XVII International Scientific Conference on Industrial Systems (IS'17)
Novi Sad, Serbia, October 4. – 6. 2017.
University of Novi Sad, Faculty of Technical Sciences,
Department for Industrial Engineering and Management
Available online at http://www.iim.ftn.uns.ac.rs/is17

Towards a Plug-and-Play Architecture in Industry 4.0

Milan Pisarić
(Software Engineer, KEBA d.o.o., Nikole Pašića 11 Novi Sad, Serbia, pisa@keba.com)
Vladimir Dimitrieski
(Teaching Assistant, Faculty of Technical Sciences, Novi Sad, Serbia, dimitrieski@uns.ac.rs)
Milan Babić
(Student, Faculty of Technical Sciences, Novi Sad, Serbia, babic@uns.ac.rs)
Stefan Veselinović
(Student, Faculty of Technical Sciences, Novi Sad, Serbia, stefanveselinovic@uns.ac.rs)
Filip Dušić
(Student, Faculty of Technical Sciences, Novi Sad, Serbia, filip.dusic@uns.ac.rs)

Abstract

Having a robust Plug-and-Play architecture in the industrial context is one of the main goals of Industry 4.0. In this paper, we focus on describing one approach in reaching this goal and present an initial prototype of the solution. The approach is centered on abstract device descriptions and automatic generation of device drivers. We base our approach around open-source technologies and commonly used communication protocols. The aim of this approach is to enable better interoperability, and therefore to improve flexibility of production processes.

Key words: *industry 4.0, industrial internet of things, plug-and-play, industrial network, integration*

1. INTRODUCTION

A factory or even a single production unit comprises many different devices and machines. Most of them are rigid and inflexible since they are integrated into a system that is not supposed to change too often. In cases when changes are necessary, they usually affect many independent parts of the system. These changes are often handled by many different system integrators who are usually remote engineers. The need for participation of more engineers, especially remote ones, increases logistics and personnel costs. Some of the main goals of Industry 4.0 [6] are to reduce complexity and production costs while increasing the degree of flexibility in production processes. Ideally, such a reduction would comprise highly scalable production units. Such modernized production units would allow easy modifications and repairs while allowing for the production to run continuously, without any or just with short interruptions. Some changes to the production unit could be made online, without the need of stopping the production or involving too many employees. This would reduce a cost of running a production unit and increase an output of the system as its downtime is significantly reduced. One of the main elements of industrial production that hinders its flexibility is the communication between devices. The communication between devices of various types and purposes can be very inflexible even if they are a part of the same production unit. It often requires a specialized and in-depth knowledge of many protocols and interfaces by an engineer, just for all the devices to be connected. This may be solved with an additional connectivity layer that would be implemented by following the main Plug-and-Play concepts. In such a connectivity layer, all devices would be considered as black boxes whose description allows for the appropriate driver to be found and device to be integrated. These Plug-and-Play concepts were applied in various systems for a past few decades. We feel that these concepts are expressive enough to be applied in the industrial context as well.

The goal of our research is to deliver an approach and a software solution which would enable easier connectivity of industrial components and machines. Our approach implies following generally accepted Plug-and-Play mechanism of attaching a hardware component in a system without the need for knowing exact physical device configuration. Following this approach will enable better interoperability of devices in the industrial setting. We think that this leads to greater flexibility of production processes, which is one of the basic postulates of Industry 4.0.

In addition to the Introduction and Conclusion, this paper has five more sections. In Section 2, we provide an overview of the current state in the field with a description of existing solutions. In Section 3 we introduce the problem we aim to solve and the initial approach to solving the problem. In the same section, we derive requirements for the approach. We present some implementation details of the prototype developed in support to our approach in Section 4. Afterwards, in

Section 5, we present a use case in which we have tested our approach and the prototype in order to provide a proof of concept. In Section 6 we provide a short discussion about advantages and disadvantages of our approach which we have identified while applying it in the presented use case.

