# Device Adapter Concept towards Enabling Plug&Produce Production Environments

Kirill Dorofeev*, Chih-Hong Cheng*, Magno Guedes†, Pedro Ferreira‡, Stefan Profanter*, Alois Zoitl*
*fortiss GmbH, An-Institut Technische Universität München, Munich, Germany
{dorofeev, cheng, profanter, zoitl}@fortiss.org
‡Introsys - Integration for Robotics Systems SA, R&D Department, Quinta do Anjo, Portugal
magno.guedes@introsys.eu
†Wolfson School of Mechanical and Manufacturing Engineering Loughborough University, Loughborough, United Kingdom
p.ferreira@lboro.ac.uk

***Abstract*—Modern manufacturing systems require a transformation from mass production towards mass customization. This results in a trend towards more agile production lines. It also demands a reduction of configuration times when building the production line as well as faster reconfiguration when adding new hardware and product variants to the manufacturing line. This paper introduces the concept of a device adapter that allows the device to be seamlessly plugged into the agile production systems. The device adapter wraps the device functionality and offers it as a service, hiding away the low-level process capability (skill) implementation and allowing to formally represent the production steps. Preliminary tests have been performed on an industrial demonstrator that simulates a real manufacturing process.**

## I. Introduction

One important aspect of Industry 4.0 (I4.0) is the seamless integration of physical products and processes, and their virtual representations in one network [1]. Assets represented as I4.0 components autonomously communicate directly with each other to execute tasks, such as production plans. The logical concept of *administration shell* [2] is proposed as an enabler of such an autonomy, where a machine with an administration shell software installed is readily capable of performing I4.0 tasks. What is currently missing, however, is how to realize it in a Plug&Produce scenario, where one needs to (1) define semantic protocols for upward (Manufacturing Execution Systems, Enterprise Resource Planning) and downward (the low-level PLC's, sensors, actuators) communications and, moreover, (2) seamlessly integrate existing technology stacks such as AutomationML [3] and/or OPC Unified Architecture (OPC UA) to utilize their interoperability.

In this paper, we solve the above mentioned problem by proposing *device adapters* as a concrete realization of administration shell, where the operational requirements are carefully derived from multiple intelligent manufacturing scenarios among different industry sectors: electronics, white goods, automotive. In the Plug&Produce scenario, the requirements form a sequence of actions (modes) needed to be realized. After standard component discovery over the physical presence, we detail how a device adapter negotiates with the upper levels of control, e.g., cloud based systems. Subsequently, we propose a framework by utilizing the concept of skills [4] [5] [6] for two purposes: (1) to expose the devices functionality to the upper control level as capabilities, and (2) to allow high-level control to configure the machine via parameterized recipes. We use self-description [7] as a way to expose skills, where we carefully design it in the form AutomationML, and allow runtime translation into OPC UA information model. Although there is a natural translation from triggering a skill to a method call in OPC UA, it is surprisingly underspecified in the Companion Specification AutomationML for OPC UA [8] regarding how a skill defined in AutomationML should be realized in OPC UA. To this end, our translation incorporates such modeling deficiencies by additionally creating specialized method calls for each skill.

We experimented the above mentioned concepts on a real production setup and validated the benefits. The highly heterogeneous machines used in the demonstrator show that the proposed concept is general and can be reused for different setups. The demonstrator also serves to analyze the potential of introducing the device adapter to existing production systems.

The rest of the paper is organized as follows. After giving a related work overview in Section II, Section III describes the demonstrator mentioned above. Section IV defines the main requirements derived from the demonstrator use-cases. Section V describes the solution concept of the device adapter and surrounding components needed for realizing a Plug&Produce scenario. The details on how it was realized on the demonstrator are given in Section VI. We summarize possible further investigation steps and conclude in Sections VII and VIII respectively.

## II. Related Work

The need for higher levels of production systems flexibility in the industrial automation domain has been indicated in the literature for a quite long time. The Plug&Produce concept introduced in [9] is built upon the splitting of system modules functionality in self-contained modules. These modules are then used as building blocks of the production system. The system module is defined based on the analysis of system components and identifying the similarity between those, whereas different modules should be kept independent from each other [10]. There can be distinguished two types of such modules, the physical and logical [11]. The modularity of physical systems is nowadays widely spread in the industrial automation [11], whereas the logical components for such agile production systems are under active research. Once the modules are defined under the context of a modular architecture, a finite set of modules can potentially deal with an almost infinite set of changes [10].

Recent developments [4] [5] [12] have provided the base for the definition of a configuration methodology by defining the hierarchical skill model that allows the logical configuration of such systems. This model is in line with an open standard, which enables the definition of modular systems, such as Automation ML [12]. Moreover, in [13] it is shown how the AutomationML model can be operationalized by means of OPC UA, a family of technologies intended to provide a communication between field devices and upper control levels [14]. This, however, requires some extensions in order to make it create agile production lines [5].

## III. Running Example

The system under analysis, showed in Figure 1, is deployed at the Introsys facility and consists of two robotic spot welding cells that represent actual production lines from the automotive industry. The main actor of each cell is a 6-DoF robotic arm, equipped with a custom gripper designed to handle a body part of a car (a longeron). Motion control and safety routines are managed by an industrial field PLC. An additional vision-based quality inspection system allows to identify potential imperfections on the weld. Finally, an energy monitoring module collects and registers the energy consumed by all equipment.

