Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 79 (2019) 596–601

www.elsevier.com/locate/procedia

12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy

Implementation of the MIALinx integration concept for future manufacturing environments to enable retrofitting of machines

Dominik Lucke[a, c, *], Peter Einberger[a], Daniel Schel[a], Michael Luckert[a], Matthias Schneider[a], Emir Cuk[a], Thomas Bauernhansl[a], Matthias Wieland[b], Frank Steimle[b], Bernhard Mitschang[b]

[a]*Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Nobelstraße 12, 70569 Stuttgart, Germany,*
[b]*University of Stuttgart, Institute for Parallel and Distributed Systems (IPVS), Universitätsstraße 38, 70569 Stuttgart, Germany*
[c]*Hochschule Reutlingen, ESB Business School, Alteburgstr. 150, 72762 Reutlingen, Germany*

* Corresponding author. Tel.: +49-711-970-1897; fax: +49-711-970-3606. *E-mail address:* dominik.lucke@ipa.fraunhofer.de

Abstract

Manufacturing has to adapt to changing situations in order to stay competitive. It demands a flexible and easy-to-use integration of production equipment and ICT systems. The contribution of this paper is the presentation of the implementation of the Manufacturing Integration Assistant (MIALinx). The integration steps range from integrating sensors over collecting and rule-based processing of sensor information to the execution of required actions. Furthermore, we describe the implementation of MIALinx by commissioning it in a manufacturing environment to retrofit legacy machines for Industrie 4.0. Finally, we validate the suitability of our approach by applying our solution in a medium-size company.

© 2019 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering.

Keywords: Manufacturing, Smart factory, Industrie 4.0, Manufacturing service bus, Rules, Integration, MIALinx

1. Introduction

The trends of the market towards personalized products and shorter product life cycles pressures a company's manufacturing to continuously change and adapt. To stay competitive, these changes have to be executed in a highly effective manner. Therefore, one key element is the availability of current information on the different manufacturing levels starting from technical processes, single machine components, the shop floor and the associated business processes. Based on the gathered information, production processes can be controlled and optimized; in a first step by humans, while later on, machines can cooperatively organize themselves over the complete life cycle of products, factories and processes. This vision is enabled by massively applying information and communication technologies (ICT) and will ultimately change the way that products will be produced in the future. Therefore, legacy as well as modern, state of the art machines need to be integrated into the future production environment. To achieve this goal, retrofitting and integrating legacy machines and equipment are substantial. In this context, retrofitting focuses on the process of enhancing existing machinery to provide current information about its status.

While there are several approaches for how to extract different information from heterogeneous sources like sensors, machine controls or enterprise resource planning (ERP) systems, effectively integrating and utilizing this information remains a critical challenge. To address this issue, we previously presented the concept of the Manufacturing Integration Assistant (MIALinx) [1], [2], a lightweight and easy-to-use information and communication technology integration solution. It supports rule-based self-organization for manufacturing processes by modeling simple, reusable "IF-THEN" rules that link events in a manufacturing environment with corresponding actions. It is crucial to MIALinx that even persons without programming knowledge are able to program rules linking sensors and actuators flexibly to their current needs. Examples for sensors are electrical current and voltage, temperatures, but also database entries or production orders. Examples for actuators are linear axis, signal lights, emails or a

2212-8271 © 2019 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering.
10.1016/j.procir.2019.02.084

maintenance demand message generator for computerized maintenance management systems (CMMS) or ERP systems.

In this paper, we present the implemented architecture of MIALinx and describe the application in a manufacturing environment, where MIALinx is used to retrofit legacy machines and integrate them in a modern Industrie 4.0 environment.

2. Related work

Currently, most factories have a very heterogeneous ICT landscape. As an example, ERP systems, manufacturing execution systems (MES) or machines from different vendors are used and must be integrated e.g. using point-to-point bridges [3]. This approach is hard to maintain and is not supporting the visual modeling of integration rules by domain experts. Recently, approaches for standardizing and simplifying the integration are built based on the service-oriented architecture (SOA) paradigm. Enterprise service buses (ESB) [4] or specialized manufacturing service buses (MSB) [5] are offering such functionality. In our approach we build on top of the Virtual Fort Knox (VFK) [6] – which hosts an MSB – for executing rules and integrating sensors, actors and services.

