BIAN (Banking Industry Architecture Network) —— 编织数字化银行生态价值之网

Weaving the Value Net of Digital Banking Ecosystem

李纪华 IBM 中国金融事业部企业云转型首席架构师

Li Jihua, IBM China FSS Enterprise Cloud Transformation Chief Architect

Abstract

For a long time, banks in the transformation process of pursuing business agility will continue to receive various challenges while maintaining the existing system. In addition to the disruption of the market and competitors, they also include the challenges of various fast evolving transformation technologies, methods. The Big-bang transformation or the incremental transformation, how technology and business collaborate during the transformation, and how to layout and improve gradually the capabilities inside and outside the bank, all of these are the issues that need to be considered in light of the actual situation of the industry and the enterprise. Unlike Internet companies, banks have accumulated lots of existing systems for quite a long time, and there are many "silos", data inconsistencies, redundant functions and data, and unclear application boundaries. There have been long-standing problems, and all of them interconnected via many "point-to-point" interfaces. All this makes it difficult to improve the reuse of existing resources and improve business agility.

In recent years, with some changes in driving forces, the banking industry has begun a new round of digital transformation. These driving forces include:

- Evolution of business architecture and design method. It is important for enterprise architecture to gradually shift from technology to business. Business architecture and business models are increasingly valued by enterprises.
- The evolution of technology platforms. The cloud-based technology platform has gradually matured, driving ABCDEI (Artificial Intelligence AI, Blockchain, Cloud Cloud, Big Data, Edge Computing, Edge Computing, IoT) and 5G technologies.

- The continuous integration of IT and business, IT is also a kind of business. IT itself has also begun to think about integrating development and operation seamlessly with DevOps.
- Banks are evolving to 4.0 and developing ecosystem based business models. The crossing of banking and non-banking industries has led to an "emerging" innovation situation. Many banks have already begun their journey to open banking.

The Banking Industry Architecture Network (BIAN) came into being in this context. BIAN is a non-profit organization (bian.org) composed of the world's leading banks, technology providers, consultants and academics. This network of professionals is dedicated to reducing banking costs and accelerating innovation in the industry. Members combined their industry expertise to define a banking technology framework to simplify and standardize the core banking architecture. At the same time, this framework is based on the principle of service-oriented architecture (SOA), and the integration model formed provides banks with future-oriented solutions to promote industry collaboration in the API economy.

With the help of the BIAN reference model, banks can establish “building blocks” of business capabilities, and map and interface with existing systems to clarify the boundaries between applications. Rely on industry standards (such as ISO20022, FIBO, etc.) to unified information definition and data standards. Refer to the message interaction of the BIAN Service Operations to achieve a service-oriented, loosely coupled future banking architecture.

From the perspective of business model innovation, just as ODBC has opened the door to open database interconnection, BIAN is promoting the establishment of an open banking ecosystem with its open service standards. Not only for the financial industry, but also for the external ecology of the bank, including non-financial industries and the developer community. From the perspective of architecture design, adhering to the essence of SOA and keeping up with the current trend of cloud transformation, BIAN has enriched the design of bank architecture from the perspective of design. Makes it possible to carry out the gradual transformation under the loosely coupled architecture system.

BIAN can be combined with an enterprise architecture approach, which can quickly set up a bank's future service panorama view and capability view in digital transformation projects. Through mapping with existing applications, customers cna find inadequacy (such as duplicate functions, gaps and misalignment). In the process of microservices and API construction, "services" are used throughout which make the overall process easier to keep consistency. At the same time BIAN help improve the rationality of services integration between internal applications, and the value of external service

exposure when planning for open banking. Furthermore, BIAN's SOA design concept can be extended to other industries, thereby weaving a wider ecological value network!

1 前言

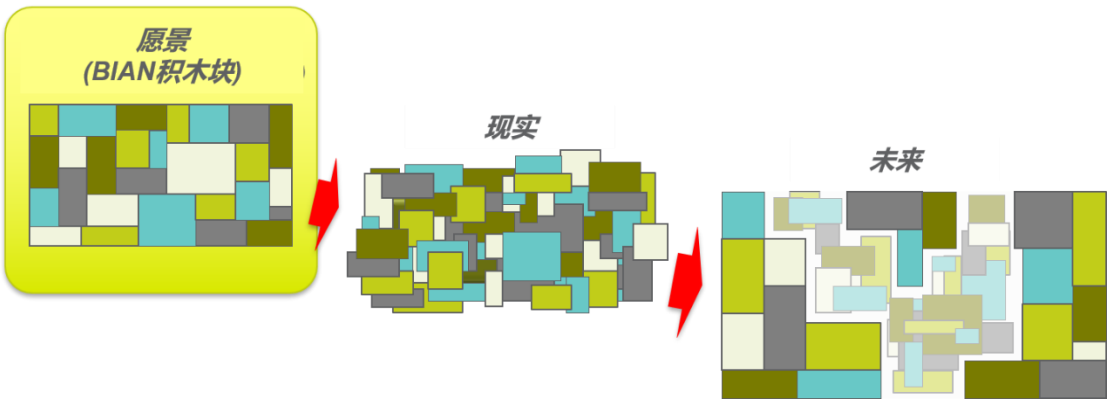
长期以来银行在追求业务敏捷性的转型过程中会在维护现有系统的同时不断接收到各种冲击，除了市场与竞争对手的冲击，还包括各种转型技术、方法和方向的冲击。突进式转型还是渐进式转型，科技如何和业务协作进行转型，银行内外如何布局和循序渐进提升，这些都是需要结合行业和企业实际情况切实思考的问题。与互联网企业不同，银行长期以来积累了大量已有系统，而且“竖井”林立，数据不一致，功能与数据冗余，应用边界不清等问题长期存在，而且所有这些都通过大量的“点对点”接口进行连接。这一切都给如何改善现有资源的重用性，提升业务敏捷性带来了困难。

近些年来，随着一些驱动力的变化，银行业已经启动了新一轮数字化转型过程。这些驱动力包括：

- 业务设计技术方面的进步，企业架构从偏重技术逐渐走向业务为要，业务架构及业务模型越来越受到企业的重视。
- 技术平台的进步，以云为代表的技术平台逐渐成熟，带动了 ABCDEI(人工智能 AI, 区块链 Blockchain, 云 Cloud, 大数据 Data, 边缘计算 Edge Computing, 物联网 IoT)及 5G 的全面技术进步。
- 业务管理意识的提升，IT 与业务的不断融合，IT 也是业务。IT 自身也开始思考结合开发运营一体化 DevOps 来逐步贯通。
- 银行在向 Bank 4.0 演进，在 API 经济互联互通下展开生态业务。银行和多个行业纵横交错形成了“涌现”创新局面。不少银行已经展开了开放银行之旅。

银行业架构网络（BIAN）正是在这一背景下应运而生的。BIAN 是一个业界多方协作的非营利性组织(bian.org)，由全球领先银行，技术提供商，顾问和学者组成。这个专业人员网络共同致力于降低银行业务成本，并加快行业创新速度。 成员结合他们的行业专业知识，定义了一个用以简化和标准化核心银行体系结构的银行技术框架。 同时这一框架基于面向服务的架构(SOA)原则，所形成的整合模型为银行提供了面向未来的解决方案，以促进 API 经济下的行业协作。

借助 BIAN 参考模型，银行可以建立起业务能力“积木块”，通过与现有系统进行映射和对接，理清应用之间的边界。依托业界标准(如 ISO20022、FIBO 等)统一信息定义和数据标准。参考 BIAN 服务操作规范 API 的消息交互，从而达成面向服务的、松耦合的未来银行架构。以此达成业务敏捷性的终极目标。这也是目前国内银行纷纷建设业务中台、服务中台的目的。



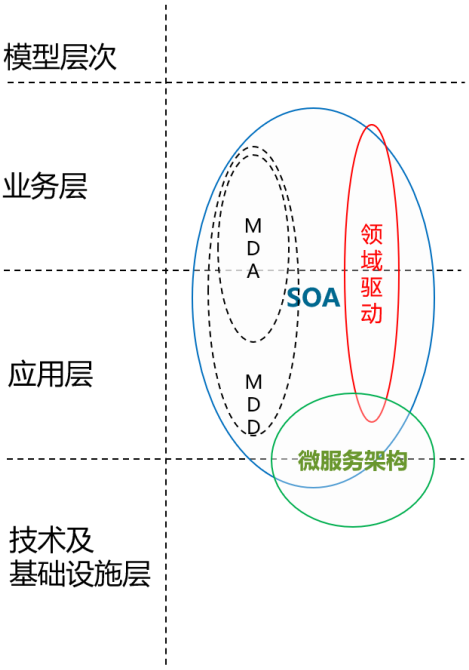
从架构及技术角度看，BIAN 融汇了业界关于银行业务模型(如 IBM IFW)和技术体系(如 API 和云)的积累、结合 SOA 架构和微服务架构理念，基于业务能力、组件及服务而形成的银行应用之间互联互通的技术标准。在过去近五年时间 BIAN 差不多以每年一个版本的速度在进行演进，截至到 2020 年初最新版本为 v8.0 (https://bian.org/deliverables/bian-standards/bian-service-landscape-8-0/)，BIAN v9.0 计划 2020 年 Q3 发布。国内一些银行也开始加入 BIAN 并结合 BIAN 来打造更为服务化和生态化的银行系统。

2 BIAN 理论基础

BIAN 是企业架构思想在银行的延伸，BIAN 一直致力于开发出一种用于企业架构设计，业务能力(Business Capabilities)和相关服务操作(Service Operations)的方法。通过选择并组装这些不同层面的“积木块”来对银行（或金融机构）进行建模。由于 BIAN 的设计是“规范的”，这意味着面向任何金融组织的不同实施情况，它们可以以一致方式进行诠释。考虑到规范性设计定义以及松耦合架构，BIAN 方法采用了面向服务的架构(SOA)方法。

2.1 BIAN 基于 SOA 体系架构的优点

盛行于本世纪初的 SOA 面向服务架构体系在系统设计和实现方面的优点已经有很多文档进行了充分论述。但是，随着领域驱动设计(DDD-Domain Driven Design)，微服务架构(Microservice Architecture)的不断流行，很多用户将 SOA 逐渐打入“遗留”系统或方法之列。更有甚者，认为 SOA 就是单体(monolithic)架构的代名词。实际上 SOA 用“服务”一致性贯穿业务和技术的思路仍然是适用的，只是技术上的局限性，SOA 在早期阶段把更多重点放到了应用整合服务整合以及流程整合方面。随着技术的不断进步特别是随着微服务、云原生技术以及 PaaS 平台云的不断成熟，我们需要对 SOA 理论体系 “温故而知新”。结合目前这些前言技术更好地将“服务”理念向业务延伸，真正落实 SOA 的核心理念。笔者将一些关系较为密切的方法体系进行了比较，供大家快速参考。关于方法体系会在方法专题专门展开讨论。



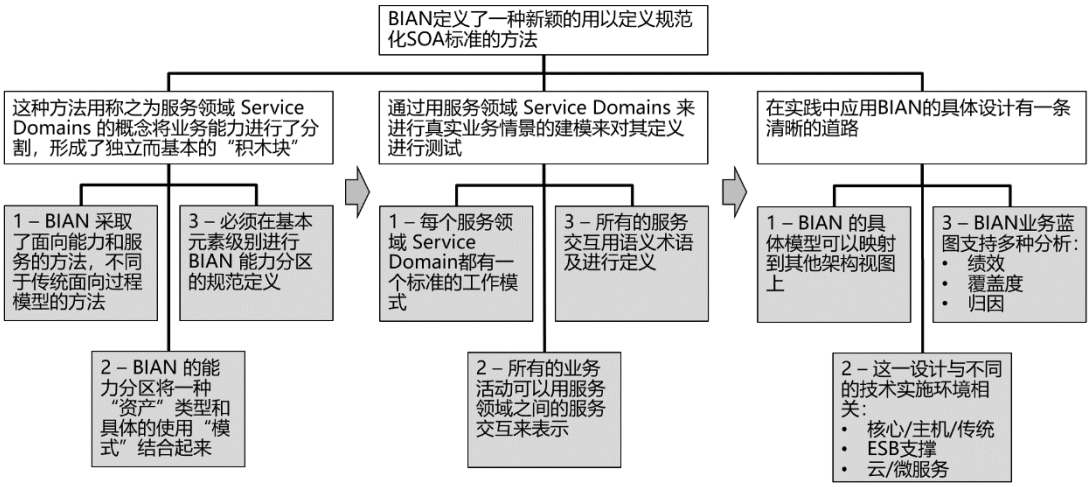
- OMG(Object Management Group)打造的 MDD/MDA(模型驱动设计 Model Driven Design/模型驱动架构 Model Driven Architecture)率先规划了模型层次及转换概念，并制定了模型的体系层次（元元模型、元模型、模型以及真实世界）及相关标准。在模型驱动的推动下逐步由模型驱动架构(MDA)向更广阔的领域推广，如模型驱动开发、模型驱动工程等。可以说 MDD/MDA 奠定了模型相关的很多理论基础，始终指导着各个行业很多基于模型和架构的系统性开发工作。许多行业基于 MDD/MDA 思想结合自身行业模型打造了行业模型体系标准，特别是在系统工程领域，MBSE(Model Based System Engineering)已成为业务 IT 融合进行快速应用交付的典范。
- 面向服务架构 SOA 则通过“服务”概念贯穿业务与 IT，打通了长期以来相对隔离的三大领域：业务流程改进、应用设计与开发以及软件的运维。通过软件架构及策略来支持业务的面向服务特征，通过服务统一企业内的业务能力，对接现有 IT 系统的服务能力。SOA 横贯业务、IT 及运维，必须以业务模型为中心来一致性执行，否则会偏离业务本质，影响企业级实施效果。本质上 SOA 更多强调自顶向下的思路，也会结合自底向上这一方向。另外 SOA 继承了面向对象 (OO-Object Oriented) 及组件设计的很多理论体系，这些均体现在 SOA 设计模式中。SOA 承接了业务架构和业务模型，并渗透到应用、技术以及运维架构层面。
- 领域驱动设计(DDD-Domain Driven Design) 在贯彻企业架构(业务架构)建立通用语言的前提下，针对特定业务领域自始至终贯彻领域(业务)驱动理念。将业务模型(实体模型)与设计模式以及代码有机紧密衔接在一起，更多采取了自底向上的思路。在向整个企业级推广时如果有企业业务架构及业务模型的有力支持，会产生更好的推广效果。DDD 是 SOA 思想在某个业务领域的细化和延伸，适合应用项目级来切入。
- 微服务架构则是在数字化转型下催生的新型分布式架构，强调服务的职责单一、自治性和独立演进，业务上承接了 SOA 服务理念，但在技术上与分布式技术、云、容器等紧密结合。因此微服务需要从 XYZ 三个维度全方位考虑，即微服务本身自治性的业务服务属性、横向扩展属性、数据分片属性。如果片面从

技术角度看待微服务的技术特性，会使微服务的效果大打折扣，例如微服务的业务切分合理性。

因此有必要重新回顾 SOA 方法体系的要点，指导 IT 解决方案乃至“中台”体系的设计：

- 服务。通过服务贯穿业务和 IT 系统架构，通过对功能的封装和暴露提升信息流动性，以及更好的组织灵活性。
- 服务重用和共享。基于服务的软件设计和架构可以有效减少开发和管理(运维)成本。
- 消息。基于消息的通信机制可以带来广泛的积极效应，包括配置灵活性，更易于监控和获取洞察，更好的控制和安全性等。
- 复杂性。服务可以简化软件的复杂度，使得复杂性高的软件更具适应性，也更容易进行方案整合。

自然，SOA 更为重要的理念在于贯穿业务和 IT。下图示出了 BIAN 的基于 SOA 的业务设计原则和技术[3]。



BIAN 的整个理论体系是基于 SOA 的，将“服务”向业务架构层面进行了延伸。通过对银行运营服务能力的组件式切分和能力组件之间交互的定义，刻画了银行业务运营实践。

- BIAN 的服务能力组件是松耦合且独立的 – BIAN 最核心的一个概念是服务域(Service Domain) 。 服务域是业务能力的划分单元，同时也是业务服务的封装单元。服务域之间相互独立、松散且唯一，互不重叠，因此通过服务域中的业务服务之间的协作可以进行真实业务场景建模。
- BIAN 服务域是完整全面的 – BIAN 立足于定义一个完整的银行服务组件集合，所有可能的业务活动都可以通过服务域之间的协作来完成。

- BIAN 服务域是基本的(原子性的) – 服务域仅支持单一的业务目的。服务域是业务服务的最小封装单位，不包含更小更细的服务域，多个识别出的服务域可能构成服务域协作集群。
- BIAN 服务域封装了业务行为和业务对象 – 服务域是银行系统“动态”行为模式与“静态”业务对象的一个结合体，进一步对这些动态行为和静态对象进行了标准化语义层面定义，从而可以准确一致地对银行业务进行描述。

基于上面的业务运营设计理念，BIAN SOA 在进行应用调整时提供了更多可能性:

- 运营能力重用 – 每个服务域构成的独立且唯一的运营能力可以在企业范围内进行广泛使用，这提升了运营能力的重用，将精力放到更需要资源的专业领域，改进资源利用率和水平。
- 运营灵活度提升 – 随着更多业务功能通过共享服务方式对外可用，业务需求和业务模式的变化可以通过服务的调整或重用来更容易地得到支持。在适当的情况下，可以通过生态协作由外部合作方也可以及时提供这些能力。
- 减少信息碎片化和不一致性 – 服务域作为一种 SOA 能力组件成为其所管控的业务信息的单一来源。由于服务域对其自身所管控的信息进行了封装，维护了一个自治疆域。这一特性可以减少数据不一致性和碎片化。
- 性能优化 – 每个服务域都肩负或实现了一个明确定义的业务目的，其内部能力可以针对某个具体操作进行优化。
- 对分布式系统方案的支撑 – 基于围绕服务域的设计框架，BIAN 可以将业务和技术实现串接起来。因为服务域所定义的业务能力划分是独立且唯一的，实现了与其所封装的业务角色相关的业务实体的全生命周期管理。基于这一自治理念，这些能力组件可以和分布式环境(如云和微服务)很好配合在一起。
- 渐进式演进 – 通过 BIAN 服务域可以建立全局统一的服务目录，这样可以将现有银行技术体系(如主机系统和企业服务总线 ESB)和新型的云及微服务体系从整体上进行映射。这样便于企业从全局从业务高度进行应用布局和技术架构演进和治理。

2.2 面向流程与面向组件的方法

谈及 SOA 体系架构的优点有一点需要延伸说明一下。也就是面向流程与面向组件的方法差异和各自的适用场景。

可以说处理逻辑与信息是应用的两大支柱，长期以来传统银行应用更多采取了面向流程的方法，围绕流程或交易进行相关业务信息的准备和访问。这样做有不少优点：

- 业务信息围绕流程逻辑精确定义，相关信息专注在对流程及交易的支持上。
- 业务信息可以加以结构化以确保高效访问，可以为了性能对信息组织方式进行加工，这在主机应用上很普遍。
- 通用的企业参考业务信息可以在可用的地方轻松整合。

然而，随着此类应用的长期演变，多种流程和交易会交织在一起。相关信息结构如果缺乏数据治理也会重重叠叠，越滚越大，也就是耦合性越来越高。这会造成：

- 跨企业模型逻辑业务信息视图的碎片化，从而导致处理及数据的不一致性。
- 不能轻易对设计进行改变和增强以适应变化。

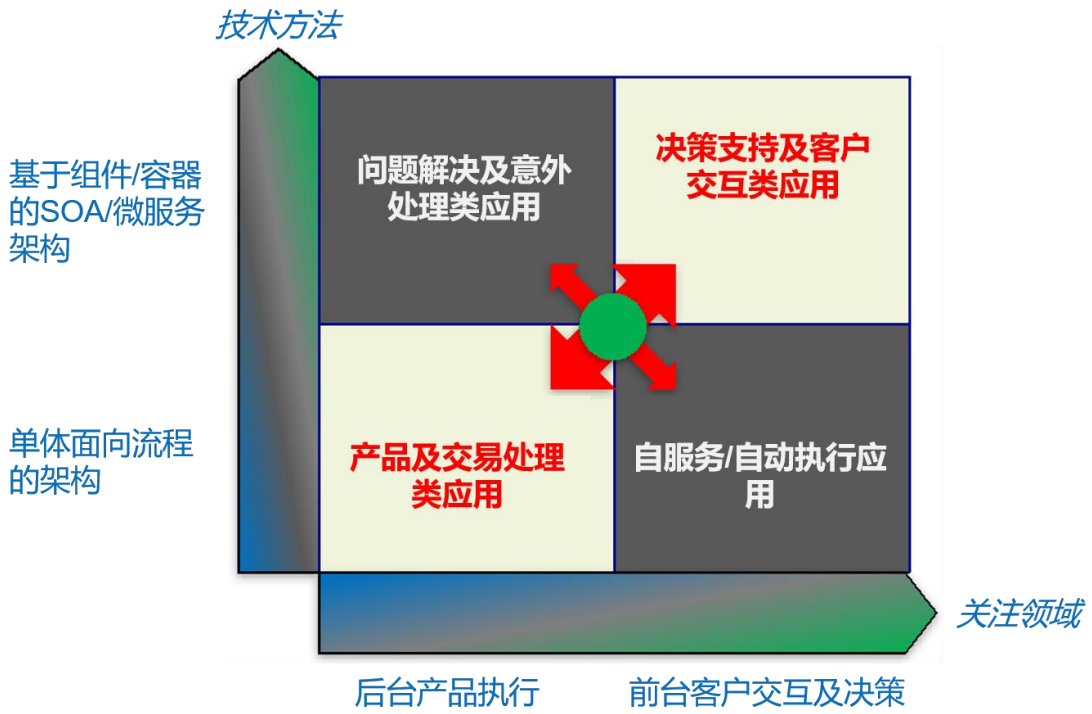
随着数字化转型对应用交付时间和频度的要求不断提升，上述面向流程这一体系架构的弱点逐步凸显。而采取面向组件的方法可以较好应对这一需求：