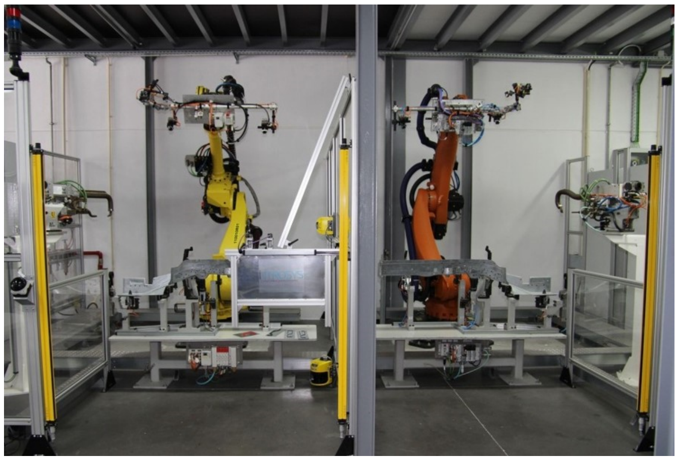
Fig. 1. Robotic Spot Welding Cells at Introsys

The assembly process executed is identical for both cells and consists of the following steps: (1) the robotic arm picks a car part from the fixture, (2) a spot welding operation is performed, (3) the robot takes the finished part to the visual inspection system and (4) the welded and inspected part is dropped back to the fixture. The difference between cells lies on the vendor standard adopted. These standards are currently used by major car manufacturers and regulate not only the equipment, but also the industrial communication protocols, electrical wiring, safety devices, software and human-machine interface used within the workstations.

Raw parts are delivered to the cells, according to production needs, by means of an Automated Guided Vehicle (AGV). The AGV has an on-board localization mechanism to safely navigate through pre-defined paths without interfering with the infrastructure. This aspect is important for having Plug&Produce enabled transportation systems as possible trajectories can be created, deleted and modified by software only and without the need for installing physical guidance mechanisms (e.g. magnetic tape or inductive wire). The backbone of the AGV runs a Robot Operating System (ROS) environment. For the current demonstration needs we use the Gazebo Simulator[1] that allows to simulate multiple vehicles. All AGV's are supervised by a central transportation management system (TMS) that is also responsible for determining the best routes and available units to meet transportation requirements.

## IV. Operational Requirements

To achieve a Plug&Produce system [9] four phases need to be completed successfully:

*Discovery Phase.* During this phase, a new device being plugged into the system broadcasts its presence to the middleware controller that initiates the communication and defines the registration procedure. Activities conducted in this stage are highly related to discovery services and can be implemented using specific protocols and different communication technologies such as OPC UA. Then the middleware is informed about the device existence, and is able to communicate with it. The middleware and the device should also keep track of any change in the system topological information, leading to changes in the process execution and system capabilities overall. After completing the Discovery Phase, the device is connected to its virtual representation in the cloud.

*Configuration Phase.* The goal of this phase is to configure the inserted device such that it can perform product-specific production tasks. Each newly inserted machine can provide its capabilities as *skills*, hiding the low-level implementation. Then the production execution can be triggered by calling an appropriate skill and by adjusting corresponding parameters. It is important to have a formalized approach for the definition of skills, to make them interpretable for both device and the cloud system communicating with each other. The skill configuration instances, *the recipes*, are also created during this phase and are validated to ensure that the system executes only acceptable action sequences, whenever the recipe is triggered. Overall, this phase can be seen as a process that transforms a product specification to a machine configuration [15].

*Production Phase.* This phase covers the manufacturing of products, new products introduction, changes in the planning, dynamic system adjustments, routing the product through the workstations, and storing of the production data. The majority

[1]http://gazebosim.org/

of the activities here are dealt automatically by the system components. Users are expected to have limited intervention in this phase. Users can deploy new products as well as optimized recipes (thus directly enforcing certain recipes), approve recipes or trigger exceptions in the operation. It is important to note that the system should have the ability to deploy new recipes during the Reconfiguration Phase, as specified in the later phase, based on the available production data, which can be constantly stored in the cloud. The exception handling is another big issue, which should be considered when building such systems.

*Reconfiguration Phase.* For an intelligent production unit, the device adapter should be designed to process multiple products. This implies that the change of production for known products, where recipes and parameters for those are already stored, does not require any reconfiguration. However, whenever there are new modules introduced in the system, or there is a need to insert a new product recipe into the system, a corresponding change phase should be implemented.

Some of the key operational requirements to be fulfilled during designing such systems are the following:

*Auto-discovery*. After the device is being plugged in the production system, the device adapter should start the communication to the middleware. The adapter should be informed about: (1) the IP address of the middleware controller, (2) the entities who initiate the communication, and (3) the methods how to perform registration. Activities conducted in this stage are highly related to discovery services and can be protocol specific being implemented in communication technologies such as OPC UA [16].