For the Internet of Things (IoT), a huge number of concepts and tools for sensor modelling and integration have been proposed: SensorML [7] for an example is an extensible XML-based language for modelling sensors and their characteristics. Furthermore, there are approaches like SitOPT that allow modelling of aggregated sensor events by executing complex events using so-called Situation Templates [8]. In related work sensor platforms like OpenMTC [9], OpenIoT [10] or FIWARE [11] were introduced for the management of sensor information or sensor networks [12], [13]. These approaches are very generic and not easy to integrate with legacy machines and hardware in manufacturing environments. In MIALinx, sensor information is modelled and accessed using so-called smart services which were presented in earlier work [2]. For further improvement, we now present an approach on how to retrofit legacy machines with sensors and show how to implement specialized adapters to gain status information of legacy machines.

After the integration of the sensors, the information has to be processed. For a rule-based processing, two areas are relevant: 1) modeling of rules and 2) rule-execution. Paschke and Kozlenkov [14] provide an overview of existing types of executable rules. For the execution of rules many rule engines like Jess [15] or Drools [16] are available and support complex rules. However, they do not provide a visual modeling of the rules. The project UC4 Decision System [17] has a similar goal as MIALinx and also provides a web-based modeling approach. However, the UC4 system focuses more on business rules. For modeling and execution of rules web automation platforms like IFTTT [18], or Zapier [19] are existing. They allow users the integration of different IT systems without any programming skills. However, they are not adapted to the use for the integration of machines, sensors, or actuators as needed in manufacturing environments.

3. MIALinx Concept

In future manufacturing environments, the capability to adapt to changing requirements will be vital to stay competitive. One approach to achieve this is the massive application of sensors and ICT. The fundamental idea of this approach is a flexible integration of sensor-equipped assets (machines, tools, but also ICT systems) on the shop floor with ICT systems. This integration could be realized by workflow technology, e.g. BPEL. However, a workflow-based approach has a drawback: It requires extensive knowledge of manufacturing workers on the one hand and skilled programmers that are able to adjust the existing solution on the other. This means, once workers identify the need to change the environment, they depend on the IT department to implement this change. In most companies, this process is time-consuming and slow.

To overcome this drawback, we introduced the MIALinx concept in previous work [1]. Our approach splits the integration effort in two tasks. The first task is making the assets available in the MIALinx environment. That is, the programmer is implementing a smart service. A smart service is an administration shell, which implements the access to get the information from the source (sensor or ICT system) or a logic operation between different sources. This should be executed when the smart service gets a specific command from the MIALinx environment. When all assets were made available to the environment, the second task for the workers is to define the current requirements to the environment [2].

Workers can specify these requirements by defining simple and reusable “IF-THEN” rules. Such a rule links events that occur in manufacturing environments to corresponding actions. The main advantage of this approach is that workers can easily create new rules or alter existing rules without any deep programming knowledge. A rule consists of four integral parts:

- The sensor: The sensor information is the input of a rule. It describes which asset of the manufacturing environment can trigger the rule. This not necessarily needs to be a physical sensor, such as temperature or vibration sensor. It is also possible that the triggering asset is an ICT system, such as a stock level or a production order status in an ERP system. Basically, everything that produces information in a manufacturing environment can be used as a sensor.

