计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200700181

微服务时代的复杂服务软件开发

吴文峻　于　鑫　蒲彦均　汪群博　于笑明
北京航空航天大学计算机学院　北京 100191

摘　要　微服务时代的软件系统变得越来越复杂,传统的软件开发理论、方法和技术不再适用。面向复杂服务软件开发的过程,微服务架构有着可扩展性强、灵活性高的优点,同时对运维能力和服务管理能力提出了更高的要求,需要借鉴群体智能的研究思路和方法,直面复杂软件系统开发过程中面临的一系列挑战。文中以复杂系统和群体智能的方法论为指导,提出以微服务架构实现智能服务适配,开发复杂服务软件系统的技术路线,阐述复杂服务软件的自适应架构、模型框架、开发技术和典型支撑工具,并通过共享出行的案例分析加以具体解释。

关键词:复杂系统;群体智能;微服务架构
中图法分类号　TP311

Development of Complex Service Software in Microservice Era

WU Wen-jun,YU Xin,PU Yan-jun,WANG Qun-bo and YU Xiao-ming
School of Computer Science and Engineering,Beihang University,Beijing 100191,China

Abstract　Due to increasingly complexity of software systems in the era of microservices,traditional software development methods and techniques are no longer applicable. With the advantages of strong scalability and high flexibility towards the development process of complex service software,the microservice architecture puts forward higher requirements for the capabilities of service operation and maintenance as well as service management. To tackle these challenges,this paper utilizes the research approaches of collective intelligence to explore new paradigms for building complex software systems. Guided by the methodology of complex systems and collective intelligence,this paper proposes a new technical approach for development of complex service software systems based on microservice architecture. It elaborates the major ideas in such an approach including the adaptive software architecture,modeling framework,development technologies and typical supporting tools. Moreover,it presents a case study to explain how to apply such an approach in the realm of ride sharing.

Keywords　Complex system,Crowd intelligence,Microservice architecture

1　引言

人们生活中的软件系统变得越来越复杂,这种由相当数量的自治异构系统耦合关联而成的系统被称为复杂软件系统。这类系统的复杂度打破了传统软件工程的还原论思想,传统的软件开发理论、方法和技术不再适用于复杂软件系统的开发。传统的软件架构有开发效率低、代码维护难、灵活性低、扩展性差的缺点,微服务架构的出现为复杂软件系统带来了可扩展性、灵活性等多种优点,但对微服务的管理和运维提出了更高的要求。我们需要从社会-技术生态系统的角度来认识现今的复杂系统,借鉴群体智能的研究思路和方法,突破传统软件工程方法的局限,直面复杂软件系统的构造、部署、运行、维护、演化和保障所面临的一系列现实挑战,以类似社会系统形成、成长和演进的思想来认识复杂软件开发过程,实现高度智能化的复杂服务适配。

本文首先介绍复杂软件系统和微服务架构的研究现状;然后分析复杂系统、群体智能系统以及微服务架构之间的内在联系,进而提出微服务时代复杂服务系统的开发方法——智能服务适配,包括软件架构、建模框架、开发技术和支撑工具等;最后以共享出行的案例加以具体解释。

2　复杂软件系统、群体智能、微服务

现实生活中存在的复杂系统具有以下特点:系统要素不仅包括信息系统,而且涉及大量与信息系统耦合关联的社会系统和物理系统;系统要素之间的耦合关联关系动态变化且日趋复杂;整个系统的行为难以通过单一自治系统特征的简单叠加来刻画;在不断适应组织和环境变化的过程中逐步成长演化而成。图1描述了复杂系统的特点。

到稿日期:2020-07-27　返修日期:2020-10-22　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划项目(2018YFB1402800)
This work was supported by the National Key R&D Program of China (2018YFB1402800).
通信作者:吴文峻(wwj@nlsde.buaa.edu.cn)

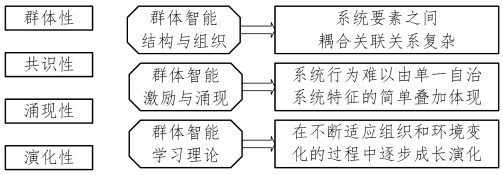

图 1 复杂系统的性质及与群体智能的关系

Fig. 1 Properties of complex systems and their connections with collective intelligence

2.1 群体智能与复杂软件系统

群体智能最早源于对蚂蚁、蜜蜂等社会性昆虫群体行为的研究，这些昆虫群体有相应的结构与组织，能够通过简单的规则涌现出群体性的智慧，同时具有一定的学习能力来适应环境的变化。群体智能系统的本质是动态认知复杂网络，由涌现强弱决定网络演化的复杂程度[1]。

