

# 华中科技大学

## 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<sup>[1]</sup>。

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<sup>[2]</sup>。

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<sup>[2,3]</sup>。

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）<sup>[4]</sup>。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<sup>[5]</sup>。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<sup>[6]</sup>。

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<sup>[2, 6, 7]</sup>。

## 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<sup>[4, 8]</sup>。

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<sup>[6, 9]</sup>。

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 Prindix 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<sup>[10]</sup>或 Fraunhofer initiative Industrial Data Space<sup>[11]</sup>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了一个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<sup>[12]</sup>。依赖于一组服务的服务也称为聚合服务<sup>[13]</sup>。

### 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<sup>[14]</sup>。

### 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<sup>[15]</sup>。

### 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<sup>[16]</sup>（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。

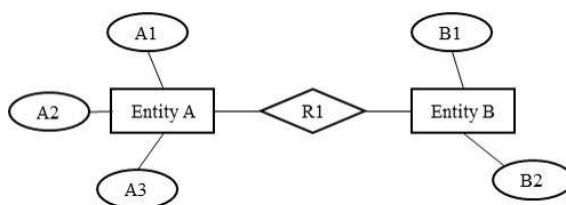


图 1 关系 R1 连接的实体 A 和实体 B 的实体关系（ER）模型及其属性。

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。

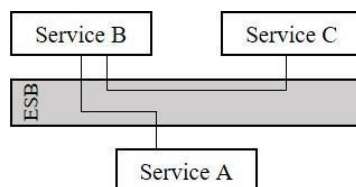


图 2 三个服务通过路线连接到一个在图的中间用灰色框表示的企业服务总线。

## 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）。

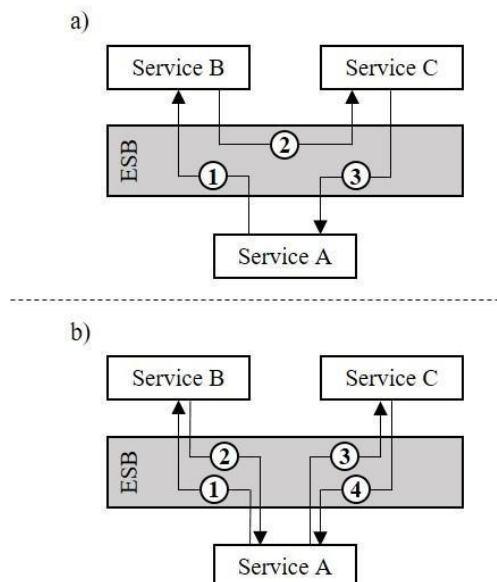


图 3 a) 断开从服务 A 开始的请求的最短路由，并将服务 B 的数据收集到 C 并将其发送回服务 A. b) 仅通过连接那些对彼此数据模型敏感的服务来表示保持服务独立性的路由。这种方法的缺点是通信信道的负载增加（图中间的灰色框）。

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。

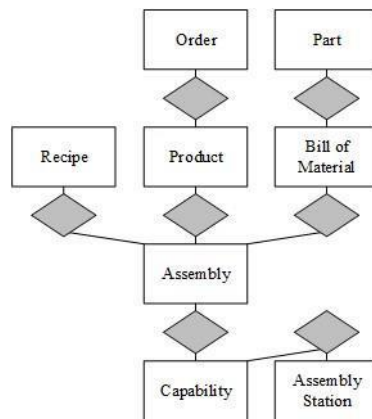


图 4 描述底层用例的数据结构的 ER 模型

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）。

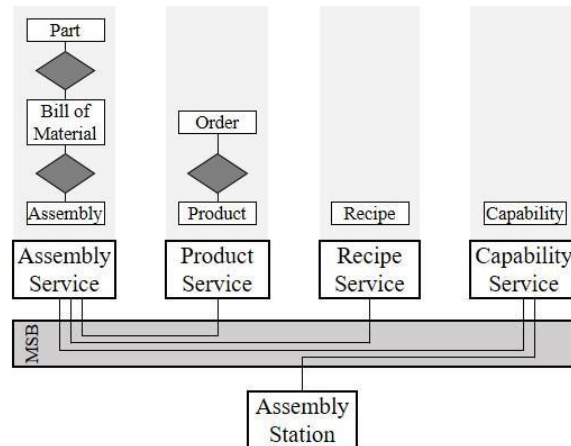


图 5 从总体数据结构及其数据模型导出的服务以及根据总体数据结构通过制造服务总线的链接

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<sup>[17]</sup>。

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。

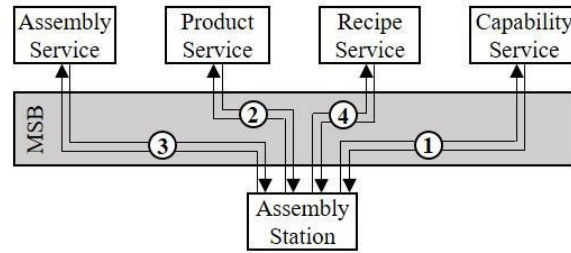


图 6 基于组装站和服务之间直接连接的呼叫顺序。为了收集所有基于订单 ID 和装配工作站 ID 的数据，必须完成①到④的请求。

## 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.