Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 67 (2018) 167 – 172

www.elsevier.com/locate/procedia

11th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '17

Challenges of production microservices

Benjamin Götz[a,*], Daniel Schel[a], Dennis Bauer[a], Christian Henkel[a], Peter Einberger[a], Thomas Bauernhansl[a]

[a]*Fraunhofer IPA, Nobelstrasse 12, 70569 Stuttgart, Germany*

* Corresponding author. Tel.: +49-711-970-1354 ; fax: +49-711-970-1028. *E-mail address:* benjamin.goetz@ipa.fraunhofer.de

Abstract

Current production systems use monolithic software solutions. This causes a lack of flexibility, scalability and prevents direct communication between network nodes which is fundamental to face challenges of highly personalized mass production. In order to overcome these drawbacks, the introduction of a service-oriented architecture (SOA) more specifically microservices in production are a promising approach. SOA enables developers to distribute applications in a number of small services which communicate via an integration layer e.g. an enterprise service bus. This paper proposes a data-driven approach for creating a SOA, based on microservices in an assembly focused production.

Peer-review under responsibility of the scientific committee of the 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering

Keywords: Microservice; Software; Service-oriented architecture; *Monolithic architecture*

1. Introduction

Global megatrends such as globalization, urbanization, demographic change, growth of population and sustainable development are not only influencing societies around the world, but also have great impact on manufacturing enterprises and lead to a paradigm change in all production factors. This includes revolutionary changes in energy and material consumption, staff and capital circulation as well as massive demand movements towards emerging markets and developing countries. It is expected, that by 2025 developing countries will account for half of the global consumption [1].

Thus, addressing various markets will be far more a key challenge than facing demand problems. Products for developed countries need to be highly individualized, while products for emerging markets need to be adapted to regional needs including functionality, design and costs. In addition, there is a trend to shortened innovation cycles. This leads to an increasing complexity of the markets as well as a rise of product variants while quantities per product and variant are decreasing [2].

The proposed solution for these challenges from an IT perspective is the concept of service-oriented architectures (SOA) and microservices. While a SOA addresses challenges of manufacturing enterprises, their implementation introduces new challenges. Among these challenges are the architecture design in terms of size of the microservices or their orchestration and integration. Thus, this paper presents an approach of how to overcome challenges of IT architecture design and implementation in industrial production environments to benefit from microservices.

2. Challenges in Production Systems

Information and communication technologies (ICT) will be a key enabler for the described challenges of manufacturing enterprises, where most of the innovations will take place. A propagated solution addressing rising market complexity as well as rising complexity within companies by ICT is the smart factory, the next evolutionary stage of the fractal factory. Cyber-physical systems (CPS) can build decentral and autonomous networks – like fractals – to self-organize and self-optimize. The level of autonomy and decentralization rises with increasing complexity [2,3].

To enable these developments, manufacturing IT is undergoing a fundamental change from the traditional automation pyramid of monolithic systems to service-orientation, also described as Everything-as-a-Service (XaaS). This paradigm describes that everything, no matter if physical or virtual, is offered as a service and originates from the three

Peer-review under responsibility of the scientific committee of the 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering

doi:10.1016/j.procir.2017.12.194

main cloud computing service layers Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [4]. Table 1 summarizes and the following chapters describe the ongoing changes in manufacturing IT.

Table 1 Comparison of traditional and emerging manufacturing IT

Traditional manufacturing IT	Emerging manufacturing IT
Hierarchical	Non-hierarchical networks
Centralized	Decentralized
Software suites	Services, Apps
Monolithic	Fine-grained services
License fees	Pay-per-Use
Complex integration	Open standards
Delayed data	(Near) real-time data
Roll-out within months/years	Deployment within minutes

2.1. Traditional manufacturing IT

Traditional manufacturing IT is characterized by a hierarchical structure defined in ISA-95 and often depicted as the automation pyramid [5]. The automation pyramid is divided in three levels: the operational shop floor level, the tactical manufacturing execution system (MES) level and the strategic enterprise resource planning (ERP) level on top. Various planning and control tasks are performed on each level [6].

