華中科技大學

本科生毕业设计（论文）参考文献译文本

译文出处：Benjamin Götz,Daniel Schel,Dennis Bauer,Christian Henkel,Peter Einberger,Thomas Bauernhansl. Challenges of Production Microservices[J]. Procedia CIRP,2018,67.

院　　系	计算机科学与技术学院
专业班级	计卓 1401
姓　　名	俞洋
学　　号	U201415219
指导教师	胡侃 肖威

2018 年 3 月

译文要求

一、 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。

二、 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页

期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

三、 译文不少于 5000 汉字（或 2 万印刷符）。

四、 翻译内容用五号宋体字编辑，采用 A4 号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。

五、 译文及其相应参考文献一起装订，顺序依次为封面、译文、文献。

六、 翻译应在第七学期完成。

译文评阅

导师评语

应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。

评分：____________________（百分制）　　　　指导教师(签名)：____________________

年　　月　　日

产业微服务的挑战

1. 引言

全球化、城市化、人口结构变化、人口增长和可持续发展等全球大趋势不仅影响着世界各国的社会，而且对制造业企业产生了巨大的影响，并且导致生产要素的范式转变。这包括能源和材料消耗、员工和资本流通的革命性变化，以及新兴市场和发展中国家的大量需求变动。据预计，到 2025 年发展中国家将占全球消费量的一半[1]。

因此，应对各种市场将是比面对需求问题更关键的挑战。发达国家的产品需要高度个性化，而新兴市场的产品需要适应区域需求，包括功能、设计和成本。此外，现在还存在一种缩短创新周期的趋势。这导致了市场的日益复杂化以及产品变体的增加，同时伴随着每种产品和变体的数量在减少[2]。

面对这些挑战，从 IT 角度提出的解决方案是面向服务架构（SOA）和微服务的概念。但是 SOA 在解决制造企业的挑战的同时，也带来了新的挑战。其中的挑战是针对微服务的规、编排和集成的相关体系结构的设计。因此，本文提出了一种方法，该方法描述了应该如何克服工业生产环境中 IT 架构设计和实施的挑战，以使其受益于微服务。

2. 生产系统中的挑战

大部分创新将在其中发生的信息和通信技术（ICT）领域将是制造企业所面临的挑战的关键推动者。智能工厂是一个由 ICT 提供的用于解决日益复杂的市场复杂度以及企业内部复杂度的普及解决方案，它是分形工厂的下一个进化阶段。网络物理系统（CPS）可以通过建立分散的和自主的网络（如分形），以实现自我组织和自我优化。自主性和分散性的水平随着复杂性的增加而增加[2,3]。

为了实现这些发展，IT 产业正在经历从传统的自动化金字塔系统到服务导向的根本性转变，后者也被描述为“一切即服务”（XAAS）。这个范例的意识是不管是物理的还是虚拟的，一切都是作为一种服务提供的，并且源于三个主要的云计算服务层：软件即服务（SaaS）、平台即服务（PAS）和基础设施即服务（IAAS）[4]。表 1 总结了这些正在进行的 IT 产业的变化，接下来的章节也对这些变化进行了描述。

表 1 传统 IT 产业与新兴 IT 产业的比较

传统 IT 产业	新兴 IT 产业
分层的	非分层网络
集中式	分散
软件套件	服务，应用
整体	细粒度服务
许可费	单次使用付费

复杂集成	开放标准
延迟的数据	（接近）实时的数据
花费几个月或几年来发布	几分钟内部署

2.1 传统 IT 产业

传统 IT 产业的特点是在 ISA95 中定义的层次结构，经常被描述为自动化金字塔[5]。自动化金字塔分为三个层次：操作车间层、战术制造执行系统（MES）层和战略企业资源计划（ERP）层。在每个级别上执行各种规划和控制任务[6]。