- 所有的业务信息治理管控功能都唯一地分配到相关的责任实体中，分而治之，各行其责。
- 信息的业务上下文定义良好，避免错误的推断，即在不同的业务环境中使用的类似类型的信息必须始终共享相同的信息值。
- 对信息完整的生命周期进行管理，确保执行适当的动作以维护信息的完整性和流转。

当然，面向组件甚至当下较为流行的微服务方法也有其弱点：

- 提供对单一受管信息的访问会带来延迟或延迟的可能性，以及可能的访问限制和约束。为了改善这一点，从技术架构层面会通过多种手段加以改善，如读写分离，但这又会引入数据一致性和补偿机制等复杂问题。

因此，两种方法有着各自的适合场景，可以在加强治理的前提下结合使用。



流程信息模型最有可能适用于可以优化信息访问性能的后台应用。应用程序之间的链接和边界比较稳定，而且它们所引用的信息集中聚焦，这意味着业务信息的本地访问和共享通常可以在单体信息体系结构中实现。在后台应用开发中也可以引入组件治理

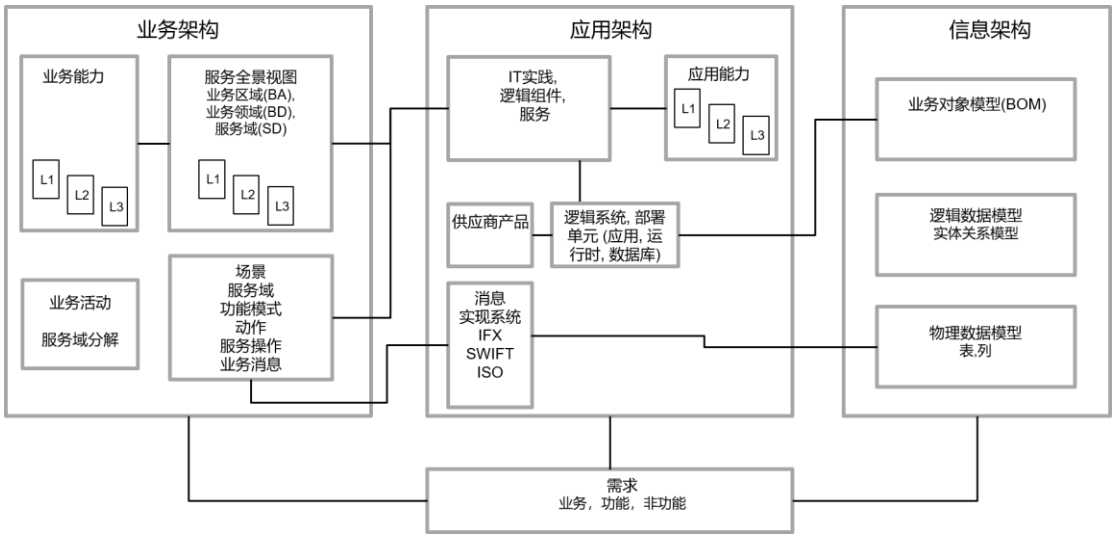
模型，这可能有助于协调应用程序之间的公共信息关联，以限制遗留应用程序中的流程和信息碎片。

组件信息模型最有可能适合前台应用。前台应用可能访问更广泛的信息来源和服务。所管控的信息模型支持在需要时灵活地以任意组合方式进行连接和访问。由于交换的信息由每个服务域自主管理，因此可以根据需要确保其完整性。组件信息模型所维护的清晰的信息上下文、定义和属性还可以跨业务地对所交换的信息值进行正确解读。

BIAN 更多采用了面向组件的方法，并在具体实施时和现有技术进行了映射。当然，在实践过程中特别是近些年来微服务实践过程中还有很多问题值得深入探讨，如服务的粒度问题，数据一致性问题等。这些从宏观角度仍然需要两种方法的结合来考虑。

2.3 BIAN 体系结构

BIAN 可以与企业已经或行将采用的企业架构体系结合起来，丰富企业架构的内涵和具体可操作性。BIAN 的整体架构如下图所示 8。



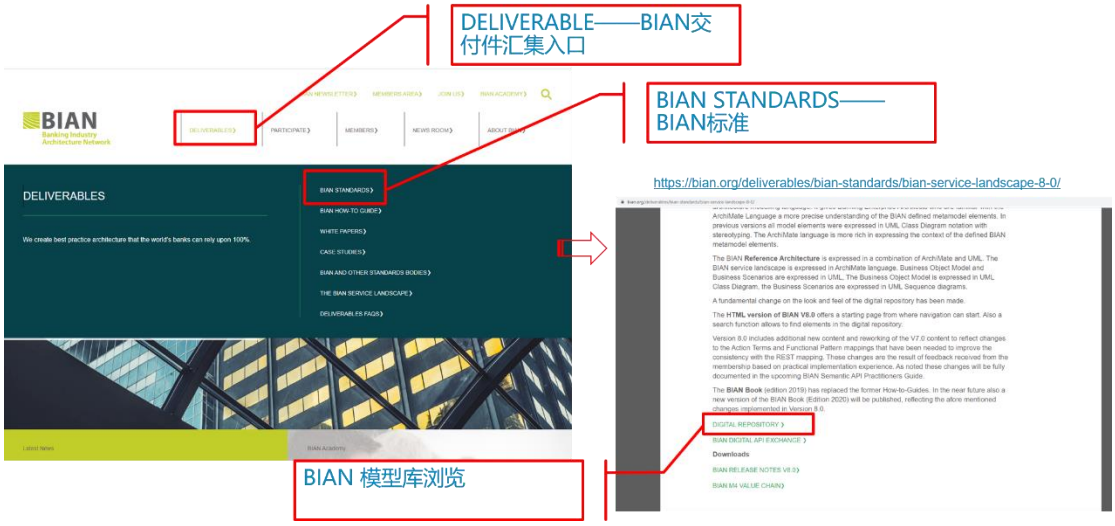
可以看到，BIAN的整个服务全景视图涵盖了企业架构的很多内容，形成了一整套架构资产，包括：

- BIAN 元模型 (Meta Model)，基于ISO 20022元模型;
- BIAN 业务术语 (Business Vocabulary);
- BIAN 高阶参考地图: BIAN服务全景视图 (BIAN Service Landscape);
- BIAN 业务架构 (Business Architecture);
- BIAN 业务能力模型 (Business Capability Model);
- BIAN 服务域定义 (Service Domain Definitions);
- BIAN 服务操作定义 (Service Operations Definitions);
- BIAN 业务场景定义 (Business Scenario Definitions);
- BIAN 应用程序接口/消息定义 (API/Message Definitions);
- BIAN 信息架构 (Information Architecture);
- BIAN 业务对象模型 (Business Object Model), 完全与 ISO 20022一致

BIAN 以UML模型库的形式进行发布，同时也以 HTML 格式提供只读版本。大家可以访问 BIAN website (https://www.bian.org/)来进行访问，后面的介绍所引用的模型快照均来自这一网站。另外，每一个BIAN标准的发布版本还带有相应的支持文档并随版本演进而进行维护。

2.4 BIAN 在线资源

BIAN是一个开放模型体系，一个不错的学习方式是直接访问 bian.org 网站，另外一个方法是从网站下载相关文档来学习。下面我们结合实际网站 bian.org 浏览一下BIAN的模型体系和文档体系。



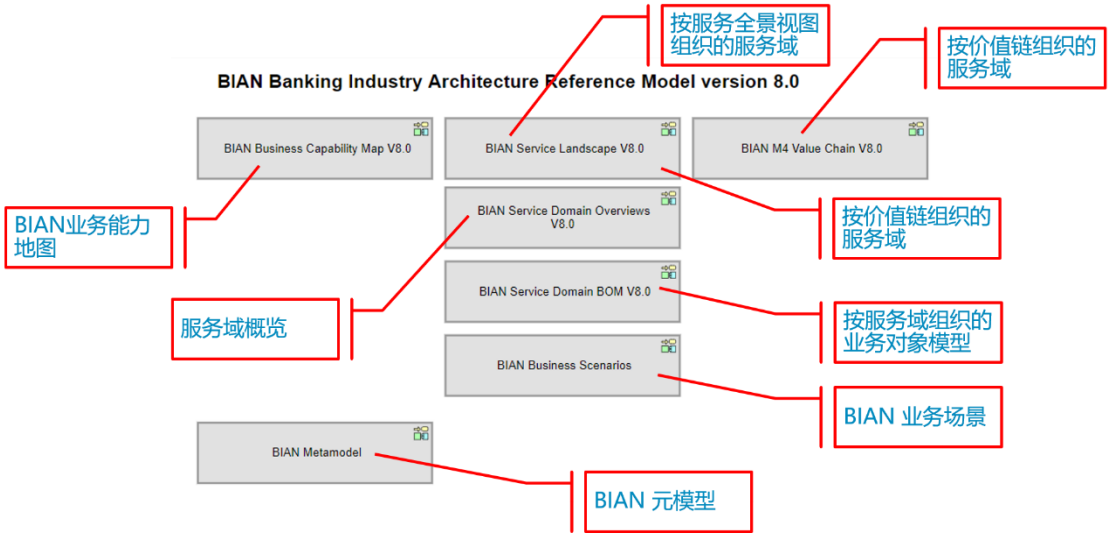
通过bian.org主页下的 DELIVERABLE 入口进入BIAN交付件入口，然后选择 BIAN STANDARDS 进入BIAN的标准。在DIGITAL REPOSITORY 即可进入BIAN在线模型库。

在BIAN STANDARDS 下的“BIAN HOW TO GUIDE”是BIAN 文档的一个完整集合。其目标读者主要是技术架构师，BIAN成员及其他金融机构。这些指南文档对于细化BIAN全景视图背后的理论体系并进行设计实践很有帮助。这些文档包括：

- “BIAN How-to Guide - INTRODUCTION TO BIAN V7.0”是一个整体How-to文档指南
- “BIAN How-to Guide - DESIGN PRINCIPLES TECHNIQUES V7.0”介绍了BIAN的设计原则及技术
- “BIAN How-to Guide - DEVELOPING CONTENT V7.0”介绍了BIAN标准体系的开发
- “BIAN How-to Guide - APPLYING THE BIAN STANDARD V7.0”简要描述了如何运用BIAN
- “BIAN How-to Guide - SEMANTIC API V7.0”是介绍BIAN API体系的文档

大家可以利用这组“How-to Guide” 作为在自己企业实施BIAN模型的工具。本文也会简要回顾一下这些文档中的要点。

下面先简单浏览一下BIAN的在线模型库 https://bian.org/servicelandscape-8-0/ 。下图的模型体系总图中包含了三条主线：其中中轴线是BIAN的主线，包括了服务域及服务操作、贯穿服务域的业务场景以及业务对象模型BOM。另外两条线是近期BIAN标准的演进方向，包括业务能力模型以及价值链模型。



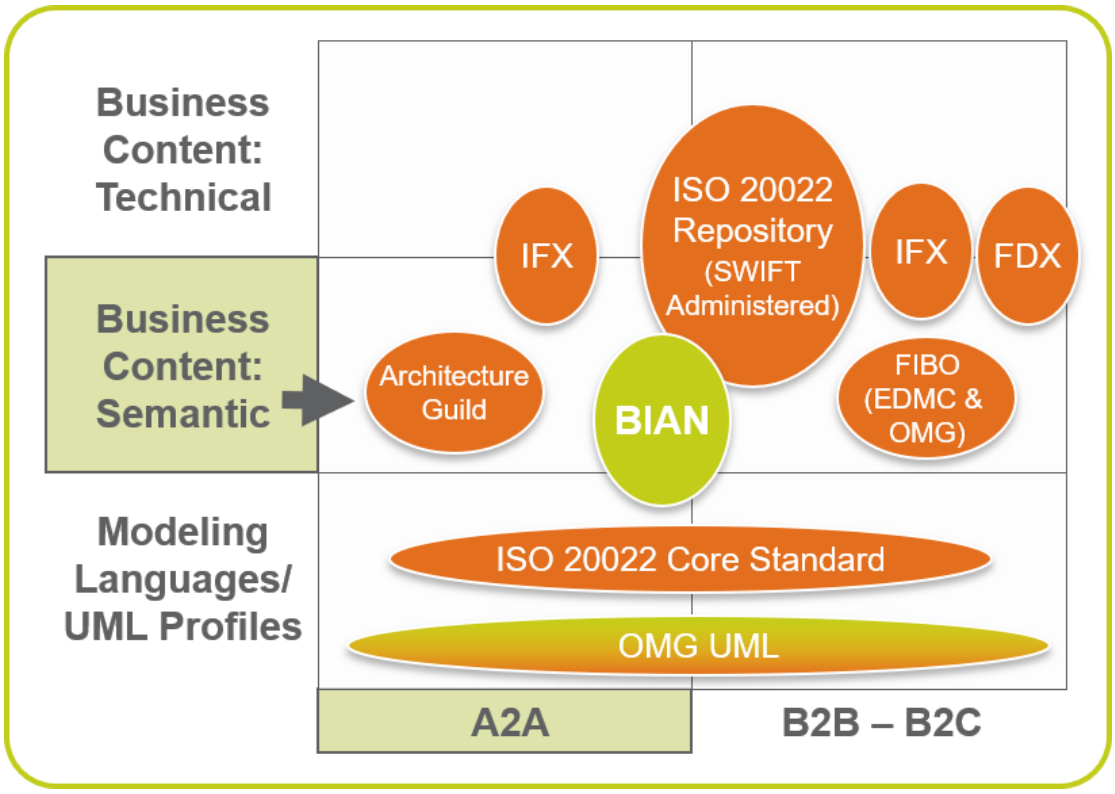
在下面的概念说明中会结合这里的模型体系进行说明。

3 BIAN 关键概念

除了服务域(Service Domain)，BIAN 基于业界标准，如 ISO20022， SWIFT，IFX(Interactive Financial eXchange)，FIBO(Financial Industry Business Ontology, EDM Council 与 OMG 联合提出的金融业务本体模型)等引入了一组概念，这些概念或多或少与其他方法如企业架构、面向服务架构 SOA 有着映射关系。企业在参照 BIAN 时可以结合企业现有的架构体系概念进行映射。下面我们结合 BIAN 的在线模型库展开分析和说明一下这些概念。

3.1 应用对应用(APPLICATION 2 APPLICATION)

也称企业应用整合(EAI-Enterprise Application Integration)。使用软件及计算机系统的架构原则来进行企业计算机应用的整合。下图清晰地说明了BIAN与几个业务标准以及A2A/B2B的相关关系。



另外，上图很清晰地阐明了BIAN的关键侧重点：

- 侧重应用到应用 (A2A)，与IFX和SWIFT侧重B2B形成互补。近些年来随着欧盟PSD2(支付服务指令 Payment Service Directive 2)，TPP(Third Party Provider第三方提供商)以及开放银行API体系的整体发展态势，BIAN的整体重心也逐渐从A2A转向A2A/B2B兼顾。
- 重心完全放在语义定义上 – 技术定义不包括在正式工作产品中，这样有助于平衡其他已经开展的行业工作。
- BIAN遵循IFX， 以及OMG(Object Management Group)，ISO 20022和SWIFT标准，与这些标准保持一致和沟通。
- 面向服务，而IFX， SWIFT，以及 ISO 20022 都是面向消息的。
- UML(Unified Modeling Language) 是基础技术，在金融服务业使用广泛。

3.2 服务全景视图 (Service Landscape)

是涵盖了每个已标识服务域的参考模型，300多个服务域以分组形式进行组织，帮助从全局角度来理解、识别和选择。

3.3 业务区域 (Business Area)

BIAN中的最高阶分类，业务区域将广泛的业务能力组合在一起。 就BIAN服务全景视图而言，业务区域从一个侧面定义了具有相似的应用支持和特定信息需求的业务活动。

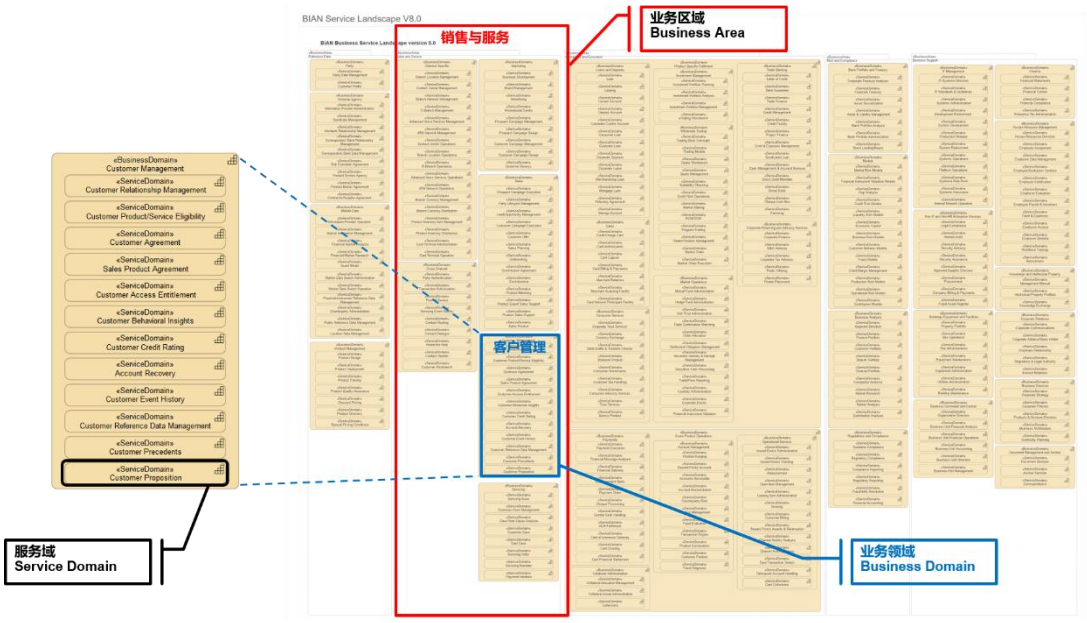
BIAN的业务区域分为：参照数据(Reference Data)，销售和服务(Sales and Service)，运营和执行(Operation and Execution)，风险和监管合规(Risk and Compliance)，业务支撑(Business Support)5个。

3.4 服务域 (SERVICE DOMAIN)

服务域封装了服务操作(Service Operation)或者业务服务，服务操作则定义了业务能力积木块之间的交互。服务域进而在组织，过程和支持它们的相关信息系统方面进行了更为全面的定义。尽管通常会将注意力集中在IT系统和系统接口上，但是这种关注不应掩盖一个事实，即服务域代表了一种业务能力划分，企业可以将其人员，流程和系统从业务层面一致性地进行整合封装。

3.5 业务领域 (BUSINESS DOMAIN)

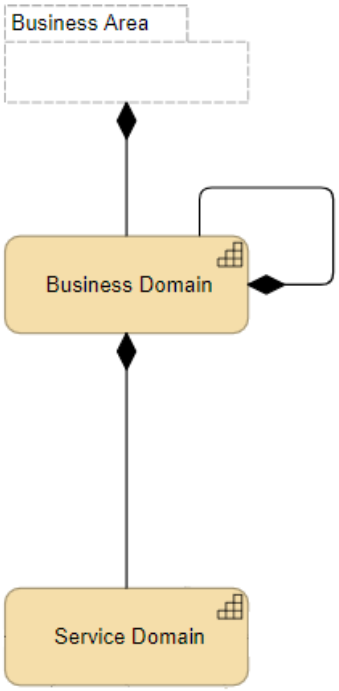
是介于业务区域(Business Area)和服务域(Service Domain)之间的一个层级，业务领域(Business Domain)在更广泛的业务区域中定义了一系列一致的能力。 在BIAN服务全景视图中，业务领域与金融服务业务中可识别的技能和知识相关。例如在销售和服务(Sales and Service)业务区域下包含了一系列与销售和服务相关能力：具体渠道(Channel Specific)，跨渠道(Cross Channel)，市场营销(Marketing)，销售(Sales)，客户关系管理(Customer Relationship Management)，服务(Serving)。下图示出了BIAN服务全景视图中的层次结构：业务区域、业务领域和服务域。



结合服务领域的概念，可以进一步进行深入拆分银行业务能力。例如客户管理(Customer Management)这一业务领域进一步包含了12个服务领域：

- 客户关系管理(Customer Relationship Management)

- 客户产品/服务资质(Customer Product/Service Eligibility)
- 客户协议(Customer Agreement)
- 销售产品协议(Sales Product Agreement)
- 客户访问权限(Customer Access Entitlement)
- 客户行为洞察(Customer Behavioral Insights)
- 客户信用评级(Customer Credit Rating)
- 账户恢复(Account Recovery)
- 客户事件历史(Customer Event History)
- 客户参照数据管理(Customer Reference Data Management)
- 客户先例(Customer Precedents)
- 客户主张(Customer Proposition)