Tools on each level of the automation pyramid are usually centralized large software suites which require a significant investment in license fees. In addition, they are often monolithic and stick to self-defined interfaces instead of using open standardized interfaces and communication protocols. Therefore, the development and maintenance of interfaces between various systems requires a high effort. With each new version of a system, all corresponding interfaces need to be updated because they are proprietary to the respective software suites. Due to this effort, a holistic vertical and especially horizontal integration is usually not realized. This lack of real-time data caused by the missing integration often requires short-term and expensive intervention to production control. Furthermore, the process to introduce new software suites is very inflexible and time-consuming, taking months to years depending on the use case specifications [2,6,7].

2.2. Emerging concept for manufacturing IT

Today, the manufacturing IT is undergoing fundamental changes enabled by technologies such as cloud computing and associated concepts. The traditional automation pyramid is dissolving and manufacturing IT is moving towards service-orientation and app-orientation [4,8].

Software suites will be divided by functionality into services and apps, decentralization offered by distributed computing approaches like edge, fog computing concepts and cloud platforms. These services and apps can be non-hierarchically orchestrated in networks, where communication between services based on open standards will become a key factor for success. This overcoming of hierarchical structures also allows for communication of real-time information [6,9].

Many manufacturing companies have noticed this shift to service-orientation and have started to build their own cloud-based platforms. Examples are the Bosch IoT Suite, GE Predix or Siemens Mindsphere. However, most of these platforms are tailored around the products and services offered by the company and lack interoperability with other platform providers. In contrast, there are platforms such as Virtual Fort Knox [10] or the Fraunhofer initiative Industrial Data Space [11] following a federative approach to enable independent software vendors to participate in the ecosystem and to prevent vendor lock-in effects.

3. Microservices

To give an introduction to the concept of microservices, we compare it in the following first to the most obvious alternative: the monolithic architecture.

3.1. Monolithic Architecture

The phrase monolithic is used to describe a software application consisting of one piece. Traditional manufacturing IT, as introduced in section 2.1, uses this typically. The architecture is designed for running solely on one computational instance. This may run multiple processes which are distributed across multiple CPUs but all share the same operating system and hardware.

If the system reaches a capacity peak, it needs to be duplicated completely. This process might be executed automatically by a continuous deployment system. The main drawback is the lack of flexibility. For example, if a number of users is reached that cannot be handled by one instance, a monolithic system lacks the required horizontal scalability. Instead, it has to be scaled vertically.

3.2. Service-Oriented Architecture

Generally, a microservice architecture is a SOA, utilized as introduced in section 2.2. A service in a SOA is a software component delivering one predefined functionality matching one business activity and its specific results. The service is self-contained which means that it does not rely on external resources. All processing required by this service is performed in itself. It includes all required resources like databases etc. To consumers using the service it appears as a black box to be accessed only via predefined interfaces. It may itself require underlying services providing a certain sub-functionality [12]. A service which relies on a set of services is also called aggregated service [13].

3.3. Microservice Architecture

Microservices refer to a new software architecture. We are aiming to present an overview of the properties of this architecture. The core concept is a fine-granular decomposition of an application into such microservices. While SOA refers to the general idea of encapsulating functionalities into separate services, microservices additionally specify the scale of this functionality as small [14].

3.4. Concept

The services are organized around business capabilities. This allows the development around key capabilities of the system instead of predefined components. The service instances are independent from each other. This applies to the full life-cycle, including development, deployment and maintenance. Microservices rely on loosely coupled, lightweight communication protocols. This means that services do not communicate directly but rather use independently defined interfaces. This reduces the dependencies between services and requires an enterprise service bus (ESB) as an integration layer. Decentralized data storage means that instead of relying on one central database, a microservice architecture splits data storage across service instances. The same applies to other resources such as computational performance [15].

3.5. Benefits

Systems based on a microservice architecture can adapt to rising capacity demands more easily. This property is commonly referred to as horizontal scalability. This is especially visible, if compared to previously mentioned monolithic architectures. In a microservice architecture every component can be duplicated for load balancing as required. This makes the reaction to demand peaks faster and can handle the demands more precisely. Furthermore, this modularity also makes the system robust to faults. If one microservice encounters an error, the rest of the system can still function independently. Using automated deployment, the faulty service may be replaced automatically. The replacement can also happen pre-emptively by keeping backup-instances of critical services running. Because the services are independent, a flexibility in technology stack and programming languages is introduced. Any microservice can be implemented in the language and use the libraries that fit best to provide its functionality.