*Fast configuration & adaptation.* Reduction of the initial build and reconfiguration effort through the use of Plug&Produce devices with standardized interfaces and built-in control capabilities should be achieved. This allows such devices to be rapidly connected together and be dynamically configured to achieve different assembly processes. The plugged machine is able to configure itself such that it can perform product-specific tasks, i.e., fulfill the process that creates a machine configuration out of product specification. Overall, this means that the production system as a whole should be able to react faster to new product variants and scheduling modifications, adapting to the market fluctuation, e.g., change in required production quantity, and production emergent situations, or re-planning and stoppages. Some of key factors to achieve are the ability to dynamically route products, load-balancing between different stations, and the real-time changes in systems functionality in response to new product variants.

*Triggering production tasks.* After the component gets all required information, it should start the execution, enabling corresponding actions to accomplish its task. For doing that, the device adapter should be able to interact with the middleware system that does the orchestration.

*Components reconfiguration, including hardware and communication reconfiguration.* Perform job switch when new production task arrives, or if there is a change in communication parameters, or if there is a change in hardware setup. This allows rapid integration and adaptation of different devices and assembly stations.

*Self-description.* Each Plug&Produce device should have an embedded description that provides, among other information, the information about its skills, data and different analysis that it does, whereas each product should comprise wireless self-identification procedures.

## V. Solution Concept

In this section, we present our solution concept realized within the device adapter, together with its surrounding environment components to enable Plug&Produce. The high-level workflow for realizing intelligent Plug&Produce is shown in Figure 2. The manufacturing service bus (MSB) provides a uniform communication framework at the shop floor layer, integrating all components involved [17]. The MSB is in charge of all vertical and horizontal communications within the system, providing a high degree of flexibility, where the devices can be easily plugged in and plugged out without harming the overall system functionality. Knowing the underlying topology and being in charge of any communications, MSB can handle routing, and protocol translation between devices. The device adapter is the interacting element between the hardware being controlled, and is in charge of harmonizing the devices and workstations with the existing network when plugged in. The device adapter makes the low level devices capable for communication with the entire network of other devices, workstations and MSB.

### *A. Skills*

The manufacturing system is composed of several equipment, which can be (1) workstations composed of any number of submodules and devices, e.g., gripper, robot and (2) transport units, responsible for moving the workpieces between the stations. Each unit in the manufacturing system is responsible for executing a certain task for completing the production, defined as device skills. In this way each manufacturing unit can be considered as a service provider that offers its skills via the means of MSB. Skills represent the control equivalent to hardware modules and follow the object oriented approach of encapsulating reusable functionality. Skills define the process capabilities offered by the equipment units to complete the required manufacturing process steps. Overall the skills can be seen as services in Service Oriented Architecture (SOA) or methods in programming. The skills have a parameter set, which can be either fixed or dynamically generated for the operation of a manufacturing system. Thus, to execute the skill it should be defined, what are the parameters for its execution. These settings can be defined in the form of a skill recipe concept which should prescribe how a skill requirement can be achieved by a skill.

*Atomic and composite skills.* The skill concept [5] [6] takes the advantage of existing concepts within the manufacturing domain, which provide a hierarchical structure of manufacturing processes. The concept provides the means to structure

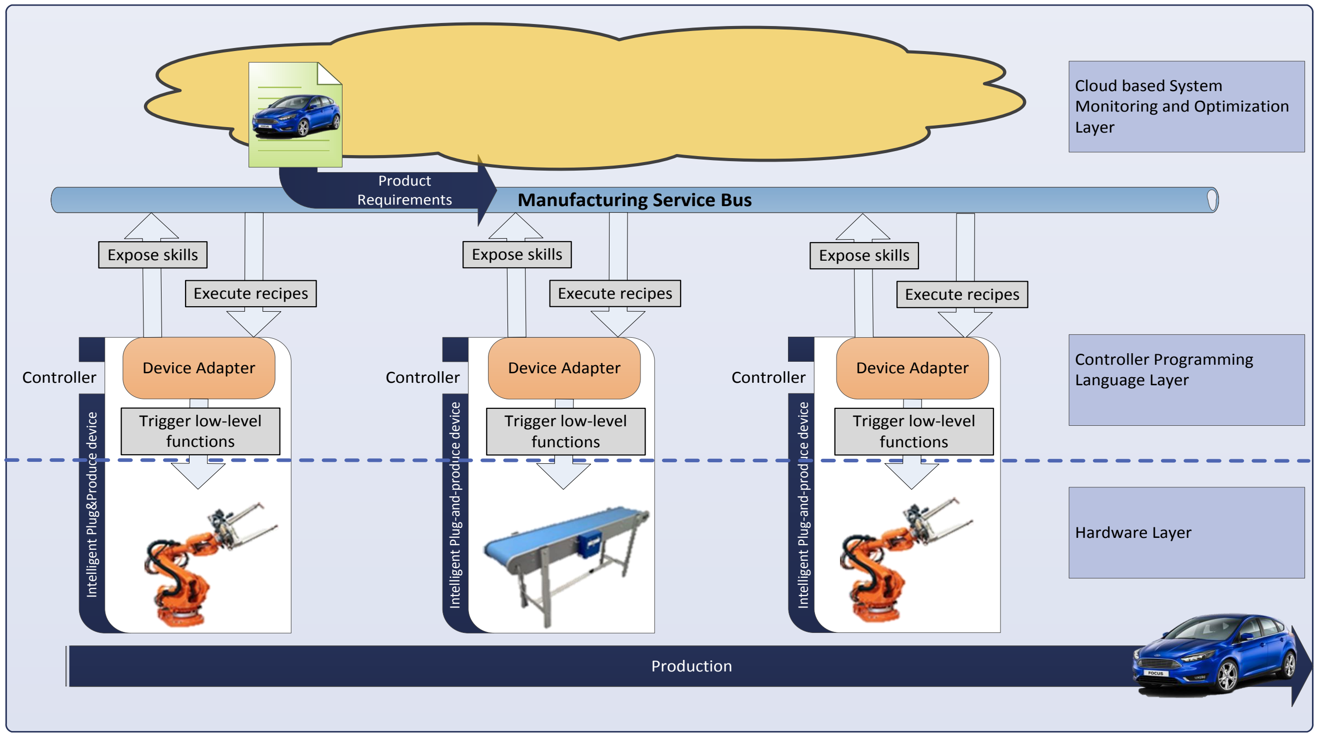
Cloud based System Monitoring and Optimization Layer
Product Requirements
Manufacturing Service Bus
Expose skills
Execute recipes
Expose skills
Execute recipes
Expose skills
Execute recipes
Controller
Device Adapter
Trigger low-level functions
Intelligent Plug&Produce device
Controller
Device Adapter
Trigger low-level functions
Intelligent Plug-and-produce device
Controller
Device Adapter
Trigger low-level functions
Intelligent Plug-and-produce device
Controller Programming Language Layer
Hardware Layer
Production

