软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005797]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

限界上下文视角下的微服务粒度评估*

钟陈星[1,2], 李杉杉[1,2], 张 贺[1,2], 章 程[3]

[1](南京大学 软件学院,江苏 南京 210023)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[3](安徽大学 计算机科学与技术学院,安徽 合肥 230601)

通讯作者: 张贺, E-mail: hezhang@nju.edu.cn

摘 要: 近年来,DevOps 日渐火热,作为支撑技术的微服务架构由于其敏捷性、灵活性和可扩展性已成为软件行业关注的热点.然而,微服务粒度的界定是微服务领域的一项难题,至今缺乏行之有效地评估微服务粒度的标准.针对此问题,本文结合几种微服务划分原则提出了 4 项评估指标用于量化地衡量微服务划分的合理性,并基于此提出了一种基于限界上下文的微服务粒度评估模型.同时,实现了工具原型自动化地计算评估结果.案例研究部分将模型的评估结果与架构设计人员的心理预期作比较,结果证明所提出的评估模型可以较好地评估微服务粒度.

关键词: DevOps;微服务架构;粒度;划分;限界上下文;评估指标

中图法分类号: TP311

中文引用格式: 钟陈星, 李杉杉, 张贺, 章程.基于限界上下文的微服务粒度界定研究.软件学报. http://www.jos.org.cn/1000-9825/5797.htm

英文引用格式: Zhong CX, Li SS, Zhang H, Zhang C. Evaluating Granularity of Microservices-oriented System Based on Bounded Context. Ruan Jian Xue Bao/Journal of Software, (in Chinese). http://www.jos.org.cn/1000-9825/5797.htm

Evaluating Granularity of Microservices-oriented System Based on Bounded Context

ZHONG Chen-Xing[1,2], LI Shan-Shan[1,2], ZHANG He[1,2], ZHANG Cheng[3]

[1](Software Institute, Nanjing University, Nanjing 210023, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[3](Computer Science and Technology, Anhui University, Hefei 230601, China)

Abstract: During recent years, DevOps gains its popularity. As the support of DevOps, microservices architecture has become a hot spot of software industry for its agility, flexibility and scalability. However, finding the adequate granularity of microservices is a big challenge. Microservices architecture still lacks the criteria for evaluating its granularity. To cope with this problem, this paper designs four evaluation metrics based on several principles of best microservices practice to quantitatively measure the rationality of microservice decomposition. Based on that, a granularity evaluation approach of microservices based on bounded context is proposed. By implementing a tool prototype, evaluation results can be automatically calculated. We use a case study to evaluate the decomposition of a microservices-oriented system and compare the results with the architects' expectations. The results prove that the proposed approach can be applied to evaluate the granularity of microservices-oriented system.

Key words: DevOps; microservices architecture; granularity; decomposition; bounded context; evaluation metrics

伴随着互联网技术的飞速发展和市场需求的急剧扩增,软件应用是否可以灵活地调整业务与规模,持续交

* 基金项目: 国家自然科学基金(61572251); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2017A13)

Foundation item: National Natural Science Foundation of China (61572251); State Key Laboratory for Novel Software Technology (Nanjing University)开放课题 (KFKT2017A13)

收稿时间: 2018-09-01; 修改时间: 2018-10-31; 采用时间: 2018-12-14; jos 在线出版时间: 2019-05-22

CNKI 网络优先出版: 2019-05-22 15:26:04, http://kns.cnki.net/kcms/detail/11.2560.TP.20190522.1525.006.html

付部署,快速更新迭代成为企业在现代化商业竞争中取胜的决定性因素.然而,传统的软件开发采用单体架构(Monolithic Architecture)[1],将应用程序的所有功能封装在一个独立的单元中,随着软件系统复杂度的剧增和开发团队的扩大暴露出其不够灵活、开发效率低下以及妨碍持续交付与部署等缺点.

针对软件敏捷性、灵活性和可扩展性需求的不断增长使得突破单体架构的限制成为亟待解决的问题,也使得微服务架构(Microservices Architecture)[2]应运而生.微服务架构将一个独立的应用程序分成多个协同工作的小而自治的服务,每个微服务通常只完成某个特定的功能,并且运行在单独的进程中.相较于单体架构,微服务在面对快速发展变化的业务需求时表现出更加敏捷(agility)、灵活(flexibility)及可扩展性(scalability)等优点.具体来说,微服务拥有技术异构,不同的服务采用适合的技术;对数据进行"去中心化"控制,不同的服务无需共享一个中心数据库;智能端点,服务管理相关业务逻辑并通过轻量级的消息交互与其他服务通信;围绕业务功能组织服务,负责服务开发的团队同时负责该服务相关业务需求的所有实现细节等特性[3].微服务的这些特点与DevOps 提倡的生产环境的弹性、可靠性、稳定性以及高频率部署是一致的,这使其在 Netflix, Amazon 以及eBay 等世界领先的互联网公司中率先流行起来.

然而,微服务的这些优势并不是一蹴而就的.微服务划分是微服务领域的一项难题.其原因主要来自于以下四个方面:第一,项目设计的初级阶段可以为解决微服务划分问题做决策提供支撑的信息过少;第二,微服务的划分不仅与系统的领域知识相关,还需要综合考虑开发团队的组织结构,项目的可用资源以及系统的非功能性需求等多个因素;第三,微服务划分问题本身就是一个极富挑战性的多目标优化问题.合理的微服务划分要求设计者在多个相互排斥的性质之间权衡利弊[4].比如,微服务提倡小而自治的服务之间相互合作[2].小而自治的服务符合单一职责原则(简称 SRP),简化了开发过程,但同时由于更多的服务需要部署也增加了运维人员的压力[5].从内聚与耦合的角度看,粒度越小的服务具有越高的内聚性,然而粒度越小的服务要实现同一个用例必然涉及更频繁的服务间的通信交互,这使得服务间的耦合性提高,系统的复杂性更大;第四,微服务领域仍缺少针对服务设计本身的质量评估方法.发展成熟的面向对象领域有许多评估面向对象设计的有效方法.然而,这些方法通常聚焦于代码实现层的概念(比如类),与面向服务的微服务架构的关注点不同.另外,在代码实现后才评估微服务划分的粒度所需要的系统重构代价实在太大.因此,将评估面向对象设计的方法直接应用于微服务架构中不具备合理性.在很大程度上,架构师对微服务划分的合理性评估仍然依赖于他们对领域知识的理解和直觉[4].