上图用BIAN元模型片段示出上述几者之间的关系。业务领域可能会有嵌套，例如在业务区域运营与执行下存在两个较大的业务领域：具体产品实现(Product Specific Fulfilment)和跨产品运营(Cross Product Operation)。这两个业务领域下又各自包含了较小的业务领域。

3.6 服务操作 (SERVICE OPERATION)

业务对象的生命周期变动是通过服务域之间的服务操作(Service Operation)调用或执行动作来触发的(也称事件触发)，或者通过内部调度机制根据条件或时间点定期进行触发(也称条件触发和时间触发)。可以将服务操作理解为具体的业务服务，目前BIAN已经识别了2,000多个。结合下图的元模型和具体例子可以加深对服务操作的理解。服务组(Service Group)实现了服务域(Service Domain)，服务组包含了多个服务操作

(Service Operation)。服务操作涉及具体的业务服务，因此与前置条件(precondition)、后置条件(post condition)和消息(Message)相关。

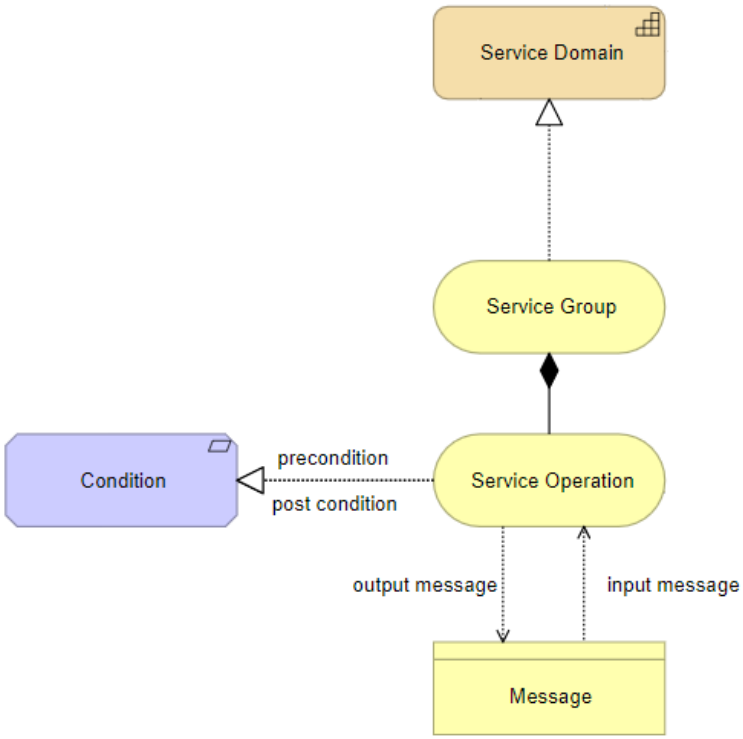
例如服务域客户关系管理(Customer Relationship Management)包含了三个服务组：

- CustomerRelationshipManagementPlan_Invocation（客户关系管理计划-调用）
- CustomerRelationshipManagementPlan_Reporting（客户关系管理计划-报告）
- CustomerRelationshipManagementPlan_Origination（客户关系管理计划-创建）

在服务组 CustomerRelationshipManagementPlan_Invocation 下又包含了几个具体服务操作：

- requestCustomerRelationshipManagementPlan（请求客户关系管理计划）
- recordCustomerRelationshipManagementPlan（记录客户关系管理计划）
- terminateCustomerRelationshipManagementPlan（中止客户关系管理计划）

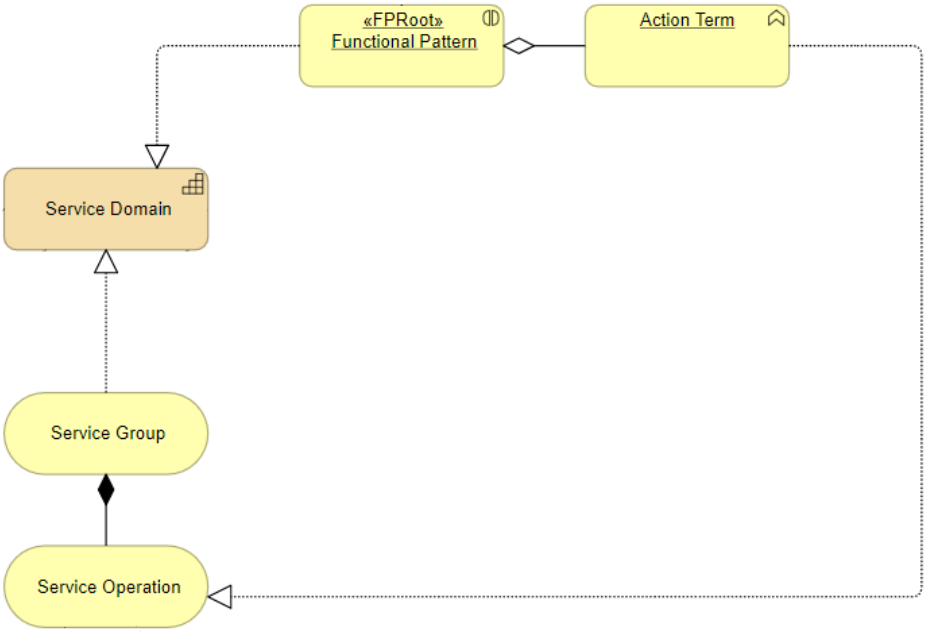
同样，在下图用BIAN元模型将这几个概念一起进行说明。



3.7 功能模式(FUNCTIONAL PATTERN) 与动作术语(ACTION TERM)

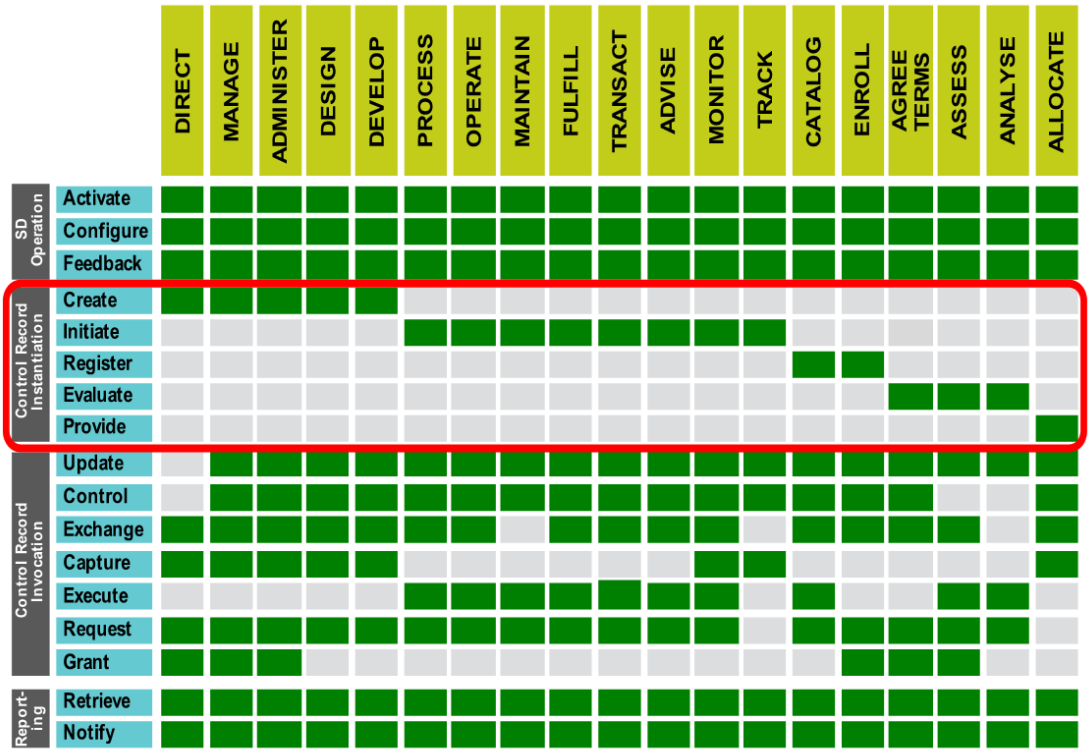
每个服务域都有一个主导功能模式，也就是服务域所提供的业务功能的主要类型或特征。例如：信用卡(Credit Card)和储蓄账户(Savings Account)的主导功能模式是Fulfill(实现或履行-根据客户要求进行产品/服务的交付)，可售产品(Sales Product)和商户关系(Merchant Relations)的主导功能模式是Agree Terms(同意服务条款)，网点货币投放(Branch Currency Distribution)的主导功能模式是Process(处理)，ATM网络运营(ATM Network Operations)的主导功能模式是Operate(运营)。目前抽象出的功能模式共19个。每个具体服务操作则对应了一个具体的动作术语(BIAN v8共定义了17个动作术语)。每个主导功能模式包含了多个动作术语。

重用上面的例子，服务域客户关系管理(Customer Relationship Management)的主导功能模式是Manage(管理)。服务组CustomerRelationshipManagementPlan_Invocation下的三个服务操作的动作术语Action Term分别是Request(请求)、Record(记录)和中止(Terminate)。



从上面的例子可以看出，动作术语刻画了所提供服务的目的(例如请求、记录或中止)。动作术语采用了类似MECE的设计原则，相互之间互不重叠，而整体上覆盖了任何服务域所支持的主要服务交换类型。

BIAN v8建立了功能模式与动作术语之间的缺省组合关系(如下图所示)。当然在某些情况下一些额外的服务操作会引入特定的动作术语，而与默认组合关系不同，这是很正常的。这一缺省组合的意义在于给开发人员了解API后面的银行语义，帮助开发人员快速了解相关业务领域(服务域)的主要功能和特征与服务操作(API)的组合关系。



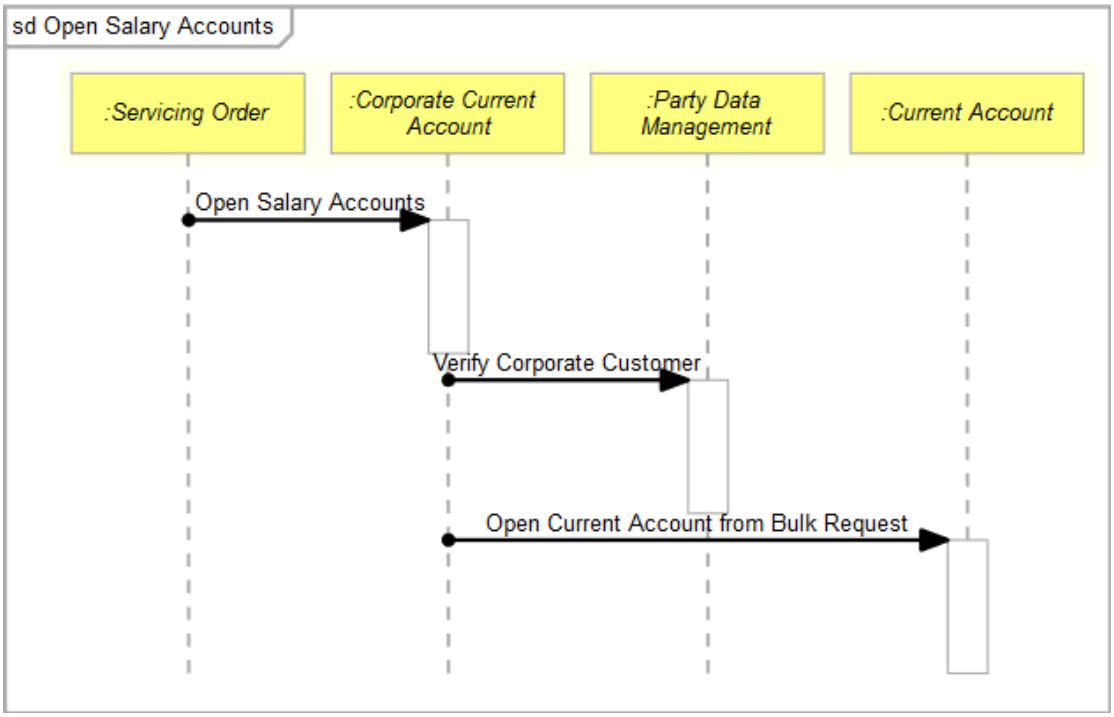
上图的横栏是功能模式(共19个)，左侧纵向是动作术语(共17个)。高亮显示了控制记录(Control Record，服务域整个执行过程中的全程记录，后面会介绍)实例化的5种类型及其对应的动作术语。这样可以更好从“动”的一面了解银行操作语义。

- Create(创建)。创建什么呢？通过动作术语来理解其语义，DIRECT(指导)，MANAGE(管理)，ADMINISTRATER(管控)，DESIGN(设计)，DEVELOP(开发)。可能涉及战略计划，管理策略，高阶管控方案，设计方案，开发计划。
- Initiate(初始化/启动)。启动什么呢？PROCESS(处理)，OPERATE(操作)，MAINTAIN(维护)，FULFILL(履行)，TRANSACT(执行)，ADVISE(建议)，MONITOR(监控)，TRACK(跟踪)。一般涉及流程或动作。
- Register(注册/登记)。CATALOG(加入列表)，ENROLL(加入目录)。
- Evaluate(检查以得到结论)。动作术语有AGREE TERMS(协议条款)，ASSESS(评估)。可能涉及度量(measurement)，测试(test)，条件(condition)，分析结果(analysis)。
- PROVIDE(提供)。ALLOCATE(分配)。涉及从库存中进行分配的资源/实物等。

3.8 业务场景 (BUSINESS SCENARIO)

如任何流程模型一样，业务场景为响应一个业务事件定义了一组服务域之间的交互链接序列。同时，业务场景也清晰定义了这一序列中相关的具体服务域，以及为执行每个动作而发生在服务域之间的交互。业务场景在BIAN中被分为两类：

- 一阶业务场景(Business Scenario First Order)。一阶交互是业务场景的简单/受限形式，利用一阶交互场景可以组装更为复杂的端到端业务场景(Business Scenario End-2-End)以及线框图(wireframe)。 它记录了单个“主要”服务域对该事件的响应，并仅捕获与该主要服务域和任何其他服务域的一阶服务交换。它显示了所选服务域（主要服务域）如何根据业务事件调用或委派任何服务域来进行响应，以便能够处理事件。因此，一阶交互场景的主要目的是识别服务域之间的服务操作连接，从而对这些连接模式进行组合以便支持更为完整/复杂的业务场景。目前一阶业务场景大多是根据经验总结出来的，因此并不是完整的银行流程集合，但可以帮助开发人员理解整个服务域的范围及目的，从而更好地处理不同的业务需求。另外，一阶业务场景提供了服务域所提供服务的某种上下文环境。当前版本的一阶业务场景已超过1000个。下面是一个批量开工资户(Open Salary Account)的例子。涉及到4个服务域(服务订单Servicing Order, 对公往来户Corporate Current Account, 相关方数据管理 Party Data Management以及个人往来户Current Account)。

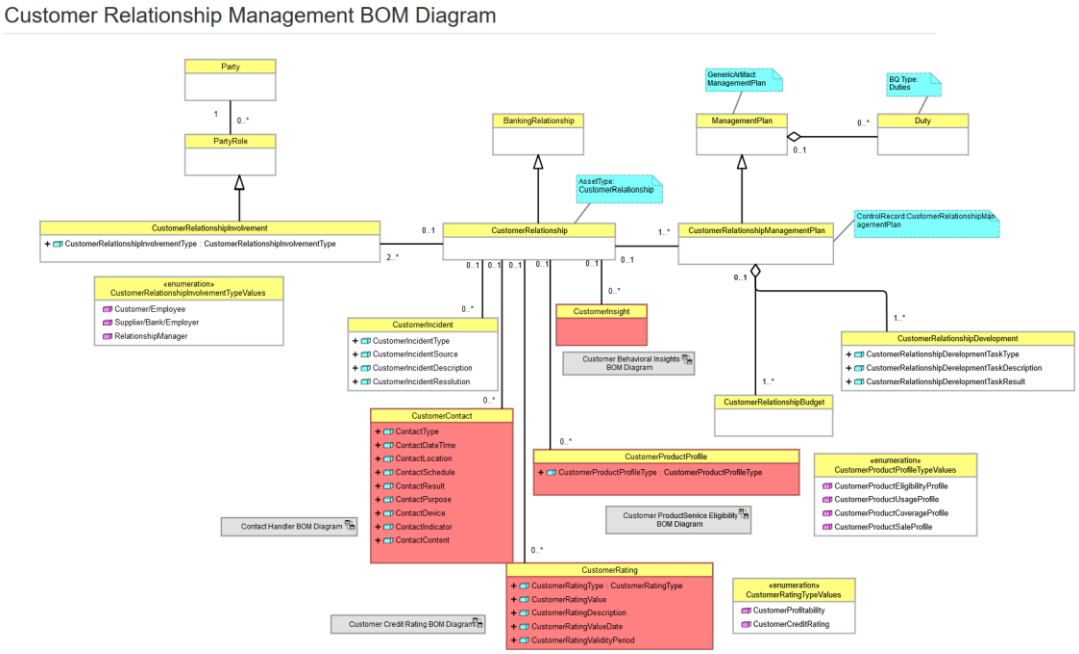


- 端到端业务场景(Business Scenario End-2-End)，更为完整/复杂的端到端业务场景。稍后在BIAN框架概览部分会结合实例进行说明。

上面介绍的概念和功能域与服务相关，在BIAN中另一条线是数据与信息。

3.9 业务对象模型 (BOM)

BIAN 当前开发的业务对象模型与ISO 20022模型可以交叉引用。BIAN BOM提供了每个服务域所管控的更为详尽的业务信息。下图示出了BIAN BOM的一个例子，也是我们上面提到的服务域Customer Relationship Management 所涉及的业务对象。

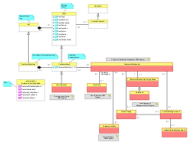


由于业务对象可能跨服务域共享，例如业务对象 CustomerContact (客户联系)在Customer Relationship Management 进行维护，但可能为其他服务域的BOM所引用，例如Contact Handler(客户联系处理),Channel Activity History(渠道活动历史记录)，Customer Event History(客户事件历史)，服务事件历史(Servicing Event History)，Point of Service(服务场点)，渠道活动分析(Channel Activity Analysis)，Fraud Diagnosis(欺诈诊断)，Delinquent Account Handling(拖欠账户处理)，Contact Dialogue(接触对话)，Transaction Authorization(交易授权)，Contact Routing(联系路由)，Card Case(卡案件)，lead/Opportunity Management(机会管理)，Customer Case(客户案件)，Fraud Resolution(欺诈解决)，EBranch Operation(电子网点运营)，Card Collection(卡托收)，Advanced Voice Operation(高级语音服务的运营)等等。下图示出了引用BOM的图示。

这样，通过服务域 BOM 逐步建立起企业级业务对象模型，在业务层面进行关键业务概念的关系梳理。

Contact Handler BOM Diagram

Channel Activity History BOM Diagram


Customer Event History BOM Diagram

Servicing Event History BOM Diagram

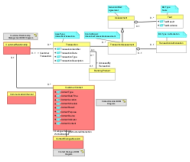
Point of Service BOM Diagram

Channel Activity Analysis BOM Diagram

Fraud Diagnosis BOM Diagram

Delinquent Account Handling BOM Diagr...

Contact Dialogue BOM Diagram


Transaction Authorization BOM Diagram

Contact Routing BOM Diagram

Card Case BOM Diagram

Lead/Opportunity Management BOM Dia...

Customer Case BOM Diagram

Customer Relationship Management BO...

