



专栏：随选网络

面向随选网络的业务编排系统研发实践

罗光峰, 李奕群, 陈慧光

(中国电信股份有限公司北京研究院, 北京 102209)

摘要: 为满足 SDN/NFV 时代业务快速上线的要求, 需要对现有网络运营系统进行重构, 以随选网络业务的实践为背景, 详细介绍了编排器在运营系统中的定位与作用, 然后阐述了编排器的核心设计思路以及核心组件工作流引擎。

关键词: 随选网络; SDN; NFV; 编排器; 运营支撑系统

中图分类号: TN929

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2017327

R&D practice of service orchestration system for on-demand network

LUO Guangfeng, LI Yiqun, CHEN Huiguang

Beijing Research Institute of China Telecom Co., Ltd., Beijing 102209, China

Abstract: In order to meet the requirements of services quickly on line in the SDN/NFV era, existing network operating system needs to be refactored. Based on the background of the practice of on-demand network services, R&D practice of orchestrator for on-demand network service was introduced, then the orchestrator's role in software system architecture was explained in detail.

Key words: on-demand network, software defined networking, network function virtualization, orchestrator, operation support system

1 引言

美国运营商 AT&T 在网络重构中, 提出了 Domain2.0 和 ECOMP 架构, 旨在通过网络的虚拟化和集中控制提升业务开通和运维的自动化, 从而提升客户体验。在 NFV 领域中, ETSI (European Telecommunications Standards Institute, 欧洲电信标准化协会) 标准组织定义了相关软件框架 MANO (NFV management and orchestration, NFV

管理和编排), 包含 orchestrator (编排器)、VNF 管理器和 VIM (virtualized infrastructure manager, 虚拟设施管理器)。ECOMP 中的 MSO (master service orchestrator, 主业务编排器) 就承担了 MANO 中 NFVO 的角色。另有 Verizon 公司的 SDN/NFV 计划, 目标是同时考虑 SD-WAN 和 NFV 的业务需求, 通过构建一个端到端编排器 (end-to-end orchestrator), 实现跨 SDN 与 NFV 的端到端的业务。

在 SDN/NFV 技术时代, 各大运营商都有各

收稿日期: 2017-11-10; 修回日期: 2017-12-08

自的技术演进计划。其中一个重要目标就是通过新技术、新网络体系实现随选网络业务。随选网络指在业务层面提供客户自定义网络业务的功能,在配置层面实现业务对应的网络配置自动化、网络资源按需指配。在业界的技术发展中,给负责实现业务层面自动配置功能的子系统命名为“编排器”。Linux 基金会在 2015 年成立了针对编排器的开源项目 Open-O,2017 年初 AT&T 开源了其 ECOMP 系统,Open-O 项目与 ECOMP 合并,更名为 ONAP 项目,并吸引了更多运营商、设备厂商、芯片厂商的加入,共同推动编排器的设计和实现。

本文针对 SDN/NFV 时代对网络运营系统的要求,分析了编排器设计的目标,提出了一种业务编排器的架构设计和实现方案,并通过开发实践,对编排器做出了验证,可为下一代运营系统的设计提供参考。

2 随选网络系统中的业务编排

编排一词在维基百科中的定义如下:编排在

计算领域主要指通过相应的工具、架构和过程部署(开通)一个业务,此过程涉及的软件与硬件统一部署,需要支持自动化的工作流程。

在随选网络系统中,编排器承担着业务设计、开发/实现、上线、开通、变更、关闭、下线整个生命周期的管理职责,业务编排在系统架构中的位置如图 1 所示。

在业务的生命周期中,编排器需要满足业务的灵活设计与快速上线要求,为业务编排设计人员(通常为网络运维人员)提供用户友好的设计工具,满足业务定义的直观、准确、可测试等特性;业务编排将接收客户订购的请求,并将其分解为预定义的工作流程,在流程中通过调用已有的原子能力实现每一个步骤。例如其中涉及配置相关的步骤,需要调用与控制器相对应的原子能力,通过控制北向接口将配置指令下达控制器,并由控制器负责最终的执行。在业务开通流程中力求无人工参与,从而大幅缩短开通时限。业务开通流程中编排器的角色和其他系统的配合示例如图 2 所示。