针对上述问题,本文旨在利用限界上下文(Bounded Context)理论[6]对微服务架构的服务划分粒度进行研究,为微服务划分粒度的质量评估问题提供理论方法及工具支持,帮助实现传统单体应用向微服务架构的迁移.本文的主要贡献在于提出的微服务粒度评估模型及其工具可用于指导微服务架构中合理的服务划分[7],辅助架构师做微服务设计决策,一方面可以减少不合理的微服务设计所造成的软件开发与部署难题;另一方面可以提高团队开发效率和系统性能,提升软件企业的竞争力.

本文第 1 节介绍背景及相关工作.第 2 节提出评估方法.第 3 节给出 4 种评估指标及指标合并过程.第 4 节实现自动化工具原型并利用案例研究验证模型的有效性.第 5 节总结论文,并提出下一步的工作.

1 背景及相关工作

合理的微服务划分是微服务架构敏捷、灵活和可扩展性的先决条件,也是微服务领域的一项严峻挑战.过去的几年里,一些研究人员致力于探究微服务划分问题.Richardson[8]提出了用于指导微服务划分的 4 个策略,包括"根据业务能力(business capability)进行划分、"基于领域驱动设计中子领域(subdomain)的概念进行划分"、"利用动词和用例图进行划分"以及"利用名词和资源定义服务"等.Richardson[8]提出的指导微服务划分的 4 个策略并没有为微服务划分问题提供系统化的指导方法,具体的微服务划分过程仍依赖于系统架构人员对领域业务需求的理解和主观经验判断.Michael Gysel[9]等开发了微服务划分工具 Service Cutter.他们根据软件架构设计的经验定义了 16 种耦合度度量标准用以衡量系统中的 nanoentities(包括数据、操作和软件制品)之间的耦合度,从而提取出以 nanoentities 为节点的系统无向加权图,在此基础上实施聚类算法实现微服务的划分.Service Cutter 存在两个方面的问题,其一,系统架构人员需要学习 16 种耦合度度量标准的定义,并且结合系统的领域业

务需求规定耦合度度量标准的优先级,这极大地依赖系统架构人员的主观经验;其二,由 Service Cutter 计算得到的候选微服务准确度较低,系统架构人员仍需结合自己的预期划分决定是否采纳 Service Cutter 提供的划分方案.Luciano Baresi[10]等以符合 OpenAPI 规范的系统 API 为输入,使用 DISCO 相似度评估算法计算 API 操作之间的相似度,将相似度高的操作及其数据划分到同一服务从而实现微服务的划分.该方法同样存在两个方面的问题,其一,该方法以系统 API 中操作之间的相关性而不是领域业务逻辑层面的业务需求作为划分依据,操作的具体实现对最终划分结果的影响较大;其二,该方法以实现层面的 API 为输入,适用于单体应用到微服务应用的迁移,而不适用于全新的微服务应用的开发.Rui Chen[11]等将系统的领域业务逻辑绘制成数据流图,将所有具有相同输出数据的操作合并为一个操作,最终得到的每个操作及其数据都成为一个候选微服务.该方法仅考虑了处理相同数据的操作应属于同一服务,而忽略了微服务划分的其他重要原则,可能存在微服务粒度过大的缺陷.显然,微服务划分问题仍缺少系统可行的理论指导方法.

更重要的是,当前的研究对于“什么样的微服务划分是合理的”这一问题还缺乏标准而统一的认识.缺乏微服务划分合理性的评估标准使得比较不同微服务划分方案变得困难.Luciano Baresi[10]等与 Rui Chen[11]等在将其工作与 Service Cutter 进行比较时,都仅讨论了两种方法在实现层面的复杂程度,而不提微服务划分结果之间的优劣对比.在评估微服务划分问题上,Sam Newman[2]指出,微服务划分的核心原则是高内聚与低耦合.高内聚将相关联的业务逻辑包含在同一个服务中,便于代码管理;低耦合降低了服务间的依赖与通信开销,允许服务的独立修改和部署.Sam Newman[2]给出的评估微服务划分的讨论属于定性分析,存在含糊不清.Uwe Zdun[12]等基于微服务架构的实践模式设计了几种评估微服务划分的标准,并定义了相关算法定量地计算当前微服务划分遵循该实践模式的程度.设计的评估标准包括“所有组件是否都可独立部署或者所有组件可独立部署的程度”以及“当前微服务划分方案是否避免了共享其他组件或者避免了以紧耦合的方式共享其他组件”.该微服务划分的评估方法仅衡量了当前微服务划分是否符合微服务架构的实践模式,而不考虑系统的具体领域业务逻辑以及当前微服务划分下系统的内聚性等其他方面.

限界上下文是领域驱动设计(Domain-Driven Design,简称 DDD) [6]的核心概念.在应对高复杂度的软件开发任务时,DDD 提倡将应用按照领域业务逻辑划分为多个限界上下文,不同上下文使用独立的模型,并且显式定义每个限界上下文的边界和限界上下文之间的映射关系.限界上下文与微服务是天然契合的.其关注点分离的思想有助于系统架构人员从每个服务内部和服务之间两个角度设计架构.从限界上下文视角讨论微服务粒度能够帮助我们分离出问题的核心点.

本文基于限界上下文理论提出了一种微服务划分的评估模型,具有以下优点:1)以用例图和实体关系图为输入,充分考虑了系统的领域业务逻辑;2)设计了 4 种评估指标,定量地衡量微服务划分在各项指标上的表现;3)设计的 4 种评估指标都结合了微服务划分的核心原则,包括高内聚与低耦合;4)进行了指标合并,提供综合评分用以比较不同微服务划分方案;5)实现了工具原型,可以自动化地评估微服务划分;6)提出的评估模型不涉及微服务应用的具体实现,既适用于单体应用向微服务应用的迁移,又适用于全新的微服务应用的开发.

2 限界上下文视角下的微服务粒度评估方法

本节为解决微服务粒度界定问题提出一种基于限界上下文的量化评估方法并给出具体的评估步骤.

2.1 服务建模

以评估微服务划分为目的,基于 UML 2.0[13]对微服务设计初级阶段的关键元素建模以便于阐述后文.建模结果如图 1 所示.

基于限界上下文划分微服务将一个独立的应用程序划分为多个协同工作的小而自治的服务,每个服务对应一个特定的限界上下文,规定了一组领域业务逻辑和相应的领域实体.服务通过完成用例实现应用的领域业务逻辑,即软件系统的功能需求.一个服务允许实现多个用例,一个用例允许由多个服务通过消息交互实现.属性描述了实体的特征,是实体的组成元素,也是微服务划分中不可再分的原子单元.一个用例涉及到的属性可以分为输入和输出两部分,分别由对应服务的读操作和写操作完成[8].实体与关联关系之间的对应关系与传统 ER

模型一致.