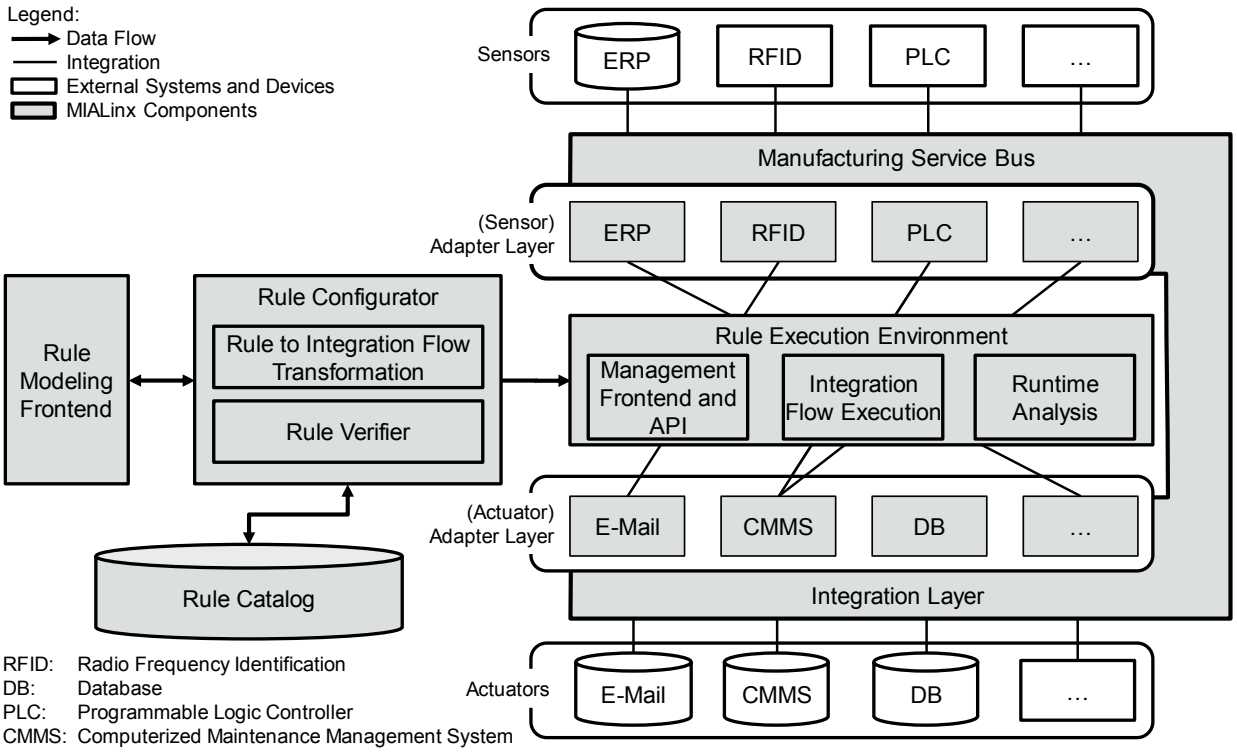


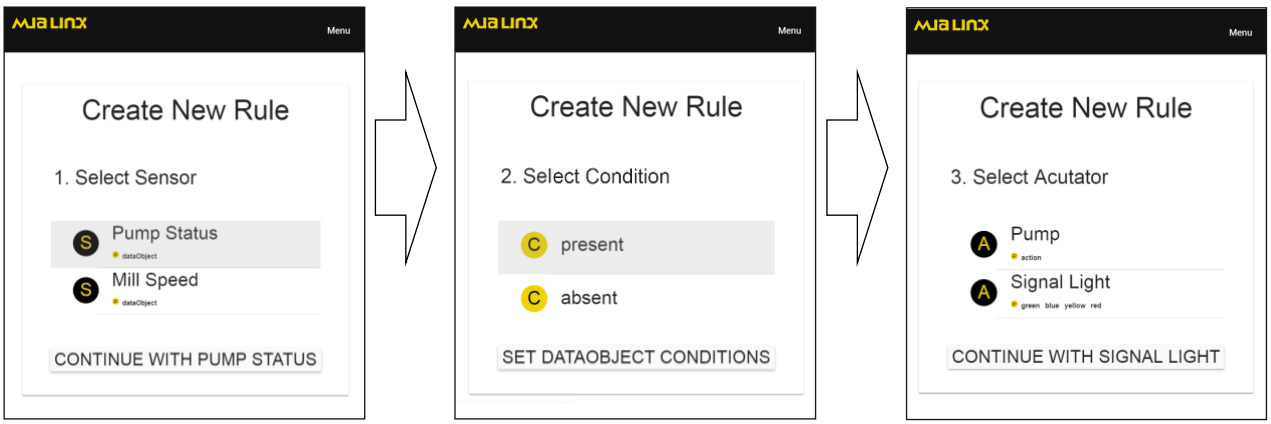
Fig. 1. Extended architecture of MIALinx based on [1]

- The actuator: The actuator is the output of a rule. It describes which asset is activated if the rule is executed. An actuator can be everything that can consume information. It could be a machine, parts of a machine like a signal light, a linear axis, or an ICT system like an ERP system.
- A set of conditions: This set describes the circumstances under which the rule is executed. A single condition is a triple consisting of a parameter name, a comparator, and a comparison value. An example for a condition would be (contamination, >, 25%). When a certain sensor publishes new event information to the MIALinx environment, MIALinx will first look for rules, which use this sensor and trigger them. Triggered rules will evaluate their set of conditions. If there is a set of conditions, whose members all evaluate to true, this rule will be executed. Our model ensures that all parameter names used in conditions can be matched to values of the sensor.
- An action configuration: The action configuration describes the action that should be executed by the actuator when the rule is executed. The configuration consists of key value pairs that are sent to the involved actuator.