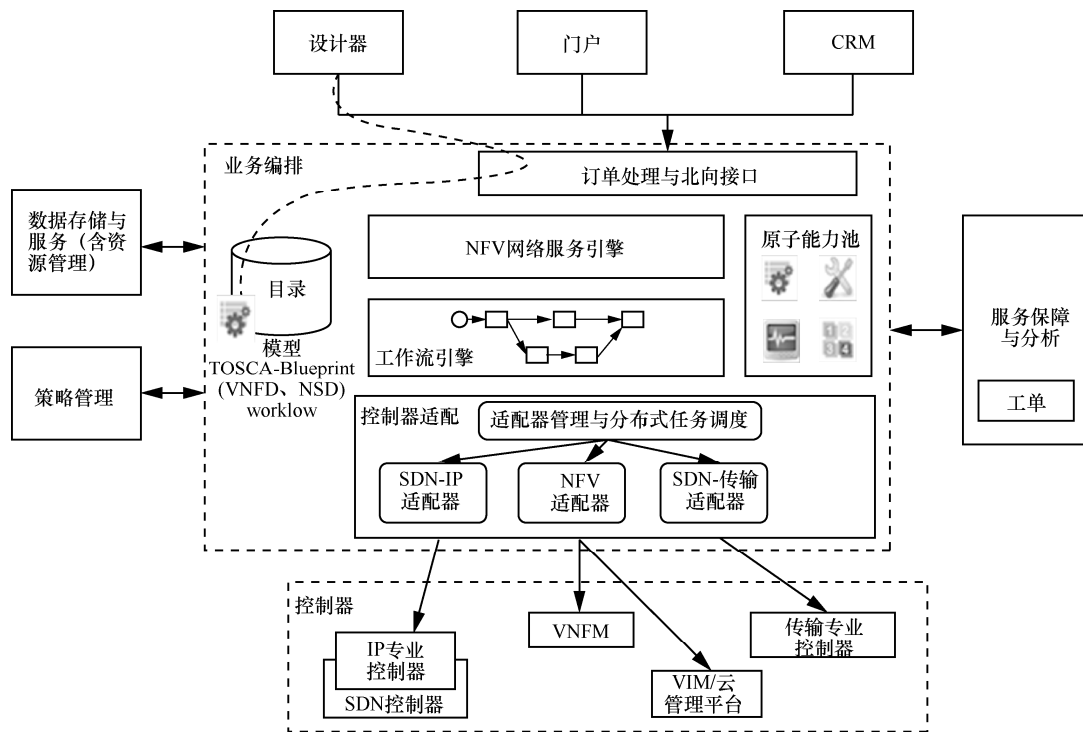


图 1 业务编排在系统中的位置

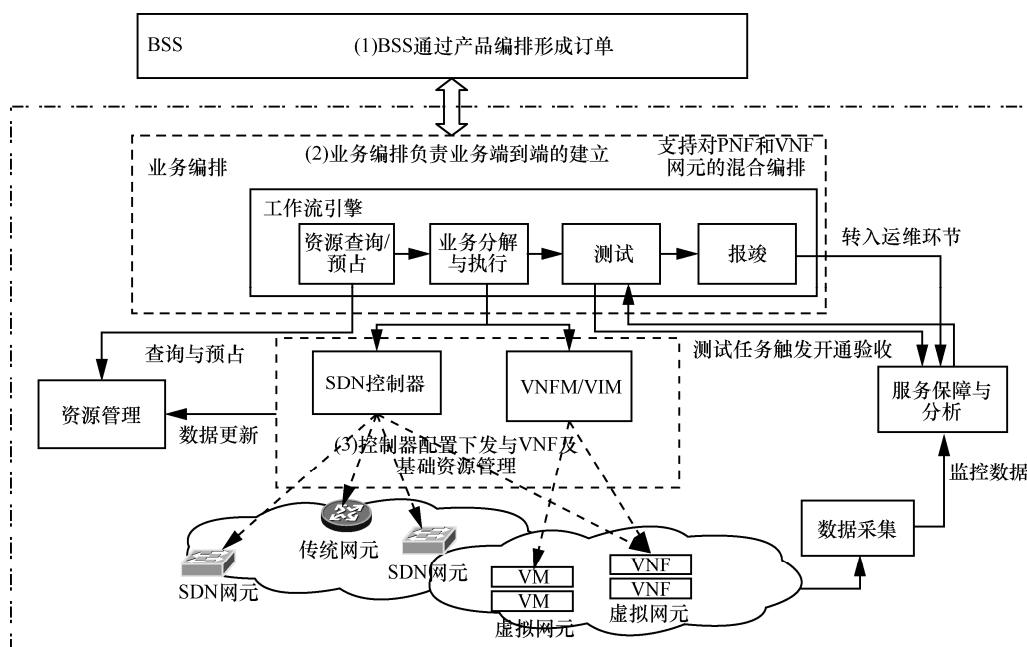


图2 自动化的业务开通流程示例

3 通过 workflow 引擎实现资源编排

编排器对外提供的业务编排功能基于工作流引擎，当出现新的业务场景时，系统可对现有能力进行重新组合，快速生成新业务以满足需求。运营人员通过可视化的拖拽、连线的操作，完成业务需要的配置。

工作流引擎主要由以下 4 个模块组成。

(1) 服务目录

用于存储系统支持哪些功能、每个功能有哪些接口、接口中有哪些参数、参数类型等。

(2) 设计器

提供 UI，并能从服务库中选择服务进行组合，形成需要的业务配置序列，存储成业务模板。

(3) 校验器

对设计好的流程和模板进行测试验证，发现潜在的错误或缺陷。

(4) 执行器

加载一个由设计器生成的业务模板，按照模板的描述执行一系列操作，完成业务配置。

workflow引擎组成与工作原理如图3所示。

对以上工作过程解释如下。

步骤 1 能力编排引擎从一个公共的服务库（后述）中获取系统可用的服务以及每个服务的接口规格，形成自己的服务库。

步骤 2 当运营人员需要设计一个工作时，设计器可从服务目录中读取可用服务（并展示于 UI），供设计人员使用。

步骤 3 设计人员使用设计 UI，通过拖拽服务及连线（接口调用），完成一个业务场景需要的流程设计。其中，接口调用过程需要指定必要 I/O 参数，这些参数的规格（类型、可取值范围等），须从服务目录中获取，并在 UI 中进行展示约束。