2. RELATED WORK

In our literature survey, we found some research papers which present work aimed at improving connectivity within the field of Industrial Internet of Things (IIoT). The approaches presented in these papers aim to improve the connectivity by using Gateway [1] or Smart-Hub [2] devices. The authors of [1], present the architecture of an IIoT Gateway and a Cloud solution that both comprise Controller and Data Manager modules. The Controller module is implementing control functions for controlling the devices, and the Data Manager module is implementing data functions for data processing, transformation and conversion. The entire architecture of the solution is based on the Hypertext Transfer Protocol (HTTP) protocol, web sockets, and other widely used web technologies making the solution easily integrated into the current web information systems. However, the authors only introduced the architecture without really presenting how to solve device interoperability issues. In [2], authors present the IIoT-based Smart-Hub that receives raw data from IIoT devices. The preprocessed data is transferred to the Microservice-based IIoT Cloud Platform. The transfer is done by serializing data in Extensible Markup Language (XML)[1] or JavaScript Object Notation (JSON)[2] formats and sending the data over the Message Queue Telemetry Transport (MQTT)[3] and Constrained Application Protocol (CoAP)[4] lightweight protocols. However, the authors of the paper also do not provide a description on how they connect devices with different protocols. Eclipse Hono [VII] architecture is based on using scalable Advanced Message Queuing Protocol (AMQP)[5] messaging middleware between devices and backend services. If a given device needs to send the data over a non-AMQP protocol, it connects to a protocol adapter responsible for conversion between AMQP and non-AMQP messages. Although Hono seems to resemble our approach, we found that it focuses more on IoT and Cloud systems in general, and not in the industrial setting. Furthermore, Hono focuses more on a manual specification of adapters, while a pure Plug-and-Play mechanism would lean more to the automatic integration.

To enable a fully functional Plug-and-Play mechanism, components of the value chain in the IIoT systems need a common language to understand each other. In paper [3], authors propose an interaction model based on The Reference Architectural Model for Industry 4.0 (RAMI 4.0), which is service oriented architecture based on the Open Platform Communications Unified Architecture[6] (OPC-UA) protocol. RAMI 4.0 is much more complex and includes many segments other than connectivity which are not needed for the solution that we aimed our focus on. However, it is interesting to see how the RAMI 4.0 positions the integration and connectivity layer in the grand scheme of things in the domain of IIoT. It regards this layer as one of the main enablers of the future "smart" industry.

Several other solutions with similar background and purpose to our solution are offered on the market, such as Allentia IoT Gateway [I], Thnghub [II], AWS IoT [III] or DGLux5 [IV]. However, all of these are commercial products that we haven't had a chance to test and further investigate. Thnghub [II] seems to follow a similar approach to the one presented in this paper. But from the available documentation, we were not able to fully investigate all the functionality and internals of the solution. Amazon AWS IoT [III] offers a solution in which devices communicate securely and efficiently with some of AWS Services like Elasticsearch[7]. The problem is that it only supports MQTT and HTTP protocols, which we find limiting, especially in the industrial setting. DGLogic DGLux5 [IV] enables access to several IoT data sources in a single, unified workspace via drag&drop data binding. It also comprises tools for building applications on top of the connected devices through visual programming and wizards and visual assistants. It is based on Distributed Services Architecture (DSA) [V], an open source Internet of Things (IoT) platform whose objective "is to unify disparate devices, services and applications into a structured and adaptable real-time data model". However, while DSA is open source, DGLux5 is a commercial product. In our opinion Allentia's product, IoT Gateway [I], has the most complete set of functionalities to tackle interoperability issue in the industrial setting. Their "semantic" Gateway bridges industrial devices and enterprise applications such as Google Analytics[8]. Allentia promises a library of thousands of already available device drivers on one hand, and a possibility to write user-defined drivers. However, the big limitations that we found in this solution are both hardware and software dependence on the Allentia itself. Moreover, since it is a commercial product, we were not able to test it and test the performance of the platform as well as the interoperability with different industrial devices.