不难发现，现实中的复杂系统是一种群体智能系统，例如系统要素之间的耦合关联构成了群体智能系统的结构与组织，系统行为难以通过单一自治系统特征的简单叠加来体现群体智能的涌现，而复杂系统在不断适应组织和环境变化的过程中逐步成长，体现了群体智能系统的学习能力。我们可以通过群体智能的观点，来观察和理解建立在社会-技术生态中的复杂软件系统，构造出具有适应环境能力和需求变化能力的软件[2]。

群体智能的研究范式一般先通过组织调控与行为预知，来实现从海量个体到有机群体的形态转化，然后利用激励机制与协同决策来达成从有机群体到智慧、涌现的有效转变，即建模-激励-决策的层次式研究思路。通过分析群体智能与复杂软件系统之间的联系，能够借鉴群体智能的研究方法和思路，对复杂软件系统进行建模、仿真、优化[3]。

2.2 微服务技术与复杂服务软件系统

微服务是近年来出现的新型软件架构，其核心思路是将复杂的整体分解为可以独立部署和维护的小型服务模块。这种从单体软件解决方案到微服务分布式结构的转变，从根本上改变了传统软件的开发、部署和管理方式。相比传统软件架构，微服务架构能够更有效地利用计算资源，更快地更新服务。

复杂服务软件系统包含大量微服务，这些微服务的实现语言可能不同，所属的租户可能不同，并且有大量状态不断变化的服务实例。在这样的动态环境中，调试和管理大规模微服务系统是一项艰巨的任务。复杂服务软件系统无法通过一次开发部署直接造就完成，而是需要借鉴自然界中群体智能的研究方法，将复杂软件系统划分为大量适应性软件单元，使得每个软件单元具有感知、决策、执行的能力，在功能和性能等方面通过组合关联和调控适应来逐步涌现服务群体的适配态势，并推动这种适配态势不断适应社会和物理环境的变化，并通过适配的成长演化走向成熟稳定的状态。

2.3 复杂服务软件系统建模

如图 2 所示，复杂服务软件系统可以分为服务业务层与服务平台层。服务业务层包括服务提供者群体、服务消费者群体、服务监管者群体，每个群体有各自提供业务功能的服务主体。

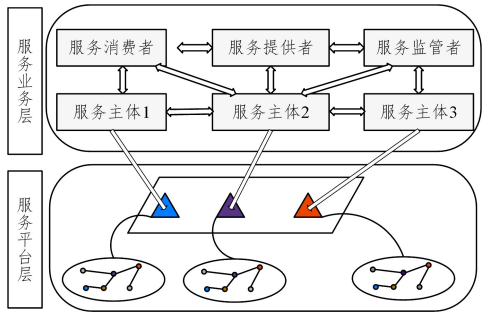

图 2 复杂服务软件系统的分层架构

Fig. 2 Architecture of complex service software systems

研究人员针对现代共享服务业提出了多主体的回路服务关系模型，适合刻画服务业务层相关的诸多要素，包含提供主体、消费主体、第三方平台、服务过程、服务目标和服务价值等[4]。本文第 4 节将以共享出行业务为例进一步介绍该分层结构。共享出行业务流程涉及网约车平台、司机以及乘客，它们构成了服务提供者、服务消费者以及服务监管者，业务流程中涉及到的主体构成了服务业务层。

这些服务主体都对应服务平台层的若干微服务群体，而所有微服务群体通过交互组合构成了服务平台层。业务层与平台层的服务主体和海量微服务有各自的组织与结构，同时每个主体和微服务也是自治的个体，构成了整个复杂服务系统。在这样一个复杂系统中，需要借鉴群体智能系统的视角，通过建模-仿真-优化的群智研究范式，支撑微服务运维实时监测与保障，找到海量微服务之间功能和性能的适配方案。

3 复杂服务软件开发方法-智能服务适配

本文提出的面向复杂服务软件的智能服务适配方法是将人工智能技术与传统开发方法相结合的一种新型软件开发方法，旨在使开发以及运维人员能够有效构建并操作大规模在线服务和应用程序。智能服务适配可以分为两个方面：1）分析设计方面，包括服务需求分析、服务发现与推荐、服务组合编排；2）演化运行方面，包括服务部署、服务演化、服务运行。

针对复杂服务软件的智能适配，主要需要解决异构服务的智能匹配问题以及服务演化和运行时的质量保障问题。在服务分析和设计方面，由于异构服务描述不精确、不一致，需要实现服务供需的精准匹配、按需组合、适应优化，来保障服务适配的正确性。在服务版本演化和动态运行方面，由于服务版本众多、更新频繁、适配态势不断变化，因此需要及时感知态势，准确评估效能，从而实现自主可靠管理，保障服务适配质量。

3.1 复杂服务软件架构

传统的基于 WS-* 框架的服务软件需要严格的接口定义，其组合模式固化，需要人工来实现更新，运维机制被动，使得建模组合过程繁杂，更新运维代价高昂[5]。面对复杂服务软件系统，其难以实现海量服务的精准匹配，难以管理复杂的动态演化适配。微服务时代的复杂服务软件开发方法，需要在微服务网格（Service Mesh）这一基础架构层，赋予软件单元适应性演化和成长性构造的能力。