步骤 4 设计完成的工作流，进入校验阶段。通过自动化测试脚本，对工作流进行测试验证。若发生错误，工作流返回设计器，需要设计人员进行调整和更正。

步骤 5 校验成功的工作流进入执行器，具备运行条件。应用层通过执行器提供的 **API** 可启动某个工作流的执行，完成配置。

步骤 6 工作流的执行可被监控，并可根据需要由系统定时或事件触发。

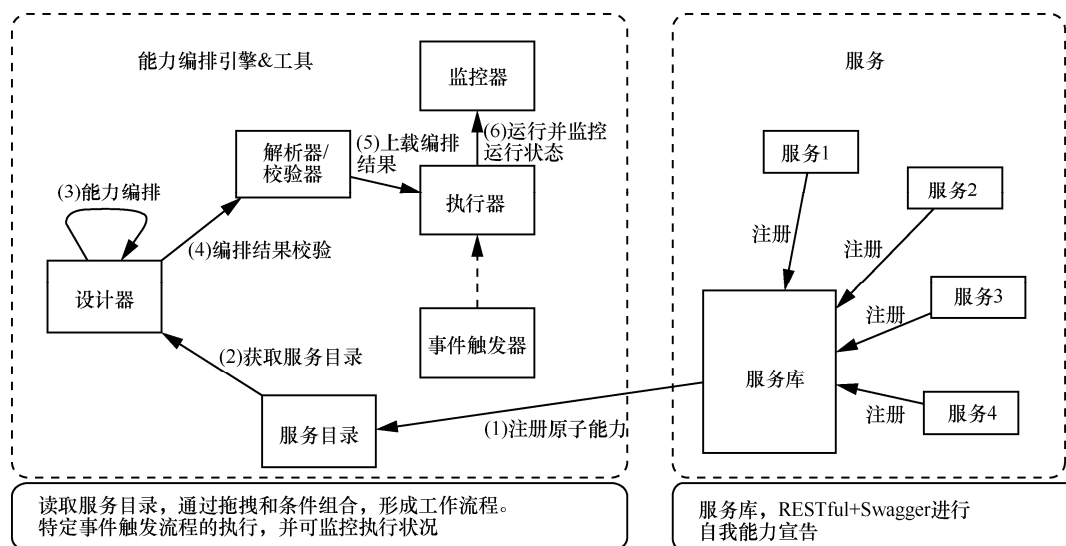


图3 workflow引擎组成与工作原理

4 业务编排的两态（开发态与运行态）

编排器管理业务的全生命周期，涉及开发态和运行态。业务的设计、开发/实现属于开发态，业务的开通、变更、关闭属于运行态。业务编排的两态在编排器中通过 DevOps 平台实现无缝的衔接，通过在 DevOps 平台中定制业务编排相关工具（例如模型语法分析与测试、运行环境与开发环境数据的同步等），实现业务从开发到部署的全流程自动化、持续化。业务编排两态示意如图4所示。

业务编排通过开发态和运行态以及 DevOps 平

台实现业务的全生命周期管理，依赖于以下能力。

(1) 业务编排的基础是良好定义的原子能力

- 从业务编排的视角看，所有能够在业务编排中调用的服务都是原子能力。
- 原子能力依照提供方大致可以分为网络类原子能力（例如通过 Device YANG 实现网元的配置）、NFV/云类原子能力（例如通过 TOSCA VNFD 拉起 VNF）、资源类原子能力（例如电路资源的查询与预占）、服务保障类原子能力（例如业务的端到端测试）。

(2) 端到端编排依赖于全面准确的资源清单在业务的开通流程中，总是需要资源的申请

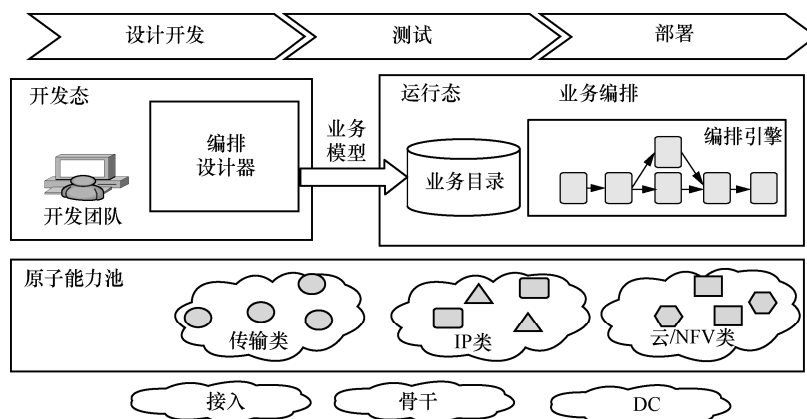


图4 业务编排两态示意