It is evident that field of Industry 4.0 will evolve in the forthcoming years. Some products like Harting Modular Industry Computing Architecture (MICA) [VI][5] are already being advertised as huge cost savers, while others like Allentia IoT Gateway [I] and Evrything Thnghub [II] promise a possibility to connect any device to any application. Germany, as one of the world leaders in industrial research and production, has a long running federal strategy [4] of investing into the

[1] XML: https://www.w3.org/XML/
[2] JSON: http://www.json.org/
[3] MQTT: http://mqtt.org/
[4] CoAP: http://coap.technology/
[5] AMQP: https://www.amqp.org/

[6] OPC-UA: https://opcfoundation.org/about/opc-technologies/opc-ua/
[7] Amazon Elasticsearch: https://aws.amazon.com/elasticsearch-service/
[8] Analytics: https://www.google.com/analytics/analytics/#?modal_active=none

research in the field. Taking this into consideration, we are certain that our solution will also be of interest in years to come. We are focused on solving multiplatform connectivity issues in the field of IIoT, while strongly relying on open-source technologies. Open-source communities are some of the key providers of technology for the software industry in last decades [7]. We find that there is still a lack of communities that tackle problems in IIoT and that this is one of the advantages of our approach over the analyzed ones.

3. THE APPROACH TO DEVICE RECOGNITION AND INTEGRATION

In this paper, we present our initial research on recognizing devices which enter a closed system/network and their "injection" into the system. Our approach closely resembles the Plug-and-Play approach encountered in contemporary personal computers and their handling of USB devices. Every device needs to freely share information about itself while at the same time being able to recognize and then communicate with other devices. This is hardly possible without some kind of a mediator. Therefore, our approach is centered on a Server and a Gateway that are connected and able to recognize new devices and enable communication between all the devices – directly or indirectly. This suggests that there are two main modules of the system architecture on top of which our approach is implemented. These modules, presented in Figure 1, are a Communication Enabling Server (in later text CES or Server) and an IIoT Gateway (in the later text just Gateway).

3.1. The Main Modules of the Solution

The main functionality of *CES* includes: (i) maintenance of the Database (DB) in which device descriptions are stored and (ii) generation of drivers for any pre-described device that enters the system. Currently, Server generates a custom driver for devices which have their description in the DB but does not generate a fully custom driver for a completely new and unrecognizable device. Instead, similar to the USB driver mechanism, it generates a "generic device" driver. *Server* stores device type descriptions that are previously created by engineers or device manufacturers. Depending on the connected device, it automatically generates a driver via an internal generator that takes the description and produces an executable flow for data transformation. Server communicates the driver via a RESTful web service (REST) to Gateway, a second module of the system in which the devices are being tracked and whose protocols are being translated.

Gateway bridges the communication gap between devices (actuators, sensors and other equipment) and the system as a whole. It serves as a multi-protocol control unit that handles connectivity between the new and existing devices. Therefore it represents a key access point for the network connectivity, regardless of protocols and interfaces. The main goal of Gateway is to "translate" various protocols that all the connected devices are communicating with. All the protocols are transformed to standardized IIoT protocols, OPC-UA and MQTT. External applications see the entire system through these two protocols regardless of the number of other devices and other protocols. This enables the end-user to develop applications that are better optimized and custom made to the system as a whole. Field devices are connected to Gateway via one of the commonly used interfaces, like EtherCAT[9], Wi-Fi, Bluetooth Low Energy (BLE[10]), Zigbee[11] etc. All additional interfaces and protocols could also be supported with additional work on our side, based on the integration of previously supported protocols and interfaces. Each interface has specific features that have to be adapted in the course of transformation to output protocols on Gateway – both OPC-UA and MQTT. These adaptations are done only once, during the initial introduction of a new interface or protocol. Once the protocol and interface are supported on Gateway, the end-user only needs to write a compatible device description. In Figure 1, we present the bottom and upper communication layers of Gateway. Regardless of the entry interface of the connected devices, OPC-UA and MQTT are the only two protocols that are visible outside of Gateway. The only entry point to the system is our Gateway enabling us to have logical independence from the concrete connectivity protocols on the production floor.