4. Proposed Solution

4.1. Approach

The proposed solution for designing microservices in production is based on a data-driven approach. This approach needs a clear picture of the data structure as well as the business processes applied in the company. Therefore, it is recommended to identify all relevant entities, their attributes and relationships as well as to visualize them. The process is similar to creating database designs by means of Entity Relationship (ER) modeling [16] (see Fig. 1). Based on that

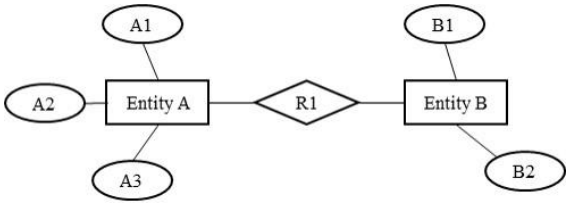

Fig. 1. Entity relationship (ER) model of entity A and entity B connected via relation R1 and their attributes visualized by ovals.

overall structure, logical units have to be created which will be managed by one microservice. The size of those logical units depends on the need of flexibility and robustness of the overall system as well as the frequency of data exchanged between those units.

The frequency of data exchange between logical units is an important key figure indicating whether two units should be combined to one microservice. The reason for that is justified by the use of an ESB for communication between all services (see Fig. 2). In case services have a small size, the load of the communication channel will rise unnecessarily because data has to be sent between services rather than within one service. Additionally, the effort for creating the routes between the services will rise. In order to estimate the frequency of data exchange between services, it is important not only to be aware of the data structure but also the behavior of the system. The business processes implemented in the company mainly influence the behavior from which the data exchange rate of services may be derived. Consequently, choosing the right size of microservices is an optimization problem with flexibility, robustness and resource consumption as criteria.

4.2. Data structure of microservices

After agreeing on a first draft of services, the data structure of each service has to be derived from the overall data structure. In order to retain relationships that exist between entities of different services, measures of identifying entities across service borders have to be introduced. A basic identification measure is using unique keys of entities. This keeps the level of redundancy low and minimizes the effort for keeping data up to date.

For example, a “Service A” that stores all data about a class of entities has to provide an application programming interface (API) for accessing entities’ data from other services. In order to address one specific entity, a “Service B” has to send a request to “Service A” containing the unique key of the requested entity. As a result, “Service A” provides all data about the requested entity in a predefined manner to “Service B”. By doing so, “Service B” will always get the latest dataset of an entity without storing it by itself.

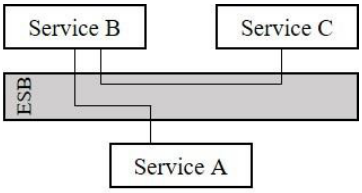

Fig. 2. Three services connected by routes via an enterprise service bus symbolized by the grey box in the middle of the figure.

The disadvantage of this approach is that each request will strain the integration layer which transports data from one service to another. In order to reduce the communicational load, a subset of data may be stored in addition to or instead of the unique key that is stored in “Service B”. This leads to increased redundancy and the data integrity is no longer guaranteed. Thus, this method is only recommended if data does not have to be up to date for each request or measures for updating redundant data are introduced among related services. Furthermore, the raised level of redundancy increases the amount of required storage. Finally, it depends on the use case which approach should be selected and whether the depicted disadvantages are relevant for the implementation.

4.3. Business processes and routing

Based on the overall data structure, a sub-structure for each service and a basic set of APIs is created. In order to realize business processes based on a set of microservices, the business processes have to be analyzed regarding temporal data demands and availability. That is, which data is available at the beginning of the process and which data is needed to fulfill the process as well as where the data is obtained. Starting from the machine or application at the starting point of the business process, each of the required services can be identified and the sequence in which the services are addressed. Based on the resulting sequence, routes between all the services have to be created by using the ESB as communication channel.

