C:\Users\zenghao\AppData\Local\Temp\ksohtml\wpsDF9.tmp.png

# 本科生毕业设计（论文）参考文献译文本

Proceedings of 2016 IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference(IMCEC 2016)

院 系 电子信息与通信学院

专业班级 电子信息工程

（基于项目信息类专业教育实验班（2+2））201401班

姓 名 曾 豪

学 号 U201414343

指导教师 钟国辉

2018 年 3 月

**一种基于微服务的企业架构参考体系结构模型**

YaleYu1，2，HaydnSilveira2，MaxSundaram2

1.Xi'anPeihuaUniversity，Xi‟an，710125，China

2.InfosysAustralia，Melbourne，Australia

yale\_yu01@infosys.com，haydn\_silveira01@infosys.com，max\_sundaram@infosys.com

**摘要***：*在面向服务、以用户体验为中心和客户需求变更驱动的市场环境中，信息和通信技术正成为现代企业业务的主导核心技术。由于面向业务和基于微服务的应用程序模块的支持，越来越多的公司逐渐地转变成更具备信息虚拟化组织的企业，对于大多数公司和企业及其内部的业务技术员工来说，这些都是前沿的。本文提出了一种基于微服务的参考体系结构模型，给出了微服务的概念和关键的体系结构组件的定义和阐述，以及实现和管理企业微服务的构建模块等。此参考体系结构模型可作为业务和IT专业人员在开发企业it转换体系结构解决方案时的务实指导，使API和微服务相关技术能够正确实现企业级无差错应用。本文还重点介绍了一组关键的体系结构问题，并提供了在企业级的公司中利用api和微服务时相应的建议.

**关键字*：***面向服务的体系结构 微服务 api 企业架构 域驱动设计 域体系结构 API网关 API平台

1. **企业IT的挑战和问题**

企业为了快速应对各种扰乱故障，他们往往需要基于敏捷模式开发IT系统及相应基础架构带来性，但同时稳定性对企业来说也是相当重要的。在组织中使用双模的IT是应对扰乱故障的一步;Gartner将双模定义为"管理两种独立的、连贯的it交付模式的做法，一个侧重于稳定性，另一种是敏捷性"。当今企业面临的挑战是，虽然传统的IT应用程序在功能上是令人满意的，但它们基本上不是以客户为中心的，而且无法满足消费者平台和竞争的快速变化需求。这个一方面是因为它的组织文化，但另一方面则是由它的企业应用程序体系结构所决定的。

创建基于组件的分布式体系结构的挑战是通过应用"面向服务的体系结构"(SOA)来解决的，该模型定义为"将自动化逻辑分解为较小的、不同的工作单元"。这些工作单元，称为"services‟，是建立在一套有明确定义的原则上的，比如标准规范，探索，松散耦合，自治权，可重用性，可组合性，无状态性和抽象性[5]。执行业务功能的服务被归类为业务服务(如地址管理服务)，而执行基础结构功能的服务则被归类为基础结构服务，如事务管理服务，除非在此上下文中明确说明，"service‟一般指的都是业务服务。

SOA服务在功能上是通用的，并且有大量的复用，因为更多的消费者发现了服务;每个地方都对服务有额外的要求，既有功能性的也有非功能性的。尽管开发确保向后兼容性，如果缺乏了演进功能，会意味着需要进行更多的测试以确保与现有的使用者和现有功能的兼容性[2]，随着时间的推移体系结构会演变为粗粒度的"整体"服务的集合。服务的更改和发布管理会变得更困难，因为多个消费者受到影响，另一方面，变化也必须得到良好协调和控制。

企业服务总线(ESB)是在许多组织中实现SOA服务的常用方法[2]，它使用Oracle、IBM等软件供应商的产品。大多数ESB产品都基于应用程序容器体系结构，并提供了一系列基础结构服务，如会话管理、流控制、工作负载分配等。将服务部署到容器时，无论是业务服务还是所有相关的基础结构服务，都包括在内。因此，每个容器都有一个在内存、CPU和授权方面运行时要求更大的成本。为了最大限度地提高效率，开发人员需要将其服务聚合到有限的运行时容器中，从而导致服务之间进一步"耦合"。这可以通过创建更多的容器实例来解决可伸缩性问题，因为它包含未使用的服务实例，因此容量将会不成比例地增加.

应用体系结构的Gartner‟s规律发展周期表明，微服务处于规律发展周期的顶峰[7]，趋势是将这些技术趋势视为一种治疗方法——全部都是为了解决所有遇到的IT问题，随着对概念的加深了解，我们能够形成一个更清晰的认知，关于技术可以做什么，不能做什么。考虑到这一点，观察SOA从2002-2010的过渡规律发展周期，组织担心他们将会变得落后，并将被诱惑投资于使用微服务体系结构来改造IT架构，然而根据历史证据，总是期望高，相应的失望也会很大。

本文提出的基于微服务和API的参考体系结构，旨在解决这些灵活性、粗粒度服务、互操作性和耦合性的挑战。

1. **在企业架构中微服务和API的概念**

Microservice是一个自己的应用程序来执行所需的功能。它独立地进化，可以选择自己的体系结构、技术、平台，可以被独立地管理、部署和扩展[1]，并具有自己的发布生命周期和开发方法。这种方法通过实现"智能端点"和将中间层作为网络资源(其功能是数据传输[6])来处理SOA和ESB的构造以及伴随的挑战。