3.2. Steps of the Approach

When a device enters the system, e.g. sensor S1 is plugged via an Ethernet cable, it communicates to a module. The device informs Gateway about the protocol over which it exchanges data. Gateway identifies the device and sends an input to Server via Application Programming Interface (API) and requests an appropriate device driver. The driver is generated on the Server side, only after the adequate device is recognized in the list of defined descriptors that is read from the database. As soon as the driver is generated and the data transformation for the device is enabled on the Gateway side, all the available variables and functions of the device are transformed to adequate OPC-UA variables and MQTT topics. With these transformations, the device is being able to (indirectly) communicate to all the other devices via its protocol or via the other supported protocols. For example, although being a predominantly MQTT device, S1 can establish a communication with S2 or A1. The means of the communication are defined at the Application layer of the system, independently of Server or Gateway layers. With the continuing improvement of all the layers it will be possible to enrich the application layer as well. Ideally it would enable end-users to have an automated process in a modern GUI (e.g. Simulink[12] like application) in which all the devices would be out-of-the-box blocks that are ready to be connected to each other.

[9] EtherCAT: https://www.ethercat.org/default.htm
[10] BLE: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/le-p2p
[11] Zigbee: https://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard
[12] Simulink: https://www.mathworks.com/products/simulink.html

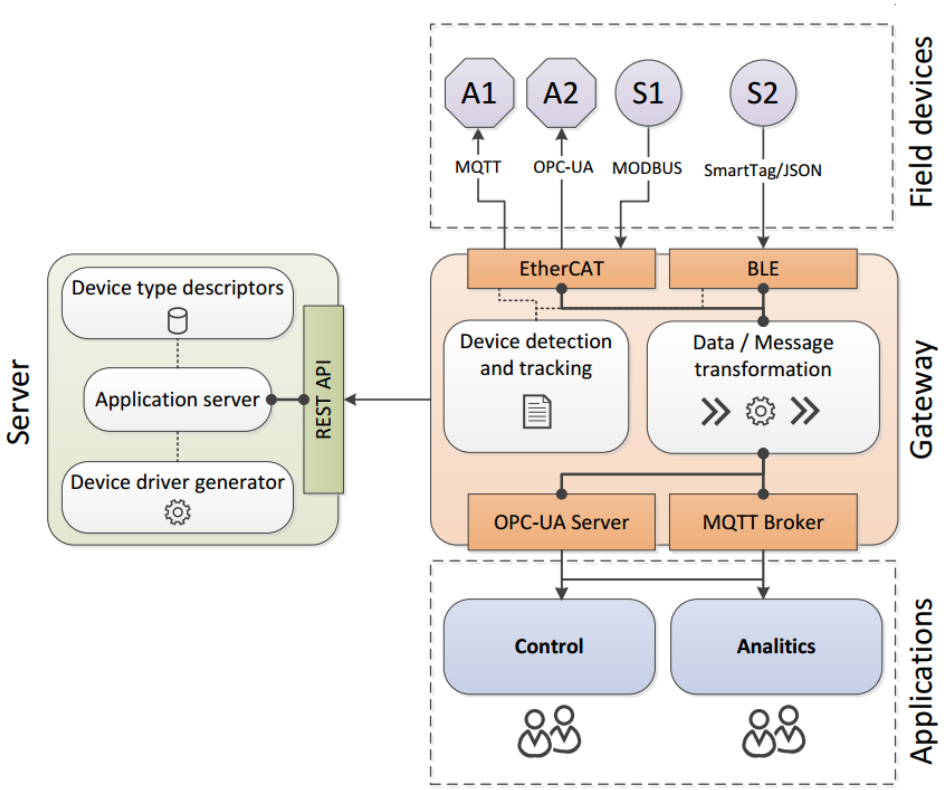

Figure 1 – The architecture of the proposed solution.

4. PROTOTYPE SOLUTION

Server is implemented as a RESTful application using the Spring framework[13]. Gateway is implemented as a combination of a Spring-based application and Node-RED[14], an open-source IoT connectivity framework. Gateway is physically stored on a Raspberry PI[15] unit that runs on Raspbian[16], a Linux-based operating system. Because of its more complex nature and potential growth of the device descriptions database, Server runs on a PC. Neither Gateway nor Server required too much hardware resources and have worked without any problems in our limited prototype.