(例如 IP 地址), 这些资源的申请需要自动化完成, 资源管理模块承担着相关职责, 并通过接口为业务编排模块提供相应的原子能力。

(3) 业务编排提供灵活性与复杂性的平衡

- 通过主工作流的编排, 实现业务开通流程的定义与管理, 可以适应不同业务不同组织的定制化要求。
- 子工作流、NFV-NS、SDN-service YANG 都可以整合细粒度的原子能力, 并能够提供事务(例如回滚操作)的支持, 作为主工作流的一个步骤(在 workflow 中定义的一个 task)。

5 随选网络编排器开发实践

在编排器原型系统实现中, OpenStack 的 Mistral 组件因其轻量、简单、易于集成和二次开发的特点, 被用于实现核心编排能力的 workflow 引擎基础。Mistral 引擎基于 YAML 定义了一套 workflow 模型语言, 并采用了 YAQL 作为数据处理的模板语言, 当前版本为 2.0, Mistral 的模型语言语法简洁、易于理解、支持任务执行结果的检查和条件分支控制, 可以满足编排引擎的功能需求。

然而, 要达到前述的“可视化”和“两态”的转换, 还需要对其进行扩展。首先, 需要一个可视化设计器, 网络运维人员使用该设计器, 通过拖拽和连线, 即能完成一个业务场景的设计。更重要的是, 虽然 Mistral 的 workflow 模型语言语法简洁, 但运维人员使用编辑器手动编写, 不仅效率低, 且容易出错。这就需要设计器不仅具备可视化, 而且能够自动生成 Mistral 的 workflow 模型语言。其次, 要实现上述的拖拽, 首先要具备一个“服务库”, 服务库的每个服务就是一个可拖拽的组件。因此, 需要设计器能够从服务库获取服务描述, 并解析出需要的信息, 供设计人员使用。例如, 一个服务的接口, 需要何种参数,