基于微服务的体系结构被定义为分布式应用程序开发的"软件架构模式"，应用程序由多个更小的"独立"组件组成;这些组件是各自形成自己的小应用体系 [1]。说得具体一点，不妨考虑在线订购电话服务的应用程序。此应用程序执行多个功能，如提供地址验证、产品目录、客户信用检查等。在使用基于微服务的体系结构模式时，为地址验证、客户信用检查和在线订购创建了应用程序，这些应用程序都拼凑在一起，为固定电话服务提供了订购能力。应用开发的这种方法解决了"单体式"应用程序和服务的挑战。

一个微服务应该围绕一个"特定的独立业务情境点"的主题范围，它的关注点应该适用于这个功能的特定情境。可以将其解释为企业系统，如客户关系管理、工资单等，或这些系统中的较小功能(如客户帐户)。微服务要求能被独立部署。每个微服务都有自己的部署体系结构，不会与其他企业软件组件共享其资源，如容器、缓存和数据存储库。比如对于微服务存储和从数据库中检索信息等逻辑等，即数据库架构应该是微服务的一部分。一个微服务的伸缩是独立于其他微服务的，也就是说微服务应该是"松散耦合的"。通过移动其宿主机器或增加实例数量来提高微服务的容量，不应对其使用者产生任何影响[6]。

应用程序往往会公开一些服务接口，这些接口可以被其他应用程序使用，被定义为"应用程序编程接口"(API)[8]。这些接口通常基于其应用编程语言，并使用自定义协议(如远程函数调用(RFC)和用于SAP应用程序的ABAP编程)。microservice应该为通信提供接口，它扩展了这个定义，包括"技术不可知"的特性。微服务的API是使用诸如HTTP这样的internet通信协议构建的，坚持开放标准，如REST和SOAP，并使用XML和JSON等数据交换技术[6]。

微服务虽然独立，但并不在仓库中存在，它们是企业架构的一部分，需要参与企业的业务流程。要做到这一点，它取决于其他企业组件，以及总是有依赖于它的那些企业组件等。微服务提供的接口应该是"可发现的"，接口的使用者必须能够检索、查找到接口，而无需明确了解底层技术的实现或位置。

在企业中，几个团队需要访问以创建或增强诸如客户帐户之类的通用功能。在缺乏对服务的明确理解的情况下，微服务很难被有效地复用。因此必须定义明确的规范，维护在企业范围内的所有微服务规范的"共享存储库"中。其他信息，如状态，所有权等，也可以包括在内。

独立是微服务建筑的一个重要特征。为了启用这一特性，微服务应该在其相关环境中"维护数据"，其他微服务可以读取其数据，但是通过该服务执行写入操作。对于客户帐户而言，微服务是客户帐户的数据管家，其他微服务可以访问客户帐户数据的只读版本。这样，开发团队就可以决定其所控制的数据资产最适合的数据模型、表示形式、持久性等，而不依赖于对企业数据模型的更新，并将其从企业治理和发布中释放出来。。

微服务通常包含三层作为典型的3级应用程序，由接口层、业务逻辑层和数据持久性层组成，但这些往往都在一个更小的特定情境中。这导致微服务可以涵盖范围广泛的技术能力，但是并非每个微服务都提供所有功能，这将根据所提供的功能的使用方式而有所不同。例如，主要由API提供程序使用的微服务将具有通信接口层、业务逻辑和数据持久性层，但不一定具有用户界面。

1. **基于微服务的参考体系结构模型和关键组成模块**

本文提出了一种基于微服务的参考体系结构模型，并对微服务的概念进行了清晰的解释和定义，就如上文所讨论的讨论。此参考体系结构模型由一些关键的体系结构组件和构造模块组成，下面详细介绍了在组织内的企业体系结构环境中实现和管理微服务。