These four parts can be combined to a rule saying: "If the involved sensor sends new information and all conditions are met, then send the action configuration to the given actuator".

The overall architecture of MIALinx is shown in Figure 1. Our architecture consists of the following components: The rule modeling frontend, the rule configurator, the rule catalog, the rule execution environment, and the manufacturing service bus, which acts as the integration layer for all sensors, actuators and services. The rule configurator provides functions to create and edit rules. Each rule is saved to the rule catalog. Once the user decides to deploy a rule it has to be transformed into an executable form. This is done by the rule to integration flow transformation. Another important task of the rule configurator is detecting rule conflicts. This is done by the rule verifier, but this is not scope of this paper. All the described functions are encapsulated into a HTTP interface that can be used by graphical user interfaces. Therefore, our web-based rule modeling tool is the presentation layer for the rule configurator. The rule execution environment takes care of evaluating the rules based on sensor data and triggering actions of actuators if a rule is evaluated to true. This component consists of three building blocks: A management API, a rule processing engine, and a runtime analysis. In our case, the management API and message processing engine components are implemented with a manufacturing service bus MSB [20]. This MSB also simplifies with its integration layer the integration of different sensors and actuators, which implementation is detailed in chapter 5.

To make sure our rules can be created or altered by everybody, we have implemented a mobile-first, intuitive, and easy-to-use graphical user interface (GUI). Our GUI guides the user through the process of rule creation (Figure 2). When a user wants to create a new rule, s/he starts by picking a sensor from the list (1). In future work, we plan to implement a further text field to search for a certain sensor. Furthermore, we plan to use context information like position to place currently recommended sensors on top of the list. After s/he has chosen a sensor, it is possible to select a parameter (2) of the sensor to define a condition (3). As soon as at least one condition is created, there are two options to select an actuator. The first one



Step 1: Select and configure sensor | Step 2: Set rule conditions | Step 3: Select actuator and configure action

Fig. 2. Creation of a MIALinx rule screenshots of the user interface

is similar to the sensor selection step: The user can select an actuator from the list of all available actuators. The second option is selecting an actuator from the recommendation list. Each sensor has its own recommendation list, where recommended actuators respectively actuators which are often used with the selected sensor are listed. When an actuator is selected, the user needs to decide which action should be executed, by specifying an action configuration (a set of key value pairs) which should be sent to the actuator. Then the created rule can be saved and executed.

4. Use case and hardware setup

To validate our approach, MIALinx is tested in a real industrial environment in a maintenance use case. In the field of maintenance, digital tools such as computerized maintenance management systems (CMMS) are already being used. Their main tasks are maintenance planning, work execution, documentation and spare part management. Condition monitoring systems complete these systems and monitor critical components to prevent unplanned failures. As the application of condition monitoring systems requires still a quite high effort and expertise they are still used for highly critical equipment. In our use case we focus on the monitoring of air filters in electrical cabinets with low cost, add-on sensors to retrofit legacy machines. Air filters are often critical, as dust on electrical components causes unforeseen failures. In the factory where our solution is applied, most air filters of the current electrical cabinets of the machines have no condition monitoring.

Currently, for the air filters a fixed, time-based maintenance strategy is assigned. Since several hundred air filters have to be regularly controlled manually, this provides potential for optimization. Currently, regular manual maintenance is indispensable to ensure the proper operation of the air filters. The goal is to install with MIALinx a low-priced sensing and monitoring solution and have it configured by the maintenance workers themselves according to the current requirements.

Figure 3 depicts an overview of the use case. A color sensor monitors the air filter of the machine and continuously sends the information to MIALinx (1). It measures the discoloration of the air filter that can be used to deduce the degree of contamination. Additionally, the temperatures are measured inside and outside of the cabinet in order to consider further indicators. For sensor data processing and transmission to MIALinx, all sensors are connected to a low-cost microcontroller with wireless communication capabilities. In our case, we used an ESP32 microcontroller with an integrated Wi-Fi network device, which sends the sensor information to the gateway component (see section 5) forwarding the received sensor information to MIALinx.