Service Mesh 的 Sidecar 模式提供了设计适应性软件的

基础，该模式将应用程序的功能划分为单独进程，无需额外组件配置和代码就能为应用程序添加其他功能，Sidecar 应用与主应用程序松散耦合，为其扩展服务路由、流量控制和监控等增强功能。服务网格中存在数据平面和控制平面：数据平面的职责是处理网格内部服务之间的通信，并负责服务发现、负载均衡、流量管理、健康检查等功能；控制平面的职责是管理和配置 Sidecar 代理以实施控制策略，并收集遥测数据。Service Mesh 抽象封装了服务实例所需的资源管理接口、消息路由接口、流量控制接口、监控测量接口，每个微服务都具有监控日志、配置流量等功能[6]。

适应性软件单元一般应具有感知、决策、执行的能力，基于 Service Mesh 的自适应软件框架基本具备了上述能力。自适应环境中的智能模型和算法可以被抽象出来，进行独立的建模、学习、演化，使之更加适应系统运行的内外因素变化。基于 Service Mesh 的自适应软件框架如图 3 所示，其中软件框架的 Service 代码和自适应模型都可以通过 DevOPS 的开发模式进行持续的集成-部署-更新。

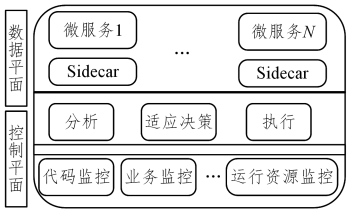

图 3 基于 Service Mesh 的自适应软件框架

Fig. 3 Self-adaptive software framework based on Service Mesh

与传统的服务架构相比，基于 Service Mesh 的自适应软件框架具有以下优势：

(1)微服务软件之间的互联调用可以按需动态进行，通过智能路由形成松耦合和灵活适配的微服务互联网络。

(2)微服务服务调用可以根据资源和请求的情况进行智能路由和流量控制，保证功能调用的可靠性和可追溯性。

(3)微服务软件运行轨迹数据能够被系统地收集和监控，支撑感知-分析-决策-执行的自适应回路。这一自适应回路可以在微服务的部署交付、运行维护的多个层次得以体现。

3.2 复杂服务软件模型

复杂服务软件系统的服务适配聚焦于以智能化手段打通分析设计-部署交付-运行维护等阶段，依托自适应的复杂服务软件架构，来实现智能高效的软件开发目标。为此，需要定义基于群体智能的复杂服务软件模型，刻画组成该软件系统的各服务主体类别、服务主体的基本画像、服务主体之间的交互、服务主体运维的相关智能模型等。

针对图 3 所示的自适应软件框架，我们可以给出复杂服务软件模型的定义：

(1)一个复杂服务软件系统应该由多类主体(Agent)组成，这些主体有的是自然主体，更多的是由软件构成的服务主体。各个主体的交互和组合关系通过多主体系统建模框架结合 BPMN 流程建模来实现。

(2)每个服务主体由一个服务智能管理模块和若干微服务实例组成。

服务主体的基本要素可以定义为五元组(G, KPI_b, M_b, KPI_s, M_s)，其中微服务关系图 G 代表该服务主体所包含的微服务实例和它们之间的调用关系。$G=(S,E)$，其中 S 是服务主体包含的所有微服务集合，每个节点 s 代表一个微服务以及它的画像，包括服务输入与输出接口、服务行为描述、服务语义描述；E 是微服务关系图 G 所有边的集合，每个边 e 代表两个服务之间的调用关系。

服务智能管理模块涉及到 KPI_b, M_b, KPI_s, M_s 这 4 个要素，M_b 和 KPI_b 定义了服务主体 Agent 的业务层模型和核心测量指标，这些指标和服务主体的业务领域密切相关。M_s 和 KPI_s 定义了服务主体的服务层模型和核心测量指标，这些模型包括执行 QoS 评估分析模型、资源调度与任务分配模型、服务告警和根因定位模型等，核心测量指标往往包括服务实例运行和通信的实时参数、底层容器的各类资源指标(CPU 计算负载、内存、网络等)。服务智能管理模块运行业务层和服务层的智能模型，使得整个服务主体达到 KPI_b 和 KPI_s 的指标要求。

3.3 复杂服务软件开发技术

在复杂服务软件系统的开发阶段，需要相应的支撑工具来帮助设计者、开发者、运维者、管理决策者共同协作，保障复杂服务软件系统的功能和性能的稳定性，满足软件服务的质量目标和功能需求。若要支撑复杂服务软件适配，需要在分析设计、部署更新、运行维护 3 个阶段引入新的技术和相应的软件工具集，包括服务需求建模、服务语义建模、服务适配组合三大类。它们的相互关系如图 4 所示。