自动化金字塔的每个级别上的工具通常是集中式的大型软件套件，这些软件需要许可费方面的显著投资。此外，它们通常是整体的，并且坚持自定义接口，而不是使用开放标准化接口和通信协议。因此，开发和维护各种系统之间的接口需要付出很大的努力。随着每个新版本的系统，所有相应的接口需要更新，因为它们是专有的相应软件套件。由于这个原因，一个整体的垂直和水平的整合通常无法实现。缺少集成所导致的实时数据的缺乏常常要求对生产控制进行短期和昂贵的干预。此外，引入新软件套件的过程是非常不灵活和耗时的，根据用例规范的不同，往往需要数月至数年的时间[2, 6, 7]。

2.2 IT 产业的新概念

如今，IT 产业正经历着由诸如云计算和相关概念等技术所带来的基础性变革。传统的自动化金字塔正在溶解，并且产业正在朝着面向服务和面向应用的方向发展[4, 8]。

软件套件将按功能划分为服务，应用程序和由如边界、雾计算概念和云平台等分布式计算方法提供的分散化组件。这些服务和应用程序可以在网络中进行非层次编排，其中基于开放标准的服务之间的通信将成为成功的关键因素。这种克服层级结构的实现也允许实时信息的通信[6, 9]。

许多产业公司已经注意到关于服务导向的转变，并开始建立自己的基于云的平台。例子是 Bosch 的 IOT 套件，GE 的 Primdix 或西门子的 Mindsphere。然而，这些平台大多是围绕公司提供的产品和服务定制的，缺乏与其他平台提供商的互操作性。相反，像 Virtual Fort Knox[10]或 Fraunhofer initiative Industrial Data Space[11]等平台遵循了联合的方法，使独立软件供应商参与生态系统，并防止了供应商锁定效应。

3. 微服务

为了介绍微服务的概念，我们在下面的第一部分与差异最明显的架构：整体架构进行比较。

3.1 整体架构

短语“整体的”被用来描述由一个整体组成的软件应用程序。传统 IT 产业，如第 2.1 节所介绍的，通常使用这种方法。该架构是仅仅在一个计算实例上运行而设计的。这样可以

在多个 CPU 上运行多个进程，但它们共享相同的操作系统和硬件。

如果系统达到容量峰值，则它需要被完全地复制出去。此过程可由连续部署系统自动执行。这种做法的主要缺点是缺乏灵活性。例如，如果用户的数量达到了一个不能由单个实例处理的值，则整体系统就会缺乏所需的水平可伸缩性。相反，它只能被垂直伸缩。

3.2 面向服务架构

一般来说，微服务体系结构是一种如第 2.2 节所介绍的那样 SOA。SOA 中的服务是一个提供一个了预先定义的功能来匹配一个业务活动及其特定结果的软件组件。服务是自包含的，这意味着它不依赖外部资源。该服务所需的所有过程都是在内部执行的。它包括如数据库等所需的所有资源。对用户来说服务就像黑匣子，只能通过预定义的接口访问。它本身可能需要一些提供一定的子功能的基础服务[12]。依赖于一组服务的服务也称为聚合服务[13]。

3.3 微服务架构

微服务指的是一种新的软件体系结构。我们的目的是提出一个这个架构的性质的概述。它的核心概念是将应用程序细粒度地分解成微小的服务。虽然 SOA 是将功能封装到单独的服务中的思想，但是微服务额外指定了被封装的功能是小规模的[14]。

3.4 概念

这些服务是围绕业务能力组织的，这允许开发者围绕系统的关键能力进行开发，而不是预定义的组件。服务实例是相互独立的，这适用于全生命周期，包括开发、部署和维护。微服务依赖松散耦合的轻量级通信协议，这意味着服务之间不直接通信，而是使用独立定义的接口，这样也减少了服务之间的依赖性，同时要求企业服务总线（ESB）作为一个集成层。分散式数据存储意味着，微服务架构在跨服务实例之间拆分数据存储，而不是依赖于一个中央数据库。这同样适用于其他资源，例如计算性能[15]。