The maintenance workers (2a) and planners (2b) can create rules for the maintenance of the air filter, based on their experience. For the rule creation, two sensors (environmental temperature and degree of air filter contamination) as well as three actuators (SMS, mail notification and ERP system adapter) can be combined. Examples of rules for maintenance workers and planners are:

- air filter maintenance demand message change rule
  - sensor: air filter
  - actuator: ERP system
  - set of conditions: contamination > 70 %
  - action configuration: create a maintenance order in the ERP system
- air filter stock rule
  - sensor: stock
  - actuator: Email
  - set of conditions: quantity of air filters in stock < 5
  - action configuration: send mail notification to the maintenance planner
- urgent air filter change rule
  - sensor: air filter
  - actuator: ERP system
  - set of conditions: contamination > 95 %

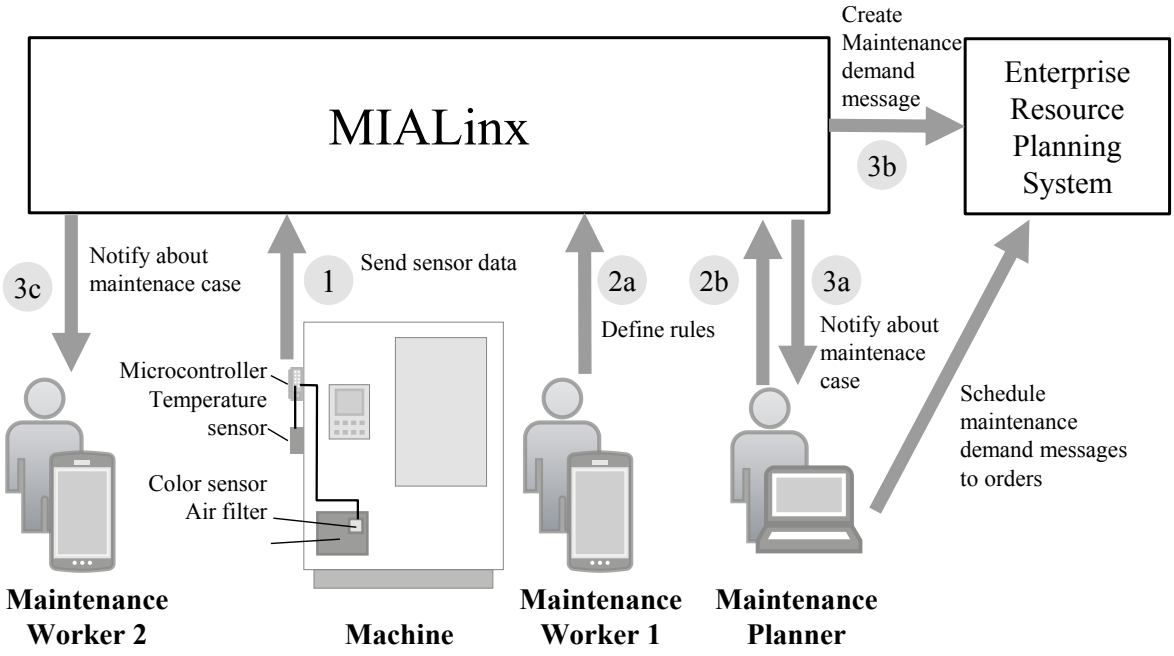


Fig. 3: Use case and hardware setup

– action configuration: mail notification to the responsible maintenance worker

The first two rules support the maintenance planner to schedule the maintenance order in time and to manage the stock of spare parts (3a, 3b). The third rule prevents overheating and unplanned shutdowns due to filter clogging, as the responsible maintenance worker is informed directly about the situation and can react on time (3c).

The rules can easily be changed, optimized or adapted to new situations by the workers based on their experience. Due to the rule creation, the knowledge of the worker is externalized and available to other workers. They can adapt the rules for other machines and benefit from the knowledge of more experienced workers.

There are several possibilities for extending the use case: On the one hand, additional environmental sensors (e.g. humidity, air quality) and actuators (e.g. signal lights) can be added to MIALinx. On the other hand, existing information can be read out from the control systems (e.g. PLCs). This enables the consideration of status messages or sensors and allows new actuators, which are directly connected to machines. This is the foundation for the already described scenario of production information acquisition [1]. In our specific use case, the collected information could be analyzed e.g. to identify patterns and negative environmental conditions or to evaluate different filter types.