Fraud Resolution BOM Diagram

EBranch Operation BOM Diagram

Card Collections BOM Diagrams

Advanced Voice Operation BOM Diagram

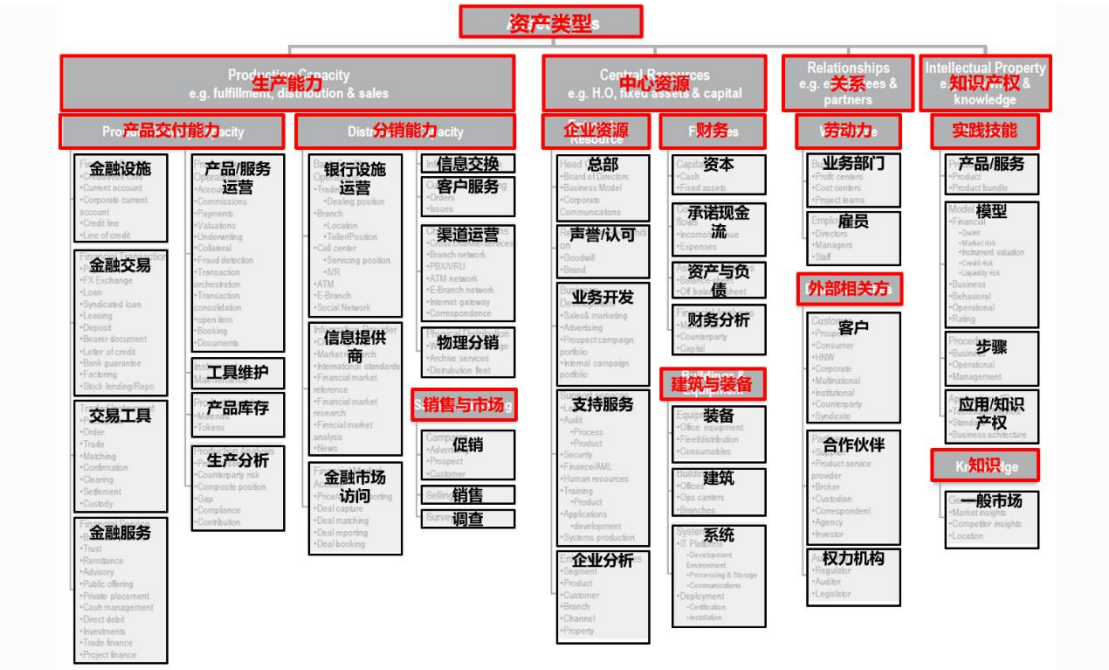
Customer Contact Overview Diagram

Customer Contact Helper Diagram

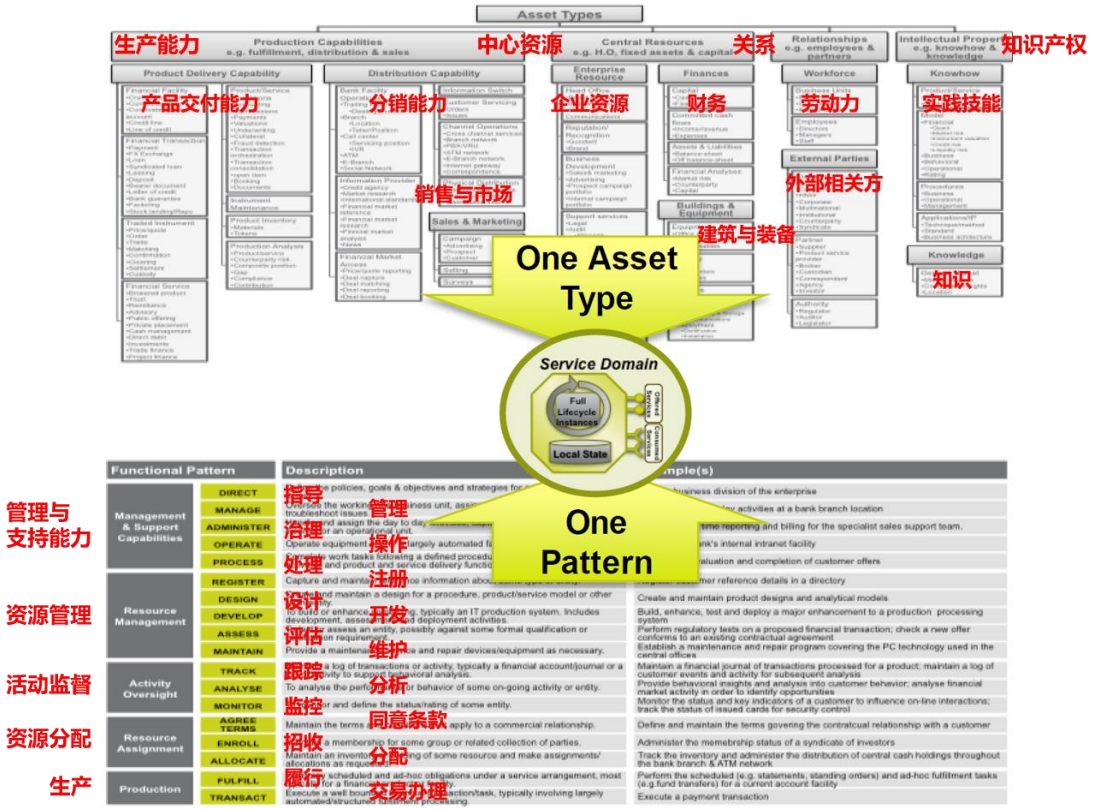
Event Helper Diagram

3.10 资产/实体 (ASSET/ENTITY)

服务域相关的功能模式所作用于的业务资产或实体/对象。是银行可以拥有或影响其行为的有形或无形的事物，例如客户关系，现金或支付能力。资产类型可以看作是业务对象的某种分类。BIAN根据MECE法则(Mutually Exclusive Collectively Exhaustive “相互独立，完全穷尽”)对银行资产进行了分类和盘点。



结合功能模式(Functional Pattern)服务域实际上代表了作用于一类特定资产类型(Asset Type)实例上的某种商业行为模式(Functional Pattern)。下图清晰表示了这一“动静”关系。



3.11 通用工件 (GENERIC ARTIFACT)

功能模式描述了执行的功能的类型。 为了使功能模式的作用更加清晰，BIAN引入了相关的设计元素， 这就是“通用工件(Generic Artifact)”。 通用工件描述了在跟踪服务域的操作从头到尾完成其操作时使用/产生的工件/文档的类型。下表显示了为每种功能模式定义的不同通用工件。

	Functional Pattern	Generic Artifact	
指导	DIRECT	Strategy	战略
管理	MANAGE	Management Plan	管理计划
治理	ADMINISTER	Administrative Plan	治理计划
设计	DESIGN	Specification	规范
开发	DEVELOP	Development	项目
处理	PROCESS	Procedure	步骤
操作	OPERATE	Operating Session	操作会话
维护	MAINTAIN	Maintenance Arrangement	维护协议
履行	FULFILL	Arrangement	合约
(交易)办理	TRANSACT	Transaction	交易
建议	ADVISE	Advice	建议
监控	MONITOR	State	状态
跟踪	TRACK	Log Record	日志记录
维护(参考信息)	CATALOG	Directory Entry	目录条目
注册	ENROLL	Membership	会员
(条款)认同	AGREE TERMS	Agreement	条款
评估	ASSESS	Assessment	评估
分析	ANALYSE	Analysis	分析
分配	ALLOCATE	Allocation	分配

添加通用工件的一个好处是，通过引用一些更具体的内容(名词)来更好地描述功能模式。例如功能模式“跟踪”不管跟踪什么用“日志”来记录。通用工件与所作用的资产类型(Asset Type)结合在一起，以定义服务域的控制记录(Control Record)。这样就完整地记录了服务域的目的(功能模式作用于资产类型以产生业务价值)以及其产生价值的全过程(控制记录将过程记录在通用工件实例中)。

例如，处理资产类型“客户关系(customer relationship)”并执行功能模式“同意条款(Agree Terms)”和相关通用工件“协议(Agreement)”的服务域具有控制记录“客户关系协议(Customer Relationship Agreement)”。 只要银行知道客户并且客户对银行有一定兴趣，服务域就会为每个客户创建并维护控制记录（客户关系协议）的实例。另一个例子是ATM网络，该网络由“操作(Operate)”功能模式作用于通用工件

“操作会话(Operating Session)”，从而产生控制记录：ATM网络操作会话 (ATM Network Operating Session)。

BIAN在模型库中维护了资产/实体的分解结构，已识别服务域的功能模式和相关的控制记录之间的关系。

3.12 行为限定符类型 (Behavior Qualifier Type)

为了给服务域的服务操作(Service Operation)和所管控的信息的规范提供足够的细节，服务域的行为特征被进一步细分，这就是“行为限定符类型”。行为限定符类型定义了如何将功能模式(Functional Pattern)进一步分解成多个组成元素。行为限定符类型与功能模式在本质上差异很大，而且与具体的服务域紧密相关。必要时使用行为限定符可以阐明服务域及其所提供的服务的工作性质，更准确地描述其其目的。行为限定符也可以是定义由服务域管理的信息的更详细规范的基础。

行为限定符类型中的“类型”定义了行为细分的性质应该是什么样的，但是细分工作本身必须针对某个具体的服务域展开。

例如，服务域“干系方身份验证(Party Authentication)”具有“评估(Assess)”功能模式，其业务目的是“评估”个人身份是否正确。 “评估”功能模式的行为限定符类型是“测试(tests)”。那么具体的行为限定符就是服务域用来检查身份而执行的具体测试类型（例如密码检查，生物特征测试等）。

再以“个人往来户(Current Account)”服务域为例，其中涵盖了很多业务请求，例如打印对账单，发起转账，设置定扣业务等。为了清晰表达所请求的具体业务或产品特性，可以使用产品特性作为行为限定符类型，而具体的银行产品如支付、存取款、转账等则是行为限定符。

下图集中示出了功能模式、通用工件及行为限定符类型几者的对应关系。

Functional Pattern	Generic Artifact	Behavior Qualifier Type	Example
DIRECT	Strategy	Goals	Increase market share
MANAGE	Management Plan	Duties	Relationship development, Troubleshooting
ADMINISTER	Administrative Plan	Routines	Time-sheet recording
DESIGN	Specification	Aspects	Business requirements
DEVELOP	Development	Deliverables	Functional module specifiation
PROCESS	Procedure	Worksteps	Invoice generation
OPERATE	Operating Session	Functions	Message capture/routing
MAINTAIN	Maintenance Arrangement	Tasks	Preventive maintenance task
FULFILL	Arrangement	Features	Current account standing order
TRANSACT	Transaction	Tasks/Steps	FX pricing, market trade, clearing & settlement
ADVISE	Advice	Topics	Tax advice, Corporate finance
MONITOR	State	Measures	Composite position, Customer alert
TRACK	Log Record	Events	Customer life event, Servicing event
CATALOG	Directory Entry	Properties	Product pricing rules, Customer reference details
ENROLL	Membership	Clauses	Qualification/membership purpose
AGREE TERMS	Agreement	Terms & Conditions	Required disclosures,
ASSESS	Assessment	Tests	Password verification
ANALYSE	Analysis	Algorithms	Average balance calculation, Propensity to buy
ALLOCATE	Allocation	Criteria	Staff assignment, Facility allocation

3.13 控制记录(CONTROL RECORD)

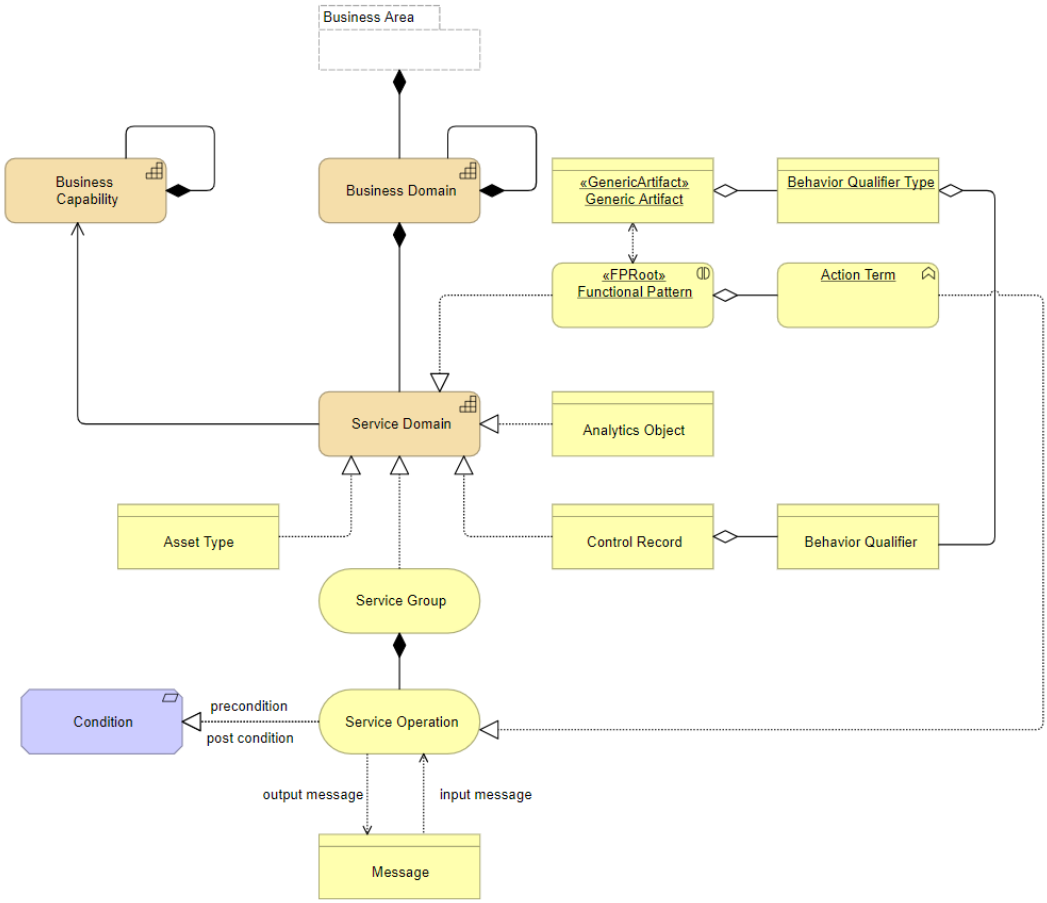
管理服务域中对象的生命周期状态。服务域在对具体资产类型(Asset Type)的实例执行功能模式(Functional Pattern)，从而行使其职责时利用控制记录对每次执行过程自始至终进行(状态的)记录和跟踪。如果将服务域的动静两方面用“动宾”结构来比喻，控制记录可以看成是对“动宾”执行过程的记录。从信息角度看资产类型及通用工件的组合定义了服务域的控制记录。例如功能模式操作(OPERATE)的通用工件是操作会话(Operating Session)，操作的作用对象或资产类型可能是ATM，这样操作会话及ATM构成了操作ATM这一动作的全部控制记录。开发人员对控制记录的规约应该非常感兴趣，因为其包含了服务域所管控的主要业务信息。并且控制记录包含了非常广泛的信息，既包含了控制处理所需的所有信息，也包含了任何可能会被引用到的信息，如服务域在完成一个工作周期期间所产生的任何信息。

3.14 分析对象 (ANALYTICS OBJECT)

服务域中用来存放分析信息的业务对象。

3.15 BIAN 元模型

结合 BIAN 的整个元模型，可以加强对上面概念的理解。

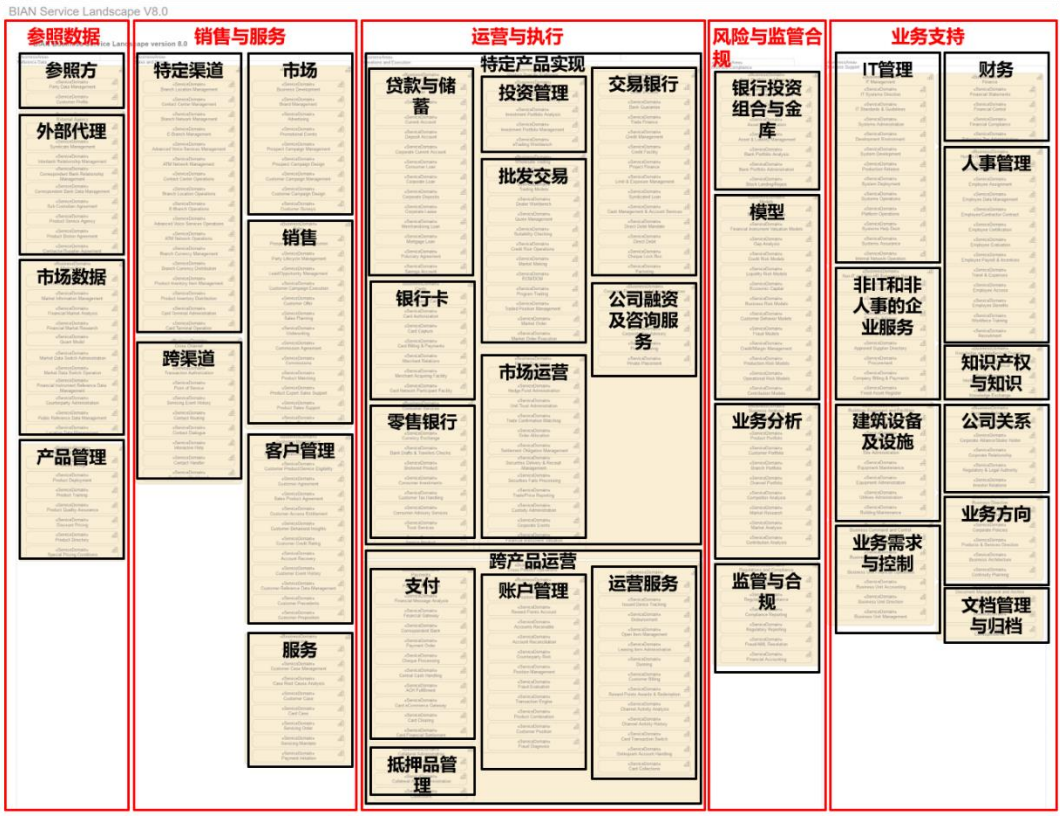


4 BIAN 框架概览

在上面介绍 A2A 的概念时提到，BIAN 的目标是定义标准语义的服务操作 (Service Operations)，特别着重于金融机构的内部运营以帮助改善银行的内部运营绩效。 虽然 BIAN 的服务操作(Service Operations)涵盖了所有类型的业务服务交互，但其重点是基于系统的交互。因此其更为具体的一个潜在目标是改进银行内部应用对应用（A2A）的互操作性。

4.1 通过业务场景贯穿服务全景视图

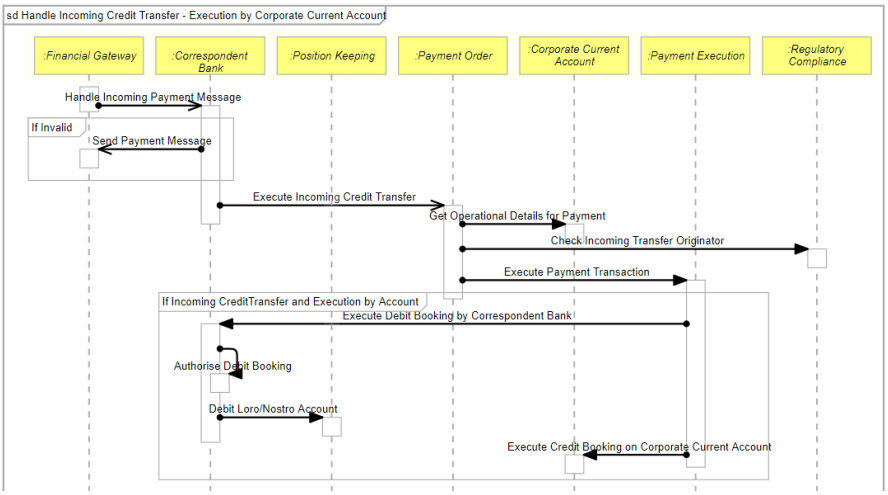
BIAN SOA 设计框架旨在包含任何一家银行都可以运用的所有业务功能。通常，一家银行可能只需要此集合的一部分功能。回顾一下服务域这一关键概念，BIAN 开发了相应的设计原理和支持技术以将金融服务能力划分为松散、非重叠的 “服务域 (Service Domain)”。 服务域的整个集合 BIAN 被称之为服务全景视图(Service Landscape)。下图示出了 BIAN 8.0 的服务全景视图，目前业务领域大约 30 多个，服务域大约 300 多个。这些服务域实际上类似业务架构中的业务组件。



所有可能的金融业务活动都可以从上述全景视图中选择一部分服务域 Service Domain，并对其之间的协作模式建模。BIAN 使用称为 BIAN 业务场景（Business Scenario）的非正式表示为服务域交互的示例建模，这与 UML Sequence Diagram 非常类似。BIAN 业务场景提供了银行业务行为的一个上下文环境，以此来清晰区分 BIAN 服务域的角色及其之间的交互。 BIAN 业务场景不是正式设计，而只是一种可能的协作模式的原型实例。从这个角度看，BIAN 目前的目标并非提供一个正式的银行业务流程模型，而是通过业务场景来对服务域之间的交互进行建模，以此验证服务域定义的合理性。

在概念部分介绍业务场景时用了一个一阶业务场景的例子。下图示出了一个端到端(E2E)业务场景的示例，一个由企业往来户执行的支付请求。我用这个例子再把上面的一些概念贯穿起来帮助大家加深对 BIAN 的基本概念的理解。

Handle Incoming Credit Transfer - Execution by Corporate Current Account

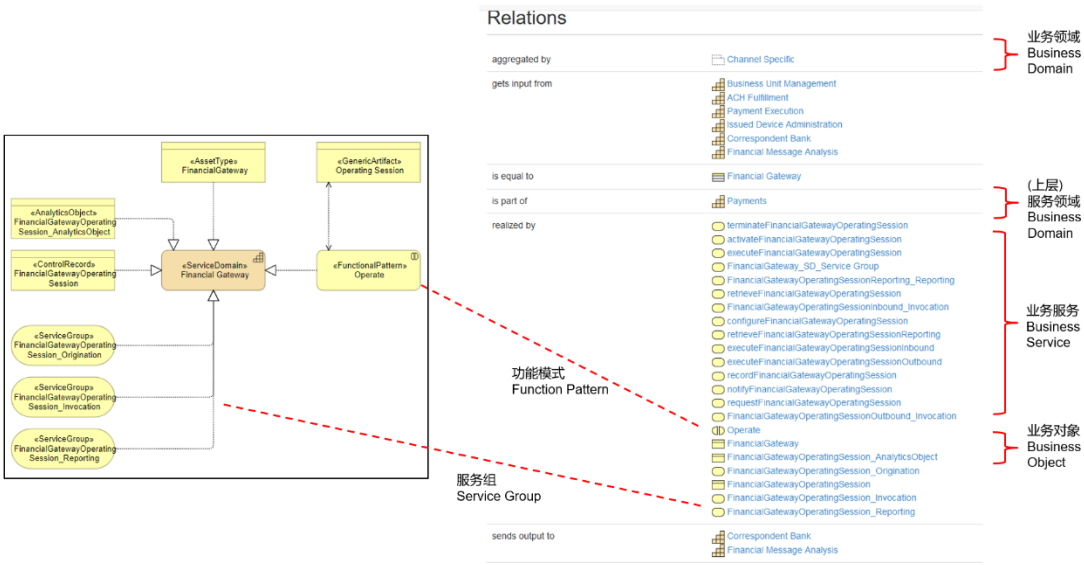


上图以 UML Sequence Diagram 示出的业务场景涉及了 7 个类实例之间的交互，而这 7 个类分属不同的服务域。

服务域之间的交互通过“服务操作”的提供和消费进行建模。实际上，一个服务域可以参与到任何业务场景之中。但由于一个服务域始终实现唯一且独立的业务目的(业务功能积木块)，它提供(和消费)的服务操作可以定义为一个相对固定或“有边界”的集合。