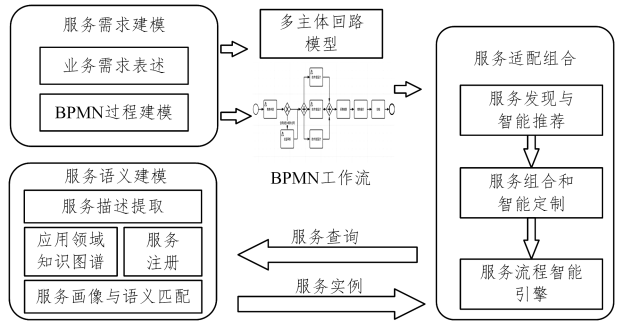

图 4 复杂服务软件的分析设计

Fig. 4 Analysis and design of complex service software

3.3.1 分析设计阶段

分析设计阶段的主要目标包括以下 3 个方面：构造反映服务需求的多主体回路模型，表达与业务相关的服务流程；汇聚多方面的数据来聚合服务实例的语义信息，从而形成服务画像和服务关联的知识图谱；智能地组合适配微服务流程，支持微服务流程的任务规划和执行。

(1)服务需求建模。服务需求建模的目标是形成刻画复杂服务软件系统的多主体回路模型，该模型将根据 3.2 节中的模型定义，描述复杂服务系统涉及的 Agent 主体和各主体之间的交互通讯模式，涵盖复杂社会-技术系统中的相关因素，定义核心的业务指标集合 $KPI(b)$ 和业务模型 $M(b)$。对于每个服务软件 Agent，系统设计者可以根据 BPMN[7] 的规范，定义 Agent 功能的 BPMN 工作流模型，并规定 Agent 的主要服务流程和这些流程的 QoS 规范。除了采用传统的人工方式来实现该设计过程，对于一些相对简单的流程，可以基于自然语言处理技术并综合以往的业务流程数据，自动地生

成业务流程模型。每个 Agent 的 BPMN 流程模型需要最终转换为这个 Agent 拥有的微服务实例所组成的微服务流程，才能在 Service Mesh 环境中得以执行。BPMN 流程模型由若干业务活动节点组成，每个业务活动可以映射为一个或多个微服务实例，并进一步组合映射为微服务流程。服务组合适配工具集将根据 BPMN 的语义，在服务知识图谱中查询匹配的微服务实体，自动适配生成微服务实体的连接图，产生基于 YAML 的微服务流程执行代码。具体地，需要设计自然语言处理建模、BPMN 服务过程模型生成、BPMN 业务流程映射、BPMN 模型仿真和评价等工具。

(2)服务语义建模。服务语义建模的目标是针对大规模、多样化、关系复杂的服务元数据，精准而有效地表示并建模服务语义以支持智能服务适配及应用。服务语义建模需要形成 3.2 节所述的模型中微服务的描述信息，其不仅包含静态的语法、语义等信息，还包括各类动态行为属性，以刻画服务内部的复杂动态行为及其协同特点。与传统的人工构建语义表示和知识图谱不同，服务语义建模需要采用多源数据融合、人工语义标注、基于图神经网络的语义和知识图谱表示学习等新方法，从大量的非结构化服务描述数据中，自动构建面向服务适配的服务画像和语义知识图谱[8]。

(3)服务适配组合。服务适配组合的目标是支撑智能化的服务发现、推荐、组合和定制，以自动或半自动的方式生成复合式的微服务流程，完成服务主体所需要的 BPMN 业务逻辑，其具体包括以下研究内容。

首先，构造语义驱动的服务智能组合功能[9]，基于微服务的语义画像进行微服务的语义聚类，根据服务主体的 BPMN 流程的语义特征和结构特征，发现与其语义匹配的微服务集合，推荐与语义和结构特性相近的微服务组合片段，以辅助服务主体的建模和开发人员自动完成 BPMN 流程到微服务流程的映射实现。在流程组合过程中，具备自动插入微服务适配功能的能力，实现上下游微服务的智能衔接。

其次，根据服务主体中的业务核心指标集 KPI_b，把 BPMN 流程的 QoS 规范转化为微服务流程的 QoS 要求，并在部署和运行中予以保证和落实。为此，需要构造微服务流程规划与编排功能，对微服务组合方案进行动态和持续性管理，实现连续化的集成和编排部署，动态维护业务流程和微服务组合的功能一致性，支持服务流程的版本管理、动态演化和自动部署，并根据流程适配的功能要求、QoS 要求、成本约束要求，由逻辑方案生成可根据流程执行的物理方案，进行相应的流程优化。

3.3.2 交付更新阶段

交付更新阶段的主要目标是形成智能、高效、可靠的服务代码集成和交付演化流程，实现低耦合服务组件代码的快速更新封装、自动化测试，实现复杂服务交付演化过程的可审计、高可靠等需求。复杂服务软件由于微服务组件具有低耦合、高重用、迭代速度快的特点，已经无法与传统的软件开发集成交付流程相契合，需要采用图 5 所示的事件驱动的持续交付集成方法。