5. Implementation of the architecture

For our use case the architecture of the manufacturing service bus (MSB) with its several integration interfaces, such as RESTful, WebSocket or OPC UA [20], as used in the original MIALinx design [2], has been extended with a so-called MSB gateway component. It provides a seamless connection solution between MIALinx, which is hosted in a cloud computing environment and production machines that are often integrated in factory local networks with no internet access. Apart from smart sensors, the MSB gateway also connects OPC UA capable machines to the MSB and makes their information available for MIALinx (Figure 4). Via the OPC UA client interface component in this MSB gateway, it is possible to get access to an OPC UA server of a machine in a closed local network. Further advantage of the MSB gateway concept are to reduce the outgoing connections of the factory network by bundled communication between several devices and services with the MSB. It also helps to avoid new security problems, compared to a solution that every device or machine

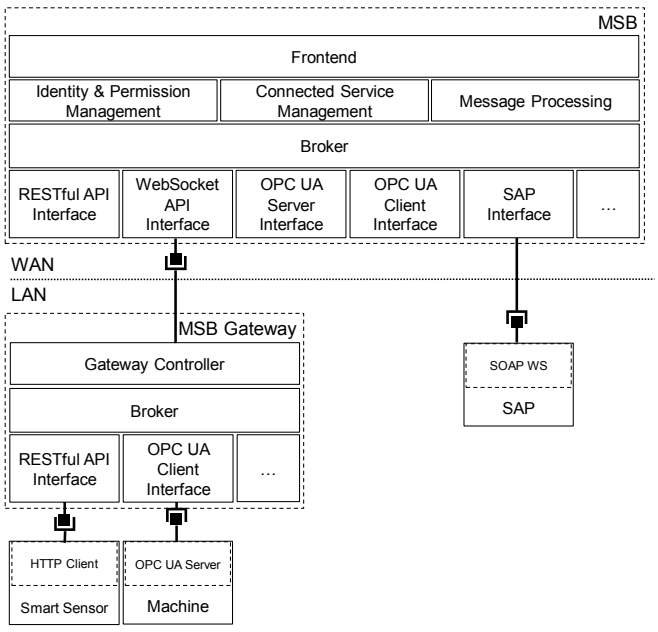


Fig. 4. Manufacturing Service Bus (MSB) architecture for MIALinx

communicate directly with the MSB. The MSB gateway is structured similar to the MSB architecture. It consists of a gateway controller, a broker and the interfaces. The gateway controller registers itself as a smart service of type gateway via the WebSocket API interface to the MSB. The gateway controller presents itself as connected service management and message processing component to the broker of the MSB gateway. This enables the reuse of the broker and existing MSB interfaces.

Using OPC UA as an example, the OPC UA client interface is able to connects to several OPC UA servers of machines. For an easy integration then the MSB frontend is used to configure the subscription of events and the functions for reading and writing variables of the OPC UA server. The advantage is that the information, the events and the functions of the configured smart services can be used by other connected MSB smart services, so a point-to-point integration of the ICT systems is obsolete.

It is also often required to integrate existing IT systems; in our use case, the integration of SAP as ERP system is required. This requires an SAP Interface as a separate MSB interface. An SAP instance must be configured so that it makes its BAPI available as a SOAP-based web service. This SAP web service is connected to the MSB via the SAP interface. It loads the WSDL file of the SOAP-based web service and analyzes it for the self-description required for registration with the MSB. At MSB level, the SAP web service methods are represented as functions and can be directly called in the code.

6. Conclusion and future work

In this paper, we show how to integrate sensors installed in a shop floor with the existing IT systems of a company. We achieved that by using a service-oriented architecture and a rule-based approach for modeling the behavior based on smart services. We enable domain experts to model individual rules in an easy to use tool. Afterwards, these rules are transformed into integration flows for the manufacturing service bus. The manufacturing service bus integrates all sensors, actors and enterprise IT systems. We apply the presented approach in a real-world usage scenario based on the needs of small and medium-sized enterprises SME to improve the production on the shop floor. The research presented in this paper lowers the barriers for SMEs to enter the field of Industrie 4.0.

As future work, we plan to deploy MIALinx in the domain of production information acquisition. Furthermore, the goal is to provide a catalog of different sets of rules for different use-cases and domains. Based on our findings, we plan continuously improve our approach. Finally, we intend to evaluate multiple business models to find the approach most suitable for SME. This will lower the effort needed by new adaptors of MIALinx; hence, they can reuse the rules and adapt them to their concrete production environment.