3.5 益处

基于微服务架构的系统可以更容易地适应不断增长的容量需求。此属性通常称为水平可伸缩性。与前面提到的整体结构相比，这是十分明显的差别。在微服务架构中，可以根据需要复制每个组件以进行负载平衡。这使得对需求峰值的反应更快，也可以更精确地处理需求。此外，这种模块化也使得系统对故障变得健壮。如果一个微服务遇到错误，系统的其余部分仍然可以独立工作。通过使用自动部署，错误的服务可以被自动替换。通过保持关键服务的备份实例的运行，这种替换也可以预先发生。服务的独立使得技术栈和编程语言具有灵活性。任何微服务都可以在某种语言中实现，并使用最适合完成其功能的库。

4. 提出的解决方案

4.1 方法

在生产中设计微服务的建议解决方案是使用基于数据驱动的方法。这种方法需要清楚地

描述数据结构以及公司应用的业务流程。因此，建议确认所有的相关实体、它们的属性和关系并且可视化它们。该过程类似于通过实体关系（ER）建模来创建数据库设计[16]（见图 1）。基于该整体结构，必须创建由微服务管理的逻辑单元。这些逻辑单元的大小取决于整个系统的灵活性和健壮性以及这些单元之间交换的数据的频率的需要。

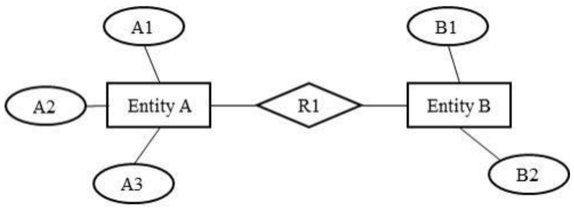

图 1 关系 R1 连接的实体 A 和实体 B 的实体关系（ER）模型及其属性。

逻辑单元之间的数据交换频率是指示两个单元是否应该合并到一个微服务的重要关键数字。通过使用 ESB 来实现所有服务之间的通信是合理的（参见图 2）。在服务具有小尺寸的情况下，通信信道的负载将不必要地上升，因为数据必须在服务之间发送而不是在一个服务内发送。此外，创建服务之间的路由的工作量将上升。为了估计服务之间的数据交换的频率，重要的是不仅要知道数据结构，还要知道系统的行为。可以从中导出服务的数据交换率的行为被公司中实施的业务流程主要影响这。因此，选择合适的微服务规模是一个以灵活性、健壮性和资源消耗为标准的优化问题。

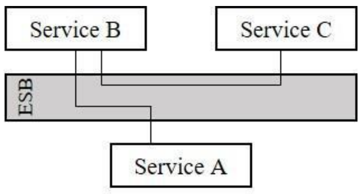

图 2 三个服务通过路线连接到一个在图的中间用灰色框表示的企业服务总线。

4.2 微服务的数据结构

在同意了服务的第一个草案之后，每个服务的数据结构必须从总体数据结构中得到。为了保留不同服务实体之间存在的关系，必须引入跨服务边界标识实体。一个基本的识别措施是使用实体的唯一密钥。这保持了低冗余度，并且最小化了保持数据最新的工作量。

例如，存储着关于一类实体的所有数据的“服务 A”必须提供用于其他服务访问实体数据的应用程序编程接口（API）。为了定位一个特定的实体，“服务 B”必须向包含这所请求实体的唯一密钥的“服务 A”发送请求。作为结果，“服务 A”以预先定义的方式将所请求的实体的所有数据提供给“服务 B”。通过这样做，“服务 B”将总是获得实体的最新数据集而无需自行存储。

这种方法的缺点是，每个请求都会向从一个服务传送数据到另一个服务的集成层施加压

力。为了减少通信负载，一个数据子集可以以额外或替代唯一密钥的方式存储在“服务 B”中。但是这会导致冗余增加，并且不再保证数据完整性。因此，仅在数据不必对每个请求都是最新的情况下才推荐该方法，或者在相关服务中引入冗余数据更新的措施。此外，提高的冗余度还增加了所需存储的量。最后，选择哪种方法取决于实际用例，以及方法所描述的缺点是否与实现相关。