Fig. 2. Concept Overview

and define higher and lower level processes with the ability to describe their composition, inter-dependencies and parametric constraints. It should be possible to create a sequence of low level skills, combining those into more complex skills. To separate these two cases, we define two types of skills: (1) atomic, which are defined as elementary skills at the lowest level of granularity, and (2) composite, which are defined as an ordered set of elementary or lower level composite skills. A composite skill should have exactly the same main characteristics as the atomic skill, in fact from an outside point of view there should be no difference. The difference consists in the existence of a structured definition for its lower level of granularity. This definition establishes both the internal process sequence and the information flow which results in the composite skill [4]. The composite manufacturing process can have different combinations of composite and atomic skills and/or different set of parameters that fulfill the skill requirements. In other words, there can be multiple recipes fulfilling the requirements of one skill. The skill can be seen as an IEC 61499 function block, having its event inputs that allows to represent the execution sequence of different skills. The role of data inputs and outputs is to provide the parameters to each of the skill, i.e., instantiate it for executing a specific task. These parameters are not mandatory for all skills, but some manufacturing sequences might require an information flow between different manufacturing processes. In other words, this provides the means for information flow of a given manufacturing process configuration.

## B. *Recipes*

The execution of skills is driven by skill recipes. Recipes are an aggregation of input parameters for the execution of a skill. Before the recipe can be executed during the Production Phase, it should be validated during the Configuration Phase and deployed defining not only the skill's parameter ports but also the event ports which define the sequence of skills being executed. The recipe concept enables the execution of the skills to fulfill a certain skill requirement. This means that the process of matching the skill requirements, which will be defined by the product, to the skills, which are defined by the equipment, will result in the creation of a skill recipe. The different sets of product parameters that describe different product variants should be mapped onto the machine parameters that execute the low level skill. The device adapter should interpret the product specific parameters provided by a production management service and map from product-specific to internal, skill-specific configurations. Recipes can also serve as the basis for improvements. For example, for a given product there might exist multiple recipes that are valid for its production. Subsequently, during the runtime one of the existing recipes can be chosen to optimize the current execution, e.g., execute $Recipe\_fast$ to speed up the whole production or $Recipe\_slow$ to optimize the energy consumption. Existing recipes can be also optimized and tuned during Reconfiguration Phase. The optimized (tuned) parameters should be validated by the user and subsequently returned to the device adapter to supersede the previous version of the recipe.

Similarly to the skill concept, recipes can also be atomic or composite, since they define the input parameters for atomic or composite skills respectively. A composite recipe can be an aggregation of both atomic recipes and composite recipes, which describes the manufacturing process with respective precedencies. An atomic recipe has the set of parameters to execute a skill. One should note that the same recipe can be executed any number of times with a different parameter set. For doing that, the recipe update can be done during the Reconfiguration Phase.

### C. *Execution*

Bringing together these concepts of atomic skills, composite skills and skill recipes, it is possible to describe the overall view of execution of such a manufacturing system. In Figure 2 the device adapter plugged into the system offers the skills of the underlying devices to the MSB. The MSB can afterwards use one of the recipes, pre-validated during the Configuration Phase, and download them back to the device adapter. The product, arriving at the workstation, using the product ID , e.g., using Radio-frequency identification (RFID), triggers then the execution of the corresponding recipe that is translated into the low-level function call by the device adapter. Anytime a new recipe can be introduced into the system. This means that at different moments in time the same product ID can be executed with different recipes. This allows changing of running recipes, e.g., for optimizing the process in terms of production speed or energy consumption.