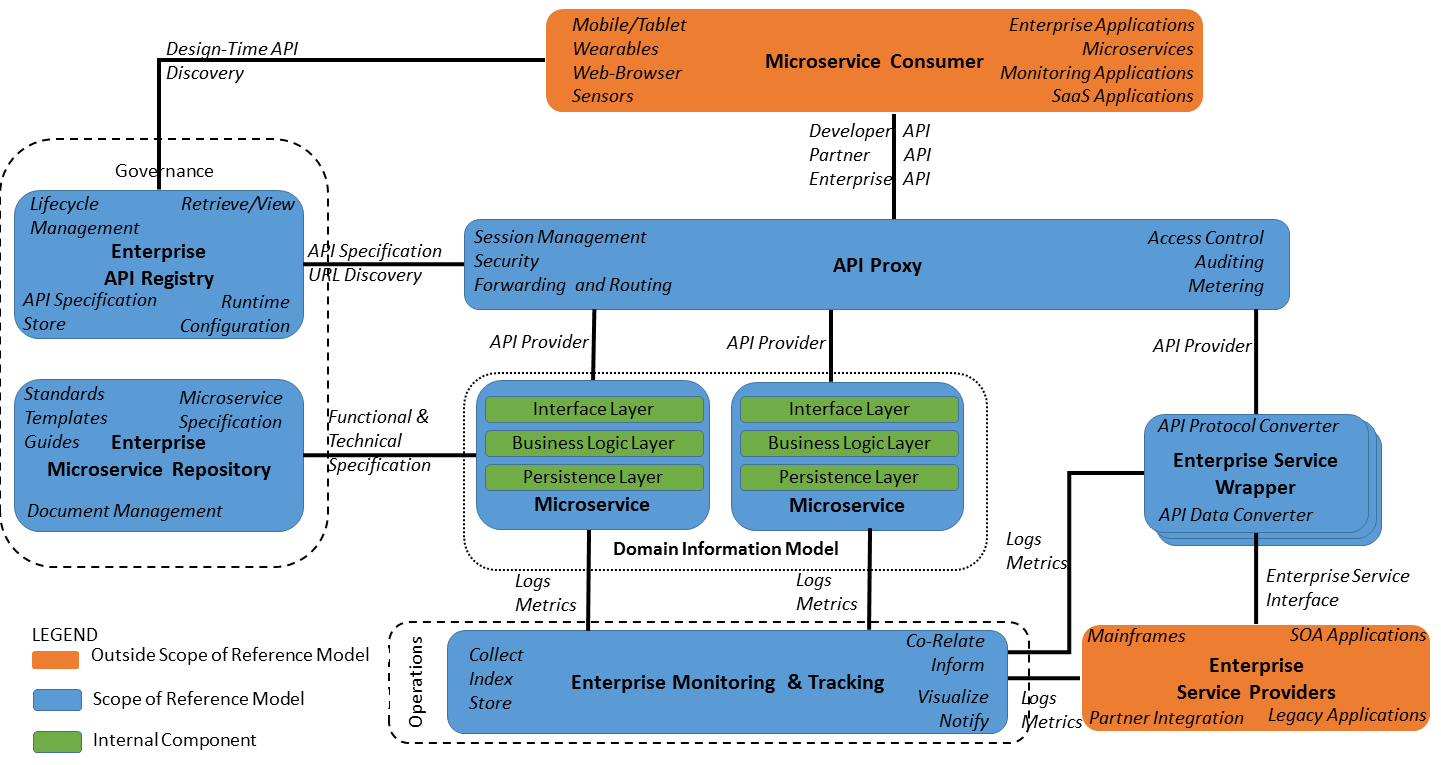


图1.基于微服务的参考体系结构模型

1. **微服务域**

当此基于微服务的体系结构模型应用于组织或公司内部的企业级时，此应用程序分解的结果体系结构将由细粒度模块组成，从而导致较小的应用程序或微服务。例如，问题诊断功能[3]根据各种技术类型(如铜服务诊断、光纤服务测试、电缆服务诊断等)细分为较小的块，微服务的灵活性意味着可以使用不同的工具和技术构建每个微应用，并将其扩展到一个在管理中产生复杂性的企业。

"微服务域"通常是一个业务逻辑实体，包含相关微服务的集合，其目的是帮助管理与微服务拓展相关问题的复杂性，如多个数据模型、平台和技术的拓展、对其他微服务的交叉依赖等。业务功能域将作为业务逻辑容器的界限。它可以用于从业务运营角度分配所有权，例如，与提供帐单功能相关的微服务将由与计费过程和活动相关联的业务单元管理和治理。

"微服务域"也可应用于企业信息管理。对于信息体系结构的微服务方法是以开发人员为中心的，它允许每个微服务都有自己的表示方法和命名约定，如客户一样的通用数据实体。但由于缺乏"通用信息模型"，这可能无法在一个有数百微服务的企业中进行扩展。在这种情况下，它将导致在微服务中添加中介层以进行语义翻译。由于企业信息模型过于庞大，微服务信息模型过于精细，因此应将信息模型应用于中等粒度层次。企业信息体系结构的"微服务域"可以提供此中间级别，在该逻辑域内的所有微服务都将遵循"域信息模型"。这样做的好处是多重的，因为它会增加域内的互操作性，因为所有的微服务都会是使用同样的语言，它还会为这些函数的API提供域的使用者，它还提供该领域所有微服务的发展标准。

**b. 企业API注册表**

通过使用API注册服务，可以满足微服务的"发现"要求。它的目的是使微服务所暴露的接口在企业内部和外部服务的消费者可见。"企业API注册表"是整个企业的共享组件，其位置必须是众所周知的和可访问的。其信息内容是以标准格式发布的，信息应该是一致的和可读性良好的格式，并且必须是访问控制的。它必须具有搜索和检索功能，以便用户能够在设计时查阅可用API规范的详细信息。此外，它还必须提供有关API功能的操作、其使用上下情境和可访问性要求(如安全协议)的信息。它还可以包含api目标URL的运行时配置，从而允许api代理组件动态地将API路由到目标url。它还将包括API生命周期管理的能力;它应该为企业API组合的管理提供一个中心位置。