参数类型是 string 还是 integer, 参数允许何种取值范围等。这些约束, 必须在服务描述模型中包含。所以, 这点要求, 需要服务库“生成”这种模型, 设计器解析模型来展现。最后, 工作流需要一个运行状况的可视化, 当一个工作流编写验证完成, 从开发态进入运行态, 需要对其运行状态进行监控和展现。便于在发生错误的时候, 由专业人员定位和解决问题。

同时, 为了满足编排的需要, 编排器的核心能力集必须原子化、组件化。这不仅满足了编排引擎的需要, 而且由于各个原子能力的自治、解耦, 给系统的分布式部署、独立负载均衡带来了极大方便。本次原型开发, 采用了微服务架构实现了编排器的核心服务库。为实现这些服务的管理, 引入统一的微服务总线 (microservice bus, MSB)。微服务向 MSB 注册自己的接口信息和节点信息, MSB 负责管理微服务的负载均衡、故障检测, 对微服务接口的调用, 也是经由 MSB 代理转发。最后, 上述工作流的设计器需要的服务库, 也由 MSB 统一导出。从实现上, MSB 采用业界经典的 Nginx 作为基础服务调用代理模块, 通过扩展的 OpenResty 插件, 完成服务的管理。

值得提出的是, 为了方便开发, 保持与工作流引擎的一致性, 微服务开发的数据建模, 也采用了基于 YAML 的 Swagger 标准。开发态和运行态, 以 YAML 语言作为基础传递信息。

对于编排器来说, 南向要对接各种设备厂商的控制器或者设备。这些厂商的接口规格通常有较大差别。因此, 在编排器实践中, 采用一个驱动层 (driver), 将各个厂商的接口统一抽象成公共模型。这样, 微服务层即可针对该抽象模型进行无差异编排。

随选网络的编排器实现的开发架构如图 5 所示。

为了验证在实际的生产环境下, 上述设计是否符合业务需求, 检验业务编排系统架构是否满

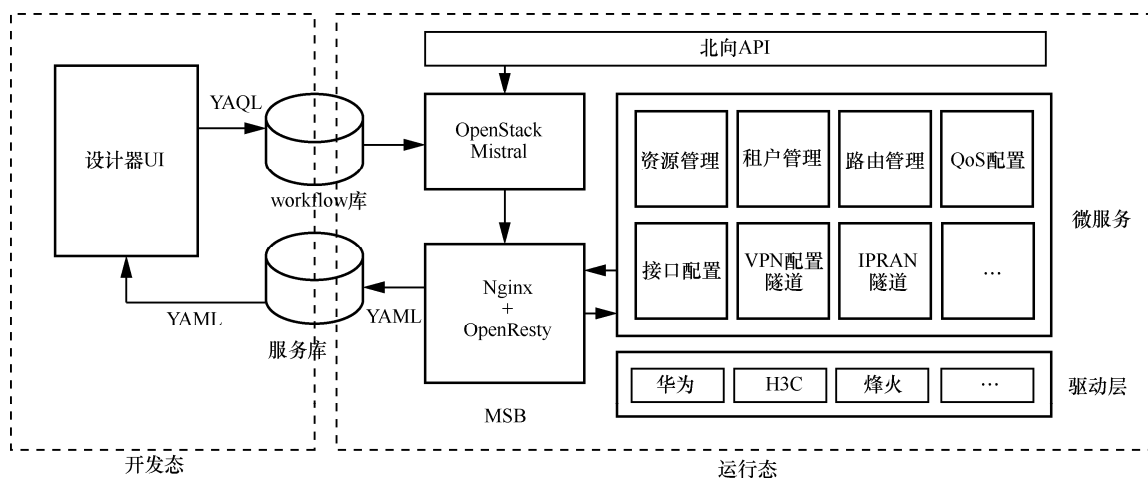


图5 随选网络的编排器实现的开发架构

足设计目标，需要从以下两个方面进行。

- 需要满足业务需求：业务模型的定义、业务开通、业务变更、业务关闭。
- 需要满足运维管理需求：业务流程执行过程的监控、业务执行过程故障的检查、系统扩展、系统运维管理。

在实际验证过程中，为了更贴近实际业务需求，在实施方案中采用一个典型的业务场景作为业务需求，用于验证业务编排系统。图6给出了一个在园区通过一个虚拟化网元 vCPE，为客户提供 site-to-internet 的场景。