### D. *Architecture concept overview*

All the communication is done through the MSB. The devices with the device adapter being connected to MSB are able to advertise their skills. The device level requires atomic devices that are aggregated up to workstation level, the highest level of granularity. This means all devices internal to a workstation will require communication with the ability to expose services to make their skills available. This communication will be critical for the performance of the system, as it implies the ability to execute skills, and therefore requires a higher level of priority. Parallel to the workstation, the transport is considered at the same level of granularity as changes to the transport would not require human interventions. The existence of these two device types establishes the need for an interaction between the transport and the workstation, which represents the movement of the product from the transport to the workstation. The device data for every device in the system will be stored in the cloud and is subsequently available for any further analysis and decision-making processes.

## VI. Implementation

For implementing the above described concept we chose Automation Markup Language (AutomationML) for devices self-description and OPC UA as a communication protocol. AutomationML supports object-oriented approach and allows modeling of plant components as data objects, not only

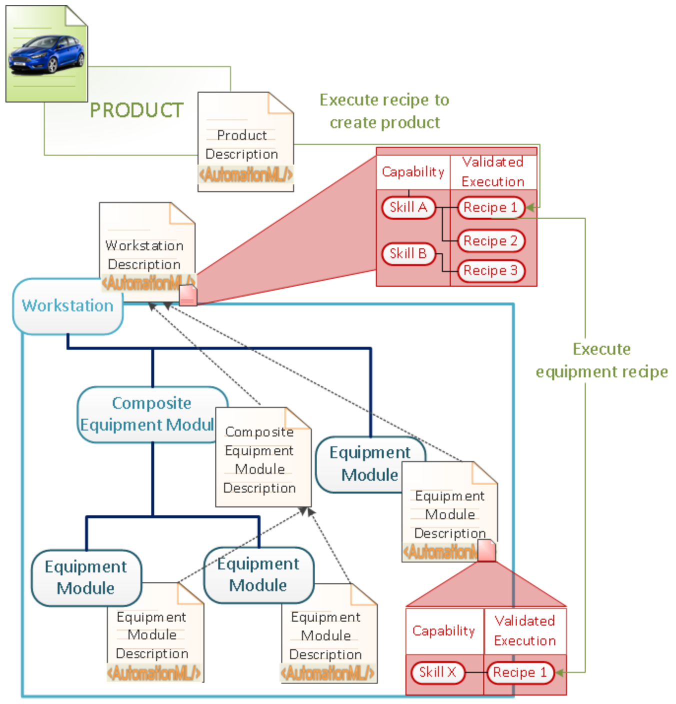

Fig. 3. Informational Model

capturing information of different domains, as their properties within the hierarchical plant topology, but also the associations with other objects [3]. The AutomationML modeling approach is compatible with the required component-based, modular structure of Plug&Produce systems. OPC UA is an open, cross-platform, SOA-oriented machine to machine communication protocol for industrial automation. It was chosen as an approach for implementation of a communication layer in the Reference Architectural Model RAMI 4.0 [2]

*AutomationML model.* The AutomationML for a workstation is an aggregation of all the device self-descriptions files. These are also AutomationML, which is based on XML, and therefore easily joint as aggregated nodes. The self-description file for each device requires common terminologies and concepts for both the Role Class and Interface libraries [3]. This will establish the available skill types and interfaces to be used on the description of the devices. The first step for defining the self-description file, is to create the generic device description in the system unit class library. Once this is done, one can simply create an instance of generic device in the Instance Hierarchy to define the specific device. This node is communicated to the next hierarchal level where it is aggregated, and so on until reaching the highest level (workstation or transport), as it can be seen in Figure 3. It is also important to note that a device description has approved recipes that can be used to fulfill requirements, from both products and composite skills.

All the information, defined in AutomationML description is passed up to the cloud based system towards having a virtual representation of an I4.0-compliant component.

| Recipe 'Produce A' | Skill Requirements |
|---|---|
| Transport A to WS | AGV_Load() |
| | AGV_Route(A, WS) |
| | AGV_Unload() |
| Execute TaskFull | Robot_Pick(speed=75%) |
| | Robot_Weld(speed=15%, program=14) |
| | Vis_Inspect(tolerance=0.5mm) |
| | Robot_Drop(speed=75%) |

TABLE I
DEMONSTRATOR RECIPES

The recipe execution process looks as follows: whenever there is a demand for a new product, either triggered by an operator or by schedule, the MSB captures this request alongside with the respective product ID and informs the device adapter correspondent to the workstation, which previously advertised the skill, to handle such a product. The device adapter consequently retrieves the skill recipe and product parameters for the received product ID. Then it iterates through each skill enclosed in the recipe, finds the required parameters and sends them to the underlying functional interface to execute the skill.

As it is shown in Figure 4, at first, the user creates and validates the recipes to be used by the production system during the Configuration Phase. At the same time the device adapter should get the self-description of the device(s) it is controlling. Then the device adapter is able to expose the skills of the underlying devices to the MSB. Subsequently, after user triggers the production, the MSB sends the commands to the corresponding device adapter to invoke the low-level functions to process the required skill.