**c. API代理**

代理或代理模式在面向对象的编程中的起源是GOF设计模式，它被定义为"另一个对象控制访问的占位符"。它提供了一些功能，如为底层资源提供保护，为远程资源、节流等提供已知的位置。要将微服务从其用户中"消除"，此代理模式将通过"API代理"组件应用于微服务接口级别。组织将为不同的消费者提供API，其中一些是在企业之外的其他用户。这些微服务将因服务级别协议(SLA)、安全要求、访问级别等而异。API提供的代理函数可以分为三类。

企业API：与组织的消费者使用组织内的"企业API"，它们的特点是较低的安全要求、更大的访问程度、企业级SLA、不计量和无节流。这些API将提供修改核心业务实体和撤销协作API的权限；随着业务的增长，it与其他企业实体合作，提供it服务或交付产品，但一个领域中的合作伙伴可以成为另一个竞争对手，因此必须适当限制访问权限。随着API技术的使用获得了动力，"协作API"可以用来与企业外部的合作伙伴进行集成，取代传统的B2B集成。"协作API"还将用于服务由企业外部的提供者托管的方案，如在SaaS平台上托管的服务。它们的特点是通过使用安全网络、通过协商和签署的接口协议进行更大的形式化处理、企业级SLA、限制访问API的子集相信受保护的信息，从而提高安全性。

公共API：作为服务提供者，企业可以选择为其开发人员提供一些服务作为API，使他们能够提高服务。例如，电信服务提供商可以提供SMS作为api，一个移动应用程序开发人员，通过SMSAPI将"移动用户验证"功能添加到应用程序中。这些接口被支持为"公共API"，灵活性和简单性是这些接口的关键，它们具有简单的自动登入过程，其特点是使用行业标准安全机制，如"OAuth‟，低访问度，受计量、使用节流和货币化的限制。

**d. 企业微服务资料库**

"企业微服务存储库"将是一个共享存储库，用于存储有关微服务的信息，它提供诸如微服务生命周期状态、版本、业务和开发所有权等信息，详细信息如其设计功能目标，它如何实现功能目标、工具、技术、架构、它提供的服务、任何API、数据保持和查询、任何特定的非功能性要求。在缺乏明确定义的存储库标准的情况下，企业必须为微服务定义自己的标准规格文物。像微服务本身这样的表示应该是技术不可知的。此外，此存储库还将包含用于微服务设计和规范的支持标准、指南和模板。

**e. 企业服务包装**

大多数大型企业已经投资于像SOA这样的体系结构，并且总是有很大一部分的核心业务逻辑和遗留软件的数据。替换这些的转换程序将等效微服务可能不会在短期内出现。"企业服务包装"组件允许通过API技术访问这些资源。它是一个访问器组件，它将挂钩到企业资源(如ESB)和使用消息队列的旧式应用程序，或者SOAP或其他技术来公开服务并使它们可用作API。它还必须能够在API数据格式之间执行转换。在创建此组件的实例时，可以应用像函数分解这样的微服务，以便与企业的其余部分对齐。此组件是通过微服务快速解锁核心业务能力的关键。

**f.企业监测和跟踪管理**

在任何企业中，操作智能是一个重要的决策工具，任何分析工具都高度依赖于数据。通过部署基于微服务的体系结构导致的碎片化意味着基于微服务的不同实现，有多个不同的信息源，每个都有一个相当小的信息子集。企业监视和跟踪管理器要求所有事件源(微服务和其他应用程序)发布它将收集和存储的标准化事件(操作和日志事件、性能参数等)。这些事件源将包括微服务和其他企业应用程序。它将作为所有事件的中心集合，可在其中执行各种分析操作，此存储必须存储当前和历史数据，并可为业务和操作智能功能。操作团队可以在单个微服务中或整个交互生命周期中查询与单个事务相关的所有事件的存储。基本事件合作关系可用于实时业务活动监视。必须支持报告和仪表板以进行可视化。事件监视程序可以监视需要操作的事件类型的数据流。

1. **主要关注事项和问题以及建议**

从商业所属人的角度来看，微服务的所有权是模糊的。在一个不同业务线支付开发和支持能力的企业中，可以想象，每一个都在微服务周围建立自己的体系结构。从企业架构的角度来看，这可能并不可取，因为这会导致IT支出的所有领域的重复。可以创建一个治理结构或功能点来管理企业微服务在整个组织中的设计和实现，而一条业务线可以完全控制微服务，其中业务功能显著与微服务存储库中存在的内容不同。

微服务和API技术是由互联网上可用的软件和基础设施的出现所驱动的。HTTP是网络通信体系结构的基本通信协议之一。微服务和APIs‟是围绕用于通信的HTTP协议构建的，它解决了许多互操作性问题。但是，当业务功能需要事务管理、原子性、可靠的消息传递、仅有一次的传递和广播事件驱动时，它并不像企业通信系统所期望的那样提供保证的服务质量。我们需要在这一领域进行更进一步的研究开发工作，并有可能根据企业级别要求调整一些WS。

基于微服务基础架构的要点是"技术的选择"，在构建微服务时，这对开发人员有很大的好处，但不是从企业业务的角度来看，因为它们没有绑定到特定的技术栈或版本的软件中。考虑到100个微服务应用的几个版本都用它自己独特的工具集、跟踪和管理许可证来构建的场景