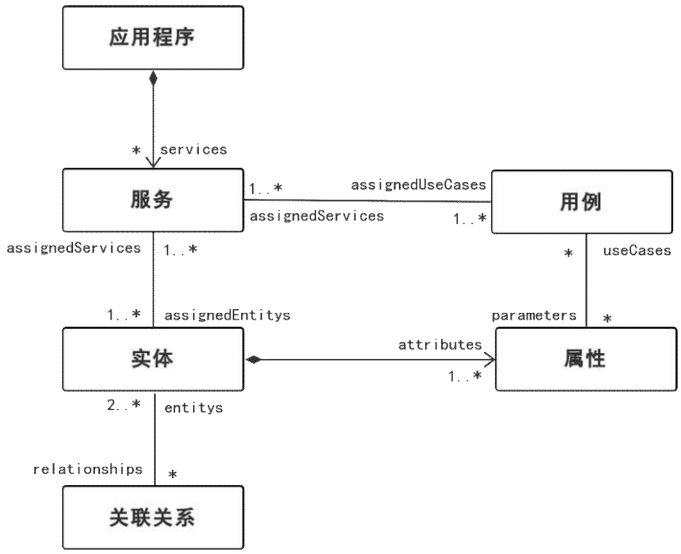

Fig.1 Modeling of key elements in microservices decomposition

图 1 微服务设计中关键元素的建模

在评估微服务粒度的工作中,服务模型的表现形式包括软件制品(Artifact,包括用例图和实体关系图等)和候选微服务.

(1) 用例图描述了软件系统的功能需求.每个用例对应软件系统的一项业务逻辑,完成一个用例有时需要多个服务之间的协同合作.Richardson[8]指出用例可以作为微服务划分的依据.

(2) 实体关系图与自然语言相对应,描述了软件系统中关键的领域实体及实体与实体之间的关系,可用于指导微服务的拆分.关联关系紧密的领域实体倾向于划分到同一个服务中.

(3) 候选微服务描述了微服务划分中服务与用例、实体之间的对应关系,规定了每个服务负责的领域业务逻辑和相关限界上下文.本文通过评估候选微服务在各量化指标上的表现评估微服务划分的合理性.

2.2 评估过程

本文提出的微服务粒度的评估过程可以分为 4 个步骤,如图 2 所示.

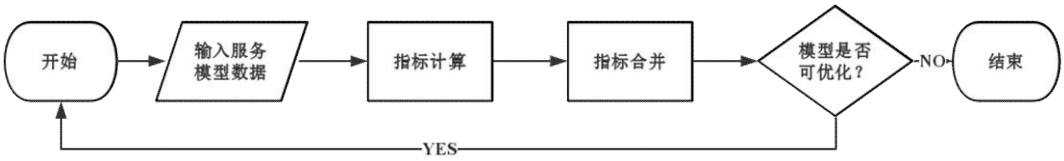

Fig.2 Evaluating the rationality of microservices decomposition

图 2 评估微服务划分合理性的过程

(1) 输入服务模型数据:包括用例图、实体关系和候选微服务 3 种表现形式.

(2) 指标计算:以服务模型数据作为输入,定量计算候选微服务在所有评估指标上的值,衡量该微服务划分在各项指标上的表现.

(3) 指标合并:对指标计算的结果做归一化处理与加权求和得到该微服务划分的综合评分,以比较不同的微服务划分方案.

(4) 评估模型是否可优化:设计人员根据指标计算及合并过程得到的结果对微服务划分进行分析,判断能否重新调整划分以获得更优的微服务粒度.

容易看出,微服务粒度的评估是一个向更优划分靠近的不断迭代的过程.架构设计人员应选择评估过程得到的最优划分方案作为最终的微服务划分结果.

3 评估指标及合并

本节根据业界广泛采用的微服务划分原则[2,8]定义一系列评估指标用于衡量微服务划分,指出各项指标的有效性, 并给出具体的指标计算公式.

3.1 服务内聚性

服务内聚性指服务内用例之间的功能相关性[14].服务的内聚性越高,设计模型的可理解性越强,系统也就越敏捷[14].

3.1.1 指标有效性分析

内聚性是软件系统中的主要属性,也是微服务设计质量评估的重要因素.服务的内聚性越低, 一方面意味着该服务存在越多功能逻辑不相关的代码, 其他服务对该服务存在越多的依赖, 该服务越难以独立修改; 另一方面意味着不同服务中存在越多功能逻辑相关的代码,代码的冗余度越大.与此同时,系统的技术可选性越小[2].另外, 高内聚性的服务由于实现了功能逻辑紧密相关的用例, 因而具有更清晰的限界上下文, 可理解性更强. 当系统的微服务划分与开发团队的组织结构一致时,具有高内聚性的微服务开发速度更快.并且,由于服务的内聚性减少了微服务内部组件间的消息交互,因此具有更高的系统效率[15].

正确定义服务边界、分析系统语义是提高服务内聚性的前提[15].S Newman[2]指出,高内聚是微服务领域的重要概念,是指将逻辑相关的代码尽可能放在相同服务中.通过领域驱动设计中的限界上下文划分微服务边界,每个服务聚焦于一个具体的业务逻辑,可以避免服务粒度过大,实现高内聚从而实现微服务好处的最大化.

3.1.2 计算方法

服务的内聚性可通过服务内部的实体间内聚性来度量.现有的许多研究[16,17,18]通过为 ER 模型中的关联关系信息定义优先级来衡量实体间的内聚性,涉及到的关联关系信息包括类型、基数(cardinality ratios)和参与约束(participation)等.[18]使用距离的概念度量实体间的内聚性,并为不同优先级别的实体间内聚性设置了不同的距离值,距离的值越大,实体间的内聚性越小.[17]指出,强实体与弱实体(Strong-Weak)之间的内聚性最高,拥有第一优先级.拥有内聚性第二优先级的是具有特殊化/概化(Generalization)关系的超类和子类.另外,聚合(Aggregation)关联关系由于具有部分与整体生存周期上的联系,同样属于第二优先级队列.具有限制(Exclusive Binary)关联关系的实体间内聚性较低,拥有第三优先级[18].具体的实体间关联关系类别与距离值的对应见表 1.

Table 1 Correspondences between relationship of entities and distances in ER diagrams

表 1 ER 图中实体间关联关系类别与距离值的对应

关联关系类型	距离值
Strong – Weak	1
Generalization	10
Aggregation	10
Exclusive Binary	100

对于 ER 模型中的其他关联关系类别,我们根据基数与参与约束信息定义不同的优先级以表现实体间不同

程度的内聚性特征.针对关联关系的基数,不同基数的关联关系对应的实体内聚性特征如下:一对一 > 一对多 > 多对多;针对关联关系的参与约束,不难理解,强制(mandatory)性参与的关联关系比可选(optional)参与的关联关系具有更强的内聚性,因此,不同参与约束的关联关系对应的实体间内聚性特征如下:强制-强制(M-M) > 强制-可选(M-O) > 可选-可选(O-O).据此, Mohammad [14]列出了表 2.