In the proposed solution the shortest route between all services and the route preserving the highest level of independency are investigated. In case of the shortest route, the service requesting a set of data (further referred to as “Service A”) which is distributed on many data source services (further referred to as “Service B” and “Service C”) sends a request to Service B which provides the basis of the overall data set. Because of the microservice structure and the distribution of data to many services, Service B is not able to fulfill the request entirely. Therefore, Service B has to forward the result of the first request to Service C which is able to complete the data set and to send the result back to Service A (see Fig. 3a). By doing so, the service requesting data is at the start as well at the end of the route. As a consequence, the route is short and therefore less demanding on the ESB but creates a high level of dependency among all participating services. The dependency level is raised because each service has to be aware of the data structure of the other services which means that each change of the data structure of one service causes a change of the structure of the other services, too.

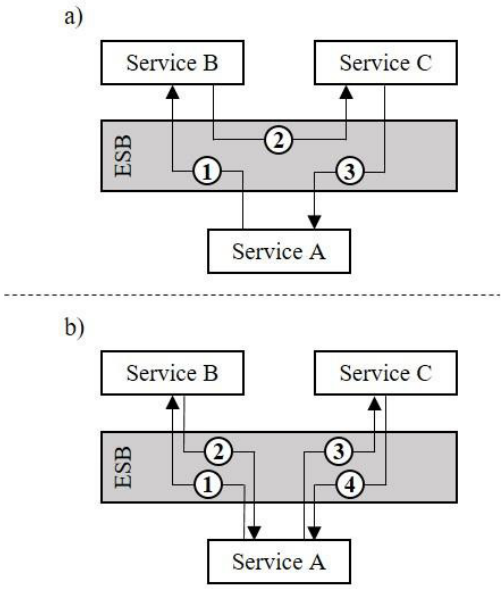

Fig.3. a) Repesents the shortest route of a request starting at service A and collecting data from service B to C and sending it back to service A. b) Represents a routing preserving the independencey of the services by only connecting services which are aware of each others data model. The drawback of this approach is a raised load on the communication channel (grey box in the middle of the figure).

In order to avoid such a high level of dependency among the services, a routing which is formed as a loop and starts at the requesting Service A and enriches the data set on the fly from Service B and Service C back to Service A is not recommended. Instead, a direct routing between the service sending the request (Service A) and the data sources (Service B and C) is proposed (see Fig. 3b). By using this approach, only Service A is aware of the overall data structure, while Service B and Service C only have to be aware of the data structure they were assigned to (see section 4.2). The drawback of this routing method is a raised load on the ESB because more requests are necessary to fulfill the request of Service A.

Regardless of whether we chose the shortest route or the route which avoids a high level of dependency among all services, we are now able to map business processes to the microservice structure and create routes to fulfill those business processes.

5. Implementation / Example

The implementation of the proposed solution has been carried out for an isolated assembly station at which a worker builds sub-assemblies manually and feeds a station of a main assembly line with those sub-assemblies. Furthermore, the assembly station has a display to show the worker all necessary information for the current order. The information is intended to support the worker in correctly performing the assembly.

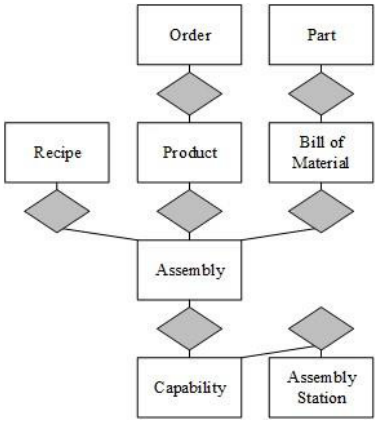

Fig.4. ER Model describing the data structure of the underlying use case.

In the first step the station receives an order number which identifies a specific order. Based on this order number all information necessary for the assembly has to be collected to support the worker, starting with getting the product which has to be produced. Each product consists of many assemblies which are put together to produce the final product. To get the right assembly according to the assembly station each assembly station has capabilities that describe the assemblies they are able to produce. Furthermore, a recipe is linked to each assembly which describes necessary steps to perform the assembly. By knowing the recipe, the capabilities as well as the list of necessary assemblies, it is possible to identify the assembly which has to be produced at the station. To finally get the part information necessary to support the worker at the station, the bill of material that is attached to the assembly will be identified and based on that the part information will be requested. The ER model describing all entities and their relations is shown in figure 4.