4.3 业务流程和路由

根据整体数据结构，会创建每个服务的子结构和一组基本的 API。为了实现基于一系列微服务的业务流程，必须对时间数据需求和可用性进行业务流程分析。也就是说，要知道哪些数据在流程开始时可用，哪些数据用于完成流程以及获取数据的位置。 从业务流程起点的机器或应用程序开始，每个所需的服务以及服务的处理顺序都会被识别。 基于结果序列，必须使用 ESB 作为通信通道来创建所有服务之间的路由。

在所提出的解决方案中，研究了所有服务之间的最短路线和保持最高独立性的路线。在最短路由的情况下，请求分布在许多数据源服务（又称为“服务 B”和“服务 C”）上的一组数据（进一步称为“服务 A”）的服务会发送请求到服务 B，而服务 B 提供了整个数据集的基础。但是由于微服务结构和数据分配到许多服务，服务 B 不能完全满足要求。因此，服务 B 必须将第一个请求的结果转发给服务 C，服务 C 能够完成数据集并将结果发送回服务 A（参见图 3a）。

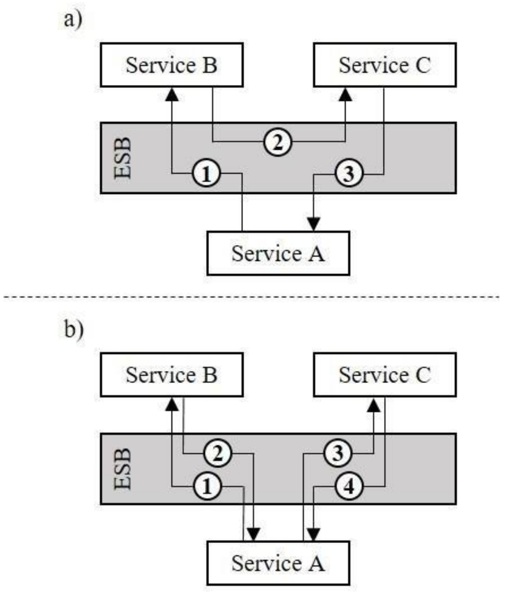

图 3 a）断开从服务 A 开始的请求的最短路由，并将服务 B 的数据收集到 C 并将其发送回服务 A.b）仅通过连接那些对彼此数据模型敏感的服务来表示保持服务独立性的路由。 这种方法的缺点是通信信道的负载增加（图中间的灰色框）。

通过这样做，请求数据的服务即在路由的开始也在路由的结束。因此，该路线很短，而且对 ESB 的要求较低，但是会在所有参与的服务中产生高度依赖性。因为每个服务都必须对其他服务的数据结构保持敏感，这意味着每个服务的数据结构的每次更改都会导致其他服务的结构发生变化，因此会提升依赖关系级别。

为了避免这些服务之间的高度依赖性，建议不要建立一个从服务 A 开始请求，然后将服务 B 和服务 C 的动态数据集返回到服务 A 的循环形式的路由。相反，建议使用发送请求的服务（服务 A）和数据源（服务 B 和 C）之间的直接路由（参见图 3b）。通过使用这种方法，只有服务 A 需要敏感整个数据结构，而服务 B 和服务 C 只需要敏感它们被分配的数据结构即可（参见 4.2 节）。但是此路由方法的缺点是会增加 ESB 上的负载，因为需要更多的请求来完成服务 A 的请求。

5. 实施/例子

提出的解决方案的实施在一个隔离的组装站中被部署，在该组装站中，工人手动制造子组件并向主组装线的工作站供应这些子组件。此外，装配站还有一个显示屏，向工作人员显示当前订单的所有必要信息。该信息旨在帮助工作人员正确地执行装配。