Table 2 Correspondences between cardinality and participation of relationship and distances in ER diagrams [14]

表 2 ER 图中一般关联关系的基数与参与约束信息对应的距离值[14]

关联关系的基数 \ 参与约束	M – M	M – O	O – O
1 – 1	1	10	100
1 – N	5	50	500
M – N	10	100	1000

在计算三元及其他 n ($n>2$)元关联关系时,我们将表 2 数据乘以 10^{n-2} 得到对应的距离值.表 1 和表 2 中距离值的数值选择只反映了 ER 模型中不同关联关系对应的实体内聚性的强弱, 与具体数值多少无关. 比如, 1 和 10 的差距使得具有 M-M 参与约束的两个实体比具有 M-O 参与约束的两个实体内聚性更强. 最后,出于完整性的考虑,规定两个相同实体之间的距离值为 1.

那么,任意实体 E_1 和 E_2 之间的距离值可以表示为 ER 图中两个实体之间所有路径上的边的距离值之和的最小值,即最短路径的长度,公式表示如式 1.

$$Distance(E_1,E_2)=\begin{cases}\min_p \sum_{i=0}^{e_p} d_i, p>0\\ \infty, p\le 0\end{cases} \quad (式\ 1)$$

其中, p 表示 ER 图中实体 E_1 和 E_2 之间的路径数, e_p 表示当前路径的边数, d_i 表示边的距离值.实体 E_1 和 E_2 的内聚性如式 2 所示.

$$Cohesion(E_1,E_2)=\frac{1}{Distance(E_1,E_2)} \quad (式\ 2)$$

由式 1,得到任意 ER 图对应的 ER 模型中所有实体之间的距离值, [14]称其为距离表(Distance Table).接下来,利用 ER 模型中实体之间的距离值度量服务内聚性.根据第 1 节提出的服务模型,一个服务允许实现多个用例.一个服务实现的用例越多,该服务的内聚性就越低[2].对于服务内的任意两个用例 1 和 2,我们得到对个实体集 (BE_1,BE_2) 以及所有的 1 中实体与 2 中实体之间的距离值(由距离表可得),从而构建二分图 $G(V,E)$,二分图中每条边 $E(i)$ 的距离值用 $Weight(E(i))$ 表示.使用贪心算法选出连接用例 1 和 2 中所有实体的距离值最小的边的集合 R 可用于度量两个用例间的内聚性.具体算法如 Algorithm 1 伪码所示.

Algorithm 1 使用贪心算法选出符合条件的 R
输入: BE_1, BE_2,距离表 *DistanceTable*
输出: R :连接用例 1 和 2 中所有实体的距离值最小的边集合
1: $R \Leftarrow \varnothing$
2: $S \Leftarrow \varnothing$
3: *while*$(|S|<|BE_1|+|BE_2|)$*do*
4: 从 G 中选出具有最小距离值的边 e
5: *if* e 不属于边集 R 且 e 不与 R 内其他边构成环 *then*
6: 将 e 加入集合 R
7: 将 e 的端点加入集合 S
8: *end if*

9: *end while*

根据得到的符合条件的边集 R,我们由式 3 计算用例之间的内聚性 UCC(Use Case Cohesion).不难理解,式 3 的计算结果是边集 R 的内聚性平均值.

$$UCC(i,j)=\frac{\sum_{i=1}^{|R|}\frac{1}{Weight(R(i))}}{|R|} \tag{式 3}$$

对应的服务 k 的内聚性 SC(Service Cohesion)可以表示如下,其中 a 表示第 k 个服务中的用例数:

$$SC_k=\begin{cases}\sum_{i=1}^{a}\sum_{j=1,i>j}^{a}\frac{UCC(i,j)}{\frac{a(a-1)}{2}}, a>1\\ 1, a\le 1\end{cases} \tag{式 4}$$

同时,得到微服务系统设计的服务内聚性 SDC(Service Design Cohesion)如下, s 表示系统中的所有服务数:

$$SDC=\frac{\sum_{k=1}^{s}SC_k}{s} \tag{式 5}$$

3.2 服务耦合性

服务耦合性描述了一个服务依赖于其他服务的程度,或者说一个服务的改变对其他服务的影响程度[15].

3.2.1 指标有效性分析

微服务的特征之一是服务自治,允许服务独立地修改和部署.这就要求减少服务之间的依赖,降低服务的耦合度.服务之间的低耦合可以带来两个好处.一方面,服务之间的通信交互越小,意味着系统具有更高的效率和更好的性能.另一方面,低耦合度意味着其他服务的修改对当前服务的影响较小,系统具有更好的灵活性,能快速适应需求的变化[2].低耦合是软件工程中衡量系统设计的重要标准.

依据限界上下文定义微服务边界,正确实现微服务划分可以保证实现服务间的松耦合[19].Robert Martin[20]的单一职责原则—— “Gather together the things that change for the same reasons. Separate those things that change for different reasons.” ——同样可以降低微服务划分的耦合度.具体来说,将同一个用例涉及到的属性划分到同一个服务,将具有紧密关联关系的实体划分到同一个服务,减少服务之间的通信数量,消除不必要的服务之间的关联和依赖[9].

3.2.2 计算方法

服务间的耦合度体现了服务间的相互依赖程度.若服务 A 访问服务 B 的实体以实现其业务用例,则称服务 A 依赖于服务 B,服务 A 与 B 之间存在耦合.在微服务应用中,服务之间通过轻量级消息机制实现交互.服务之间的消息交互越多,系统的通信开销就越大,系统性能也就越差.因此,可以用服务间传递消息的大小,包括传递的属性个数以及属性的复杂度等,来度量服务间的耦合度[11].其中,属性的复杂度由属性本身的组成结构决定.类级别的属性具有比基本数据类型属性更高的复杂度,集合类型的属性也具有比基本数据类型属性更高的复杂度.为此,定义两种不同的属性复杂度级别:类与集合(ClassOrList)复杂度为 10,原子(Primitive)级别复杂度为 1,所有属性的默认复杂度为原子级别.另外,服务之间以不同的频率传递消息体现的服务耦合程度是不同的.在项目设计阶段,消息频率表现为用例频率(Frequency of Use).用例频率属于用例图的内容,根据业务逻辑预估该用例在系统运行时单位时间内的调用次数获得.比如,修改用户密码等涉及到基础配置的用例调用次数较少,频率较低.我们定义三种不同的用例频率:低、中、高(分别对应频率值 1、10、100),并设置默认用例频率为中.为属性和用例频率设置默认值降低了设计人员使用该工具进行服务粒度评估的输入工作量.基于以上分析,用式 6 度量系统设计的服务耦合性.