在实际验证环境中按照实际生产环境进行部署，系统完整支持热备份和集群，业务编排系统

包含了几个关键组件。

(1) 设计态

服务目录管理：为了实现直观便捷的编排，所有业务编排系统中的微服务接口是通过 Swagger 规范定义的，这些接口需要登记到服务目录中，用于辅助设计器的实现。

workflow设计器：通过图形化的界面，为运维人员提供可视化的 workflow 编排设计工具，管理设计好的 workflow 模板，并将确认版本发布到 workflow 引擎。

(2) 运行态

- **微服务总线：**业务编排系统完全基于微服务架构实现，所有的功能模块和驱动以微服务的方式开发和部署，微服务总线实现

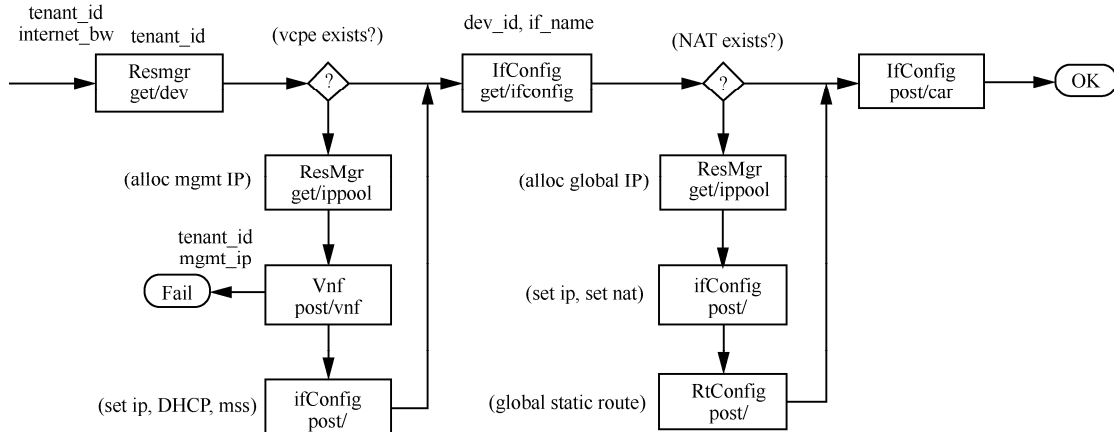


图6 园区随选业务开通流程示意



了微服务的注册、路由和负载均衡。

- 工作流引擎：基于 OpenStack 的 Mistral 引擎，为了满足业务编排中订单处理的需求，对引擎进行了扩展，用以实现 workflow 执行的控制和工作流执行信息的通知。
- 北向接口和订单处理：API、订单校验与分析、订单监控、在途订单管理。
- 微服务：资源管理、租户管理、路由管理、VPN 配置、QoS 配置、接口配置。
- 南向驱动层：实现了多厂商（中兴、华为、华三）SDN 控制器的适配。

验证结果记录如下。

步骤 1 完成业务编排系统的搭建，所有的微服务都要完成部署和测试，并注册到微服务总线。

步骤 2 完成服务目录的更新，将所有的微服务登记到服务目录中，并完成适当的注释说明，比如输入/输出参数的中文简介、是否为异步调用接口等。

步骤 3 使用设计器完成业务模型的定义，通过图形化的界面，拖拽服务目录中的服务项作为 workflow 模型中的任务，并实现任务之间的先后顺序，完成任务中服务需要的参数配置。

VPN 业务开通 workflow 模板样例如图 7 所示，工作流的模型以 YAML 格式保存。

这个 workflow 模型中定义了多个任务，任务采用了顺序执行方式，首先执行的是 task1，当 task1 执行成功后，执行 task2，然后是 task3。从 workflow 模型中可以看到，工作流中的一个任务就是调用一个微服务，微服务的调用都是通过微服务总线的统一入口实现，微服务的数据输入参数都可以在模型中指定，微服务的输出亦可以保存并用于后续任务。