The following is an example using the herein described device adapter concept to enable the production of a welding operation on a longeron, as described in Section III. In this case, the longeron should receive reinforced welding and be inspected for weld spot misplacement with a tolerance below 0.5 mm. The recipes, validated in the system are given in the Table VI.

For the above mentioned example, the process starts when the adapter receives an order from the MSB to process a new product ($A$), which is already delivered by the AGV (Transport A to WS completed). The adapter then matches the recipe, retrieved from the 'Recipe List', which allows the execution of product ID $A$ and delivers the list of skill requirements to the 'Skill Executor' for iteration. Inside of the 'Skill Executor', InterpretRecipe first executes Robot_Pick(speed=75%) to  that is in charge of executing Robot_Pick(), and subsequently Robot_Weld(speed=15%, program=14), Vis_Inspect(tolerance=0.5mm) and Robot_Drop(speed=75%) to other EquipmentModule-Runtime that are in charge of these skills. The 'program' parameter from the Robot_Weld() corresponds to an identifier of the robot program that must be executed to perform the weld operation for that skill, which includes the number and position of the weld spots. Other parameters are self-explanatory.

When all skills required by the recipe are executed, the device adapter resets its $productID$ variable to $NULL$ to avoid duplicate execution and informs the MSB about all Key Performance Indicators (KPIs), e.g., power consumption, execution time, during this production task. Finally, the device adapter notifies the MSB controller that the task execution has been finished. The MSB then triggers the corresponding transport unit to execute the product transfer, such that further processing over the product may be conducted by other machines.

It is also worth mentioning that for evaluation purposes the link between the adapter and the low-level devices is carried out using third-party solutions (KepServerX for the field PLC, Unified Automation Wrapper for the robotic arm and energy module) and a custom implementation based on the open62541 library[2] for the visual inspection system. These allow the exchange of triggers, control flags and data between the devices and adapter to start execution of skills, coordinate the execution process and receive back the relevant information (KPIs). This additional layer is what allowed to seamlessly integrate and adopt the Plug&Produce concept introduced by the device adapter in the legacy devices that were not designed or meant to at first place for this kind of functional paradigm.

*Mapping between AutomationML and OPC UA.* For the evaluation we used Java Eclipse Milo project[3] (licensed under EPL-1.0). As the AutomationML standard has been established and the focus now is towards applicability and the connection to other standards we have implemented the creation of OPC UA Information Model out of Automation ML description following the Companion Specification AutomationML for OPC UA [8] [13]. The specification describes how to model AutomationML based information sets using OPC-UA information model and how to transmit them. We take the AutomationML to OPC UA conversion companion, and create a reference converter implementation, where one can automatically convert the model description in AutomationML to Java code for Eclipse Milo OPC UA Server Information model. This implementation allows end-users to concentrate on developing models in AutomationML format, and the code generation feature is automatically handled by our converter software.

After running through the described procedure using the companion specification [8], a created OPC UA Server contains a bunch of Objects, ObjectTypes and Variables that map to the AutomationML model. However, we identified that there is no mapping described between AutomationML model and an OPC UA method call. For providing an interface to trigger the underlying workstation function, the device adapter should offer the capabilities of the low level device that it controls via method calls. For doing so, while parsing an AutomationML file, we create a method node for each skill, defined in the description file. The input parameters for each method can also be automatically generated from the AutomationML file,