$$\frac{1}{s}\times\sum_{l=1}^{s}\sum_{t=1}^{u_l}f_{lt}\times\left(\sum_{m=1}^{\eta_{lt}}c_{ltm}+\sum_{n=1}^{w_{lt}}c_{ltn}\right) \quad \text{(式 6)}$$

其中:

u_l 表示服务 l 实现的用例总数 $(u_l\geq 0)$;

f_{lt} 表示服务 l 中的用例 t 的频率 $(f_{lt}\geq 0)$;

r_{lt} 表示服务 l 中的用例 t 包含的在服务间读属性的总数 $(r_{lt}\geq 0)$;

w_{lt} 表示服务 l 中的用例 t 包含的在服务间写属性的总数 $(w_{lt}\geq 0)$;

c_{ltm} 表示服务 l 中的用例 t 包含的在服务间读属性 m 的复杂度 $(c_{ltm}\geq 0)$;

c_{ltn} 表示服务 l 中的用例 t 包含的在服务间写属性 n 的复杂度 $(c_{ltn}\geq 0)$.

3.3 用例收敛性

用例收敛性指代单个服务聚焦于处理特定领域业务功能的程度.

3.3.1 指标有效性分析

微服务划分的原则之一是每个服务要足够小.著名的面向对象设计原则——单一职责原则(SRP)——规定每个类/模块都应该有单一的功能,并且该功能应由这个类/模块完全封装起来.SRP 对于面向服务的微服务设计同样有效[8].服务实现的用例体现了该服务为软件系统提供的业务功能.服务实现的用例越多,意味着微服务设计不符合 SRP 的程度越大,系统可复用性和内聚性就越低.

另一方面,实现单个用例涉及的服务越多,需要的服务之间的通信开销就越大,系统延迟也就越高.不仅如此,这还需要越多的开发团队之间的合作,一定程度上提高了服务间耦合和开发团队间的依赖,降低了服务及其开发团队的自治性,应尽量避免.

3.3.2 计算方法

基于 3.3.1 的分析,微服务设计中,1)每个服务实现的用例应尽可能少;2)实现单个用例涉及的服务应尽可能少.为了评估 1),计算微服务设计中服务实现的用例平均数.为了评估 2),计算设计中实现每个用例涉及的服务平均数. 微服务划分的用例收敛性用公式表示如下:

$$\frac{1}{s}\sum_{k=1}^{s}U_k+\frac{1}{u}\sum_{l=1}^{u}S_l \quad \text{(式 7)}$$

其中:

s 表示当前微服务设计中的服务数;

U_k 表示服务 k 中对应的用例数;

u 表示当前微服务设计中的用例数;

S_l 表示实现用例 l 涉及的服务数.

3.4 实体收敛性

实体收敛性指代单个服务聚焦于处理特定领域实体的程度.

3.4.1 指标有效性分析

微服务倾向于让每个服务管理自己的数据库[19],包括对领域实体数据的增加、删除、更改和查询.为满足

单一职责原则,降低服务的数据库管理负载,单个服务管理的实体数应尽可能少.

另一方面,共同闭包原则（Common Closure Principle, 简称 CCP）——“the packages should not have more than one reason to change.”——强调从变化的角度看待软件架构设计,可用于提高微服务设计的可维护性.具体来说,访问同一实体的用例趋向于划分到同一服务.这样当某一领域实体发生需求变更时,需要做出修改的服务尽可能少,带来的服务间耦合也尽可能小.

3.4.2 计算方法

基于 3.4.1 的分析,微服务设计中,1)每个服务管理的实体数据应尽可能少;2)同一实体数据涉及的服务应尽可能少.为了评估 1),计算微服务设计中服务管理的实体平均数.为了评估 2),计算设计中实体涉及的服务平均数.微服务划分的实体收敛性用公式表示如下:

$$\frac{1}{s}\sum_{k=1}^{s}E_k+\frac{1}{e}\sum_{l=1}^{e}S_l \tag{式 8}$$

其中:

s 表示当前微服务设计中的服务数;

E_k 表示服务 k 中管理的实体数据数;

e 表示当前微服务设计中的实体数;

S_l 表示实体 l 涉及的服务数.

3.5 指标合并

合理的微服务划分要求设计者在多个相互排斥的性质之间权衡利弊[4].减少服务实现的用例数以降低微服务粒度,使微服务设计更符合单一职责原则.一方面,这不仅提高了服务内聚性,同时还降低了对应开发团队的工作强度.另一方面,这带来了更多的服务间数据流,增加了服务间的耦合与依赖.在划分微服务时,需要更多数据交互的用例及其数据应趋向于被划分到同一个服务内.另外,当微服务划分具有良好的实体收敛性时,服务需要接收更多的服务间消息才能完成同一用例,服务间的耦合度较高.而当微服务划分具有较小的用例收敛性时,一方面要求服务包含较少的用例,另一方面要求用例涉及的输入输出属性尽量包含在更少的服务中,这就与较小的实体收敛性目标,即要求每个服务包含的实体较少发生了冲突.根据上述分析,我们需要合并计算得到的所有指标以得到对微服务划分的综合评分.假设当前存在 n 个不同的微服务划分,对每个划分都能通过前文所述的公式得到 4 个指标, 从而得到指标矩阵 M , M 的每一列分别对应服务内聚性、服务耦合性、实体收敛性和用例收敛性:

$$M=\begin{bmatrix} m_{coh1} & m_{cop1} & m_{ucc1} & m_{ec1} \\ m_{coh2} & m_{cop2} & m_{ucc2} & m_{ec2} \\ \cdots & \cdots & \cdots & \cdots \\ m_{cohn} & m_{copn} & m_{uccn} & m_{ecn} \end{bmatrix} \tag{式 9}$$

由于每种指标表示的含义不同,其量纲和取值范围也存在很大的差别,将不同指标直接累加可能造成某一维指标对结果影响过大.因此,为了更公平地合并不同的指标,我们需要对指标计算结果进行归一化处理,使所有指标数据的归一化结果在[0,1]之间.另外,为了使加权结果越大与微服务设计越合理的目标相统一,在归一化过程中还应考虑到指标本身.根据前四小节的分析,更合理的微服务划分具有更高的服务内聚性,更低的服务耦合性、用例收敛性以及实体收敛性.

(1) 对于划分目标为更高的指标 i ,对 m_{ij} 做如下归一化处理:

$$m_{ij}{}'=\frac{m_{i\max}-m_{ij}}{m_{i\max}-m_{i\min}} \tag{式 10}$$

(2) 对于划分目标为更低的指标 i ,对 m_{ij} 做如下归一化处理:

$$m_{ij}^{'}=\frac{m_{ij}-m_{i\min}}{m_{i\max}-m_{i\min}} \quad (式\ 11)$$

其中, $m_{i\max}$ 和 $m_{i\min}$ 分别是指标 i 在 n 个不同划分结果下的最大值和最小值.