步骤 4 通过用户门户，用户可以订购产品，并将订单发送给业务编排系统。

步骤 5 业务编排系统接收到订单后，通过订单处理查询并确认通过哪一个 workflow 模型实现

```
version: '2.0'
serviceopenworkflow:
  description: workflow
  type: direct
  input:
    - ORDER_IPVPN
  tasks:
    task1:
      action: yihe.sync
      input:
        headers:
          Content-Type: application/json
        method: post
        body:
          orderid: <% $.ORDER_IPVPN.get(orderId) %>
          atom: Bisop.OrderOrcha.iomReviewOrderInfo
          url: http://172.18.1.105:8080/iom/reviewOrderInfo
      publish:
        task1_r: <% task(task1).result.content %>
      on-success:
        - task2
    task2:
      action: yihe.async
      input:
        headers:
          Content-Type: application/json
        method: post
        body:
          dispatchType: <% $.ORDER_IPVPN.get(OperType) %>
          totalInterVPC: <%
$.ORDER_IPVPN.get(TotalInterVPC) %>
          vpclDZ: "
          ctYUNRegEmail: <%
$.ORDER_IPVPN.get(CTYUNRegEmail) %>
          speedA: <% $.ORDER_IPVPN.get(AppSpeedA) %>
          yunResCityA: <%
$.ORDER_IPVPN.get(YUNCityA) %>
          routeProtocol: <%
$.ORDER_IPVPN.get(RoutProtocol) %>
          custId: <% $.ORDER_IPVPN.get(CustId) %>
          vpnNetCode: <% $.ORDER_IPVPN.get(vpnNbr) %>
          ctYUNACCNBR: <%
$.ORDER_IPVPN.get(CTYUNACCNBR) %>
          netStruct: <% $.ORDER_IPVPN.get(Networkexp) %>
          celP: <%
$.ORDER_IPVPN.get(VPC_IPSegmentA) %>
          yunRespoolldZ: "
          yunRespoolldA: <%
$.ORDER_IPVPN.get(YUNNameA) %>
          formType: <% $.ORDER_IPVPN.get(FormType) %>
          comments: <%
$.ORDER_IPVPN.get(ProjectRemark) %>
          speedZ: "
          yunResCityZ: "
          changeType: "
          vpclDA: <% $.ORDER_IPVPN.get(VPCIDA) %>
          custName: <% $.ORDER_IPVPN.get(CustName) %>
          proInstId: <% $.ORDER_IPVPN.get(CircuitID) %>
          workId: <% $.ORDER_IPVPN.get(orderId) %>
          centerPoint: <% $.ORDER_IPVPN.get(CenterPoint) %>
          vpnLinkCode: "
          celPZ: "
          workType: <% $.ORDER_IPVPN.get(ProductID) %>
          atom: Bisop.OrderOrcha.cloudResourceRequest
          url: http://
172.18.1.105:8080/CloudClientService/cloudResourceRequest
      publish:
        task2_r: <% task(task2).result.content %>
      on-success:
        - task3
    ...
```

图 7 VPN 业务开通 workflow 模板样例

此订单,然后通过 workflow 引擎的接口启动相应的流程。

步骤 6 workflow 引擎将依据模型中定义的步骤和任务执行的条件逐一完成任务。

步骤 7 workflow 引擎在执行过程中会通过通知订单,处理任务执行的进展情况和结果。

步骤 8 用户门户随时可以通过北向接口获得订单执行的进展情况,并向用户展示。

在业务验证过程中,完成全部业务流程的实现,包括业务开通、业务变更、业务关闭,也包括了一些异常流程的验证,例如业务开通过程中的停开、业务开通过程中异常情况的处理等。

在完成的验证过程中,异常情况的处理最为复杂,例如任务的执行异常,需要重新执行,在重新执行时还需要修改任务的输入参数, Mistral 开源项目并不完全支持,需要通过扩展实现。

通过开发实践,对业务编排系统的设计做出了验证,基本满足了最初对业务编排的要求。

新系统架构带来的优势如下。

- workflow 引擎不仅实现了控制流的调度管理,同时实现了数据流的传递,可以结合微服务总线完美实现对微服务的调用。
- 微服务作为基础能力可以通过 workflow 方式完成编排,有利于微服务的重用。
- 微服务由于实现的是基础功能,功能需求稳定变化少,对应的微服务软件版本稳定,通过编排随时可以为新的业务需求创建工作流模板,无需编写软件代码,即可为新业务提供服务。
- 在实践验证过程,对 workflow 引擎进行了必要的扩展。
- 通过对开源项目的扩展,实现了异常情况下任务的干预(重试),对工作流的执行实现回滚操作。