[2]http://open62541.org/
[3]https://github.com/eclipse/milo/

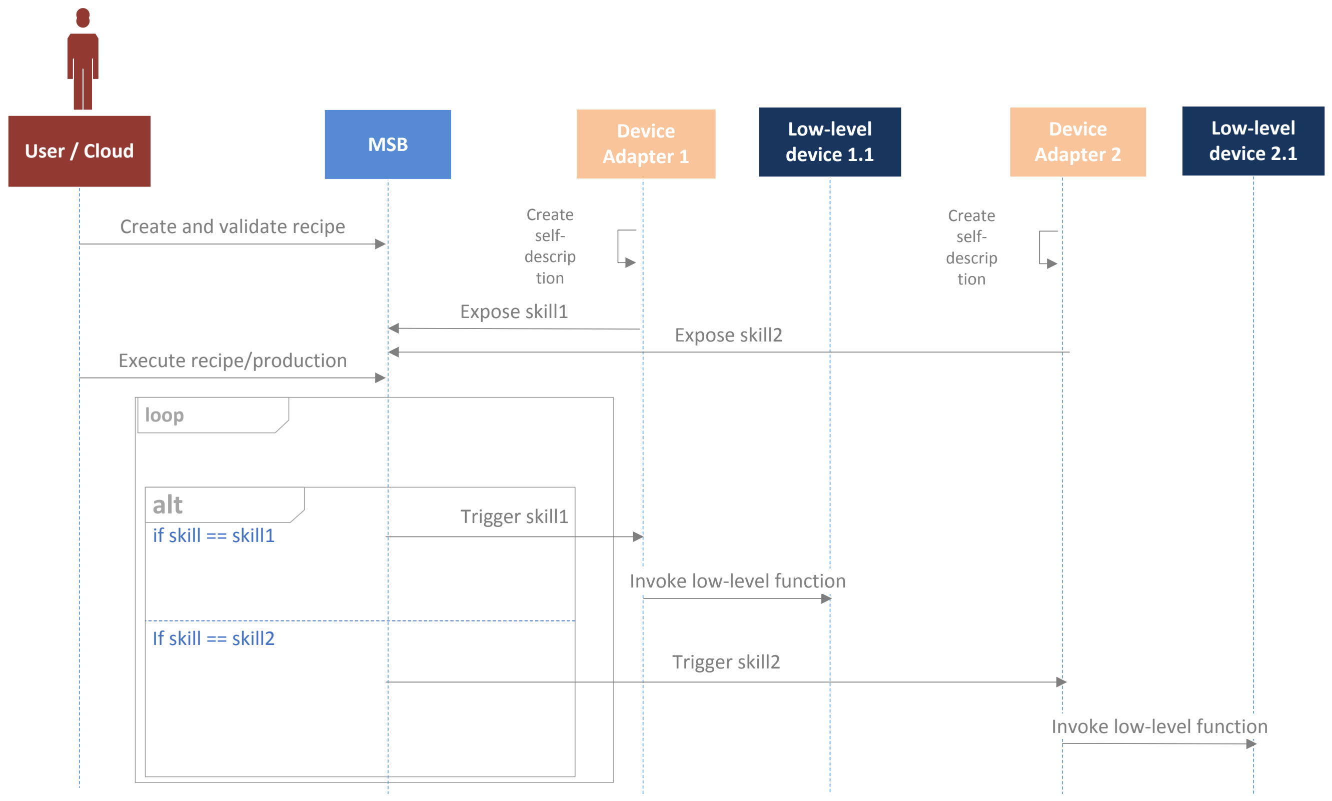

Fig. 4. Sequence Diagram

as it defines the parameters needed for the skill execution. Thus, after generating the definition of an OPC Server out of AutomationML model, there is a ready interface for triggering the production from the MSB. Moreover, a workstation can have tasks, i.e. the sequences of particular skills. These are defined in AutomationML and will be device adapter server methods which can be called to execute multiple skills at a time.

Figure 5 describes the implementation of the proposed device adapter concept realized in the demonstration example. (1) The device adapter for the workstation generates the description of an OPC UA server out of workstation description AutomationML file. This server contains OPC UA server method nodes that represent the skills of the workstation devices. (2) The OPC UA server discovers the MSB and registers there. The MSB is then able to create an OPC UA client that communicates back to the device adapter OPC UA server and triggers the production by executing the production recipe. (3) The device adapter instantiates the OPC UA clients to communicate with different system components and translates the skill call to the corresponding low-level command.

In the current evaluation activities carried out for deploying the device adapter, we used physical legacy equipment, where the current industrial standards were met. Such legacy systems are still designed to operate under very strict and static procedures, with fixed topology and predictive execution timing. This means that low-level data and protocols could only be changed to some extent to facilitate the integration of Plug&Produce concept and unified semantic model. The result is an additional interfacing wrapper for the device adapter that, currently, still needs manual mapping of server function calls to client functions to be called in low level devices.

## VII. Future Work

While introducing skills to the existing system, the programs that execute the production tasks in our demonstrator had to be rewritten to realize each skill as a standalone subprogram. Each of those subprograms has an interface consisting of multiple data inputs and outputs to interact with the device adapter. Subsequently, the system loses in the performance compared to the initial control program, where the control was seen as a one complete program. On the other hand, the agility of the system increases. As one of the further steps we want to assess how much we lose in the performance compared to the dedicated system.

One other aspect to explore would be a generation of the device subprograms out of the recipes, defined in the AutomationML file, during the Configuration and Reconfiguration Phases without the need for manual programming.

We will also investigate how the proposed approach deal with heterogeneous communication interfaces being used on the device level and explore more about the Plug&Produce concept on the communication layer. We could show how

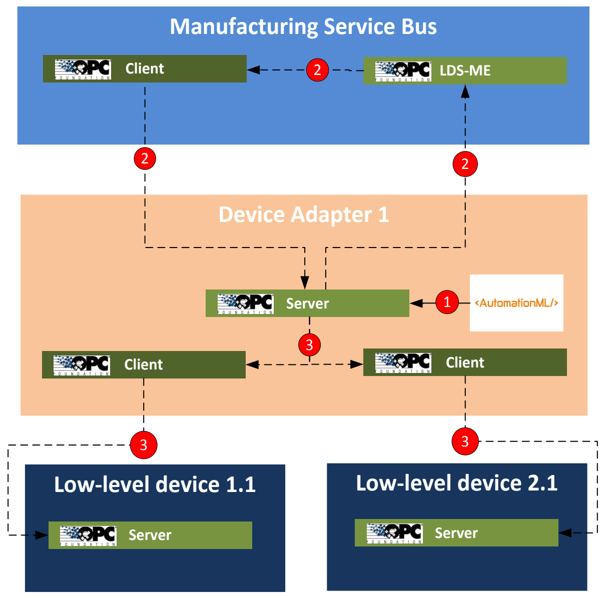

Fig. 5. Implementing the Proposed Concept in Demonstrator Example

heterogeneous platforms can be seamlessly integrated into a system and configured for use in the factory.