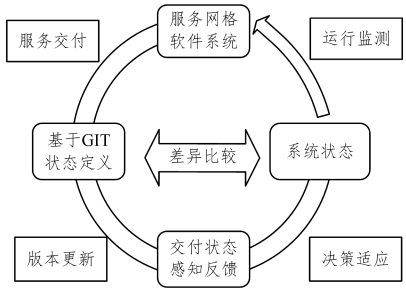

图 5 事件驱动的持续交付集成方法

Fig. 5 Event-driven continuous delivery integration method

如 3.1 节所述，微服务时代的复杂软件架构强调 Service Mesh 的基础设施和声明性容器编排方法。这一架构将复杂服务组件中描述系统相关的策略、代码、配置、版本控制等信息存放在软件仓库 Git 中，并以其作为核心形成一个交付流程操作的反馈和控制回路[10]。该控制回路持续地对比系统实际状态与 Git 中目标规定之间的差异，启动版本更新步骤。如果在预期时间内状态仍未收敛，则会触发告警事件，并上报差异，请求人工干预。这一闭环控制系统让交付流程具备了自愈能力，能够处理一些非预期操作造成的系统状态偏离。交付更新阶段可进一步划分为持续集成、持续部署、金丝雀部署以及灰度发布等具体步骤。

(1)持续集成。持续集成是以软件仓库为存储位置，共享代码和文档，通过集中编译生成目标容器镜像。典型的持续集成工具 GitLab CI[1] 是 GitLab 内置的持续集成工具，在仓库根目录下创建 CI 配置文件，并配置好 build 组件 GitLab Runner，便可实现各种提交场景下的自动构建。

(2)持续部署。持续部署是以容器镜像库为基础，在 YAML 文件的指导下，持续地把 Docker 镜像部署到目标环境(以 K8S 为代表的容器云平台)。典型的持续部署工具 Argo CD[2] 是遵循声明式 GitOps 理念的持续部署工具。它的应用定义、配置和环境信息是声明式的，并且能够进行版本控制，其应用部署和生命周期管理是全自动化并且可审计的。Argo CD 是独立的部署工具，支持对多个环境、多个 K8S 集群上的应用进行统一部署和管理。

(3)金丝雀发布与灰度发布。金丝雀部署是一种发布策略，与灰度发布是同一类策略。金丝雀策略只有一套系统，在应用更新的过程中新版本逐渐替换为旧版本。金丝雀发布与灰度发布的典型工具包括 Argo Rollouts[3]，Istio[4] 和 Flagger[5]。Argo Rollouts 是典型的面向 K8S 的渐进式服务发布工具，支持蓝绿的灰度发布和金丝雀部署。Istio 是一种完全开源的服务网格，作为透明的一层接入到现有的分布式应用程序中，拥有可以集成任何日志、遥测和策略系统的 API 接口[11]。Flagger 是一个开源的 Kubernetes operator，旨在解决持续交付需要复杂的管理以及部署失败而影响系统的可用性等问题，它使用 Istio 切换流量并通过监测指标分析业务应用在更新发布期间的状态表现。

1) https://docs.gitlab.com/ee/ci/

2) https://argoproj.github.io/argo-cd/

3) https://argoproj.github.io/argo-rollouts/

4) https://istio.io/

5) https://flagger.app/

复杂服务软件完整的适配演化流程如图 6 所示，包括灰度发布、持续部署、演化质量检验等一系列环节。其根据演化策略，逐步更新微服务流程部件，通常采取灰度发布方式，只更新少量服务实例，针对测试用户进行 A/B 测试，并通过金丝雀统计分析来判断演化的新版本是否稳定，并决定是否需要扩大部署的范围，或回滚到旧的版本。根据演化部署策略，分配必要的容器资源、消息路由和转发策略，生成所需的容器镜像，上传、更新和重启相关容器。同时，在持续部署中需要不断执行演化质量检验过程，对新部署案例的功能和性能、代码质量等进行综合评估。

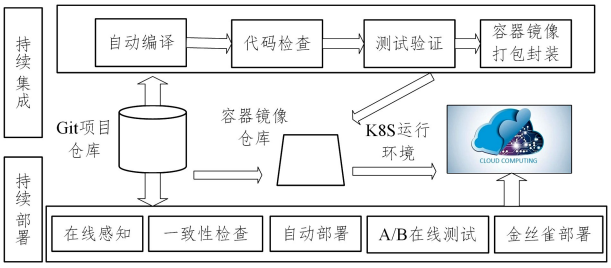

图 6 复杂服务软件的持续集成及持续部署

Fig. 6 CI-CD of complex service software

3.3.3 运行维护阶段

该阶段的核心目标是服务系统的整体效能评价与优化、服务可靠性的维护与保障。基于对服务执行轨迹的自动化追踪，服务智能适配的控制系统需要实现服务适配的动态监控、效能评测、智能调度、自动恢复等一系列智能运行维护的功能。这部分的开发技术是目前业界所提的 AIOps[12] 的重要组成部分，需要综合实时大数据处理、服务语义关联、数据驱动的机器学习和统计分析等基础工具，设计实现基于服务画像的多维度适配效能评价工具、服务任务智能调度和优化工具，以支持主动的服务质量管理和运行状态的优化。同时，通过实时数据采集和日志分析，引入全景关联信息挖掘技术，设计开发服务适配运行时的监测、诊断及故障处理等服务适配保障工具，以快速地识别故障原因并及时进行动态处置。

