将 SOA 定义为一种体系结构风格

使您的业务模型与技术保持一致

Boris Lublinsky, 企业架构师, Freelance Consultant

2007 年 5 月 09 日

了解如何将 SOA 定义为一种体系结构风格，以促进将与业务保持一致的企业服务作为设计和构建解决方案的基本单元。了解为什么使服务与业务模型保持一致非常重要，并探索一种可用于实现该体系结构风格的模式语言。

引言

存在大量关于 SOA 的讨论，但是很少就这个流行的三字母缩写词的实际含义达成一致。面对很多竞争的定义，很难揭示其真正的本质。SearchWebServices.com 宣布了一个关于其最佳定义的竞赛，并收到了大量的建议。很难选择单个“最佳”定义，因为 SOA 对不同的人具有不同的含义。（请参见参考资料 [1] 和 [2]。）

所处的角度：	**SOA 是**
业务主管和业务分析人员	一组服务，它们构成 IT 资产（功能），可用于构建解决方案并向客户和合作伙伴公开这些解决方案。
企业架构师	一组体系结构原理和模式，用于处理解决方案的总体特征：模块性、封装、松散耦合、关注事项分离、重用、可组合性，等等。
项目经理	一种支持大规模并行开发的开发方法。
测试人员和质量保证工程师	一种模块化从而简化总体系统测试的方法。
软件开发人员	一种包括诸如 Web 服务等标准、工具和技术的编程模型。

尽管所有这些观点都绝对是正确的，但是理解 SOA 的关键是 *体系结构 (Architecture)* 的字母 *A*。难题在于，还不存在一个得到普遍认可的软件体系结构定义；软件工程协会 (Software Engineering Institute) 维护的列表有 50 多种不同的软件体系结构定义。

对于本文，我使用 IEEE 标准 1471-2000 (IEEE Recommended Practice for Architectural Description of Software-Intensive Systems) 提供的定义（请参见参考资料 [3]）：

“体系结构是一个系统的基本组织形式，体现为其组件、组件之间的相互关系和与环境的关系以及指导其设计和发展的原理。”

特定体系结构的组件集和组件之间的关系被定义为一种体系结构风格（请参见参考资料 [4]）：“组件和连接器类型的词汇表以及关于如何组合它们的约束集。”

Convergent architecture: Building model-driven J2EE systems with UML 中提供的更全面体系结构风格定义认为“体系结构风格是通过共同的原理和属性相关的一系列体系结构”。IT 体系结构风格既是 IT 体系结构的一种整体方法，又是一种特定方法。其整体性在于，它涵盖整个软件生命周期，包括项目和工具设计方面。其特定性在于，它合并和集成了许多在传统方法中作为单独实体来处理的结构化、过程化和描述性方面。“体系结构风格提供一个有用的合理替代选择集——并非所有替代选择——并协调它们以便良好地协同工作。”（请参见参考资料 [8]）。

SOA 可定义为这样一种体系结构风格，它促进了将与业务保持一致的企业服务作为设计、构建和组合企业业务解决方案的基本单元的思想。多种模式、定义设计、实现和 SOA 部署完善了此风格。

为什么选择 SOA？

业务和技术负责人等都对 SOA 感兴趣，并将其视为用于实现以下目的的万能解决方案：

- 实现业务与 IT 的更好一致性
- 创建更灵活和反应更灵敏的 IT 基础设施
- 简化集成实现

人们坚信，SOA 能够以双赢的方式，允许您将业务领域与信息技术 (IT) 领域保持一致。SOA 就像是在两者之间创建共生和协作关系的桥梁，这种关系比以往所遇到过的任何关系都更为强大和有价值。而且，SOA 还可通过业务和 IT 实现更好的一致性来获得理想的业务成果。（请参见参考资料 [5]）。

为了理解当前的 SOA 推动力，让我们首先更密切地查看一下当今的典型企业 IT 体系结构及其缺点。

以应用程序为中心的体系结构

当今的企业 IT 体系结构通常被视为应用程序的集合。软件系统的设计、开发、增强和维护都以应用程序为中心。此方法导致在企业体系结构中创建隔离的竖井，从而导致代价很高和不灵活的 IT 系统。每个应用程序都基于单一目的（例如贷款发放、索赔管理，等等），具有自己的数据存储区并针对一组单一用户。结果，它仅实现了企业功能的子集，仅使用和产生企业数据的子集，通常不关心企业中的其他处理。这些竖井将自身表示为*数据岛 (island of data)* 和*自动化岛 (island of automation)*。

数据岛

每个数据岛分别具有自己的企业对象含义或定义（请参见参考资料 [6]）。例如，在一个应用程序中，“价格”可能定义净价，而在另一个应用程序中，同一术语可能还包括营业税。即使诸如“地址”之类的对象在两个应用程序中具有相同的含义，其中一个应用程序也可能将其定义为一组地址行，而另一个应用程序则将其视为街道地址、城市、州、邮政编码和国家/地区。这两种情况都在应用程序之间造成了语义不一致。