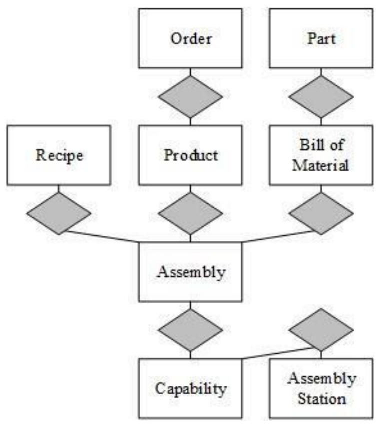

图 4 描述底层用例的数据结构的 ER 模型

在第一步中，工作站会收到一个标识特定订单的订单号。根据这个订单号，从获得必须生产的产品开始的所有组装所需的信息都会被搜集以帮助工人。每种产品都由许多装配在一起生产最终产品的组件组成。为了根据组装站获得正确的组装，每个组装站都有能力描述他们能够生产的组件。此外，配方链接到每个组件，其中描述了执行组装的必要步骤。通过了解配方，功能以及必要组件的列表，可以确定必须在工作站生产的组件。为了最终获得必要的零件信息以帮助工作站上的工作人员，附在该装配上的物料清单会被识别，并且基于该零件的信息会被请求。描述所有实体及其关系的 ER 模型如图 4 所示。

用于装配，产品，配方和功能的逻辑单元/服务会根据数据结构以及在一个工作日内发

生的请求频率进行创建（另请参见图 5）。

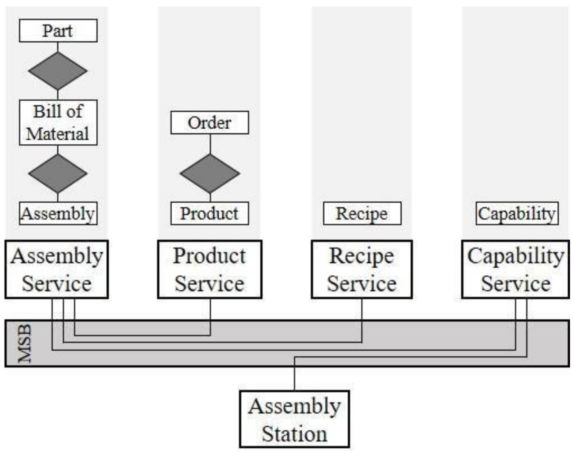

图 5 从总体数据结构及其数据模型导出的服务以及根据总体数据结构通过制造服务总线的链接

此外，图 5 还示出了在服务之间被管理的实体之间的路线关系。在这个例子中，制造服务总线（MSB）用于实现这些路线。MSB 的思想是所有服务都提供一组事件和功能，并且一个服务的事件可以通过所谓的集成流程与另一个服务的功能连接。如果发生连接事件，集成流不仅会触发服务的功能，而且能够使服务之间交换数据[17]。

装配服务会对零件，物料清单以及组件进行处理。如 ER 模型中所示，负责产品和订单的产品服务连接到组装服务和配方服务。配方服务只管理描述组件如何生产的配方。最后，保存和提供工作站功能的能力服务引入了真正的组装站以及组装服务之间的联系。

为了完成本节开头描述的业务流程，组装站首先会请求功能服务的功能。因此，每个工作站都拥有一个唯一的标识号码，该标识号码将发送给该请求的服务。作为结果，装配工作站接收到工作站能够生产的装配 ID 列表。接下来，工作人员扫描表示要生成的下一个订单的订单 ID 的条形码。该订单 ID 被发送到产品服务，该服务返回相应订单的产品 ID。为了知道产品由哪些组件组成，组装工作站会将产品 ID 以及组件列表发送给组装服务。组装服务现在能够选择产品在其数据库中包含的所有组件，并通过工作站能够生成的组件列表来过滤该查询。于是，组装服务会发送必须为当前订单生产的组件 ID 以及包含必要部件所有信息的物料清单。在最后一步中，组装工作台将组装 ID 转发给配方服务，该服务查找组装工作的所有工作步骤并将其发送回工作站。现在工人拥有完成组装工作所需的所有信息。组装完成后，扫描下一个订单的条形码，并重新开始过程。