## VIII. Conclusion

We propose the semantic protocol of realizing the administration shell concept that can be reimplemented in multiple ways using different technologies. The universal concept of the device adapter can be then created and extended for different setups.

Creation of an OPC UA information model out of existing AutomationML data allows to reduce configuration effort and enables re-usability of information over the time. This also simplifies the re-engineering and maintenance process. Once the AutomationML description is changed, the device adapter changes its OPC UA server automatically. Overall it leads to more agile, more adaptable production systems.

The concept proposed aims not only to serve as a basis for the future production lines, but also to allow the integration of the legacy systems in the future Industry 4.0. To enable the pluggability of devices, an embedded discovery service is provided to ensure that current legacy devices, such as industrial robots and PLCs, automatically register themselves in MSB after being connected to the whole system. Device adapter also serves to expose low-level device functionality as skills to the upper control layers, reducing the initial build and reconfiguration effort. Each module gets its description, which is a part of a common semantic model, allowing various modules to inter-operate in a standardized way. AutomationML is used to describe not only the device information, but also its execution workflow. This makes the reconfiguration as easy as deploying a new recipe into the system.

On the other hand, the proposed approach means bigger initial effort when executing the system, requiring a definition of the devices. However, our approach will allow to decrease the overall operational time during the system life cycle due to re-usability and fast change-over process.

## Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680735, project openMOS (Open Dynamic Manufacturing Operating System for Smart Plug-and-Produce Automation Components).

## References

[1] Plattform Industrie 4.0 *Interaction Model for Industrie 4.0 Components*. [Online]. Available: http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/interaction-model-I40-components.pdf?__blob=publicationFile&v=8

[2] *Implementation Strategy Industrie 4.0 Report on the results of the Industrie 4.0 Platform*. 2016. [Online]. Available: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/Implementation_Strategy_Industrie_4.0_-_Report_on_the_results_of_Industrie_4.0_Platform/Implementation-Strategy-Industrie-40-ENG.pdf

[3] AutomationML consortium. *Whitepaper AutomationML Part 1 - Architecture and general requirements*. 2014.

[4] P. Ferreira, and N. Lohse. *Configuration model for evolvable assembly systems*. 4th CIRP Conference On Assembly Technologies And Systems, 2012.

[5] M. Onori, N. Lohse, J. Barata, and C. Hanisch *The IDEAS project: Plug & produce at shop-floor level.* Assembly Automation 32(2), pp. 124-134, 2012.

[6] J. Pfrommer, M. Schleipen, and J. Beyerer. *PPRS: production skills and their relation to product, process, and resource*. In IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-4, 2013.

[7] M. Schleipen, A. Lder, O. Sauer, H. Flatt, and J. Jasperneite *Requirements and concept for Plug-and-Work* Automatisierungstechnik 63(10), pp.801-820, 2015.

[8] *Companion Specification AutomationML for OPC UA*. 2016. [Online]. Available: https://opcfoundation.org/news/opc-foundation-news/bridging-the-gap-between-communication-and-semantics-for-industrie-4-0-companion-specification-automationml-for-opc-ua

[9] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota. *Agile Assembly System by Plug and Produce*. CIRP Annals Manufacturing Technology, vol. 49, pp. 1-4, 2000.

[10] Z.M. Bi, L. Wang, and S.Y.T. Lang. *Current status of reconfigurable assembly systems*. International Journal of Manufacturing Research 2(3): p. 303 - 328, 2007.

[11] P. Ferreira. *An Agent-based Self-Configuration Methodology for Modular Assembly Systems*. In Department of Mechanical, Materials and Manufacturing Engineering, The University of Nottingham, 2011.

[12] B. Brandenbourger, M. Vathoopan, and A. Zoitl. *Engineering of Automation Systems using Metamodel implemented in AutomationML*. In International Conference on Industrial Informatics (INDIN), pp.363-370, 2016.

[13] R. Henssen, and M. Schleipen. *Interoperability between OPC UA and AutomationML*. 8th International Conference on Digital Enterprise Technology DET 2014 Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution, pp. 297-304, 2014.

[14] A. Frejborg, M. Ojala, L. Haapanen, O. Palonen, and J. Aro *OPC UA Connects your Systems - Top 10 reasons why to choose OPC UA over OPC*. Finnish Society of Automation, Biannual seminar no. XX, 2013.

[15] C.-H. Cheng, T. Guelfirat, C. Messinger, J. Schmitt, M. Schnelte, and P. Weber. *Semantic Degrees for Industrie 4.0*. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 1010-1013, 2015.

[16] S. Profanter, K. Dorofeev, and A. Zoitl. *Device Adapter Concept towards Enabling Plug&Produce Production Environments*. In Emerging Technologies And Factory Automation (ETFA), Limassol, 2017.

[17] C. Morariu, O. Morariu, T. Borangiu, and S. Raileanu. *Manufacturing Service Bus Integration Model for Implementing Highly Flexible and Scalable Manufacturing Systems*. Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing, 2012.