Based on the data structure as well as the frequency of requests that will take place during a working day, the logical units/services for assembly, product, recipe and capability have been created (see also Fig. 5).

Furthermore, Fig 5 also shows the implementation of the relations as routes between entities which have to be managed among the services. For this example, the manufacturing service bus (MSB) is used to implement those routes. The idea of the MSB is that all services provide a set of events and functions and that an event of one service may be connected with a function of another service through so-called integration flows. The integration flows do not only trigger the function of the service if the connected event takes place but also enables the services to exchange data [17].

The assembly service handles parts, bill of materials as well as assemblies. The product service which is responsible for the products and the orders is connected to the assembly service and the recipe service as shown in the ER model. The recipe service just manages the recipes which describe how assemblies have to be produced. Finally, the capability service, which saves and provides the capabilities of working stations, introduces the link between the real assembly station as well as the assembly service.

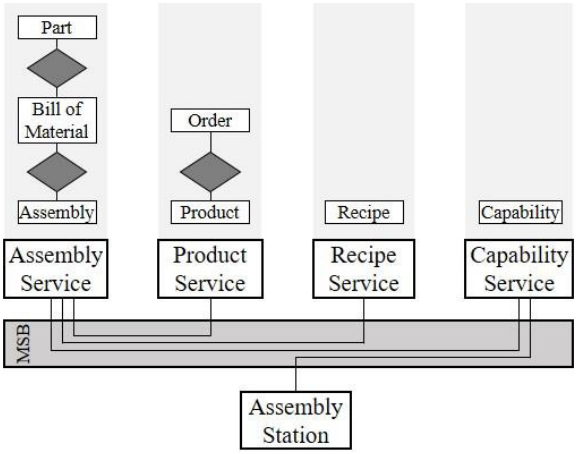

Fig. 5. Services derived from overall data structure and their data model as well as the links through the manufacturing service bus according to the overall data structure.

In order to fulfill the business process described at the beginning of this section, the assembly station first requests the capability service's capabilities. Therefore, each workstation owns a unique identification number which is sent to the service for that request. As a result, the assembly station receives a list of assembly IDs that the station is able to produce. Next, the worker scans the barcode that represents the order ID of the next order to produce. This order ID is sent to the product service which returns the product ID of the respective order. To know which assemblies the product consists of, the assembly station sends the product ID as well as the list of assemblies, it is able to produce, to the assembly service. The assembly service is now able to select all assemblies which the product consists of in its database and filters this query by the list of assemblies the station is able to produce. As a result, the assembly service sends back the assembly ID which has to be produced for the current order and the bill of material containing all information about the parts necessary. In the final step, the assembly station forwards the assembly ID to the recipe service which looks up all working steps for the assembly work and sends it back to the station. Now the worker has all information that is necessary to perform the assembly work. After finishing the assembly, the barcode of the next order is scanned and the process starts again.

Fig. 6 shows the alternative approach as described in section 4.3, where the sequence of all requests between the assembly station and the services. The requests have a star topology with the assembly station in the center. This provides a low level of dependency between all the services. In comparison to the routes shown in there is no direct connection between two services which would cause a mixing of the different data structures of each service. Consequently, each service may be replaceable if it provides the same interface to the assembly station as the previous one.

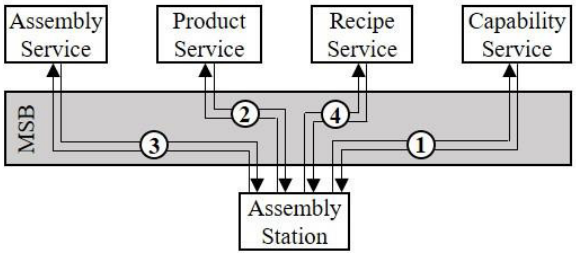

Fig. 1. Sequence of calls based on direct connections between assembly station and services. In order to gather all data based on the order ID as well as the ID of the assembly station the requests from ① to ④ have to be done.

6. Conclusion

The current megatrends (e.g. globalization, demographic change, growth of population) force manufacturing companies to accept paradigm changes in production in order to fulfill new demands. One of those paradigm changes is abandoning monolithic software systems in favor of SOA and the concept of XaaS that support companies to create production systems with an enhanced scalability, robustness and flexibility. A specific concept of a SOA is designing services as microservices in order to not only encapsulate functionalities but also keep the size of the services small. Microservices in particular improve the horizontal scalability and modularity/robustness of the overall system.