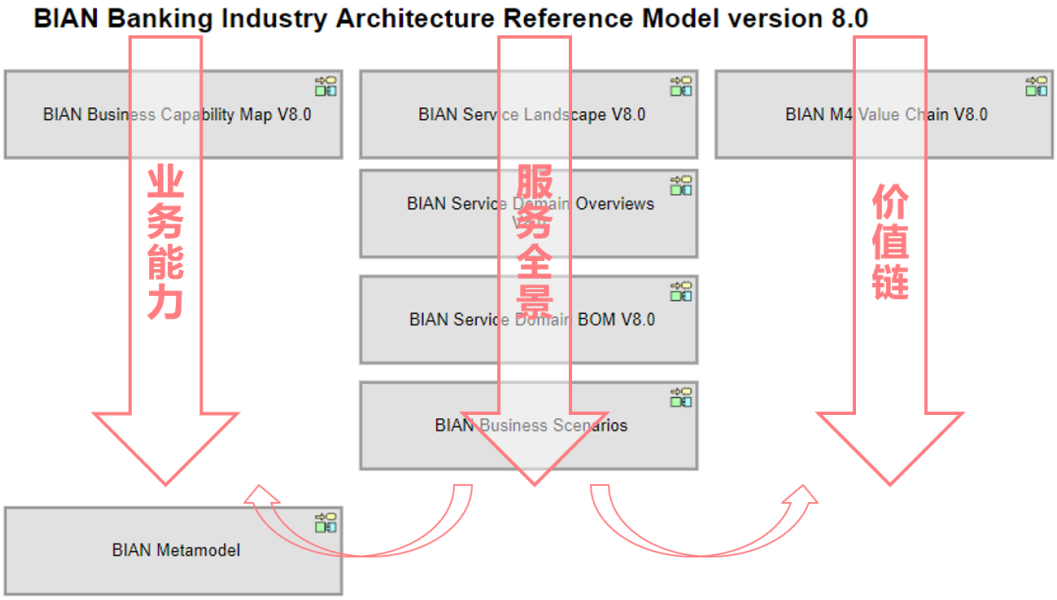
下图示出了上例中的其中一个服务域 FinancialGateway(金融网关)的例子。结合 BIAN 元模型可以很清晰地看到以服务域为中心的组织关系，包括：

- 服务域所归属的上层服务域(如 Payments)
- 服务域所归属的业务领域(如具体渠道 Channel Specific)
- 服务域所包含的服务(Service Operation)、服务组(Service Group)以及业务对象(Business Object)
- 服务域的功能模式 Functional Pattern(如操作 Operate)



4.2 BIAN 的业务能力模型和价值链模型

通过上面的概念以及框架概览可以看到 BIAN 的整个脉络体系。BIAN 从 2015 年 v4.0 演化到 2019 年的 v8.0 也一直在探索其他维度和方向。 业务能力(Business Capability)和价值链(Value Chain)是由服务全景视图延伸出来的新方向。



下面我分别展开介绍一下。首先看看业务能力的概念。

如果仅在服务域层面进行研讨而不向顶层全面企业业务设计层面进行延伸，那么这种服务域概念层面的研究并值得投入。业务规划人员或战略家可以在服务域概念分区之上进行更多层面或角度的探索和发现。这些更多层面或角度构成了业务的“价值视图”。详细刻画了业务交互以及动机(通过适时调用更多的服务域)，并将业务价值创造和更多的通用业务绩效度量与结果关联起来。

可以借助业务价值层用于范围广泛的战略规划及企业投资决策活动。BIAN的业务能力模型(Business Capability Model)意在将BIAN服务域与业务价值分析层联系在一起。下面我们通过概念定义来澄清两者之间的关系和区别。

业务能力(Business Capability) – 业务能力表示业务在做什么，或者业务能做什么，是一组业务功能的封装。在BIAN中业务能力是一个比较新的发展方向，它表示了一组应用于特定对象用以实现特定目标的一组一致的动作集。从业务架构角度看，业务能力是一组嵌套结构，大多数达到3级。

服务域(Service Domain)与业务能力(Business Capability)的区别

这里需要体会和区分一下服务域和业务能力。服务域表示松散的通用业务功能，更准确地说，是执行某些操作(例如维护客户关系的参考信息或运营一个网络)的能力。 对“业务能力(business capability)”的更为正式和完整的定义则描述了整个企业希望在所定义的业务环境中可以做的事情，可以为此定义一些相关的价值或目的。因此，

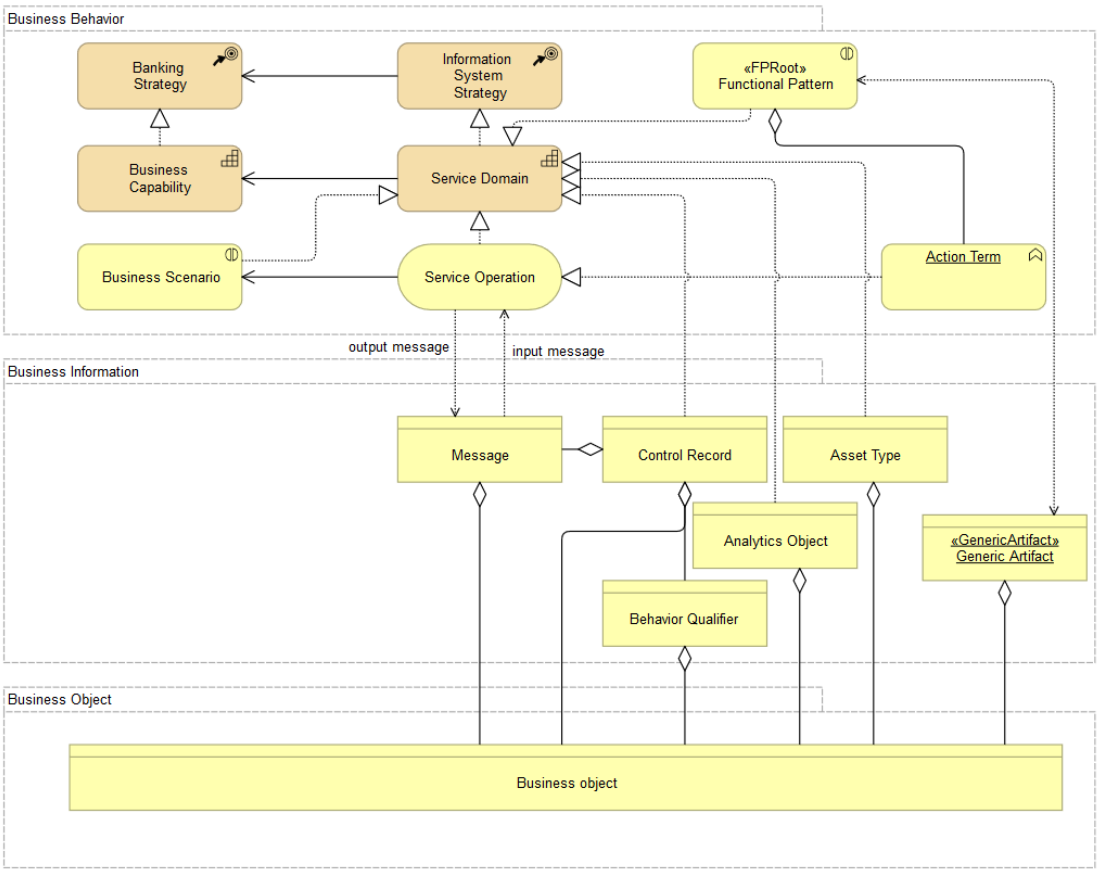
在BIAN中，业务能力定义为结合特定业务环境的执行能力。可以利用服务域执行的业务功能(business function)来支持具有不同业务上下文以及关联的价值和/或目的的不同业务能力[4]。一个业务能力是一或多个服务域的组合，一个服务域可以参与到多个业务能力的建设中。

再举一个例子将有助于阐明这种区别。 BIAN定义了一个服务域，用于跟踪/确定银行对客户的信用评估（客户信用评级）。该服务域可能涉及许多不同的业务能力，下面是两种可能的业务能力：

- 使产品与客户匹配的能力
- 与客户协商产品价格的能力

可以使用BIAN业务场景对上述每个业务能力进行建模。 在这两种情况下，方案都将包括对客户信用评级的引用，但是尽管银行拥有关于客户信用的准确信息，但这一信息对银行的价值或影响在这两种能力之间会有所不同。

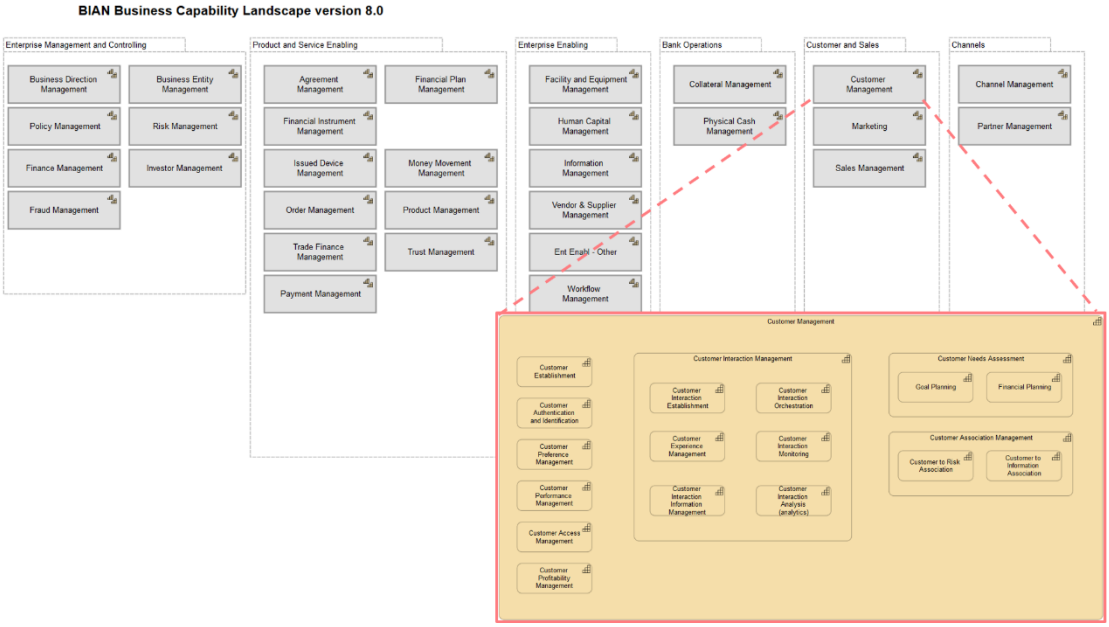
配合下面的BIAN分层元模型(https://bian.org/servicelandscape-8-0/views/view_29386.html) 可以很清晰地看到上面这些概念之间的关系。



在业务行为层面，银行战略(Banking Strategy)由业务能力(Business Capability)来实现，与信息系统战略(Information System Strategy)相关。核心概念服务域

(Service Domain)实现了信息系统战略，同时与业务能力紧密相关，可以是多对多关联。

通过 bian.org 在线模型库访问业务能力，可以看到 BIAN v8.0 能力地图如下图所示。



目前 BIAN 业务能力地图是一个多级嵌套结构，大部分可以到三级。第一级是一个能力分类，包括：

- 企业管理与控制(Enterprise Management and Controlling)，
- 产品与服务支持(Product and Service Enabling)
- 企业支持(Enterprise Enabling)
- 银行运营(Bank Operations)
- 客户与销售(Customer and Sales)
- 渠道(Channels)

每个分类下又包含了能力分层结构，如上图示出的客户管理(Customer Management)能力结构。客户交互管理(Customer Interaction Management)是一个业务能力，指能够记录，跟踪，控制，组织与监控事件的时间顺序，交互点和客户的其他行动，而与渠道或会话无关。

这一能力之下又细分为六个基本业务能力：

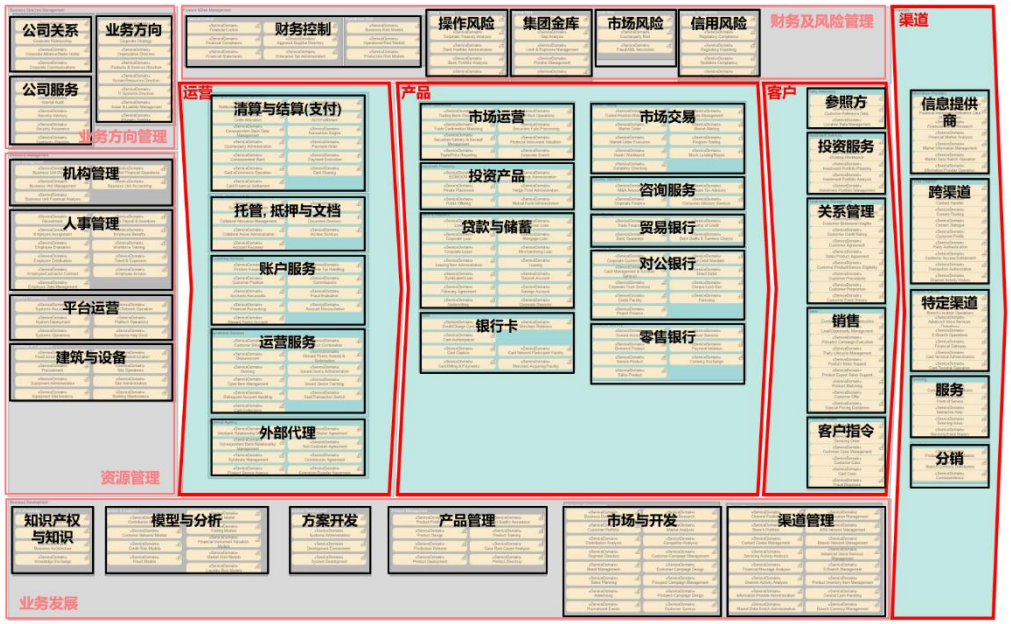
- 客户交互建立(Customer Interaction Establishment)，发现并建立新的客户互动的能力
- 客户交互编排(Customer Interaction Orchestration)，监督，规范和控制客户互动的能力
- 客户体验管理(Customer Experience Management)，在客户与组织交互时对客户行为，感知和满意度的理解能力
- 客户交互监控(Customer Interaction Monitoring)，跟踪并记录客户交互过程的能力

- 客户交互信息管理(Customer Interaction Information Management)，收集，组织，跟踪，报告或以其他方式传播有关客户的基本事实，统计信息，属性和信息的能力
- 客户交互分析(Customer Interaction Analysis)，对客户交互进行审视以确定交互模式以及其他有助于改进的有意义的数据的能力

可见， BIAN 通过业务能力地图提供了一个可以参考的业务能力分层结构，目前还在建设过程中。不同企业也可以根据自身需要来定义自己的能力地图。当然，对某个具体能力还有有待于企业根据自身情况进行具体定义的。例如在客户体验管理中具备实时记录客户手机 App 浏览轨迹并与呼叫中心进行共享以便于客户问题诊断和解决的能力等。对业务能力进行建模的结果是形成了一整套结构化的高阶业务需求为未来系统建设提供输入。

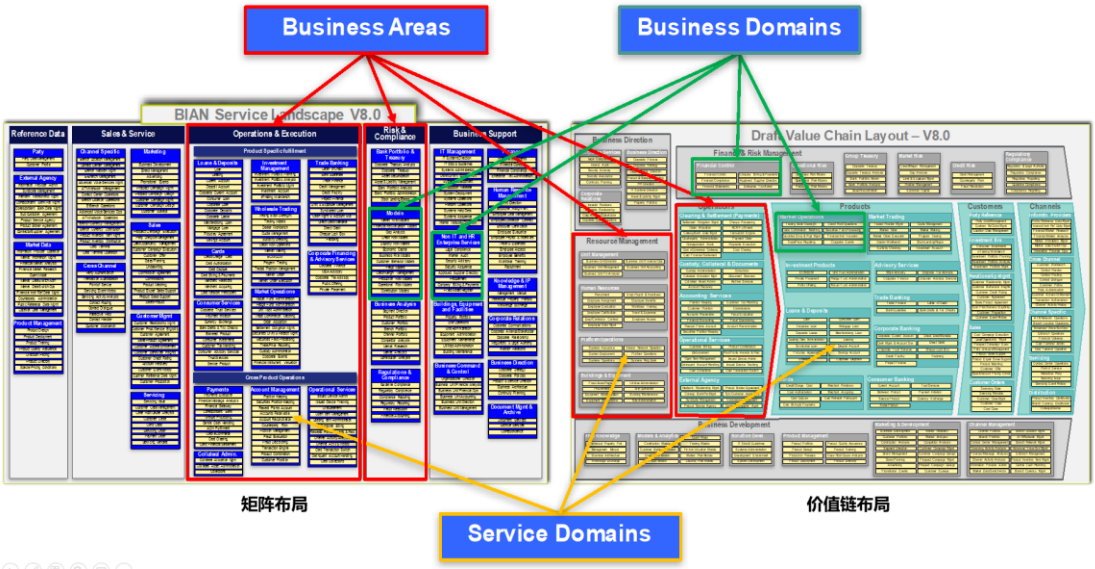
服务域的另一种布局选择——BIAN 价值链视图

BIAN 服务全景视图(Service Landscape)的主要用途是作为一个完整的参考框架来组织服务域集合，以此来发现和开发基本能力积木块(服务域)。因此，只要满足完整性和松散性，可以创建不同的业务区域(Business Area)和业务领域(Business Domain)来重新布局。也可以创建不同的布局来涵盖所识别出的 BIAN 服务域，并突出不同服务域之间的关系。例如可以按照业务条线业务职能来组织，也可以按照业务部门来组织。在 BIAN v8.0 刚刚推出了按价值链来布局的方式。价值链布局可以更为清晰地看出企业的支持活动(业务管理 – 与围绕和启用为客户提供产品和服务的核心“工厂”相关的监督，定义，开发和支持功能)和核心活动(产品与服务交付 – 包含了与交付给客户的产品和服务直接相关的核心“工厂”能力)是如何与企业的价值流动过程联系在一起的。



上图示出了 BIAN v8.0 的服务域价值链布局。在上面服务全景视图中列出的所有服务域按价值链的分类体系重新进行了分类。其中从第一级看核心活动包括：运营，产

品，客户和渠道；支持活动包括：业务方向管理，财务及风险管理，资源管理，业务发展。同业务能力模型一样，价值链模型也在演进过程中，因为价值链模型与生态价值网建设密切相关，大家可以密切关注。下图示出了矩阵式与价值链两种布局下的组织关系对照。尽管业务区域(Business Areas)与业务领域(Business Domains)的分类方法有些许差异，但服务域没有变化。



5 BIAN 的应用

BIAN 的理念和模型可以在不同场景下进行运用：

- **基于 BIAN 提供的标准业务模型视图作为参照从业务到技术进行一致性映射。** BIAN 在业务架构层级进行的定义，起到了承接高阶业务模型/战略和许多实现级架构视图之间桥梁的作用。BIAN 的服务全景视图所提供的语义服务操作可以一致性地映射到行业消息通信标准以及专有消息定义上。BIAN 服务域的功能和角色还可以与常规的实现级别的架构视图(例如流程和数据模型)进行对应。
- **将 BIAN 的设计应用于不同的技术环境。** BIAN 服务域和相关的服务操作定义了业务功能分区及其之间的接口。这一关于业务行为的高阶设计规约可以用于广泛的技术环境。其中三种主要考虑的环境是：作为一个框架更好地组织或调整企业已有的“单体式”系统；作为一种设计用于采用企业服务总线(ESB)技术的服务支撑应用；作为一种“容器”服务分区模式用于高度分布的“云”技术环境，特别是最近出现的微服务架构。
- **使用 BIAN 来定义企业蓝图。** 将 BIAN 服务域作为企业蓝图“积木块”来搭建企业蓝图。服务域的一个关键特性是其业务目的/角色不会随时间而变化。服务域的工作或实现其目的的方式可以随实践和支持解决方案的发展而改变，但其核心业务目的是稳定的。 结果，使用服务域定义的业务蓝图也高度稳定，适合于不同类型的分析。包括：设置和跟踪绩效，绘制和评估业务覆盖范围，