接下来,对归一化处理后的指标进行加权以获得综合评分.显然,不同的软件系统对不同指标的优先级要求不同.比如,对于具有更高性能要求的软件系统,耦合性对系统设计非常关键,服务耦合性指标应具有更高的权重.相反,更强调可复用性的软件系统应增加服务内聚性和实体收敛性在评估微服务划分工作中的权重.假设各指标的权重向量为 $W=\{w_1,w_2,w_3,w_4\}$,那么候选微服务 k 的综合评分为:

$$\sum_{i=1}^{4} m_{ik}\times w_i \quad (式\ 12)$$

其中, $w_{coh}+w_{cop}+w_{ucc}+w_{ec}=1$.

4 工具原型及案例研究

4.1 工具原型

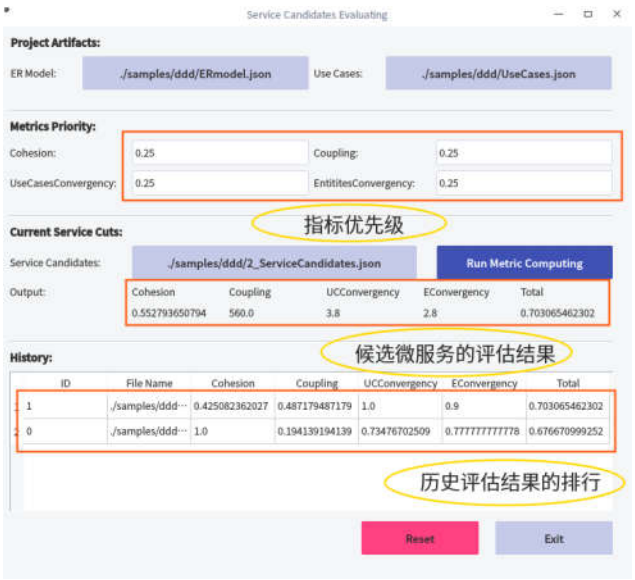

Fig.3 Interface of the SCE system prototype

图 3 SCE 系统界面

为了验证提出的微服务划分的评估模型,我们实现了 Service Candidates Evaluating(SCE)工具原型,自动化地计算候选微服务于服务内聚性、服务耦合性、用例收敛性和实体收敛性四个方面的指标并给出综合评分以比较不同微服务划分的优劣.SCE 的实现基于 Python2.7.13 与 Gui 编程技术 PyQt5.图 3 描述了该 SCE 的图形化用户接口,主要包括以下几个方面:软件制品与候选微服务的输入,指标优先级的输入,当前候选微服务的评估结果输出以及历史评估结果的排行输出.SCE 规定软件制品与候选微服务的输入应严格遵循特定格式的 JSON 文件形式.

4.2 案例研究及分析

本节,我们将评估模型应用于领域驱动设计中的经典场景——货物跟踪系统(Cargo Tracking System,简称CTS)——以阐述评估模型的具体使用过程并验证模型的有效性.利用评估模型对 CTS 案例下的几种微服务划分方案进行评估,若评估结果排序与微服务架构设计人员的预期相符,则认为评估模型有效.

4.2.1 业务场景

CTS 是 E.Evans[6]用于阐述领域驱动设计的经典案例,1)具有合适的问题复杂度;2)具有完整且合理的架构分析;3)具有公认合理的限界上下文划分方案;4)GitHub 上具有已实现的完整代码:DDDSample. 对 DDDSample 进行逆向分析, 不难得到 CTS 的业务需求如下:

(1) 将 Cargo 从 Location A 运送到 Location B.每个 Cargo 创建时都伴随着一个 TrackingId, Cargo 的具体说明用类 RouteSpecification 来实现.一旦 Cargo 被创建,一到多个 Itinerary 将分配给该 Cargo;

(2) 系统通过计算现有的 Voyage 给 Cargo 分配合适的 Itinerary,每个 Voyage 都包含一系列的 CarrierMovement;

(3) 在 Cargo 的路线确定以后,HandlingEvent 跟踪 Cargo 的每个 Itinerary 完成货物跟踪;

(4) Cargo 的 Delivery 实例描述了具体的运输状态,预计到达时间以及该 Cargo 当前是否被跟踪.

进一步分析可得到 CTS 的用例图和实体关系图. 同时,得到 CTS 的业务用例和领域实体如下:

(1) 业务用例:View Tracking, View Cargo, Book Cargo, Change Cargo Destination, Route Cargo, Create Location, Create Voyage, Add Carrier Movement, Handle Cargo Event;
下文使用的缩写形式为:VT, VC, BC, CCD, RC, CL, ACM, HC.

(2) 领域实体:Cargo, HandlingEvent, Delivery, Voyage, RouteSpecification, Location, Itinerary, CarrierMovement, Leg.
下文使用的缩写形式为:Car, HE, Del, Voy, RS, Loc, Iti, CM, Leg.

表 3 给出 CTS 系统中业务用例与领域实体之间的对应.

Table 3 Use cases of CTS

表 3 CTS 的业务用例及其领域实体

用例名称	读取实体	写入实体
ViewTracking	Cargo HandlingEvent Delivery Voyage RouteSpecification	-
ViewCargo	Cargo RouteSpecification Delivery Itinerary	-
BookCargo	Location	Cargo RouteSpecification
ChangeCargoDestination	Cargo RouteSpecification	RouteSpecification
RouteCargo	Crago RouteSpecification Location Voyage CarrierMovement	Itinerary Leg
Create Location	-	Location
Create Voyage	-	Voyage
AddCarrierMovement	Voyage	CarrierMovement
HandleCargoEvent	Voyage Cargo	HandlingEvent Delivery

4.2.2 模型验证

接下来,通过验证 CTS 案例下 5 个候选微服务在评估模型中的表现来验证提出的微服务划分评估模型的有效性.5 个候选微服务的具体细节如表 4.