所有这些工具集的一致在企业层面是不可能的。必须根据需要标识和更新"有限的工具集"，并且必须使用在企业级别提供的工具集开发所有微服务。本文提出的基于微服务的参考体系结构模型可以作为企业和IT专业人士在整个组织中的微服务和api体系中评估和选择正确的技术和平台的关键参考。

1. **结论**

本文提出了一种基于微服务的参考体系结构模型，明确理解和定义了微服务的概念和关键的体系结构组件，以及实现和管理企业微服务的构建块。企业架构的上下文。

我们认为，这种参考体系结构模型可以作为业务和IT专业人员在开发企业it转换体系结构解决方案时的务实指导，以便API和微服务相关技术能够在企业级适当使用，不产生混淆。本文还重点介绍了一组关键的体系结构问题，并在企业级的公司中利用api和微服务时提供了相应的建议。

**参考文献**

[1] Dmitry Namiot, Manfred Sneps-Sneppe. “关于微服务架构”，国际开放信息技术杂志ISSN: 2307-8162 vol. 2, no. 9, 2014, p24-27

[2] Mahmood,Zaigham. “面向服务的体系结构的前景和局限性”，国际计算机学报，第3期，第1卷, 2007, p76-77

[3] “应用程序框架（TAM）” https://www.tmforum.org/Browsable\_HTML\_Frameworx\_R15.0/main/

[4] Chief Information Officer Council(2001) “A Practical Guide to Federal Enterprise Architecture”. Version 1.0

[5] Erl, Thomas. “面向服务的体系结构：概念，技术和设计”，Prentice Hall / PearsonPTR, p32-37

[6] Newman, Sam. “微服务构建”, O‟Reilly Media, p410-420

[7] Gartner, “应用程序体系结构的发展规律周期”, 2002, 2010, 2015.

[8] API架构: 构建API的蓝图

**参考文献原文**

A Microservice Based Reference Architecture Model in the Context of Enterprise Architecture

Yale Yu1,2 , Haydn Silveira2 , Max Sundaram2

1. Xi'an Peihua University, Xi‟an, 710125, China
2. Infosys Australia, Melbourne, Australia

yale\_yu01@infosys.com, haydn\_silveira01@infosys.com, max\_sundaram@infosys.com

***Abstract*—In the service oriented, customer experience centric and customer changing demand driven market environment, ICT is becoming the leading enabler and partner of the modern enterprise business. More and more companies are transformed into more or pure digital style and virtual organized enterprises that are enabled and supported by a group of business oriented and microservice based applications and modules that are new to most of the companies and enterprises, both business and technical solution staff. This paper presents a microservice based reference architecture model with clear understanding and definition of the concept of microservice and key architectural components and building blocks for implementing and managing enterprise microservices in the context of enterprise architecture. This reference architecture model can be used as the pragmatic guidance for both business and IT professionals when they develop the enterprise IT transformation architecture solutions so that the API and microservice relevant technologies could be used properly at enterprise level without confusion. This paper also highlights a set of key architectural issues and provides corresponding recommendations when APIs and microsevices are leveraged within a company at enterprise level.**

***Keywords—service oriented architecture; microservice; API; enterprise architecture; domain driven design; domain architecture; API gateway; API platform***

***Challenges and Issues with Enterprise IT***

The business organization needs to be able to respond to disruptions quickly, they need to bring agility to their IT systems and infrastructure, at the same time business also craves stability. The use of BiModal IT in the organization is one-step in the response to disruption; Gartner defines BiModal IT as “the practice of managing two separate, coherent modes of IT delivery, one focused on stability and the other on agility”. The challenge facing organizations today is that while the traditional IT applications are satisfactory in function, they are essentially not customer centric, and are unable to keep up the fast changing demands of the consumer platforms and the competition. This is in part due to its organization culture, but another part is its enterprise application architecture.

The challenge to create a component based distributed architecture is addressed thru the application of “Service Oriented Architecture” (SOA) which is defined “a model in

which automation logic is decomposed into smaller, distinct units of work”. These units of work, known as „services‟, are built around a set of well-defined principles, such as standard contract, discovery, loose coupling, autonomy, reusability, composability, statelessness and abstraction [5]. Services that perform business functions are classified as business services

e.g. address management service, while those that perform infrastructure functions are classified as infrastructure services

e.g. transaction management service, unless explicitly stated in this context „service‟ refers to a business service.

SOA services are generic in function and have a great deal of reuse, as more consumers discover the service; each places additional demands on the service, both functional and non- functional. As additional features are added, the absence of an approach to evolve features while minimizing the need to ensure backward compatibility, means more testing is required to ensure compatibility with existing consumers and existing features [2], over time architecture evolves into a collection of coarse-grained “monolithic” services. Change and release management for these services is difficult as multiple consumers are impacted and changes must coordinated and controlled.

Enterprise service bus (ESB) is a common way to implement SOA services in many organisations [2], using a product from a software vendor like Oracle, IBM, etc. Majority of these ESB products are based on an application container architecture and provide a set of infrastructure services such session management, flow control, workload distribution, etc. Deploying a service to a container, includes both business service and all the associated infrastructure services irrespective of the requirement. Consequently, each container has a comes at the cost of greater runtime requirements in terms of memory and CPU and licensing. To maximize efficiencies, developers then aggregate their services into a limited number of runtime containers, leading to further “coupling” between services. Scalability can be addressed by creating more instances of the containers, which would provide a disproportionate increase in capacity, because of its containing instances of services that are unused.