关联行为属性以更好地约定需求。前面描述的业务能力地图，服务全景视图，价值链模型都是蓝图的具体形式。

前面也提及在 bian.org 提供了一组“How-To-Guide”，可以结合起来学习和应用 BIAN 的主要思想和资产。其中：

- *BIAN How-to Guide - Introduction to BIAN* - 是整个文档体系的整体介绍。其核心理念已经在上面的章节中结合在线模型库进行了介绍。BIAN的其他主要内容体现在下面四个文档中。
- *BIAN How-to Guide - Design Principles & Techniques* - 目标读者是技术和业务架构师。侧重BIAN的理论和设计实践，是理解BIAN方法的不错参考。
- *BIAN How-to Guide - Developing Content* - 这个目标读者是 BIAN 的工作组成员。侧重解释当前工作方法和用于记录BIAN标准内容的不同工具和模板。
- *BIAN How-to Guide - Applying the BIAN standard* - 目标是 BIAN 成员及其他希望在不同的技术环境应用 BIAN 的设计内容的金融机构。
- *BIAN How-to Guide - Semantic API* - 概述了如何使用 BIAN 标准和相关内容为应用程序接口（API）开发提供高级设计。可以结合 BIAN API Platform Sandbox (https://portal.bian.org/) - 一个提供了 RESTful 端点定义的开源开发者环境一起参考。目标读者是在 API 生态系统进行开发和部署的业务和技术架构师。因此需要对 BIAN 的设计原则和内容有基本理解。

需要注意的是这组文档会随着 BIAN 大版本演进而不断更新。下面我们以 Applying the BIAN standard 及 Semantic API 为主抽取一下 BIAN 标准的应用要点。

5.1 基于 BIAN 提供的标准业务模型视图作为参照从业务到技术进行一致性映射

从前面的介绍可以看到，BIAN 可以与企业架构规划一起结合进一步从设计层面将业务战略层层贯穿贯彻下来。这体现在从业务到技术多个层次上：

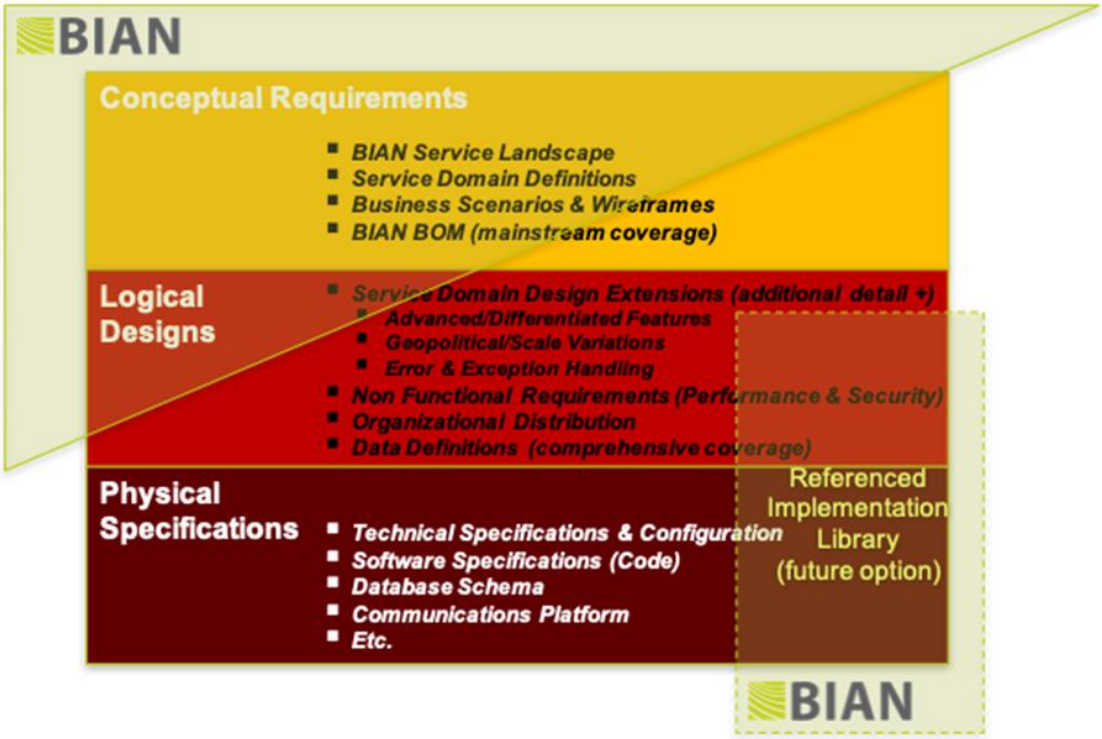
- 业务架构层面。服务域定义了松散的操作分区，实际上构成了业务能力积木块。在 BIAN v9.0(2020 Q3)计划完成所有 308 个服务域的定义，可以结合 BIAN 服务域来映射企业已有的业务架构规划。还可以结合 BIAN 业务能力地图、服务全景视图和价值链构造企业服务蓝图。利用 BIAN 服务操作构造统一的企业服务目录。
- 信息/数据架构层面。BIAN 与一些金融标准(如 ISO20022,FIBO 等)相互参照，维护了一组业务语义词汇。加之 BIAN 基于服务域提供了业务对象模型(BOM)，基于此可以逐步建立起统一的企业级数据模型体系。统一的数据定义也有助于服务域之间，企业内外的服务交互。因此，服务域可以在数据架构层面进行延伸，帮助进行业务信息划分，职责划分，有助于企业级数据治理工作的展开。

例如，信息在服务域/服务操作之间如何暴露，数据格式在不同业务上下文中的控制等。另外，BIAN 通过控制记录和行为限定符类型/行为限定符等对业务信息模型中的“动态”特性进行了建模。控制记录如同服务域中的核心实体生命周期记录仪，全程记录了围绕核心实体的业务价值实现过程。行为限定符则对服务域对应的功能模式落实到服务操作层面进行了细节刻画。

- 应用架构层面。将 BIAN 服务域作为逻辑能力框架与现有银行应用进行对照，帮助理清应用边界以及冲突的部分，有助于应用和服务的更好封装。由于 BIAN 服务域的松散性和唯一性，可以更专注地提升和优化银行具体专业领域能力。在提升服务“外部化”和重用的同时，减少紧耦合架构带来的技术和运营的妥协/折衷，践行 SOA 共享服务中心的理念。在向应用架构翻译时需要结合应用架构原则和目标进行更深层次的考虑，特别要结合数据架构统一考虑，例如功能与数据的映射关系，是逻辑组件独立但数据共享，还是数据分开组件独立运营等。
- 基础设施层面。服务域进而可以映射到更为底层的支持技术基础设施上。技术基础设施层面基本上都提供运行实例隔离或分区机制，如容器、虚拟机等。因此服务域与相关的应用模块只要有相似的运行特征都可以共享同一类型的技术设施。考虑到不同服务域之间的交互时，特别是在分布式服务框架、微服务体系架构逐渐流行时需要在服务域交互层上考虑更多关于流量、路由、服务发现和注册等方面的优化。考虑到互联网金融类应用的交互特点，还要考虑共享数据的访问和优化问题(是否需要缓存)、数据一致性问题(ACID 还是 BASE)等。

但 BIAN 并非要事无巨细，无所不包。BIAN 的一个理念是保持与实现无关并支持规范定义，因此 BIAN 设计必须限制在概念级别。BIAN 提供了概念性组件设计，并提供了一些指南，说明在需要更全面的逻辑设计和物理规范的开发中如何解读这些指南。BIAN 的概念设计意在仅仅提供足够的细节来与开发对接，通过采用标准的应用模块/边界来支持基于组件的开发并改进互操作性。基于组件的应用设计与 SOA 实现方法高度吻合，可以在实际实现过程中映射到不同的技术环境中。

通过下面的范围示意图可以更为清晰地看到这一点。BIAN 更多关注概念需求，包括前面介绍的服务蓝图，服务域定义，业务场景及协作线框图，以及业务对象模型 BOM。关于 BOM 再做一些说明， BIAN 将服务域信息属性与现有的行业概念对象模型(如 ISO20022)进行匹配，以便更为一致地对业务信息进行解读。但由于现有模型的一些差距及失配，BIAN 还是维护了一套自己概念性的业务对象模型(BIAN BOM)。



在逻辑设计层面，本质上解决了概念性需求接下来“如何”实现的问题。在实际设计过程中，随着更多信息的收集和注入，BIAN 在逻辑设计层面可以进行扩展。开发人员可以将这些逻辑设计扩展加入到服务域概念需求以及语义控制记录属性中，这些逻辑设计扩展可以以任何适当的形式出现以满足开发环境和部署技术的具体要求。

BIAN 列出了可以采用的逻辑设计选择的一般性指南，但为了保持实现的独立性，应避免定义太多的细节信息。设计扩展的一般类型包括：

- **差异**- 添加特定于高级或差异化行为支持的细节、考虑规模要求的细节（对于大型企业）、处理地缘政治特定需求的细节。
- **设计选项** - 在可用的可能工作方法之间进行选择，例如支持交互处理与离线处理，或支持不同的交付渠道。
- **组织安排** - 处理特定的地理分布和组成企业的不同业务线
- **非功能需求** - 应用运行目标定义，如性能和安全。

物理规约涵盖了用于服务域功能实现的实际代码和数据模式。BIAN 标准是实现无关的，因此不会涉及服务域的任何物理属性。也就是说，当将 BIAN 用于组件/容器类型部署时，有一些更为具体的操作属性可以帮助确定最适合的软件体系结构和公共设施类型。

可以考虑的软件方法和设施列表包括：

- ***消息队列和事件*** - 服务交换及服务触发，包括适用于所有服务域类型的序列化，安全及弹性恢复功能。
- ***(有限)状态机*** - 适用于控制记录及其子分区的生命周期管控，子分区必要时由行为限定符类型来定义。

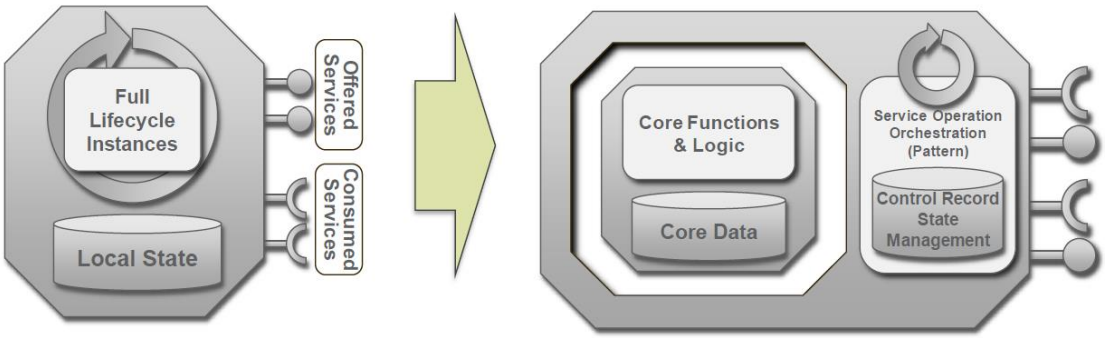
- ***事件驱动处理*** – 与状态机设计一起使用时有利用事件驱动设计的巨大潜力。可以应用于控制记录(及其行为限定符定义的子分区)的具体属性及其相关的规则/策略，或者控制记录状态或状态转移模式。
- ***工作流管理*** – 可以广泛应用于大多数服务域类型。在一些情况下可以适当进行工作流“嵌套”以与行为限定符类型形成的控制记录分解层次结构一致。
- ***规则引擎*** – 与工作流管理常一起使用，规则引擎也有广泛应用的可能。
- ***数据管理设施*** – 特别是在服务域管控自身自治性数据存储库的同时，需要广泛的数据管理设施。高级数据管理功能可能用于具体的高度分布式环境中（用于复制和恢复）。
- ***分析及报告设施*** – 广泛应用的通用分析及报告。
- ***命令及控制*** – 由于每个服务域都可以充当自己的操作单元，因此有可能开发与服务域一致的标准跟踪和报告设施，以帮助实现服务域之间的指挥和控制结构。

当在 SOA 中作为容器实现时，一些设施可以应用在所有服务域分区。一些设施可能更适于某些特定的功能行为模式。例如，“处理/流程(Process)”类型的服务域更适合使用工作流管理。

5.2 将 BIAN 的设计应用于不同的技术环境

BIAN 作为一种逻辑设计提供了关于银行业务服务域以及其下业务服务操作的一个完整且规范的服务目录。这一服务目录将银行、客户、第三方合作方都紧密联系起来，构成了新型生态银行互联互通的基础。同时，这一服务目录在不同业务交互场景不同语境下也赋予了不同的含义，因此这一服务目录不是单纯的技术 API，而是包含了业务含义的语义 API 服务。当面对客户和第三方时银行在这一语义服务层的封装下可以看作是一个“黑箱”，其具体实现技术对外屏蔽。如果进入到银行系统内部，SOA 的理念在与具体技术环境结合时可以有不同的选择。

在描述如何在不同的技术环境中应用 BIAN 之前，有必要对服务域的规范进行细化以便与不同的技术实现对接。从业务层面看，服务域行使着一定的业务角色并通过其提供的服务操作参与到其他业务活动中，或者根据需要从其他服务域订阅服务操作。这种描述是一种基于服务的实现，但是相同的业务功能也可以在不太灵活的传统技术环境中实现。在该技术环境中，连接是点对点的，而不是通过某些基于服务的灵活机制(如服务总线，服务的发现和注册)来实现。

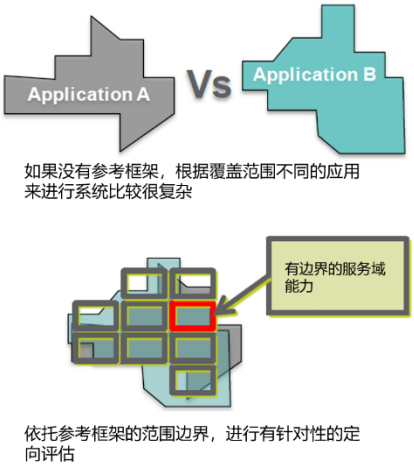


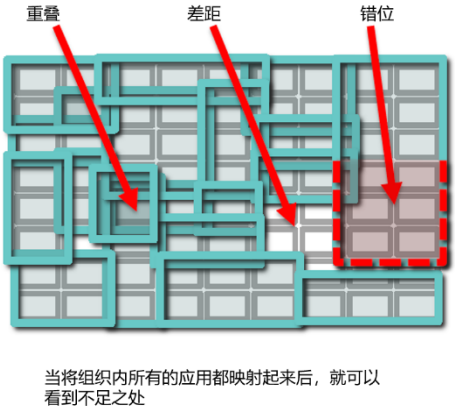
从设计层面看，上图示出了服务域的两个主要组成部分：功能核心(functional core)和“服务化”打包器(“service enabling” wrapper)。打包器主要用来处理与其他服务域的交互，如上图所示。需要区分服务域的这两个方面，因为这两个方面在不同技术环境将以不同的方式进行实现和解释。

从技术环境层面看，银行由于长期积累，存在多种不同技术环境共存的情况，其典型环境包括：1)基于主机(Host-大致代表了 IBM 主机及各类小型机环境)的核心业务系统环境；2)SOA 时代打造的基于消息队列/企业服务总线的应用互连环境；3)新型微服务容器云平台环境。

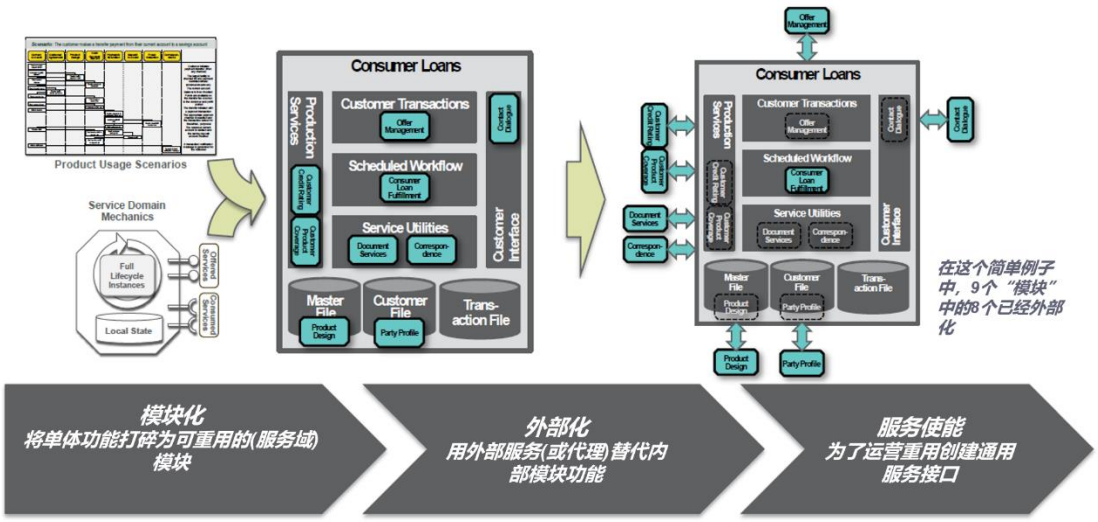
5.2.1 常规核心业务应用的合理化

BIAN 服务域可以用来评估现有应用组合，用服务域划分来识别业务应用之间的业务逻辑及业务信息的碎片及重叠。因为 BIAN 服务域作为一个稳定的框架定义了不重叠的功能分区，这些分区可用于评估现有/核心应用的疆域，从而找到不足之处。大多数现有银行核心业务应用涵盖了多个但不同的服务域集合，因此直接进行应用程序与应用程序进行比较意义不大，因为两个应用通常具有不同的功能范围。 由于在将应用映射到服务域时服务域不会重叠，因此通过服务域和现有应用的一一映射，然后合并所有服务域的评估结果，可以直观看到现有应用的情况。包括下图示意出的重叠、差距/空白以及错位。





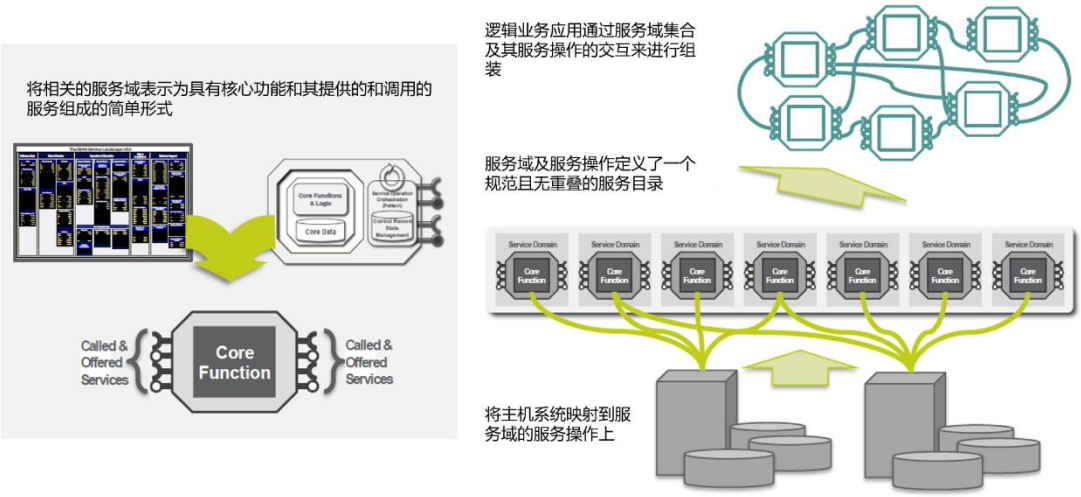
关于功能冗余问题，对于绝大多数核心业务应用来说，可能会发现下图这种模式。 由于银行应用是逐一建立起来的，某些应用逻辑和/或业务信息可能涵盖了不少其他服务域的内容(即使是很少的内容)。通过服务域映射可以形成下图中部 as-is 现状功能分布图，然后逐步过渡到下图右侧的 to-be 理想状态。这样通过服务外部化逐渐剥离耦合的应用，将应用之间的边界清晰化。同时保证应用业务目的更加明确和专注，从而有利于进行业务服务自治。这是技术层面践行松耦合、分布式或微服务架构的关键。



银行应用中有很多领域容易落在多个服务域中，特别是与客户信息、客户关系管理、客户服务相关的部分，以及许多共享的后台功能。利用服务域的松散性，非重叠特性可以更好地指导共享服务功能建设(国内也称之为“中台”)，可以帮助减少同步开销。 在这种环境下 BIAN 更多是用作参考评估框架。

5.2.2 基于企业服务总线的应用互连/主机翻新架构

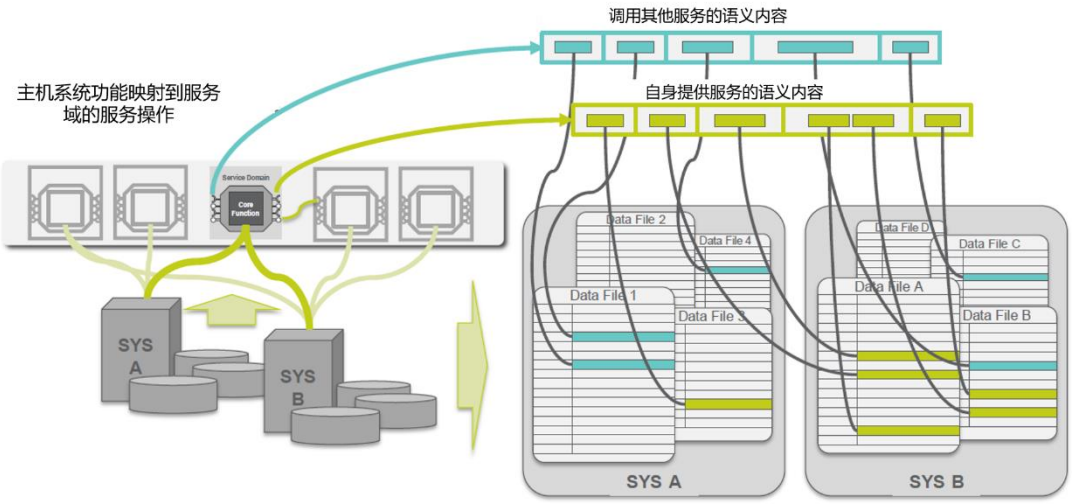
第二种技术环境是对现有核心主机系统进行服务化改造，其目的是支持对功能和数据的共享访问。通过服务化技术(如企业服务总线 ESB)来提供对已有主机系统的访问，同时基于 ESB 上所封装的服务可以进行应用组装，如下图所示。