Table 4 Microservices candidates of CTS application

表 4 CTS 应用程序中的候选微服务

候选微服务	服务	用例	实体
SC0	S00	VT	HE,Del
	S01	VC	Car,RS,Iti
	S02	BC	RS,Car
	S03	CCD	RS,Car
	S04	RC	Leg, RS, Car
	S05	CL	Loc
	S06	CV	Voy
	S07	ACM	CM
	S08	HCE	HE,Del
SC1	S10	VT, VC, BC, CCD, RC,CV,HCE,ACM,CL	RS,Car,Leg,Del, Voy, Iti, HE,CM,Loc
SC2	S20	CCD,CV	RS,Car
	S21	HCE,VT	HE, Del
	S22	ACM	CM, Voy
	S23	CL, BC	Loc
	S24	VC,RC	Iti,Leg
SC3	S30	VC, HC	HE, Del
	S31	RC,VC	Iti, Leg
	S32	BC, CCD	Car, RS
	S33	CV, ACM	Voy, CM
	S34	CL	Lo
SC4	S40	VC, BC, CCD, RC	RS,Car,Leg, Del,Iti
	S41	VT,HCE	HE
	S42	ACM,CV	CM,Voy
	S43	CL	Loc

候选微服务 SC0 到候选微服务 SC3 在设计时仅考虑评估模型的一项指标.在候选微服务 SC0 中,每个服务仅负责一个用例的开发与实现,服务的内聚性最大.在候选微服务 SC1 中,服务 S10 将系统中所有业务用例与领域实体都组合在一起,实现任意用例都只需访问本地领域实体,无服务间消息交互,具有最小的服务耦合性.候选微服务 SC2 从用例收敛性的角度出发, 在保证服务实现的用例平均数较小前提下,尽可能将实现同一业务用例涉及到的领域实体划分到相同服务.比如,实现业务用例 ViewTracking 涉及到的领域实体有 Cargo, HandlingEvent, Delivery, Voyage 和 RouteSpecification,若将此 5 个领域实体(CTS 共有 9 个领域实体)都划分到同一服务,则违背服务实现的用例平均数较小目的.注意到,领域实体 Cargo 和 RouteSpecification 在业务用例 ViewTracking, ViewCargo, BookCargo, ChangeCargoDestination 以及 RouteCargo 中皆有涉及,因此将领域实体 Cargo 和 RouteSpecification 划分到一个服务中.候选微服务 SC3 中, 在保证服务管理的实体平均数较小的前提下,尽可能将处理相同领域实体的业务用例划分到同一服务,划分方式与 SC2 同理.SC4 对领域实体的划分与领域驱动设计对 CTS 的划分一致.在领域驱动设计中,E.Evans 根据限界上下文方法将 CTS 划分为 4 个服务:1)Voyage 服务负责所有航运的信息管理,包含领域实体 Voyage 和 CarrierMovement;2)系统中的地点数据具有“多读少写”的业务特点,由 Location 服务负责管理,包含领域实体 Location;3)Planning 服务负责管理系统中所有的货物及其航段信息,包含领域实体 Cargo, RouteSpecification, Itinerary 和 Leg;4)Tracking 服务用于跟踪货物的具体事件,包含领域实体 HandlingEvent.SC4 实现了在 4 个指标之间的权衡.

由于 SC4 是公认合理的限界上下文划分方案,而 SC0 到 SC3 都仅考虑了 4 个评估指标中的一项,因此预计在评估模型给出的评估结果中候选微服务 4 即 SC4 应具有最好的综合评分.

在运行 SCE 进行微服务划分评估时,根据 CTS 的项目需求指定 4 个指标的优先级参数为[0.25,0.25,0.25,0.25].运行 SCE,对 5 个候选微服务分别计算其指标,得到如下指标矩阵 M(M 的每一行对应 SC0 到 SC4):

$$M=\begin{bmatrix} 1.0 & 336.0 & 5.444 & 5.111 \\ 0.234 & 0.0 & 10.0 & 10.0 \\ 0.457 & 1140.0 & 4.02 & 4.13 \\ 0.596 & 920.0 & 3.911 & 3.911 \\ 0.736 & 1080.0 & 4.028 & 4.028 \end{bmatrix} \quad (式\ 13)$$

对指标计算结果进行归一化处理得到矩阵 M' 如下.分析矩阵 M' 发现,候选微服务 SC0 具有最大的服务内聚性,SC1 具有最小的服务耦合性,SC2 具有较小的用例收敛性,SC3 具有最小的实体收敛性,与此前的服务设计的分析一致.

$$M'=\begin{bmatrix} 1.0 & 0.112 & 0.748 & 0.793 \\ 0.016 & 1.0 & 0.0 & 0.0 \\ 0.302 & 0.699 & 0.982 & 0.951 \\ 0.480 & 0.757 & 1.0 & 0.987 \\ 0.660 & 0.714 & 0.981 & 0.968 \end{bmatrix} \quad (式\ 14)$$

再按照优先级参数对 M' 做加权求和后,得到 5 个候选微服务的综合指标为[0.663,0.254,0.733,0.806,0.831].候选微服务 SC4 由于权衡了微服务划分的多个方面,具有最好的综合指标结果.综合指标的计算结果与预想一致,一定程度上说明了评估模型的有效性.

4.2.3 与 Service Cutter 对比

由于微服务领域缺少评估微服务划分的系统化工作,我们选择划分微服务的工具原型 Service Cutter,使用本评估模型对 CTS 在 Service Cutter 中的几种划分结果进行评估,将评估结果与 Service Cutter 的人为评估结果进行比较.

Service Cutter 的微服务划分过程包括 4 个步骤:系统定义(System Definition),系统说明(System Specification),服务划分(Service Decomposition)和分析服务划分结果(Analyze Service Cuts).在利用 CTS 进行微服务划分的模型验证时,Michael Gysel 等[9]同样通过对 DDDSample 的逆向分析得到相关用户描述(User Representation),并以此作为系统定义和系统说明步骤的输入,得到的用户描述包括用例图、实体关系图、nanoentities 的内容易变性和结构易变性特征等.Service Cutter 通过逆向分析得到的用例图和实体关系图与 4.2.1 基本一致.除用例图和实体关系图,Service Cutter 分析 DDDSample 得到的部分用户描述列举如表 5.Service Cutter 对 Location 的内容易变性与结构易变性的定义与本文 4.2.2 部分的讨论相一致.在服务划分步骤中,M.Gysel 等还根据 CTS 的需求特性对 16 种耦合度度量标准的优先级进行了调整,以使微服务划分结果更接近预期结果.

Table 5 Part of user representations of CTS application in Service Cutter

表 5 Service Cutter 得到的 CTS 部分用户描述

用户描述 / 易用性	内容易变性	结构易变性
Often	HE.type HE.completionTime HE.registrationTime HE.location Del.transportStatus	-
Rarely	Loc.unLocode Loc.name	Loc.unLocode Loc.name

以用户描述和调整后的耦合度度量标准的优先级作为输入,Service Cutter 在划分 CTS 系统时调整聚类算

法得到了两种划分方案,具体细节如表 6.