图 6 显示了 4.3 节中描述的替代方法，包括组装站和服务之间的所有请求的顺序。这些请求具有组装站位于中心的星形拓扑结构。这种结构提供了所有服务之间的低度依赖关系。

与所示的路线相比，在两个服务之间没有会导致每个服务的不同数据结构混合的直接连接。因此，如果每个服务提供的接口与前一个接口相同，则每个服务都可以更换。

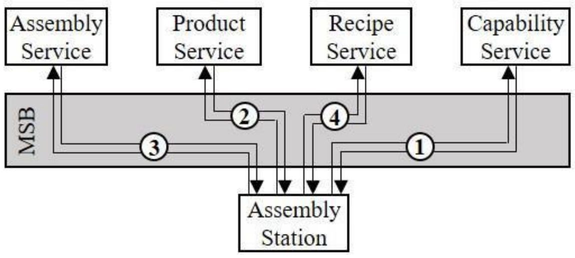

图 6 基于组装站和服务之间直接连接的呼叫顺序。为了收集所有基于订单 ID 和装配工作站 ID 的数据，必须完成①到④的请求。

6. 结论

目前的大趋势（如全球化，人口变化，人口增长）迫使制造公司接受生产范式的变化以满足新的需求。其中一种模式变革是放弃整体软件系统而采用 SOA 和 XaaS 的概念，这些概念支持公司创建具有增强的可扩展性，稳健性和灵活性的生产系统。 SOA 的一个特定概念是将服务设计为微服务，以便封装功能和保持较小的服务大小。微服务显著地提高了整个系统的水平可伸缩性和模块化/健壮性。

为了获得最适合的服务规模，引入了数据驱动的设计方法，该方法从创建整体数据结构和建立相关业务流程开始。然后，根据微服务将整个数据结构划分成逻辑单元。单元的大小取决于灵活性和稳健性的需求，以及单个服务的资源消耗水平和分配给所有参与者的电力负载。例如，如果微服务的平均大小很小，则链接所有服务的通信信道的负载会更高，因为需要在服务之间交换更多数据。相比之下，如果服务更大，则单个服务上的通信量将减少但负载更重，因为每个服务都必须执行更多的任务。除了服务的大小，服务之间的路由也会影响所有组件的负载。本文研究了两种不同的路由方法：a）减少通信信道负载的最短路由和 b）通过在请求数据的服务和提供数据的服务之间创建星形路由来保持每个服务的最高独立性的路由。

本文在工人操作的组装站的基础上部署了一个实例。工作人员在开始时收到订单 ID，并根据此 ID 收集所有必要数据以支持工作人员。最后，这个用例是一个用于检测该方法的业务影响实验站点。关于服务和硬件的可交换性，一般的可用性以及业务影响的实验结果正在进一步研究。

致谢

本文中的结果来自 Fraunhofer IPA 在 BEinCPPS 项目（GA 号 680633）中的工作。 它由欧盟地平线 2020 研究和创新计划资助，并且是 I4MS 倡议的一部分。 作者负责本出版物的

内容。

参考文献

[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.

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.

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 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

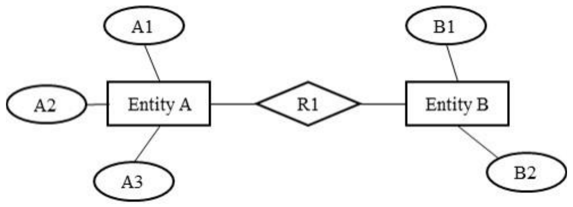

Fig. 1. Entity relationship (ER) model of entity A and entity B connected via relation R1 and their attributes visualized by ovals.

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.