Acknowledgement

MIALinx is a joined research and implementation project of the Fraunhofer Institute for Manufacturing Engineering and Automation IPA and the Institute for Parallel and Distributed Systems of the University of Stuttgart. It is funded by the Baden-Württemberg Stiftung gGmbH.

References

[1] Wieland M, Hirmer P, Steimle F, Gröger C, Mitschang B, Rehder E, et al. Towards a Rule-based Manufacturing Integration Assistant. Procedia CIRP 2016; 57: 213–218.
[2] Wieland M, Steimle F, Mitschang B, Lucke D, Einberger P, Schel D, et al. Rule-Based Integration of Smart Services Using the Manufacturing Service Bus. Proc. 14th IEEE Int. Conf. Ubiquitous Intell. Comput. UIC2017, Fremont, USA: 2017, pp. 1–8.
[3] Kletti J. Manufacturing Execution Systems (MES). Berlin; London: Springer; 2007.
[4] Chappell DA. Enterprise Service Bus. 1st ed. Sebastopol, Calif: O'Reilly; 2004.
[5] Minguez J, Lucke D, Jakob M, Constantinescu C, Mitschang B. Introducing SOA into Production Environments - The Manufacturing Service Bus. Proc. 43rd CIRP Int. Conf. Manuf. Syst., Vienna, Graz, Austria: 2010, pp. 1117–1124.
[6] Holtewert P, Wutzke R, Seidelmann J, Bauernhansl T. Virtual Fort Knox Federative, Secure and Cloud-based Platform for Manufacturing. Procedia CIRP 2013;7:527–32.
[7] Open Geospatial Consortium I. Sensor Model Language (SensorML) 2007.
[8] Hirmer P, Wieland M, Schwarz H, Mitschang B, Breitenbücher U, Leymann F. SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates. In: Barzen J, Khalaf R, Leymann F, Mitschang B, editors. Proc. 9th Symp. Summer Sch. Serv.-Oriented Comput., vol. RC25564, IBM Research Report; 2015, pp. 113–127.
[9] Boosting the Development of Innovative M2M and IoT Applications n.d. http://www.openmtc.org/ (accessed December 18, 2015).
[10] Soldatos J, Kefalakis N, Hauswirth M, Serrano M, Calbimonte J-P, Riahi M, et al. OpenIoT: Open Source Internet-of-Things in the Cloud. In: Žarko IP, Pripužić K, Serrano M, editors. Interoperability Open-Source Solut. Internet Things, Springer International Publishing; 2015, pp. 13–25.
[11] FIWARE n.d. https://www.fiware.org/ (accessed April 24, 2018).
[12] Aberer K, Hauswirth M, Salehi A. A Middleware for Fast and Flexible Sensor Network Deployment. Proc. 32Nd Int. Conf. Very Large Data Bases, Seoul, Korea: VLDB Endowment; 2006, pp. 1199–1202.
[13] Aberer K, Hauswirth M, Salehi A. Invited Talk: Zero-Programming Sensor Network Deployment. Int. Symp. Appl. Internet Workshop 2007 St. Workshop 2007, 2007, p. 1–1. doi:10.1109/SAINT-W.2007.57.
[14] Paschke A, Kozlenkov A. Rule-Based Event Processing and Reaction Rules. In: Governatori G, Hall J, Paschke A, editors. Rule Interchange Appl., Springer Berlin Heidelberg; 2009, pp. 53–66.
[15] Jess, the Rule Engine for the Java Platform n.d. http://jessrules.com/ (accessed April 24, 2018).
[16] Drools - Business Rules Management System (Java™, Open Source) n.d. http://www.drools.org/ (accessed April 24, 2018).
[17] Obweger H, Schiefer J, Suntinger M, Kepplinger P, Rozsnyai S. User-oriented Rule Management for Event-based Applications. Proc. 5th ACM Int. Conf. Distrib. Event-Based Syst., New York, NY, USA: ACM; 2011, p. 39–48. doi:10.1145/2002259.2002266.
[18] IFTTT - Make Your Work Flow n.d. https://ifttt.com/ (accessed December 18, 2017).
[19] The best apps. Better together. - Zapier n.d. https://zapier.com/ (accessed April 24, 2018).
[20] Schel D, Henkel C, Stock D, Meyer O, Rauhoeft G, Einberger P, Söhr M, Daxer M, Seidelmann J. Manufacturing Service Bus: an Implementation. In 11th CIRP Conf. Intell. Comput. Manuf. Eng., 2017, pp. 179–184.