Server application comprises two main components: (i) the DB component which stores device descriptors and (ii) the device driver generator component. We are using an IoT-based descriptive language that enables us to write abstract device descriptions. These technology agnostic descriptions are practically information models of a single device. Based on a device description, e.g. device description provided by a manufacturer, a compatible, platform specific device driver is generated and the device is put to use. Therefore the key starting point of this process is to have a proper device description. For the needs of the prototype, we have chosen the open-source Eclipse Vorto[17] project because of its features – Vorto Toolset and Repository primarily. Definition of a function block written in Vorto is straight forward and easily readable by a regular user of the system. The key elements of Vorto descriptions are device status and operations fields. Device descriptions are written manually using the Vorto GUI, but the goal is to have this process partially automatized in the future. Also, there is an existing repository of existing device description that can be reused. In our solution, all descriptions are stored in the MySQL DB and are later retrieved from the DB by providing a device ID. The device ID can be a MAC address of BLE or LAN devices or some other, device-specific ID that is broadcast by the device upon connecting to Gateway. The device driver generator is implemented in Java and takes the device description fetched from the DB as an input. A generated driver represents a Node-RED communication flow that is delivered via REST API to Gateway. It is then inserted into a running Node-RED instance and immediately it starts to translate data from the new device to OPC-UA and MQTT protocols which Servers are incorporated in Gateway.

The heart of Gateway is a Node-RED-based application that runs and continuously listens to and detects the arrival of new devices into our closed network. One of the nodes that is repeatedly running is the Linux Address Resolution Protocol (ARP) scanner, a tool that scans and refreshes the list of all the connected devices on a specified network interface. Devices are recognized and their IP or MAC addresses are cached. Gateway then sends a request for the adequate driver for the device to Server. As previously explained, this driver is being generated on the Server side and sent back to Gateway. If the driver is received correctly, the Data transformation block on Gateway is enabled and

[13] Spring Boot: https://projects.spring.io/spring-boot/
[14] Node-RED: https://nodered.org/
[15] Raspberry PI: https://www.raspberrypi.org/
[16] Raspbian: https://www.raspberrypi.org/downloads/raspbian/
[17] Vorto: http://www.eclipse.org/vorto/

appropriate variables on the internal OPC-UA Server and corresponding topics on the MQTT Broker are dynamically created.
At this moment, the third component of our system, the Application, is also implemented in Node-RED. Within it, we create several nodes that represent simulated production process. As soon as another device comes into the system, its variables/topics are created and can be manipulated with in this temporary application layer. The Application itself was not one of the targets of our research and this was made just for testing purposes.

5. THE PLASTIC MOLDING MACHINE USE CASE

In order to make a proof of concept of the given approach, we have set up a small model of a production unit which is used as a prototype. Plastic parts are produced in a molding machine whose heart is an industrial PC. The parts are transported through the production unit via several conveyor belts. During the transportation the plastic parts are being tested and sorted, as part of quality assurance process. Our model consists of a functional industrial PC, a modeled conveyor belt, a few small single-board computers, a set of LEGO Mindstorms[18] and a smart sensor unit. The used IPC is a commercial device, high-end controller of KEBA's CP3xx series. It runs an IEC61131-3[19] application with all the variables being shared via integrated OPC-UA Server. These variables are later used in the prototype.