Table 6 Microservices candidates of CTS application in Service Cutter

表 6 Service Cutter 对 CTS 应用程序划分得到的候选微服务

候选微服务	服务	用例	实体
CUT0	S00	VC, BC,CCD, RC, CV	RS, Car, Leg, Iti, Voy, Del
	S01	HCE, VT	HE
	S02	CL	Loc
	S03	ACM	CM, Loc
CUT1	S10	VC, BC, CCD, RC,CV, ACM	RS,Car,Leg, Iti, Voy,CM
	S11	HCE, VT	HE, Del
	S12	CL	Loc

Service Cutter 对划分结果的讨论认为,CUT0 将领域实体 Location 划分到两个服务,并且依据划分结果与预期结果的一致程度判定 CUT0 为“坏”划分,CUT1 为“可接受”的划分.运行 SCE 得到本评估模型对两种划分方案的评估结果如下:

Table 7 Evaluation of two decompositions in Service Cutter by SCE

表 7 SCE 对 Service Cutter 两种划分方案的评估结果

候选微服务	服务内聚性	服务耦合性	用例收敛性	实体收敛性	综合评分
CUT0	0.706	1120	4.138	4.278	0.800
CUT1	0.598	350	4.556	4.556	0.787

SCE 的运行结果显示微服务 CUT0 具有更高的综合评分,与 Service Cutter 的人为评估结果不一致.尽管候选微服务 CUT1 的服务耦合性更小,但由于 CUT1 中只有 3 个服务,每个服务实现的业务用例和管理的领域实体更多,因此用例收敛性和实体收敛性都更大.并且,对比候选微服务 CUT0 和 CUT1 的具体划分,CUT0 将业务用例 AddCarrierMovement 划分出来成为一个服务,使服务 S00 更符合 SRP 原则,提高了划分的服务内聚性.从服务内聚性,服务耦合性,用例收敛性和实体收敛性四个方面综合考虑,CUT0 对应的微服务划分更合理.与本模型相比,Service Cutter 对划分结果的人为评估具有更强的主观性.

5 总结及下一步工作

微服务和 DevOps 在缩短交付周期、提高团队自治性等特性方面是相辅相成的.然而,合理的微服务粒度是微服务一切特性的前提,也是微服务领域的一项难题.本文从限界上下文视角提出一种微服务划分粒度的评估模型.首先,针对微服务划分问题给出完整的评估过程;然后,结合微服务划分的核心原则包括高内聚、低耦合等设计 4 项评估指标以量化评估过程;其次,引入指标合并得到微服务划分的综合评分以比较不同微服务划分方案;再次,实现工具原型 SCE 自动化地评估微服务划分;最后,比较评估结果与架构师心理预期验证模型的有效性.

目前,该方法仍存在如下待改进的问题:首先,评估模型仅考虑了微服务划分的 4 个优化目标,评估结果不够精确,评估过程考虑的因素不够全面,后续工作将围绕微服务架构本身及限界上下文讨论更多的微服务划分原则,并结合实际工程项目完善评估模型;其次,工具原型 SCE 要求架构设计人员将软件制品转化为格式规范的 JSON 文件作为输入,后续工作考虑将 SCE 集成到微服务工具链,使图形化软件制品直接转化为 SCE 输入,减少架构设计人员的工作负担.

References:

[1] Miika Kalske, Niko Mäkitalo, Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. Current Trends in Web Engineering, 2018, 32-47.

[2] S Newman. Building microservices: designing fine-grained systems. 2th ed., the United States of America: O'Reilly Media, 2015. 2-3,9,16,32,47.

[3] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, Stefan Tilkov. Microservices: The Journey So Far and Challenges Ahead. IEEE Software, 2018, 35(3), 24-35.

[4] Qian Ma, Nianjun Zhou, Yanfeng Zhu, Hao Wang. Evaluating Service Identification with Design Metrics on Business Process Decomposition. IEEE, International Conference on Services Computing. 2009,160-167.

[5] Sara Hassan, Rami Bahsoon. Microservices and their Design Trade-offs: A Self-Adaptive Roadmap. IEEE International Conference on Services Computing, 2016.

[6] E. Evans. Domain-Driven Design: Tracking Complexity in the Heart of Software. Pearson Education, 2003:30.

[7] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu, Zhiming Liu. Identifying Microservices Using Functional Decomposition. Dependable Software Engineering. Theories, Tools, and Application, 2018, 50-65.

[8] C.Richardson. Pattern: Micro service Architecture (22 June 2017). http://microservices.io/patterns/microservices.html.

[9] Michael Gysel, Lukas KÖlbener, Wolfgang Giersche, Olaf Zimmermann. Service Cutter: A systematic approach to service decomposition. Springer, Service-Oriented and Cloud Computing, 2016, 185-200.

[10] Luciano Baresi, Martin Garriga, Alan De Renzis. Microservices Identification through Interface Analysis. European Conference on Service-oriented & Cloud Computing, 2017,19-33.

[11] Rui Chen, Shanshan Li, Zheng Li. From Monolith to Microservices: A Dataflow-Driven Approach. IEEE, Asia-Pacific Software Engineering Conference, 2018,466-475.

[12] Uwe Zdun, Elena Na, varro, Frank Leymann. Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns. Service-Oriented Computing, 2017, 411-429.

[13] Bran Selic. Tutorial: an overview of UML 2.0. International Conference on Software Engineering, 2004, 1069-1070.

[14] Mohammad Daghaghzadeh, Ahmad Baraani Dastjerdi, Hossein Daghaghzadeh. A Metric for Measuring Degree of Service Cohesion in Service Oriented Designs. International Journal of Computer Science Issues, 2011, 83-89.

[15] Abdelkarim Erradi, Naveen Kulkarni, Piyush Maheshwari. Service Design Process for Reusable Services: Financial Services Case Study. Springer, International Conference on Service-Oriented Computing, 2007, 606-617.

[16] Toby J. Teorey, Guangping Wei, Deborah L. Bolton, John A. Koenig. ER Model Clustering as an Aid for User Communication and Documentationin Database Design. Communication of the ACM,1989,975-987.

[17] Peter Jaeschke, Andreas Oberweis, Wolffried Stucky. Extending ER Model Clustering by Relationship Clustering. International Conference on Conceptual Modeling.1993,451-462.

[18] Jacky Akoka, Isabelle Comyn-Wattiau. Entity-Relationship and Object-Oriented Model Automatic Clustering. Data & Knowledge Engineering Journal.1996, 87-117.

[19] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen. Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media, 2016,18.

[20] Robert Cecil Martin. Agile software development: principles, patterns, and practices. Prentice Hall PTR, 2003.

附中文参考文献:

[2] Sam Newman.微服务设计.人民邮电出版社,2016, 2~3,9,16,32,47.

[6] E. Evans.领域驱动设计——软件核心复杂性应对之道.清华大学出版社,2003,30.