ESB 环境是 SOA 初级阶段建设的产物。在没有任何组织蓝图或企业架构来指导服务的范围和内容的情况下，企业 SOA 建设容易倾向于定义相对细粒度的服务，以提供对共享实用程序的功能访问。这些细粒度的服务可以通过提供可重复使用的软件实用程序来改善新业务应用的开发，但这种方法有一些挑战：

- 缺乏业务架构和业务模型建设 – 软件实用程序的重用并不总是能推动业务功能的合理化或重用。这也是历史上 SOA 建设的一个难题，不认真对待 SOA 方向可能走偏。
- 复杂且非系统化的服务库 – 由于服务的粒度很细，因此可能导致重叠，庞大而复杂的服务集合，从而难以对其进行分类，维护和引用。目前国内一些银行的服务种类已经上万，里面有不少这样的情况。由于历史原因，不同渠道不同产品的同类服务由不同服务实例来实现。当进入到微服务/服务网格技术体系后这一混沌情况会造成新的复杂性，使得新技术的引入难以或无法预期体现到业务价值上。
- 专有服务 – 由于服务的颗粒度较细，所定义的服务可能包含了特定主机应用的专有功能。这些功能因为与特定系统相关，从而在面对多个不同主机源封装并组装服务时的简便性受到影响。
- 同步 – 如果主机系统只是简单进行了服务化，而没有在蓝图或设计的指导下减少主机系统之间的冗余和冲突，则功能和数据的可追溯性问题仍然存在。

换句话说，技术层面的“解耦”不能替代业务层面的解耦。利用 BIAN 服务域及其服务操作来定义 ESB 的服务目录，由于服务域的高度稳定性可以通过服务域下操作服务的实现来逐步消除应用程序中的冗余，并大大减少应用同步的开销。在有 ESB 或服务目录这种类型的技术环境中，可以用到 BIAN 服务域的功能核心和服务打包器组件(如在 ESB 平台进行服务打包)。同时由于在这种技术环境下已经细化到了服务操作层面，需要参考 BIAN 模型在三个层面进行映射:服务域、服务操作和业务信息。下图示出了这一环境下的 ESB 服务如何与主机系统的数据结构/消息结构进行映射。



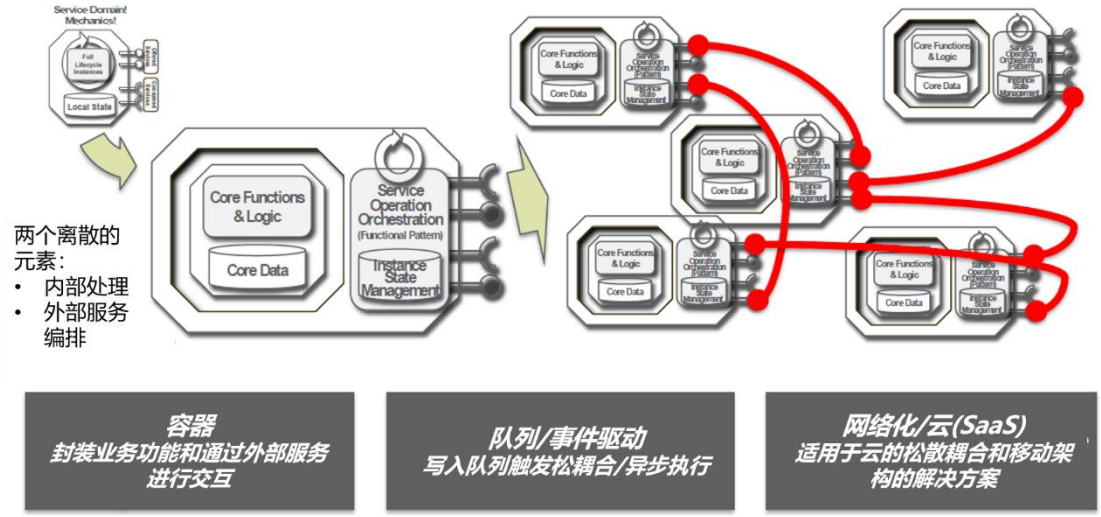
5.2.3 松耦合的分布式/云平台

第三类技术环境是高度灵活，分布式的云平台以及微服务架构。 这种类型的环境可以看作是从 ESB 环境到“纯”面向服务架构的过渡。在 ESB 上所映射呈现出的功能被独立的业务功能“容器”所取代，这些容器可以通过网络作为自治服务中心进行相互协作。 这种“松散耦合”的高度分布式网络服务环境的一些主要操作特征包括：

- 功能独立运行 – 每个服务中心都可以充当独立的功能“容器”，并具有自己的内部处理逻辑，数据存储和状态管理。 它会按照自己的计划/时间运行，调用其他服务并在认为适当时响应外部事件。
- 通过服务调用进行通信 – 服务中心之间的交换/交互全部通过服务操作调用进行。
- 所需的信息精度级别各不相同 – 两个协作的服务中心可能只需要在更近似的语义级别上同意/协调信息元素的含义。这减少了对机器可表示数据采用通用数据格式和结构的要求。
- 交易所使用基于队列和事件的机制 – 网络服务中心异步运行。 当一个人向另一个人请求服务时，它可以继续执行其他任务，并在收到请求时监视响应并采取行动。 操作也应考虑“例外”处理，即如何处理延迟，丢失或错误的响应(和请求)。以及微服务架构中的例外运营模式，如断路器、水密舱等。

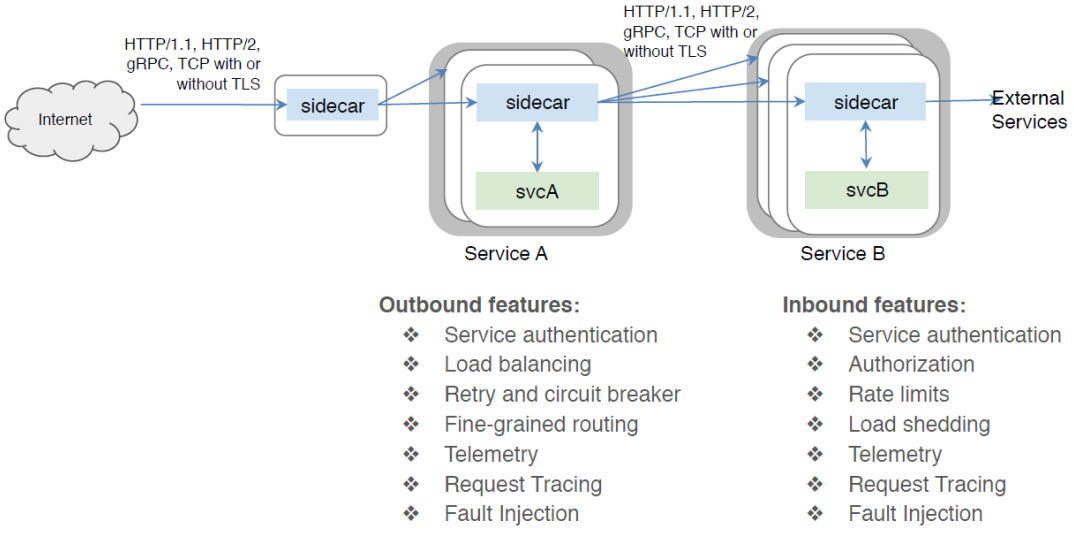
高度分布式环境中的业务执行通常是事件驱动的。 某些事件会触发一个中心的活动，然后根据需要调用其他服务中心的服务。然后，这些“辅助”服务中心也可能又会呼叫其他服务中心，然后才能做出响应。初始触发的处理可能会导致整个网络上许多服务交互的“级联”反应，直到完成所有必要的处理和响应，并且网络达到新的稳定状态为止。这种服务之间这种分布式协同(Choreography)方式与 ESB 总线中心式的编排(Orchestration)方式有很大不同。

应该说，微服务这种服务协同方式更接近“纯”面向服务架构，因此 BIAN 服务域可以与上述服务中心“容器”一一对应。服务域规范的两个组成部分都会用到。功能核心定义了容器的角色，概述了容器所需的内部功能。服务打包器则处理网络中的服务启用、服务/状态管理和目录/连接处理等。服务打包器包含必要的逻辑，以确保服务域了解其可能调用的所有其他服务域的状态。通常会使用某种形式的队列和事件管理来实现自身提供的服务操作和被(其他服务域)调用的服务操作。下图示出了云平台方案经常看到的网状连接形态。

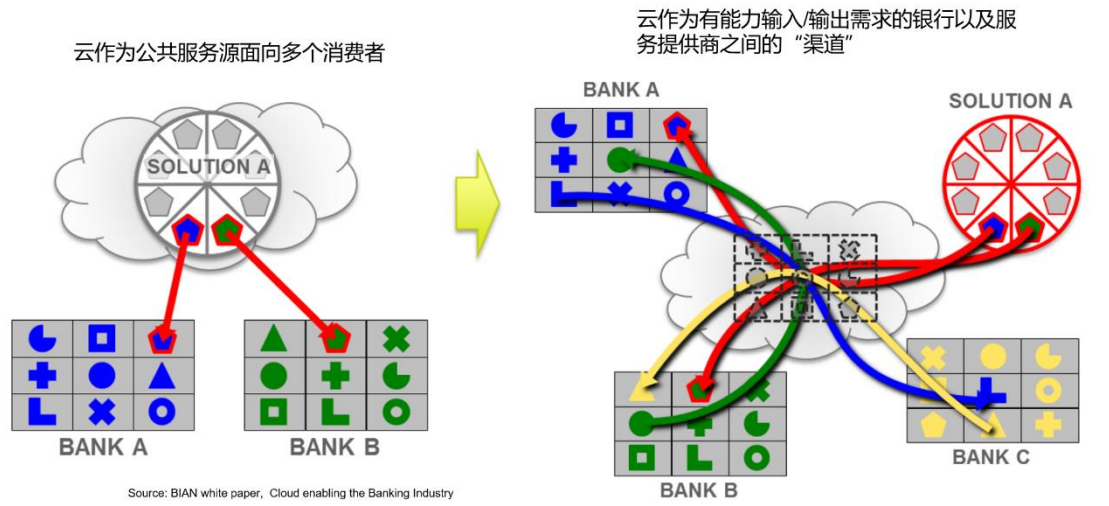


目前在微服务架构领域快速演进的服务网格(Service Mesh)技术如 Istio，和 BIAN 服务域的功能拆分有类似的理念。如下图 Istio 示意图所示，SvcA，SvcB 专注于业务核

心功能，运行在专属容器中。除了核心功能之外的功能，如路由连接，安全，监控遥测，策略控制能均剥离到另外的专属机制 sidecar 上。

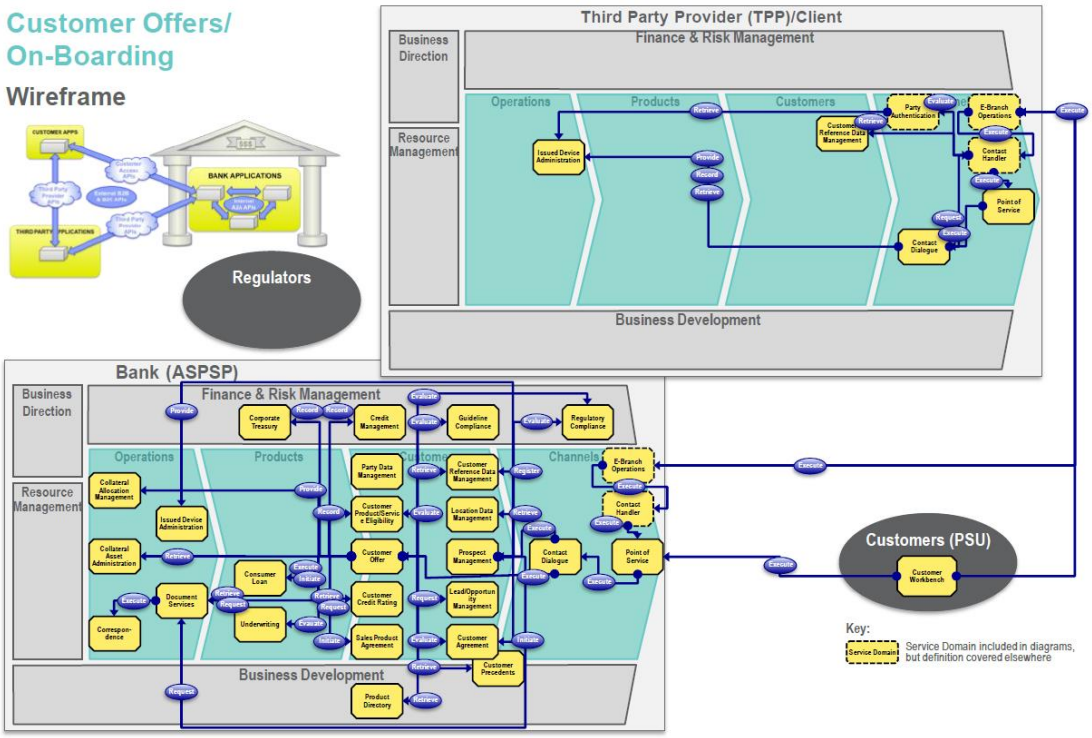


当然，BIAN 与云结合主要还是体现在业务层面，如 SaaS 上。SaaS 的关键在于业务的标准性规范性， BIAN 服务域恰好为银行之间，银行与服务提供商之间在业务层面进行无缝衔接提供了桥梁。配合云平台可以搭建起围绕银行生态的超级平台[7]。



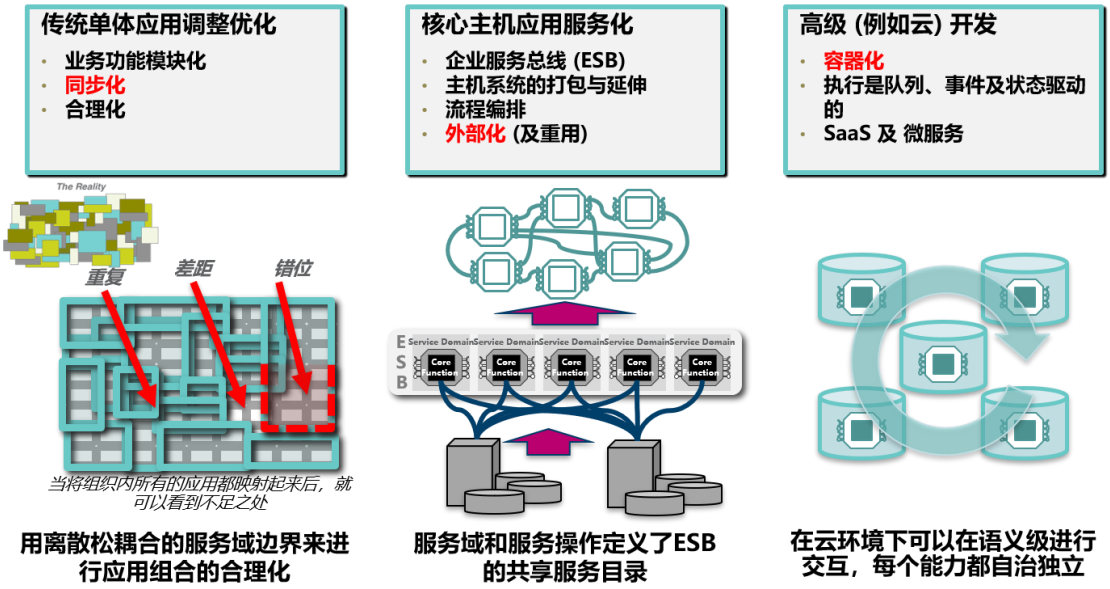
目前不少国内银行已经通过生态云、金融云平台将自身的银行能力向外辐射和输出。同时，积极展开生态合作，通过云平台托管合作伙伴的应用。配合 BIAN 标准可以进一步将多个云平台进一步织成更大的生态网络，基于 BIAN 标准的 SaaS 统一对外(消费者)服务接口，同时和各个银行及服务提供商以标准接口在语义层面对接。进一步延伸这一理念，可以改变国内外银行之间的“服务”对接方式。配合技术层面的“平台”技术如区块链，可以从业务到技术打造更为统一的全球性生态平台。

在 BIAN 目前正在进展的工作中有一项内容是用线框图(Wireframe)来勾勒 BIAN 服务域的协作集群关系，更为准确地反映生态相关各方(B2C,B2B)以及系统内部(A2A)之间的交互。下图是基于欧盟 PSD2 而勾勒出的开放银行生态线框图示例，示出了客户推荐/开户场景下第三方提供商(TPP)/客户，支付服务用户(PSU)，监管机构(Regulators)以及银行(提供账户服务的支付服务提供商 ASPSP)之间通过服务的协作关系。线框图采用了价值链布局方式，从而更为直观地看到价值在各方之间的流动。



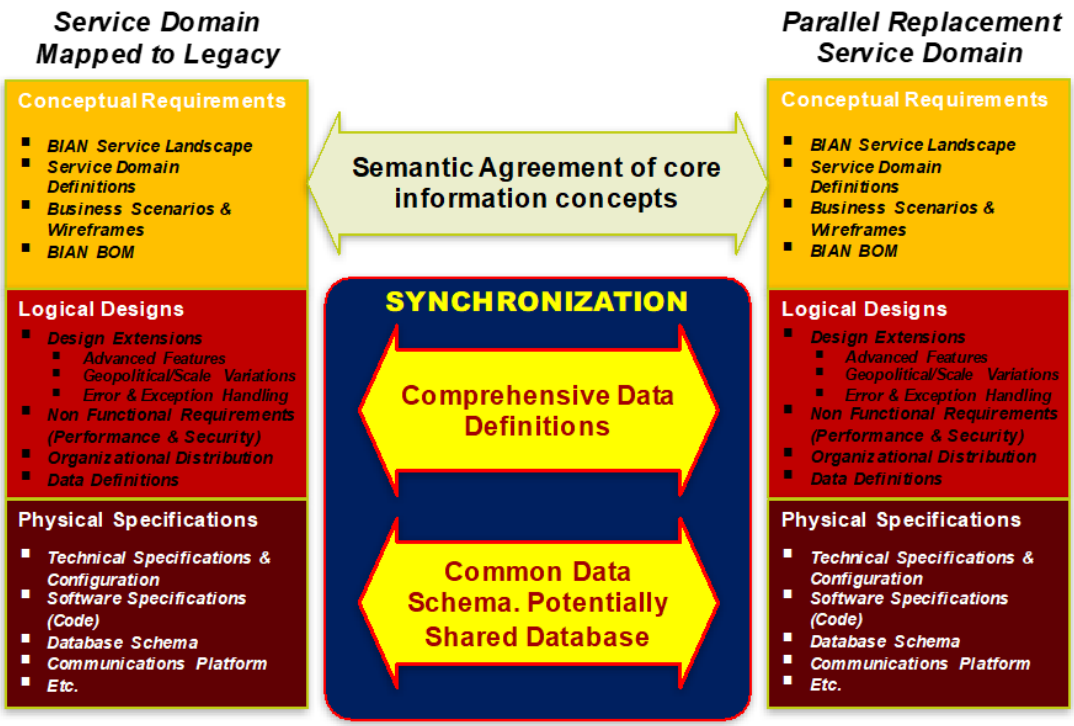
5.2.4 三种环境结合运用

如前所述，大多数银行的应用组合都结合了上面所有三种类型的环境。 BIAN 服务的一个优势是可以在任何这些技术环境中对 BIAN(银行业务)进行一致的解释。在每种情况下服务域功能分区都以相同的方式来进行概念定义和逻辑设计。 因此，模型可以促进在不同环境中开发和运行的业务应用之间的集成。



另外，需要注意的是：尽管分布式微服务架构看上去接近“纯”面向服务架构。但其毕竟是有代价的，坦率讲企业需要从业务价值，总拥有成本全面考虑技术选择。在这种情况下，BIAN 这一统一逻辑模型的指导和治理意义更为重大。

在新旧系统交替过程中，可以用 BIAN 来整理概念需求，作为演进参照框架统一业务需求，业务架构。进一步指导逻辑设计和物理规约设计。例如，在整个演进或转型过程中，随时根据各个层次的具体需求来进行对应和调整。某些情况下可以采用更为灵活的方式。在银行系统现代化改造时，一些应用开始以云的方式进行开发和部署。其逻辑设计过程与传统面向过程/流程的方式有不少区别，但逻辑数据模式仍然可以与传统系统进行映射和参照。在物理数据库层面甚至也可以根据性能、吞吐量、容灾等要求进行共享。在云系统完全达到要求时再考虑逐步进行物理分离。这样在银行系统演进过程中可能出现“双层架构”，并且这一架构可以以服务域为单位进行展开。