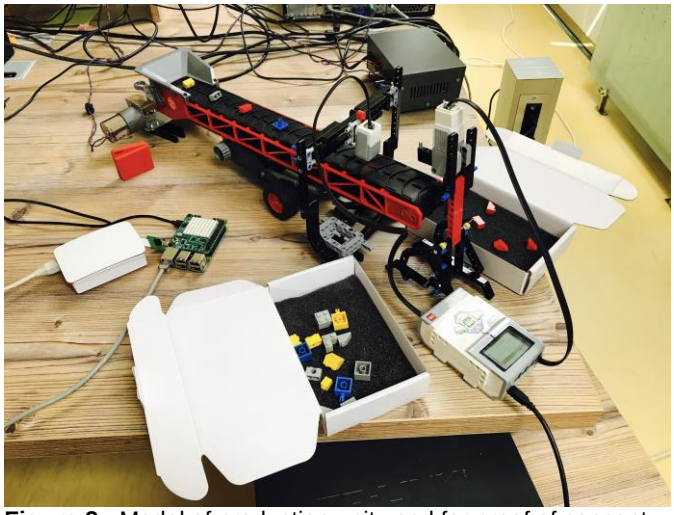

Figure 2 - Model of production unit used for proof of concept

LEGO Mindstorms serves as a simulator of two devices – a color sensor on one hand and a sorting actuator on the other. Texas Instruments SensorTag[20] is used as a temperature sensor that enters the system later during the production and enriches the process. It's already prepared IoT profiles and interfaces enabled us an easy introduction of sensors into our production process in real-time. Raspberry PI, used to host Gateway and IPC both run on Debian based Linux operating systems. Server can run on both Windows and Linux operating systems. Conveyor belt used in the production unit is only a toy model that is adjusted to our needs with adding a small DC motor that controls the flow. The process itself is straight forward. The simulation of the plastic brick production runs on the IPC. Bricks are laid down on the conveyor belt in a predefined time interval. They run underneath the color sensor and depending on their color are sorted in one of two stocks. Simulation is partially depicted on Figure 2.
All the components of the production unit are inserted into a closed network one by one, while their drivers are being generated dynamically. As soon as the connectivity is successful, the full process can be monitored via a 3rd party client application, e.g. OPC-UA Server can be analyzed through UA Expert[21]. When a temperature sensor is added to the network, the process of generating the driver is repeated and the "production" process in our production unit is enriched with the possibility to follow and react to additional conditions for the actuator operations. The basic option would be to add a third level of selection or to stop a whole production process if a condition is satisfied. For example, if the ambient temperature is above 45°C the production process should be stopped. Potentially, we could replace the color sensor or the temperature sensor in real-time and still be able to keep the process running, which argues towards the robustness of the presented solution.

6. DISCUSSION

In the previous sections, we presented an initial implementation of our approach. Not only that we focused on delivering a simple realization of the goals, but we also tried to fit it in a real-life use case, which enabled us to analyze advantages and disadvantages of the current solution. We wanted to build a system in which devices could be easily detected and connected, with as less coding for the end-user as possible.
One of the first benefits of our implementation is the easiness in describing devices. An engineer does not need to write a complete driver for any particular devices that enter the system – he can just reuse the previously written description for a similar device with small amount of changes. All the descriptions are adaptable but straight forward. The main goal is to easily replace one device with another of the same type, e.g. switching temperature sensors. Although two sensors may have different realization, their descriptions can still be the same, since their main function is to measure temperature before anything else.
On the other hand, our generative approach may lead to potential performance issues, as a manual implementation of device drivers most probably leads to a faster communication. Our assumption here is that the process of automatized generation of drivers is several times faster than manual implementation while

[18] LEGO Mindstorms: http://www.education.rec.ri.cmu.edu/content/lego/ev3/files/EV3%20teachers%20guideWEB.pdf
[19] IEC61131-3: http://www.plcopen.org/pages/tc1_standards/iec_61131_3/
[20] TI SensorTag: http://www.ti.com/ww/en/wireless_connectivity/sensortag/
[21] UA Expert: https://www.unified-automation.com/products/development-tools/uaexpert.html

the system is more flexible at the same time. The assumption is yet to be confirmed through the detailed tests planned for the next phase of the research.
Ideally, we would like to have fully functional and robust Plug-and-Play mechanism on the similar level compared to personal computing. This is obviously a hard task to accomplish in one step but does not seem so far-fetched with the assumption of the industry really evolving towards standardized protocols like OPC-UA. There are several commercial products that already offer solutions similar to ours, but none of them is offering a "free of charge" approach. Also, none of the solutions that we have analyzed offers a level of automatized generation of drivers like it is offered by the solution presented in this paper.
At the moment we are focused on implementing support for OPC-UA and MQTT standardized devices, but it is clearly a doable task to translate the current knowledge to other standards as well. We expect that with the further company and men support we would be able to cover as many interfaces, protocols and devices as possible, in order to already have a widely supported and powerful solution.