上述智能运行与维护工具主要建立在实时的流式大数据处理框架下（见图 7），基于 Service Mesh 的测量方法和运行日志生成海量的实时运行数据，实现分布式的运行轨迹追踪。这些数据经过处理，以事件流的方式不断分发给服务效能评估和质量保障模块、服务可靠性维护模块。这些智能模块需要采用在线持续机器学习、基于强化学习的资源调度和分配、面向时间序列异常检测、基于贝叶斯图的根因分析等智能学习模型，来实现相关智能调控分析与决策[13-14]。

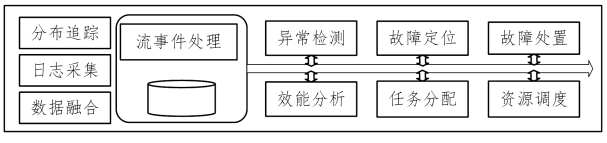

图 7 复杂服务软件的运行维护

Fig. 7 AIOps of complex service software

（1）运行监控和分布式追踪。基于日志和远程数据收集等方式，对业务指标集 KPI_b 和系统运行指标集 KPI_s 的相关时间序列进行不间断的数据采集和汇聚。

典型的运行监控工具 Prometheus[1] 是一套开源的系统监控框架，采用强大的多维度数据模型，支持灵活而强大的查询语句，目前在微服务平台中被广泛使用[15]。分布式追踪工具包括 Zipkin[2] 和 Jaeger[3]，其中 Zipkin 是最早的开源可扩展分布式追踪系统，Jaeger 工具自身没有实现测量模块，而是采用一组与 OpenTracing 标准兼容的开源追踪测量模块，用于监视和诊断基于微服务的分布式系统[16]。服务拓扑可视化工具 Kiali[4] 提供了请求路由、请求速率、请求延迟等信息的可见性，从而提供了从抽象应用程序层到具体服务和工作负载等不同级别的网格组件的可视化展示。

（2）服务质量保障与资源调度。根据运维 KPI_s 的 QoS 约束要求，基于 M_s 模型进行容器资源的动态分配和调度。当有新的微服务需要部署时，平台调度器会根据当前集群的节点状态及服务所需的具体资源来选择最优的节点。在微服务运行结束后会将分配给该服务的资源及时回收以便重新分配。同时，通过实时数据采集和日志分析，利用全景关联信息挖掘技术对服务的运行质量进行保障，当检测到服务的容器所在节点出现故障或资源不足时，可以智能地将容器调度到其他正常节点，从而保证服务的运行质量。

（3）故障检测、报警、故障排查。采用规则设置和机器学习相结合的思路，对服务适配时效前后的相关数据进行智能化的挖掘分析，实现异常服务调用拓扑分析、执行轨迹分析、故障预测与告警、适配失效分类与定位、失效原因回溯和推断等功能，从而支持服务适配异常情况的实时检查和排查，并对出现的异常情况进行及时通知和长期记录。

使用面向时间序列的异常检测算法（包括动态阈值、指数平滑、基于循环神经网络的异常检测算法等），对服务适配失效前后的指标时序数据进行实时分析，标记异常的时间点[17]。使用基于服务调用拓扑的故障分析和定位算法对故障的根因进行定位，通过对服务调用拓扑和执行轨迹数据进行分析和建模，结合服务适配失效前后的故障数据，过滤重复无效的告警，定位到最可能发生故障的位置，为故障的处理提供帮助[18-20]。

同时，智能运维工具应支持自动故障处理和人工故障处理相结合的模式。自动故障处理基于适配失效预测和定位结果，主动地采取故障服务隔离、容器动态调整、服务版本回滚等一系列动作，替换失效以及受影响的所有微服务实例，以保证服务流程的正常工作。运维人员在定位服务故障之后进行人工故障处理，把故障服务转接到本地开发容器，手工完成错误复现、原因排查以及代码修改，彻底排除服务适配失效的根因。

4 案例分析

本节以共享出行为例，介绍复杂服务软件模型。共享出行软件系统可划分为服务业务层与服务平台层，其业务流程中涉及到的主体构成了服务业务层，包括共享出行提供者群

1) https://prometheus.io/

2) https://zipkin.io/

3) https://www.jaegertracing.io/

4) https://kiali.io/

体、乘客与司机群体、政府监管群体等。每个群体有各自提供业务功能的服务主体，各主体交互形成的业务流程涉及共享出行的订单生成及分配、路径规划、订单计价等功能。服务平台层主要由业务层的每个服务主体对应的若干微服务构成，用于实现具体的业务流程。

