trends in robot software domain engineering," in Software Engineering for Experimental Robotics. Springer, 2007, pp. 3–8.

[5] E. Byres, P. E. Dr, and D. Hoffman, "The myths and facts behind cyber security risks for industrial control systems," in In Proc. of VDE Kongress, 2004.

[6] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. Van Der Hoek, and A. L. Wolf, "A characterization framework for software deployment technologies," COLORADO STATE UNIV FORT COLLINS DEPT OF COMPUTER SCIENCE, Tech. Rep., 1998.

[7] A. Dearle, "Software deployment, past, present and future," in Future of Software Engineering (FOSE'07). IEEE, 2007, pp. 269–284.

[8] N. DeMarinis, S. Tellex, V. Kemerlis, G. Konidaris, and R. Fonseca, "Scanning the Internet for ROS: A View of Security in Robotics Research," arXiv preprint arXiv:1808.03322, 2018.

[9] B. Dieber and B. Breiling, "Security considerations in modular mobile manipulation," in Proceedings of the 3rd International Conference on Robotic Computing. Naples, Italy: IEEE, Feb. 2019, pp. 70–77.

[10] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the Robot Operating System," Robotics and Autonomous Systems, vol. 98, pp. 192–203, 2017.

[11] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ROS-based applications," in IEEE International Conference on Intelligent Robots and Systems, vol. 2016-Novem, 2016, pp. 4477–4482.

[12] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic, "A domain-specific software architecture for adaptive intelligent systems," IEEE Transactions on software engineering, vol. 21, no. 4, pp. 288–301, 1995.

[13] N. Hochgeschwender, L. Gherardi, A. Shakhirmardanov, G. K. Kraetzschmar, D. Brugali, and H. Bruyninckx, "A model-based approach to software deployment in robotics," in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2013, pp. 3907–3914.

[14] A. Koubaa, "Service-oriented software architecture for cloud robotics," arXiv preprint arXiv:1901.08173, 2019.

[15] V. Mayoral, A. Hernndez, R. Kojcev, I. Muguruza, I. Zamalloa, A. Bilbao, and L. Usategi, "The shift in the robotics paradigm the hardware robot operating system (h-ros); an infrastructure to create interoperable robot components," in 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), July 2017, pp. 229–236.

[16] M. Mukhandi, D. Portugal, S. Pereira, and M. Couceiro, "A novel solution for securing robot communications based on the MQTT protocol and ROS," in Proceedings of the 2019 IEEE/SICE International Symposium on System Integrations (SII 2019), 2019.

[17] OMG, Deployment and Configuration of Component-based Distributed Applications Specification Version 4.0, 01 2006.

[18] M. Quigley, E. Berger, A. Y. Ng, et al., "Stair: Hardware and software architecture," in AAAI 2007 Robotics Workshop, Vancouver, BC, 2007, pp. 31–37.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in ICRA workshop on open source software, vol. 3, no. 3.2, 2009, p. 5.

[20] K. Roufas and M. Yim, "Software architecture for modular self-reconfigurable robots," in IROS, 2001.

[21] V. M. Vilches, G. O. Mendia, X. P. Baskaran, A. H. Cordero, L. U. S. Juan, E. Gil-Uriarte, O. O. S. de Urabain, and L. A. Kirschgens, "aztarna, a footprinting tool for robots," arXiv preprint arXiv:1812.09490, 2018.

[22] R. White, H. Christensen, and M. Quigley, "SROS: Securing ROS over the wire, in the graph, and through the kernel," in Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)., 2016.

[23] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in ICRA, 2000, pp. 514–520.