它们分别都有与另一个岛重叠的内容。例如，处理健康和牙科索赔管理的应用程序还存储被保险人的人口统计信息。与此同时，客户关系管理（Customer Relationship Management，CRM）应用程序同时包含被保险人的地址和人口统计信息。这种重复导致了完整性问题。

没有哪个应用程序能够提供企业数据的完整视图。例如，抵押管理应用程序未包含关于借款人从其他部门贷款的信息。创建企业数据的统一视图要求集成来自多个来源的信息。

自动化岛

每个自动化岛集中于企业中有限的一组活动（请参见参考资料 [6]）。例如，健康索赔管理应用程序仅负责健康索赔的处理，而不考虑这些活动在总体企业业务流程中的作用和地位。这要求用户“切换应用程序 (application hop)”以执行他们的工作，从而影响他们的工作效率。

不同岛中包含的业务流程之间存在重复。例如，由于合并和收购，保险公司可能有多个索赔处理系统。这要求在多个应用在程序之间同步变更，从而确保流程和支持这些流程的业务规则的一致性。

数据岛和自动化岛的影响在各个应用程序级别是不可见的。然而，它们在企业级别导致重大问题，最明显的是：

- *信息保真度*：数据岛之间的业务数据冗余导致企业数据的不准确表示形式——甚至是在执行定期同步的时候。这些表示形式本身通常是矛盾的，很难协调。由于各个应用程序独立发展，因此该问题的复杂性与日俱增。
- *业务流程碎片*：各个应用程序提供企业功能的有限部分。实现业务流程需要链接包含部分流程实现的应用程序。

解决这些问题已使得企业应用程序集成（Enterprise Application Integration，EAI）和企业信息集成

（Enterprise Information Integration，EII）成为许多企业项目的焦点。这些活动目前（并将继续如此）消耗了企业 IT 预算的很大一部分。

这些集成计划代价昂贵、复杂和劳动力密集的一个主要原因在于，它们尝试将从未打算协同工作的不同应用程序中的数据和处理集合在一起。花在这个任务上的大部分时间都用于同时对现有应用程序中的数据和流程进行彼此参照以实现合理化。

从应用程序到流程和服务

Grady Booch 在 2004 年接受 *InfoWorld* 杂志采访时表示，“诸如良好的抽象、良好的关注事项分离等工程基本因素从来不会从风格中消失……存在再次提高抽象级别的真正机会”。（请参见参考资料 [7]）。

SOA 引入了两种高级抽象：企业业务服务和业务流程。企业业务服务表示现有的 IT 功能（与企业的业务功能保持一致）。对业务服务进行编排的业务流程定义业务的总体功能。

SOA 的宗旨是消除当前存在的应用程序（具有刚才描述的所有缺点），并将软件系统创建为一组由业务流程进行协调的交互服务。每个服务实现总体企业上下文中定义的特定业务目标或功能，业务流程表示必须实现的业务解决方案。

基于与 Jean-Jacques Dubray 的通信联系，表 1 总结了以应用程序为中心的方法和 SOA 方法之间的关键区别。

表 1. 以应用程序为中心的方法与 SOA 实现的比较

特征	以应用程序为中心的体系结构	SOA
设计和实现	• 面向功能 • 一次性构建 • 长开发周期	• 面向协作 • 按变更构建 • 渐进地构建和部署
最终系统	• 应用程序竖井 • 紧密耦合 • 面向对象的交互	• 企业解决方案 • 松散耦合 • 面向语义消息的交互

SOA 和分解

企业业务服务通常被定义为企业 IT 系统的分解结果。*分解 (Decomposition)* 是由 20 世纪 50 年代晚期的经典系统理论形式化了的一种技术。系统理论认为，某个系统越复杂，其包含的内容就越不可知，因此要自动化它就越困难。该理论建议将复杂系统分解为较小、更加可管理、更容易控制的系统，然后将整体系统视为其各个部分的组合。同样的原理也适用于复杂的软件开发计划。

软件分解的发展

在 20 世纪 60 年代初引入的最初软件分解方法是将大型机应用程序划分为单独的作业，每个作业分别由一个单独的程序实现。后来，随着人们加深对程序内部原理的认识，每个程序本身又按照其功能被划分为模块或子程序。

Simula 和 Smalltalk 在 20 世纪 70 年代引入的面向对象（Object-Oriented，OO）范例通过引入*对象* 或代码模块加强了分解方法的采用，其中每个对象实现某个实际事物的模型。其基本思想是在软件中表示“问题域的事物”，如客户、订单或交易。但是，对象所提供的抽象被证明太细粒度化，并与技术概念交织在一起来表示业务级别的含义。