4.1 共享出行服务业务层

使用 BPMN 对服务主体 Agent 的业务层流程进行建模，同时建立描述业务层的核心指标KPI_b和模型M_b。共享出行业务层服务主体主要涉及用户、客户端、司机，该流程可分为用户下单阶段、派单阶段以及行程阶段。下单阶段 Agent 之间的交互关系如图 8 所示。用户 Agent 与客户端 Agent 交互，调用地图服务、车型库服务、计价服务等完成下单阶段，其中每个服务对应平台层的若干微服务。派单阶段各个 Agent 之间的交互关系如图 9 所示，用户 Agent 与客户端 Agent 交互选择用户附近合适的车型及司机，调用地图服务、路径规划服务等并映射到服务平台层的若干微服务，物理环境中的司机 Agent 接单，完成派单阶段。图 10 描述了司机前往起点直到服务结束的交互关系。业务层核心测量指标KPI_b主要包括预约时间、派单合理性、响应时间、规划效率等。

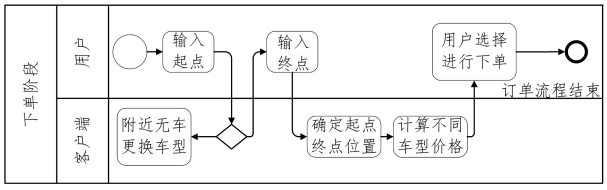

图 8 共享出行下单阶段

Fig. 8 Order stage of carsharing

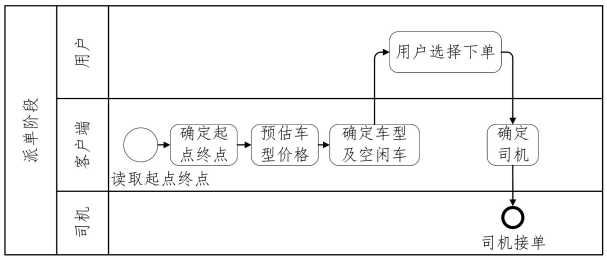

图 9 共享出行派单阶段

Fig. 9 Dispatch stage of carsharing

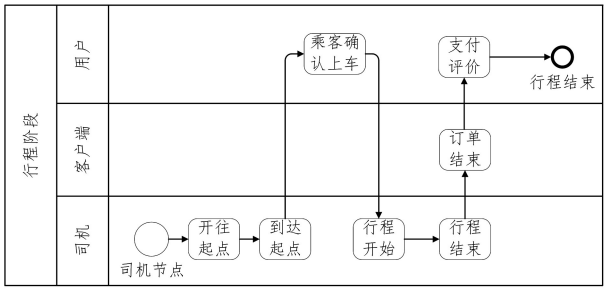

图 10 共享出行行程阶段

Fig. 10 Itinerary stage of carsharing

4.2 共享出行服务平台层

共享出行平台层包括服务主体调用的若干软件功能模块，大致包括：客服、支付、车型选择、地图、路径规划、订单支付等。这些软件模块可定义为软件智能体，乘客和司机则定义为物理环境中的外部智能体。软件智能体和外部智能体互相交互完成共享出行流程。每个软件智能体内部包含若干微服务以及相关核心测量指标。共享出行平台层流程简化图如图 11 所示，其描述了各个服务模块与司机以及乘客的交互关系。

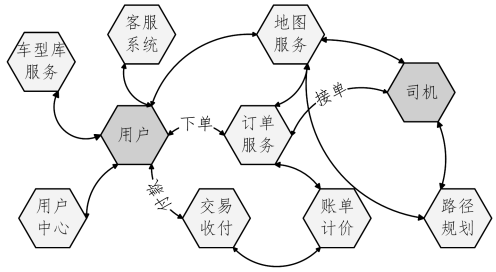

图 11 共享出行平台层流程简化图

Fig. 11 Simplified process of conceptual ride-sharin platform layer

开发和运维上述服务平台必须遵循 3.3 节所述的分析设计、部署更新、运行维护 3 个阶段，并基于这些阶段的智能工具完成。在分析设计阶段，根据共享出行的服务模型，服务组合适配工具将根据共享出行 BPMN 的语义，在服务知识图谱中查询匹配的微服务实体集合，自动适配并生成微服务实体的连接图，形成如图 11 所示的服务交互与流程组合。在交付更新阶段，开发人员需要根据乘客司机群体和交通监管部门的要求，不断更新服务主体的模型与微服务代码，并持续地进行部署演化。在运行维护阶段，服务主体通过分布式的监控组件，分析重要的业务指标，并采用一系列业务智能模型完成分析乘车需求、预测到达时间、规划行车路线等任务，对司机资源进行优化分配，提高城市的出行效率。系统的运维人员需要对微服务软件的基础平台进行监控和管理，应对各种突发事件对微服务运行的影响，及时处理各类报警事件，保证底层的微服务实例正常且稳定地运行。