7. CONCLUSION

In this paper, we presented our approach for bringing Plug-and-Play concept closer to production processes in industrial automation. We proposed an approach which is based on open-source technologies and available for all the users without the need for additional hardware. We also showed that our solution is generic, platform independent and easily expendable, which may prove as a valuable asset. In order to evaluate the approach, we presented a prototype which was built using small electronic devices and PCs and applied in a simulation of a real use case. There are two key benefits of our implementation. First is the increased degree of automatization in device driver creation process. Second is the increased level of abstraction in writing device descriptions. The mentioned increases may lead to lower performances in communication, but additional tests must be carried out before final conclusions are made.
One direction of future work is to try out other description languages, e.g. Franca[22], in order to evaluate other approaches of specifying the device description. These other description languages might prove to be easier to learn or to use than Vorto which would further ease the only manual task that still needs to be performed. One of our main goals in the future is to have a Process tool that would be block-based. The solution presented in this paper would be a connectivity layer for fully functional, advanced GUI that would show all newly arriving devices as basic function blocks. This would enable us to have, at least logically, direct communication between various devices, irrelevant of the background implementation of the data transformation. Further steps include thinking of more robust and more efficient recognition of the devices entry. One of the approaches is to have these device descriptions stored on the devices itself, but it is a far-fetched and inappropriate solution for older, already deployed devices in the industry.
We find that we may have a strong enough starting point for an AI solution that could bring the process of generating the drivers to a higher level. Future research will be also focused on delivering a fully automatized Plug-and-Play mechanism that would enable the recognition of completely new devices entering the system and adding them to the production process without any previous knowledge of the device. For example, device descriptions could already be generated automatically for groups of devices that use standardized communication protocols i.e. those that support OPC-UA, since there are some standardized mechanisms of serving the data.

8. ACKNOWLEDGMENT

This research was supported by KEBA AG Linz. We thank our colleagues from KEBA AG who provided insight, equipment and expertise that greatly assisted us in the research.

9. REFERENCES

[1] P. Hu, "A System Architecture for Software-Defined Industrial Internet of Things," in Ubiquitous Wireless Broadband (ICUWB), 2015 IEEE International Conference, Montreal, Canada, IEEE, November 2015, pp. 1–5.
[2] C. K. M. Lee and S. Z. Zhang, "Development of an Industrial Internet of Things Suite for Smart Factory towards Reindustrialization in Hong Kong", Proceedings of the 6th International Workshop of Advanced Manufacturing and Automation (IWAMA), Manchester, England, v. 24, pp. 285-289, Atlantis Press, October 2016.
[3] C. Diedrich, A. Bieliaiev, J. Bock, et al. „Interaktionsmodell für Industrie 4.0 Komponenten", Special Issue: Entwurf komplexer Automatisierungssysteme / Prof. Dr.-Ing. Ulrich Jumar. at, pp. 5-18, Automatisierungstechnik, 2017.
[4] "Interaction Model for Industrie 4.0 Components", Discussion Paper of the German the Federal Ministry for Economic Affairs and Energy, April 2016
[5] R. Schaefer, "Edge Computer reduziert Datenuebertragungskosten", in MaschinenMarkt Magazine, June 2017
[6] „What criteria do Industrie 4.0 products need to fulfil?", ZVEI Elektroindustrie Whitepaper, April 2017, URL: https://www.zvei.org/en/subjects/industry-4-0/en-welche-kriterien-muessen-industrie-40-produkte-erfuellen/
[7] Eclipse IoT White Paper – "The Three Software Stacks Required for IoT Architectures", September 2016, URL: https://iot.eclipse.org/resources/white-papers/Eclipse%20IoT%20White%20Paper%20-%20The%20Three%20Software%20Stacks%20Required%20for%20IoT%20Architectures.pdf

10. NOTES

[I] Allentia IoT Gateway. URL: http://www.alleantia.com/en/iot-gateway/
[II] Evrything Thnghub. URL: https://evrythng.com/resources/data-sheets/thnghub/
[III] AWS IoT. URL: https://aws.amazon.com/iot-platform/
[IV] DG Logic DGLux5. URL: http://www.dglogik.com/products/dglux5-ioe-application-platform
[V] DSA – Distributed Services Architecture. URL: http://iot-dsa.org/
[VI] Harting MICA. URL: http://www.harting-mica.com/en/home/
[VII] Eclipse Hono. URL: https://eclipse.org/hono/

[22] Franca: https://github.com/franca/franca/