In order to get a size that fits best for services, a data-driven design approach is introduced that starts with the creation of an overall data structure and modelling of relevant business processes. In the next step, the overall data structure is divided in logical units which are assigned to a microservice. The size of the units depends on the demand of flexibility and robustness as well as the level of resource consumption of single services and the distribution of the power load to all participants. For example, if the average size of microservices is small, the load of the communication channel that links all services will be higher in order to perform a business process because more data has to be exchanged between services. In comparison, if the services are bigger there will be less communication but more load on the single services, because each service has to perform more tasks. Not only the size of services but also the routing between the services has an impact on the load of all components. Two different routing approaches have been investigated: a) Shortest route that reduces the load of the communication channel and b) Route preserving highest level of independency for each service by creating a star-shaped routing between the service requesting data in the middle and the services providing data.

An implementation example has been shown on the basis of an assembly station operated by a worker. The worker receives an order ID at the beginning and based on this ID all necessary data are collected to support the worker. Finally, this use case is an experimentation site to detect the business impact of the approach. The results of the experiments are being further examined with regards to exchangeability of services and hardware, usability in general as well as business impact.

Acknowledgements

The findings presented in this paper result from work of Fraunhofer IPA in the Project BEinCPPS (GA No. 680633). It is funded by the European Union's Horizon 2020 research and innovation programme and is part of the I4MS initiative. The authors are responsible for the content of this publication.

References

[1] Bauernhansl T. Zukünftige Rahmenbedingungen und Entwicklungstrends in der Produktionstechnik. Galvanotechnik 2013;104:2185–2191.
[2] Bauernhansl T., ten Hompel M., Vogel-Heuser B. Industrie 4.0 in Produktion, Automatisierung und Logistik. 2014. Wiesbaden: Springer Vieweg; 2014.
[3] ten Hompel M. Software in der Logistik: Prozesse steuern mit Apps. 1. München: Huss. 2013.
[4] Bauer D., Stock D., Bauernhansl T. Movement towards service-orientation and app-orientation in manufacturing IT. 10th CIRP Conf Intell Comput Manuf Eng 2016.
[5] ISA-The Instrumentation, Systems, and Automation Society. ISA-95 Manufacturing Execution Systems Standards. Research Triangle Park, NC, USA: ISA-The Instrumentation, Systems, and Automation Society; 2005.
[6] Bauernhansl T. Industrie 4.0 - Opportunities for new IT Architectures. Stuttgart: AZ SAP Summit 2015;
[7] Spath D., Ganschar O., Gerlach S, Hämmerle M., Krause T., Schlund S. Produktionsarbeit der Zukunft – Industrie 4.0. Stuttgart: Fraunhofer Verlag; 2013.
[8] Yale Y., Silveira H, Sundaram M. A microservice based reference architecture model in the context of enterprise architecture. IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) 2016; 2016: 1856-1860.
[9] Xu X. From cloud computing to cloud manufacturing. Robotics and Computer-Integrated Manufacturing 2012; 28: 75–86.
[10] 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.
[11] Otto B., Auer S., Cirullies J., Jürjens J., Menz N., Schon J. Industrial data space. 1. München: Fraunhofer Verlage; 2016.
[12] The Open Group. SOA Source Book. Zaltbommel: Van Haren Publishing; 2011.
[13] Erl T. Service-oriented architecture : concepts, technology, and design. 1. New Jersey:Prentice Hall; 2005.
[14] Mazzara M., Meyer B. Present and Ulterior Software Engineering. 1. Cham: Springer International Publishing; 2016.
[15] Lewis J., Fowler M. Microservices – a definition of this new architectural term. https://martinfowler.com 2014.
[16] Elmasri R., Navathe S. Fundamentals of database systems. Boston: Pearson/Addison Wesley; 2007.
[17] Schel D., Henkel C., Stock D., Seidelmann J.. Manufacturing Service Bus: an Implementation. 11th CIRP Conf. Intell. Comput. Manuf. Eng., 2017.