Gartner‟s hype cycle for application architecture shows that microservices are at the peak of the hype cycle [7], the tendency is to treat these technology trends as a cure-all for all IT problems, as maturity around the concept grows and there is a clearer understanding of what the technology can and cannot do. To consider this observe the transition of SOA thru the hype cycle from 2002-2010, organizations fear that they will be left behind and will be tempted to invest in transforming the IT landscape using microservices architecture, invariably the expectation is high and based on historical evidence, the corresponding disappointment will be large too.

Microservices and API based reference architecture presented in this paper seeks to address these challenges of agility, coarse-grained services, inter-operability and coupling.

***Concept of Microservice and API in context Enterprise Architecture***

A Microservice is an application on its own to perform the functions required. It evolves independently and can choose its own architecture, technology, platform, and can be managed, deployed and scaled independently [1] with its own release lifecycle and development methodology. This approach takes away the construct of the SOA and ESB and the accompanying challenges, by making “smart endpoints” and treating the intermediate layers as network resources whose function is that of data transfer [6].

A Microservice based architecture is defined as a “software architecture pattern” for development of distributed applications, where the application is comprised of a number of smaller “independent” components; these components are small application in-themselves [1]. To put this in context, consider an online ordering application for fixed line telephone service. This application performs multiple functions such as providing address validation, product catalogue, customer credit check, etc. When using the microservice based architecture pattern, applications are created for address validation, customer credit check and online ordering, these application are cobbled together to provide the ordering capability for the fixed line telephone service. This approach to application development addresses the challenges of “monolithic” application and services.

A microservice should be built around a subjective scope of “business-bounded context” and its concerns will apply to that functional context. This can be interpreted be mapped to enterprise systems such as customer relationship management, payroll, etc or smaller functions within these systems such as customer account. A microservice should be “idenpendtly deployable”. Each microservice has its own deployment architecture and would not share its resources like containers, caches, datastores with other enterprise software components. For e.g. where the microservice stores and retrieves information from a database, the database schema should be part of the microservice. The scaling of one microservice is independent of other microservices, that is to say that microservices should be “loosely coupled”. Increasing the capacity of a microservice, by moving its hosted target or by increasing the number of instances, should not have any impact on its consumers [6].

Applications expose interfaces that can be used by other applications to interact with are defined as “application programming interfaces” (API) [8]. These interfaces are often based on their native programming languages and use custom protocols such as the remote function call (RFC) and ABAP programming for SAP Applications. A microservice should provide interfaces for communications, it extends this definition to include the characteristic of being “technology- agnostic”. Microservice API‟s are built using internet communication protocols like HTTP, adhere to open standards like REST and SOAP and use data exchange technologies like XML and JSON [6].

Microservices though independent, do not exist in silos, they are part of the enterprise landscape and need to participate in the enterprise business process. To do this it depends on other enterprise components and invariably have other enterprise components that depend on it. Interfaces provided by microservices should be “discoverable”, consumer of the interfaces must be able to look up, find the interfaces without having explicitly knowledge of the underlying technology implementation or location.

In an enterprise, several teams would require access to create or enhance common functions like customer account. In the absence of a clear understanding of the service, it is difficult for microservices to be reused effectively. Specifications must be defined, maintained in an enterprise wide “shared repository” of all the microservices specifications. Additional information like status, ownership, can also be included.

Independence is a key characteristic of microservices architecture. To enable this characteristic a microservice should have “stewardship of the data” within its context, other microservice would allowed to read its data, however writes would be performed thru that service. For e.g. a customer account microservice would be the data steward for customer accounts, other microservices can access read-only versions of customer account data. This allows the development team to take decision on what is the best suited data models, representation, persistence, etc. for the data assets they control, without depending on an updates to the enterprise data models, and freeing them from the overheads of enterprise governance and release.

A microservice normally comprises of three layers as a typical 3-tiered application, consisting of an interface layer, a business logic layer and a data persistence layer, but within a much smaller bounded context. This sets a broad scope of the technical capabilities that a microservice could possess, however not every microservice provide all capabilities, this would vary based on how the function provided is meant to be consumed. For example, a microservice used primarily by providers of API‟s would have communications interface layer, business logic and data persistence layers but not necessarily have user interfaces.

***Microservice based Reference Architecture Model and Key Building Blocks***

In this paper, we present a microservice based reference architecture model as following figure with clear understanding and definition of the concept of microservice as

discussed above. This reference architecture model consists of some key architectural components and building blocks, described below in detail, for implementing and managing microservices in the context of enterprise architecture within an organization.