结束语 本文聚焦面向复杂服务软件系统的微服务架构和智能服务适配方法，以有效支持复杂服务软件系统的构造、部署、运行、维护、演化和保障等。本文提出以群体智能的建模-仿真-优化为复杂服务软件系统的研究范式，以微服务网格架构作为自适应软件框架，扩展多主体模型作为复杂微服务系统的软件模型，以智能服务适配作为开发复杂服务软件系统的技术路线。本文详细介绍了复杂服务软件的分析设计、交付更新和运行维护各阶段的开发技术和典型支撑工具，强调了如何结合当前人工智能的最新进展，实现复杂服务软件的各阶段开发工具的智能化，以提升复杂软件系统开发的效率。

参 考 文 献

[1] LI W, WU W, WANG H, et al. Crowd intelligence in AI 2.0 era [J]. Frontiers of Information Technology & Electronic Engineering, 2017, 18(1): 15-43.

[2] WANG H, WU W, MAO X, et al. Growth structure and adaptive evolution of complex software systems[J]. Scientia Sinica Informationis, 2014, 44(6): 743-761.

[3] CUESTA C E, NAVARROE, UWE Z. Synergies of system-of-systems and microservices architectures[C]// Proceedings of the International Colloquium on Software-intensive Systems-of-Sys-

tems at 10th European Conference on Software Architecture. 2016:1-7.

[4] YIN J,DENG S,WU J,et al. Research on Shared Service based on loop-type relationship Model [J]. Communications of the CCF,2017,13(2):18-23.

[5] DRAGONI N,GIALLORENZO S,LAFUENTE A L,et al. Microservices:yesterday,today,and tomorrow[M]// Present and Ulterior Software Engineering. Springer,Cham,2017:195-216.

[6] LI W,LEMIEUX Y,GAO J,et al. Service mesh:Challenges,state of the art,and future research opportunities[C]// 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE,2019:122-1225.

[7] CHINOSI M,TROMBETTA A. BPMN:An introduction to the standard[J]. Computer Standards & Interfaces,2012,34(1):124-134.

[8] DE ALWIS A A C,BARROS A,FIDGE C,et al. Remodularization Analysis for Microservice Discovery Using Syntactic and Semantic Clustering[C]// International Conference on Advanced Information Systems Engineering. Springer,Cham,2020:3-19.

[9] CHÁVEZ K,CEDILLO P,ESPINOZA M,et al. A Systematic Literature Review on Composition of Microservices through the Use of Semantic Annotations:Solutions and Techniques[C]// 2019 International Conference on Information Systems and Computer Science (INCISCOS). IEEE,2019:311-318.

[10] LIMONCELLI T A. GitOps:a path to more self-service IT[J]. Communications of the ACM,2018,61(9):38-42.

[11] SHARMA R,SINGH A. Istio Gateway [M]// Getting Started with Istio Service Mesh. Apress,Berkeley,CA,2020:169-192.

[12] DANG Y,LIN Q,HUANG P. AIOps:real-world challenges and research innovations[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering:Companion Proceedings (ICSE-Companion). IEEE,2019:4-5.

[13] WANG S,GUO Y,ZHANG N,et al. Delay-aware microservice coordination in mobile edge computing:A reinforcement learning approach[J]. IEEE Transactions on Mobile Computing,2019,PP(99):1-1.

[14] QIU J,DU Q,YIN K,et al. A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications[J]. Applied Sciences,2020,10(6):2166.

[15] TURNBULL J. Monitoring with Prometheus [M]. Turnbull Press,2018.

[16] CRAWLEY K. Getting Started with Observability Lab:Opentracing,Prometheus,and Jaeger[J]. USENIX,2019,85:76-79.

[17] MALHOTRA P,VIG L,SHROFFG,et al. Long short term memory networks for anomaly detection in time series[C]// 23rd European Symposium on Artificial Neural Networks,Computational Intelligence and Machine Learning. 2015,89:89-94.

[18] BRANDÓN Á,SOLÉ M,HUÉLAMO A,et al. Graph-based root cause analysis for service-oriented and microservice architectures[J]. Journal of Systems and Software,2020,159:110432.

[19] WU L,TORDSSON J,ELMROTH E,et al. MicroRCA:Root Cause Localization of Performance Issues in Microservices [C]// NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE,2020:1-9.

[20] WANG P,XU J,MA M,et al. Cloudranger:Root cause identification for cloud native systems[C]// 2018 18th IEEE/ACM International Symposium on Cluster,Cloud and Grid Computing (CCGRID). IEEE,2018:492-502.

WU Wen-jun, born in 1973, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include collective intelligent, intelligent microservice and cognitive modelling.