现有业务编排系统版本还有如下不足。

- 开源的 workflow 引擎在性能方面存在不足,

虽然执行集群方式的水平扩展,但是单机的压力测试表明,高并发下的流程启动处理能力不足,还无法满足大型项目的需求。需要对 workflow 引擎进行重构才能实现单机性能的提升。

- 在实践过程中,遇到了不同种类的业务,既有低数量耗时长(以天为单位)的长流程,也有高并发耗时短(以秒为单位)的短流程,目前的流程引擎采用公平原则进行处理,对于需要尽快处理的订单,无法确保优先执行。建议通过划分资源分区的方式对不同类型的流程进行单独处理。
- 通过图形化的界面和服务目录结合使用,已经可以设计 workflow 模型,但是在设计过程中对于任务的输入/输出参数的配置还稍显复杂,对于运维人员要求较高,需要理解微服务的概念。
- 运维工具不足,开源项目提供的运维工具只实现了查询和管理基本功能,无法提供统计、监控和联合查询等需求,也没有与统一运维框架(例如 Zabbix)结合。

6 结束语

本文所描述的编排器在设计上参考了国际标准最新进展及业界网络运营系统的实践。通过面向 workflow 的编排引擎,组合基础设施能力形成业务能力,实现业务与资源的映射;通过模板化的 NFV 和业务建模,规范化研发实现,提高系统的可维护和可扩展能力。构建开发和运营的双态系统,使得业务需求能迅速转化成系统能力,不断扩充业务场景,实现自身的迭代演进。

该系统未来将在实际运行中根据需求持续优化,支持随选网络系统形态和相关业务形态分阶段地演进。同时通过积极加入业界开源组织,



吸纳先进设计理念，并将自身创新成果贡献于开源项目，实现开放共赢，共同推动网络技术革新和重构。

参考文献：

- [1] IETF. YANG-a data modeling language for the network configuration protocol (NETCONF): RFC6020[S]. 2010.
- [2] IETF. NETCONF configuration protocol: RFC4741[S]. 2006.
- [3] Wikipedia. Orchestration[EB/OL]. (2016-09-20)[2017-11-10]. [https://en.wikipedia.org/wiki/Orchestration_\(computing\)](https://en.wikipedia.org/wiki/Orchestration_(computing)).
- [4] 何晓明, 冀晖, 毛东峰, 等. 电信 IP 网向 SDN 演进的探讨[J]. 电信科学, 2014, 30(6): 131-137.
HE X M, JI H, MAO D F, et al. Discussion of evolution of carrier IP network to SDN[J]. Telecommunications Science, 2014, 30(6): 131-137.
- [5] 顾戎, 王瑞雪, 李晨, 等. 云数据中心 SDN/NFV 组网方案、测试及问题分析[J]. 电信科学, 2016, 32(1): 126-130.
GU R, WANG R X, LI C, et al. Analysis on network scheme and resolution test of SDN/NFV technology co-deployed in cloud data-center[J]. Telecommunications Science, 2016, 32(1): 126-130.
- [6] 马书惠, 毋涛. NFV 标准与开源技术现状[J]. 电信科学, 2016, 32(3): 43-47.
MA S H, WU T. Standards and open source technology of NFV[J]. Telecommunications Science, 2016, 32(3): 43-47.
- [7] 张松. 精益软件度量——实践者的观察与思考[M]. 北京: 人民邮电出版社, 2013.
ZHANG S. Lean software measurement-observer's observation

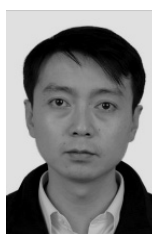
and reflection[M]. Beijing: Posts&Telecom Press, 2013.

- [8] NEWMAN S. Building microservices[M]. Sebastopol: O'Reilly Media, 2015.

[作者简介]



罗光峰（1973—），男，中国电信股份有限公司北京研究院软件工程师，主要研究方向为 SDN 领域软件架构与实现。



李奕群（1975—），男，中国电信股份有限公司北京研究院软件工程师，主要从事随选网络编排器、控制器相关开发工作。



陈慧光（1985—），男，中国电信股份有限公司北京研究院软件工程师，主要从事随选网络编排器、控制器相关开发工作。