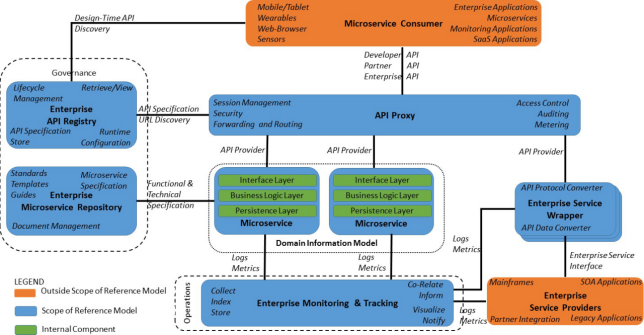


Fig. 1. Microservice based Reference Architecture Mode

1. *Microservice Domain*

by the business unit associated with billing processes and activities.

of fine-grained building blocks, creating a proliferation of smaller applications or microservices. For example, the problem diagnostics function [3], is broken down into smaller blocks based on the various technology types, such as copper service diagnosis, fibre service testing, cable service diagnosis, etc., the flexibility afforded by a microservice means that each microservice can be built using different tools and technologies, scaled to an enterprise that generates complexity in management.

A “microservice domain” normally is a business logical entity, that contains a collection of related microservices, its purpose is to aid with managing the complexity associated with governing the problems related to proliferation of microservices, such as multiple data models, platform and technology proliferation, cross-dependencies to other microservices, etc. The business functional domain would serve as the boundary of the business logical container. It could be used to assign ownership from a business operations perspective, for example, microservices associated with providing billing functions would be managed and governed

A “microservice domain” also can be applied to enterprise information management. In the case of the microservices approach to information architecture is developer-centric, it allows each microservice to have its own representation and naming convention of common data entities like customer. But this may not be able to scale at an enterprise with several hundred microservices due to lack of a “common information model”. In this case it will lead to the addition of mediation layers in microservices for semantic translations. As the enterprise information model is too large and microservice information model is too fine, an information model should be applied at an intermediate level of granularity. The “microservice domain” of enterprise information architecture could provides this intermediate level where all microservices within that logical domain would adhere to a “domain information model”. The benefits of this would be multi-fold, for one it would increase interoperability within the domain as all the microservices would speak the same language, it would also provide consumers of the domains API‟s a common language for those functions, it also provides standards for development of all microservices in that domain.

1. *Enterprise API Registry*

The “discovery” requirements of the microservices are met thru the use of the API registry service. Its purpose is to make the interfaces exposed by the microservice visible to consumers of the services both within and outside the enterprise. An “Enterprise API registry” is a shared component across the enterprise,

whose location must be well known and accessible. Its information content is published in a standard format, information should be consistent and human readable format, and must be access controlled. It must have search and retrieval capabilities to allow users to look up details on available API specifications at design time. Additionally it must also provide information on the operations provided by the API, its usage context and accessibility requirements such as security protocols. It can also contain a runtime configuration of API destination URL, thereby allowing the API proxy component to dynamically route API‟s to target URLs. It would also include the ability for API lifecycle management; it should provide a central location for governance of the enterprise API portfolio.

1. *API Proxy*

The proxy or surrogate pattern has its origins in object- oriented programming as a GOF design pattern that is defined as “a placeholder for another object to control access”. It provides features such as providing protection to the underlying resources, providing a known location for remote resources, throttling, etc. To “de-couple” the microservice from its consumers, this proxy pattern is applied at the microservice interface level, thru the “API proxy” component. Organizations will provide API‟s to a different consumers, some of whom are within and others outside the enterprise. These microservices would differ by service level agreements (SLA), security requirements, access levels, etc. The API‟s provided the proxy function can be grouped into three categories.

Enterprise API‟s: Consumers with the organization use the “enterprise API‟s” within the organization, these would be characterised by lower security requirements, greater degree of access, enterprise level SLA, unmetered and no throttling. These API‟s would provide access to modify of core business entities and retr Partner API‟s: As a business grows, it partners with other businesses entities to provide it services or deliver products, however a partner in one area can be a competitor in another, and access must be restricted appropriately. As the use of API‟s technology gains momentum, “partner API‟s” can be used to integrate with partners outside the enterprise, replacing traditional B2B integrations. “partner API‟s” would also be used in the scenarios where the services are hosted by providers outside the enterprise, such as services hosted on SaaS platforms. These will be characterised by greater security thru the use of secure networks, greater deal of formality thru negotiated and signed interface agreements, enterprise level SLA‟s, restricted access to a subset of API‟s.ieve protected information.

Public API‟s: As a provider of a service, a business could choose to provide its some of services to its developers as API‟s, enabling them to enhance their services. For example, a telecommunications service provider can offer a SMS as a

API, a mobile application developer to add “mobile user verification” capability to their application thru the SMS API. These interfaces are supported as “public API‟s”, agility and simplicity are the key to these interfaces, they have simple automated on-boarding processes and are characterised by the use of industry standard security mechanisms like „OAuth‟, low degree of access, are subject to metering, usage throttling and monetization.

1. *Enterprise Microservice Repository*

The “enterprise microservice repository” would be a shared repository for storing information about microservices, it provides information such as microservice lifecycle status, versions, business and development ownership, detailed information like its purpose, how it achieves the purpose, tools, technologies, architecture, the service it provides, any API‟s it consumes, data persisted and queried, any specific non- functional requirements. In the absence of well-defined repository standards, the enterprise must define its own standard specification artefacts for microservices. This representation like the microservice itself should be technology agnostic. In addition, this repository would also contain supporting standards, guides and templates to be used microservice design and specifications.