5.3 BIAN 语义 API

API 是“服务”概念的进一步延伸。在业务上 Open API 已成为企业向外进行能力辐射的抓手之一，API 已成为一种特殊的银行产品。因此 API 战略也是银行的产品战略之一。目前银行正在加大 Open API 的建设力度，配合 B2B 通信网关以及 SaaS 战略，构成了完整的 B2C(对消费者)、B2D(对开发人员)和 B2B(对合作伙伴)的生态渠道层。在技术层面 API 采用了更加灵活的 Restful 风格，更好适应多语言编程(polyglot programming)及微服务趋势。同时也在 API 的生命周期管理、API 运营(监控和流控)、API 安全、社区、API 开发人员网络、API 沙箱全面加强。

BIAN 的整个设计理念基于 SOA 的面向服务体系架构，因此在支持 API 解决方案方面有着天然的衔接性，这表现在：

- 支持新兴行业方法 – 包括 Open API 的开发和采用微服务架构
- 支持行业标准 – BIAN 服务域和服务操作为银行业务的组件化和服务化提出了行业标准定义。同时 BIAN 与 ISO20022、FIBO(OMG 及 EDM Council 推出的银行业务本体模型)等均有对照呼应
- 支持增量采用/迁移 – 可以分阶段逐步实施和采用 BIAN 解决方案

5.3.1 BIAN 语义 API 是与具体物理实施无关的 REST 风格 API

BIAN 服务域下的服务操作很好诠释了 API(Application Programming Interface)的初衷。服务域清晰划分了应用(Application)的边界，服务操作则针对 Programming Interface。进而，为了更好地支持 API 及微服务架构，BIAN 服务操作定义与 REST 架构风格进行了映射，这样可以帮助开发人员更好更准确地理解和使用 BIAN API。

REST(Representational State Transfer 表征状态转移)是分布式超媒体系统广泛采用的一种架构风格， 专门为 web 服务而开发。其所提出的六个指导性架构约束(Architecture Constraints，包括客户服务器，无状态，缓存，分层系统，应需代码，统一格式接口)同时兼顾了互操作性和应用通信的性能。REST 通过使用一组预定义的无状态操作集合提供了对“资源”的访问。而资源采用 URI(统一资源定位符)进行识别，服务请求的响应结果体现在消息的有效载荷(payload)中。有效载荷可以表示为不同格式，如 JSON，XML，HTML。HTTP 则是最常见的通信协议，HTTP 的动词/方法，即 GET, PUT, POST, PATCH, DELETE 则对应了对资源的 CRUD(创建 Create，读取 Retrieve，更新 Update 及删除 DELETE)操作。

从资源角度 REST 定义了四种资源典型类型：

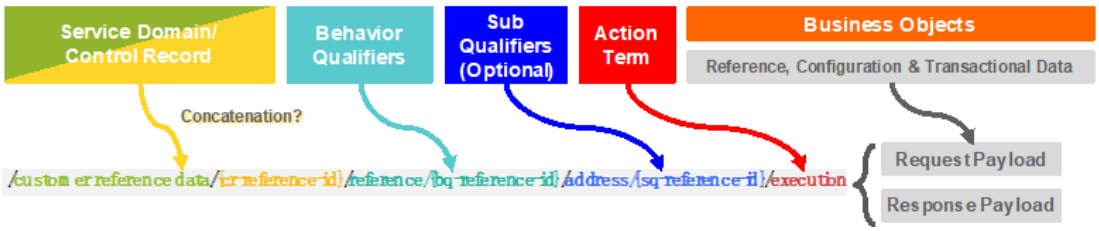
- Documents – 单个资源，如一个对象实例或数据库记录。可以使用传统层次化命名结构来进行引用。例如 http://api.example.com/building-management/office-buildings/{building-Id} 。表征状态通常会将实例的特性值进行组合。
- Collections – 表示了服务器端管理的资源目录或资源集合。collection 决定什么时候来应需创建一个新的资源实例。例如 http://api.example.com/building-management/office-buildings。
- Store – 客户端管理的资源库。store 并不创建新的资源实例但可以来使能一个 collection。例如 http://api.example.com/cart-management/users/{id}/carts 。
- Controller – 专用于处理步骤性的概念。REST API 依赖 controller 来执行与应用具体相关的动作，有输入/输出及参数，这些动作逻辑上不能与 CRUD 标准方法简单映射。例如 http://api.example.com/cart-management/users/{id}/cart/checkout 。

前面提到服务域运作的整个生命周期记录在控制记录中，因此通过 REST API 来进行调用时会与控制记录以及组成控制记录的通用工件进行交互。这里存在服务域功能模式、通用工件以及 RESTful 典型类型之间的映射，如下图所示。

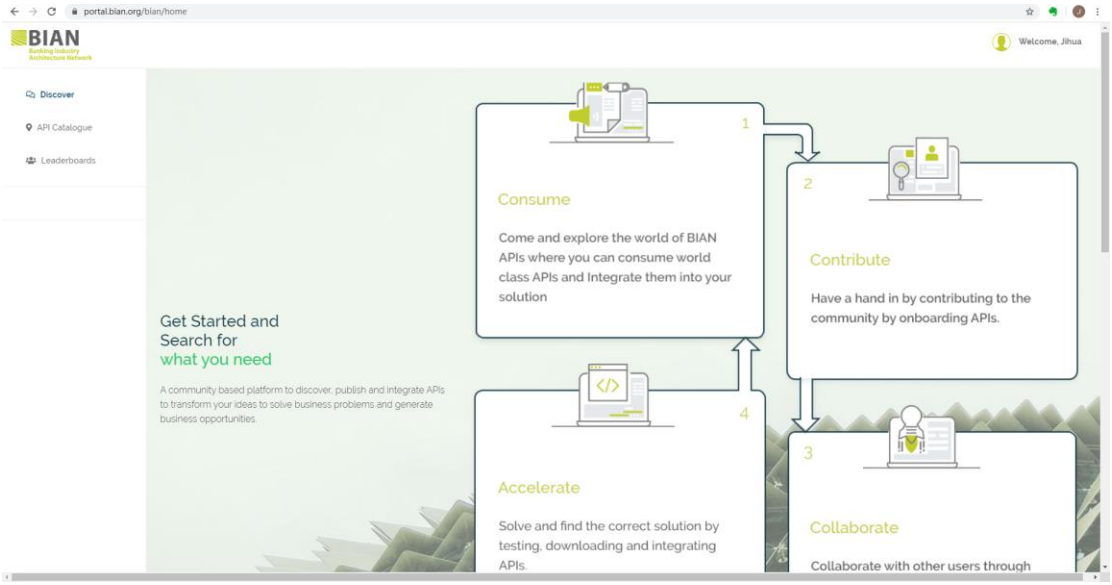
涉及步骤性执行处理逻辑的通用组件一般会对应 Controller，如步骤(Procedure)，操作会话(Operating Session)，协议维护(Maintenance Arrangement)，协议履行(Fulfillment Arrangement)，交易(Transaction)，建议(Advice)，状态监控(State),日志跟踪(Log)。涉及集合或目录的归在 Collections，如目录条目(Directory Entry)，成员注册(Membership)。其他的单一实例则归在 Document。

Functional Pattern	Generic Artifact	Mapped RESTful Archetype	Example URI
DIRECT	Strategy	Document	http://apiexample.com/enterprize-management/responsibilities/{strategy-id}
MANAGE	Management Plan	Document	http://apiexample.com/business-management/responsibilities/{management-plan-id}
ADMINISTER	Administrative Plan	Document	http://apiexample.com/administration-management/responsibilities/{administrative-plan-id}
DESIGN	Specification	Document	http://apiexample.com/model-design/customer-models/{model-id}
DEVELOP	Development Project	Document	http://apiexample.com/application-development/retail-projects/{project-id}
PROCESS	Procedure	Controller	http://apiexample.com/back-office/payments/customer-billing
OPERATE	Operating Session	Controller	http://apiexample.com/production/ATM-network/operation
MAINTAIN	Maintenance Arrangement	Controller	http://apiexample.com/systems/computer/{id}/maintenance
FULFILL	Fulfillment Arrangement	Controller	http://apiexample.com/consumer-products/current-account/{id}/arrangement
TRANSACT	Transaction	Controller	http://apiexample.com/wholesale-products/equity-trade/{id}/transaction
ADVISE	Advice	Controller	http://apiexample.com/wholesale-products/corporate-finance/{id}/advice
MONITOR	State	Controller	http://apiexample.com/customer-relations/{id}/state
TRACK	Log	Controller	http://apiexample.com/customer-relations/{id}/customer-history/log
CATALOG	Directory Entry	Collections	http://apiexample.com/products-and-services/{id}/product-specifications
ENROLL	Membership	Collections	http://apiexample.com/syndicated-loans/syndicate-members
AGREE TERMS	Agreement	Document	http://apiexample.com/customer-relations/{id}/{agreement-id}
ASSESS	Assessment	Document	http://apiexample.com/products-and-services/{id}/{assessment-id}
ANALYSE	Analysis	Document	http://apiexample.com/customer-relations/{id}/profitability-analysis/{analysis-id}
ALLOCATE	Allocation	Document	http://apiexample.com/issued-devices/device-type/{id}/{allocation-id}

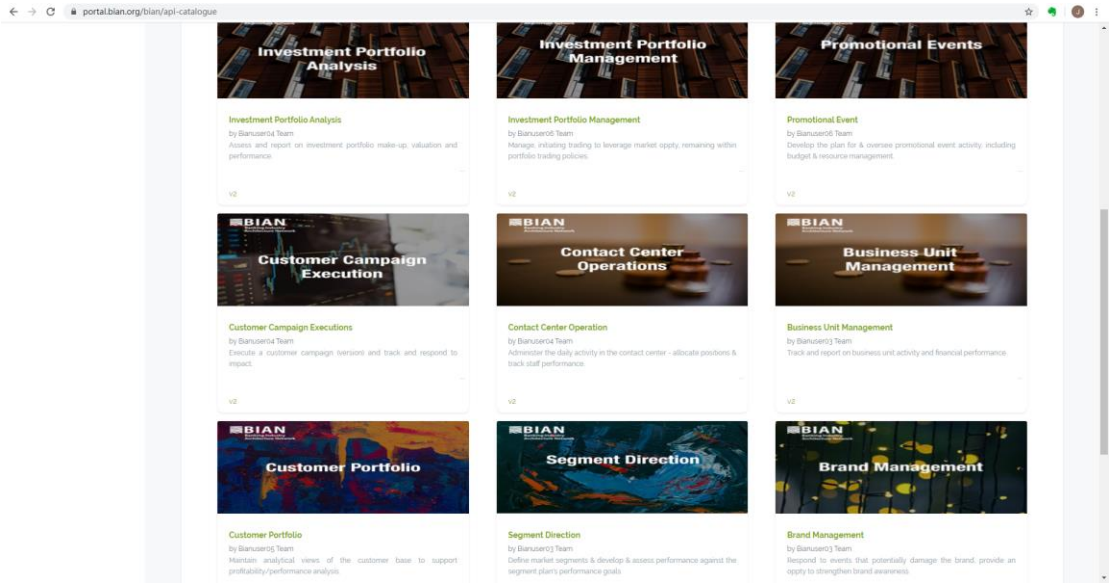
BIAN 的服务操作规约与具体实现无关，在与 REST 映射时 REST API Endpoint 成为最细的一个映射层次。Endpoint 实际上就是定位具体服务操作的 URI，URI 的层次结构也展示了 BIAN 资源定位的层次结构。如下图所示，从控制记录(根)开始，紧接着行为限定符分层结构，然后是动作术语(这里需要注意 REST URI 通常不建议包含动作，因此动作术语会转为名词方式)，最后是业务对象。



读者可以访问 portal.bian.org 搜索 BIAN 语义 API 的具体信息。

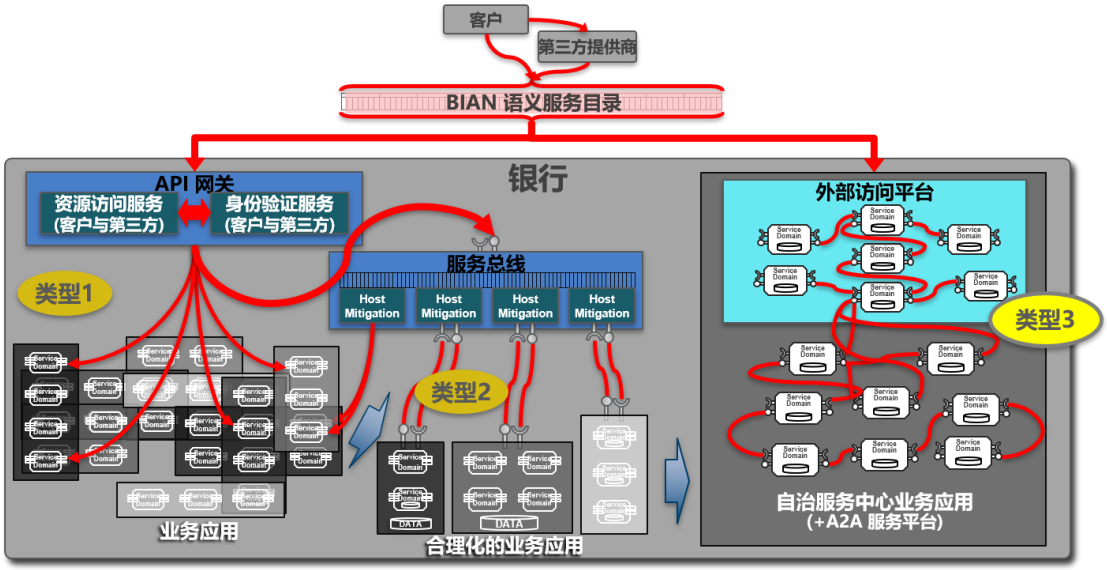


通过 API Catalog 可以浏览 BIAN 的 API 目录，下载相关文档及代码。



5.3.2 API 在不同技术环境下的实现方式

与上面介绍到的三种技术环境相对应，API 在不同技术环境下有不同的实现方式：直连核心系统(Direct to Core)，将后端主机系统打包封装(Wrapped Host)，分布式微服务架构(Distributed Microservice)。这三种方式在 BIAN 所定义的逻辑服务目录下可以共存。如下图所示，银行可以根据这三种方式的特点在不同阶段进行合理选择。

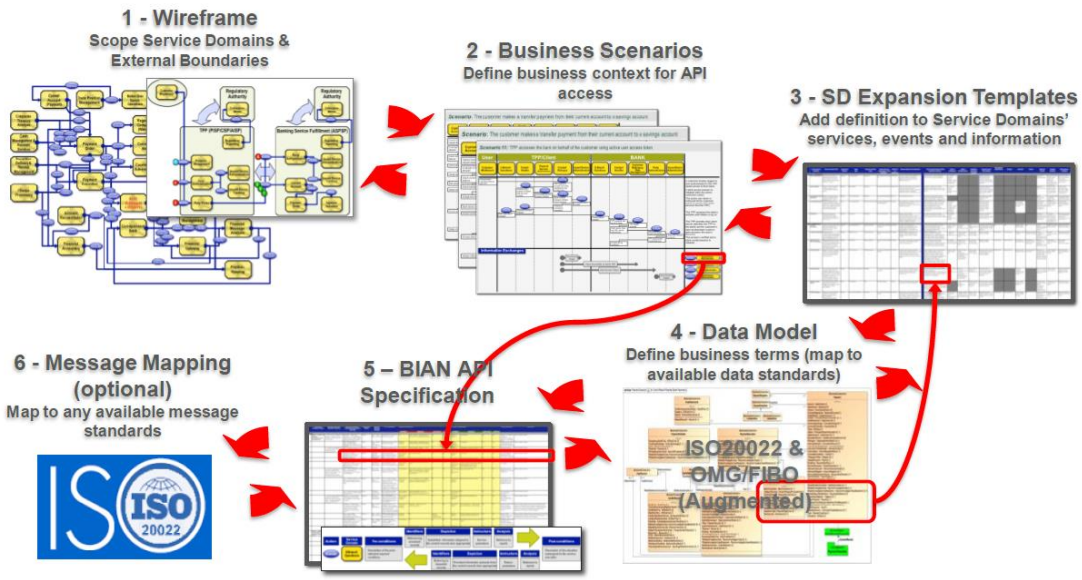


下表列出了三种不同的技术实现方式及其各自的特点。

	类型1.直连核心Direct to Core	类型2. 主机打包Wrapped Host	类型3. 分布式架构
定义	API直接路由到提供服务的核心系统。需要基于中间通道的访问控制和“缓冲”。例如通过IBM Communication Server来连接	通过集成服务中间件–服务总线–“包装”主机系统。 服务总线可以提供各种主机访问功能	主机服务可以实现为松耦合的“微服务”，并具有由复杂的连接中间件(如队列、服务网格)支持的复杂交互
API服务描述	单个主机系统支持的只读或简单的“原子”更新交易。该解决方案可能与具体的主机应用程序相关	符合既定标准的增强型“简单访问”服务。 包装可以增强服务功能，某些主机可能支持更复杂的交换类型	分布式架构结构支持涉及多个业务活动和处理/决策链的灵活而复杂的交互
业务驱动力	提供对已建立/现有客户交换类型的基于应用程序的访问	提供具有高度标准一致性的基于应用程序的访问。屏蔽/增强主机/遗留系统的限制。	· 支持复杂的交互 · 支持新的业务模式 · 支持第三方集成 · 利用先进的技术/架构
优点	· 对核心生产系统的改造工作最小 · 相对独立进行访问控制的加强，同时对现有系统影响有限 · 与业务目的一致，易于未来方案架构迁移	· 核心/遗留系统翻新，减少对现状干扰 · 访问管理，支持主机访问过滤和限流 · 完全支持标准，利用ESB作为通信中介 · 支持前端应用组装	· 支持复杂交易编排 · 全面的访问安全措施，多种安全机制并举 · 支持灵活的业务模式 · 通过外部访问平台支持后端系统的逐步迁移 · 支持高级方案的整合，组件/容器可以采用多种技术实现
限制	· 仅适于简单交易 · 有限/没有访问后交易跟踪 · 仅限于身份验证方式的访问控制	· 有限的复杂交易支持 · 有限的前端应用支持 · 仅限于身份验证方式的访问控制	· 投资较大，试错风险 · 缺乏生产系统证明，业务案例的不可预见性 · 相应标准在不断演进
示例	· 查询账户余额/对账单 · 调出产品/服务目录 · 发起一笔支付	消息遵循业界标准(如ISO20022) · 查询账户余额/对账单 · 调出产品/服务目录 · 发起一笔支付 · 客户开户/提议	· 新客户加入并开始业务 · 客户纠纷/案件解决 · 客户关系开发/追加销售/交叉销售 · 第三方服务整合

5.3.3 BIAN API 开发方法是一个系统过程

API 是 BIAN 的主要演进方向之一，是编制数字化银行生态的关键。看似简单的 API 实际上涉及整个银行内外，业务与 IT 的整体规划和协作。因此 BIAN API 的开发方法是一个系统化过程。



1) 通过线框图 Wireframe(服务域之间的协作集群)划定服务域的范围以及周边边界
2) 借助业务场景 Business Scenarios 定义 API 访问的业务上下文

3) 在服务域(SD)扩展定义模板增加对服务域所包含的服务，事件以及信息的定义
4) 对照数据标准进行业务术语定义，形成统一的数据模型
5) 根据前面的输入形成 BIAN API 规范
6) (可选)将 API 规范映射到通信消息规范

上述这些步骤是迭代反馈，逐渐求精的过程。国内已经展开银行 API 之旅的银行可以参考这一系统性方法，从另外一个维度丰富银行整体架构。

6 总结

从业务模式创新角度，如同 ODBC 打开了开放数据库互连之门一样，BIAN 正在以其开放服务标准推动开放银行生态的建立。不仅对于金融行业内部，对于银行外部生态，包括非金融行业以及开发者社区有着重大意义。从架构设计角度，秉承 SOA 的本质精髓，紧跟当前云转型趋势，BIAN 从设计角度丰富了银行架构设计。使得在松耦合架构体系下进行渐进式转型成为可能。

BIAN 可以与企业架构方法相结合，在数字化转型项目中可以较快搭建起银行未来的服务全景视图和能力视图。通过与现有应用进行映射和对接，发现不足(如重复、差距及错位)。支持银行数字化转型过程中的并行“双层架构(传统主机+云)”模式。在微服务及 API 建设过程中以“服务”贯穿始终，同时在开放银行规划时可以对内保证应用间服务及服务边界的合理性，以及外对力辐射的范围和价值。进而，将 BIAN 的 SOA 设计理念可以扩展到其他行业，从而编织起更为广大的生态价值网络！

期望大家多多关注 bian.org，衷心希望百年银行业能通过 BIAN 焕发生机！

参考文献

1. *IBM Banking Process and Service Models 在线文档, https://www.ibm.com/support/knowledgecenter/SSRA3Z_8.10.1/com.ibm.ima.toc/welcome_version.html*
2. *BIAN How-to Guide – INTRODUCTION TO BIAN V7.0*
3. *BIAN How-to Guide – DESIGN PRINCIPLES TECHNIQUES V7.0*
4. *BIAN How-to Guide – DEVELOPING CONTENT V7.0*
5. *BIAN How-to Guide – APPLYING THE BIAN STANDARD V7.0*
6. *BIAN How-to Guide – SEMANTIC API V7.0*

7. *BIAN white paper, Cloud enabling the Banking Industry*

8. *BIAN Edition 2019, A framework for the financial services industry, Van Haren Publishing*