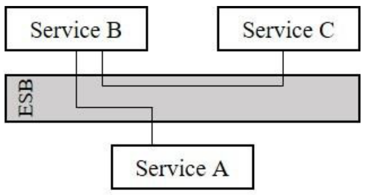

Fig. 2. Three services connected by routes via an enterprise service bus symbolized by the grey box in the middle of the figure.

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.

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.

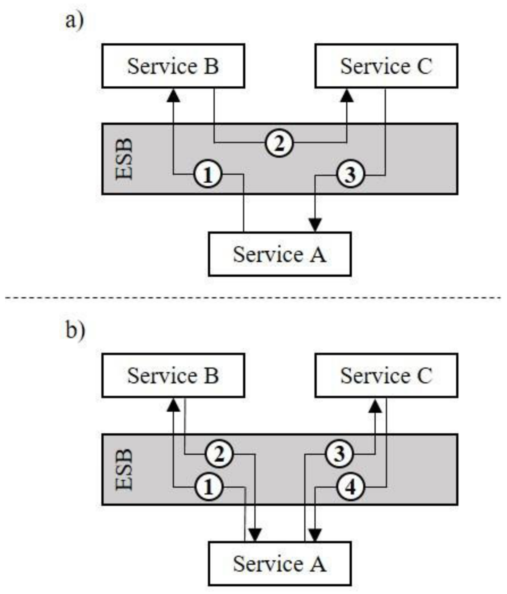

Fig.3. a) Repesents the shortest route of a request starting at service A and collecting data from service B to C and sending it back to service A. b) Represents a routing preserving the independencey of the services by only connecting services which are aware of each others data model. The drawback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

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.

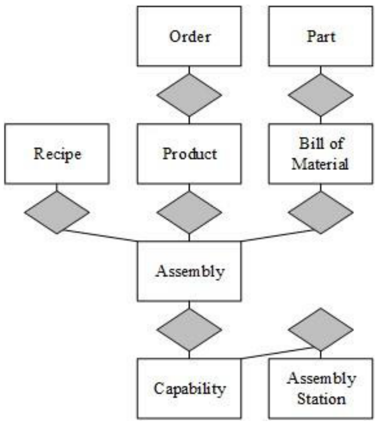

Fig.4. ER Model describing the data structure of the underlying use case.

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.

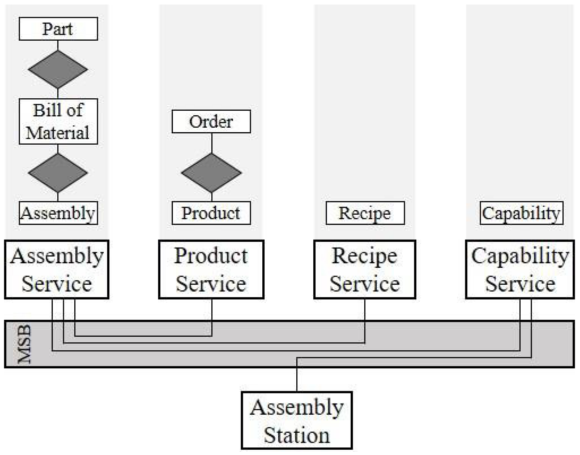

Fig. 5. Services derived from overall data structure and their data model as well as the links through the manufacturing service bus according to the overall data structure.

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.

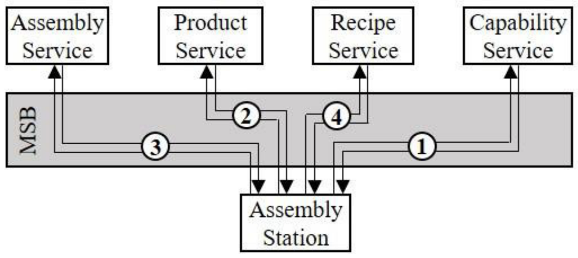

Fig. 1. Sequence of calls based on direct connections between assembly station and services. In order to gather all data based on the order ID as well as the ID of the assembly station the requests from ① to ④ have to be done.

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.