1. *Enterprise Service Wrapper*

Most large enterprises have already invested in architectures like SOA and invariably have a significant part of the core business logic and data on legacy software. A transformation program to replace these will equivalent microservices may not be forth coming in the short term. An “enterpise service wrapper” component enables the access to these resources thru API technologies. It is an accessor component that will hook into the enterprise resources like the ESB and the legacy applications using Message Queuing, or SOAP or other technologies to expose services and make them available as API‟s. It must also be able to perform transformation between to and from API data formats. A microservice like functional decomposition can be applied when creating instances of this component, to align with the rest of the enterprise. This component is key in quickly unlocking the core business capability for composition thru microservices.

1. *Enterprise Monitoring and Tracking Manager*

In any enterprise, operational intelligence is an important decision-making tool and any analytics tool is highly dependent on data. The fragmentation caused thru the deployment of microservices based architecture means there are multiple disparate sources of information based on different implementations of microservices, each with a rather small subset of information. The enterprise monitoring and tracking manager requires that all emitters (microservices and other applications) publish standardized events (operational and log events, metrics) which it would gather and store. These emitters will include microservices and other enterprise applications. It would serve as a central collection of all events, on which various analytics operations can be performed, this store must store both current and historical data and can serve

both business and operational intelligence functions. Operation teams can query the store for all events pertaining to a single transactions within a single microservices or across the entire lifecycle of the interaction. Basic events co-relation can be used for real-time business activity monitoring. reporting and dashboards must be supported for visualization. Event monitors could monitor the data streams for event types that require actions.

***Key concerns and issues and Recommendations***

From a business owner‟s perspective, the lines around ownership of microservices are blurred. In an enterprise where different lines of business pay for development and support of capabilities, it is conceivable that each create their own architectures built around microservices. This may not be desirable from enterprise architecture perspective, as this leads to duplication of across all areas of IT expenditure. A governance structure or function can be created to govern the design and implementation of enterprise microservices across an organization, and a line of business can have complete control over the microservices where business features are significantly different to what exists in the microservice repository.

Microservices and API technologies are driven by the emergence of the software and infrastructure available on the Internet (Cloud). HTTP is one the foundational communication protocol of the internet communications architecture. Microservices and APIs‟ are built around the HTTP protocol for communication, which address many of the interoperability concerns. However it does not provide the guaranteed quality of service as expected of an enterprise communication system when a business function requires transaction management, atomicity, reliable delivery of messages, once-only delivery and broadcast event-driven actions. Further and more development efforts needs to be done around this area, with the potential to adapt some WS-\* for those enterprise level requirements.

A basic tenant of microservice based architecture is “choice of technology” when building microservices, this has great benefits from developers but not from enterprise business perspective, as they are not tied into a specific stack or version of software. Considering the scenario where a few version of 100 microservices are built each with its own unique set of toolset, tracking and managing license

agreements of all these toolsets would be impossible at an enterprise level. A “limited toolset” must be identified and updated as required and all microservices must be developed using the provided toolset at an enterprise level. The microservice based reference architecture model presented above could be used as the key reference to help both business and IT professionals to assess and select the right technology and platforms for the implementation and management of the enterprise microservices and APIs across the entire organization.

***Conclusions***

This paper presents a microservice based reference architecture model with clear understanding and definition of the concept of microservice and key architectural components and building blocks for implementing and managing enterprise microservices in the context of enterprise architecture.

We believe that this reference architecture model can be used as the pragmatic guidance for both business and IT professionals when they develop the enterprise IT transformation architecture solutions so that the API and microservice relevant technologies could be used properly at enterprise level without confusion. This paper also highlights a set of key architectural issues and provides corresponding recommendations when APIs and microsevices are leveraged within a company at enterprise level.

***References***

1. Dmitry Namiot, Manfred Sneps-Sneppe. “On Microservices Architecture”, International Journal of Open Information Technologies ISSN: 2307-8162 vol. 2, no. 9, 2014, p24-27
2. Mahmood,Zaigham. “The Promise and Limitations of Service Oriented Architecture”, INTERNATIONAL JOURNAL OF COMPUTERS Issue 3, Volume 1, 2007, p76-77
3. “Application Framework (TAM)” https://[www.tmforum.org/Browsable\_HTML\_Frameworx\_R15.0/main/](http://www.tmforum.org/Browsable_HTML_Frameworx_R15.0/main/)
4. Chief Information Officer Council(2001) “A Practical Guide to Federal Enterprise Architecture”. Version 1.0
5. Erl, Thomas. “Service-Oriented Architecture: Concepts, Technology & Design”, Prentice Hall/PearsonPTR, p32-37
6. Newman, Sam. “Building Microservices”, O‟Reilly Media, p410-420
7. Gartner, “Hype Cycle for Application Architecture”, 2002, 2010, 2015.API Architecture: The Big Picture for Building APIs

**译文要求**

1. 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。
2. 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页  
期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

1. 译文不少于5000汉字（或2万印刷符）。
2. 翻译内容用五号宋体字编辑，采用A4号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。
3. 译文及其相应参考文献一起装订，顺序依次为封面、译文、文献。
4. 翻译应在第七学期完成。

**译文评阅**

|  |
| --- |
| 导师评语  应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。 |
| 评分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_（百分制） 指导教师（签名）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  年 月 日 |