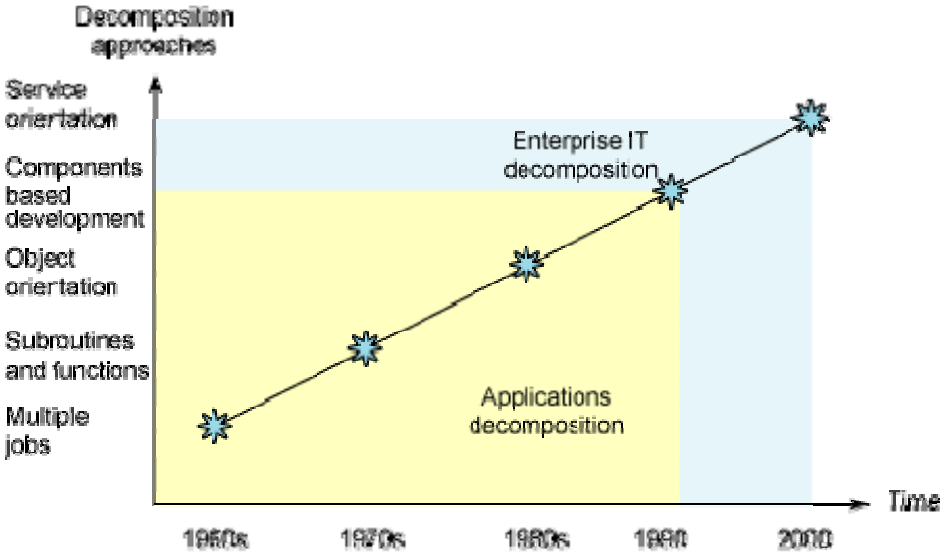
由于种种原因，许多面向对象的开发人员最终将大多数时间花在处理诸如集合、图形 Widget 等技术构造上。在大多数情况下，问题域的对象消失在乱七八糟的模块中，这些模块不再表示问题域专家可识别的任何事物。OO 的另一个问题在于，尽管在设计和实现期间，对象在分解方法中非常重要，但它们在部署或运行时不可

见，从而并不直接支持部署或运行时分解。

在对更好范例的不懈求索中，20 世纪 90 年代末引入了一种不同的分解方法。*组件*。其基本思想是通过提高抽象级别、增加粒度和创建与业务构件的更紧密联系来解决 OO 的问题。

软件组件的引入允许创建灵活、构造更好和更加可管理的软件应用程序。然而，它并没有解决主要的企业 IT 问题：其以应用程序为中心的性质。对象和组件都为各个应用程序提供了更好的设计和开发方法。SOA 将分解方法推向一个更高的层次，如图 1 所示。与尝试分解各个应用程序不同，它分解整个企业 IT 功能。

图 1. 分解方法的发展

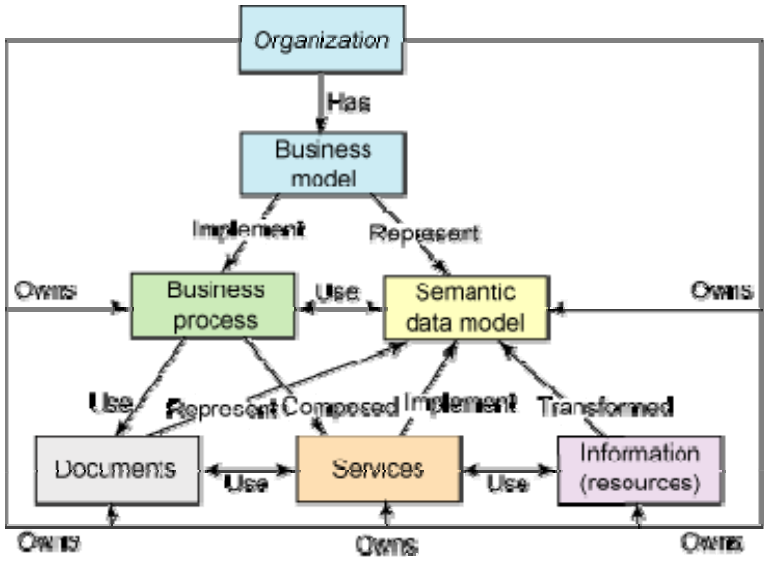


SOA 的元素

通过为其主要元素（业务服务和业务流程）提供业务含义，SOA 体系结构风格促进了业务和技术的一致性。事实上，服务和流程都可追溯到企业业务体系结构。

企业 SOA 定义一组与业务一致的 IT 服务（对整个企业中跨多个业务部门甚至企业外部的参与者可用），这组服务集合起来满足组织的业务流程和目标。可以将这些服务编排到企业业务解决方案中，并通过标准协议来调用。图 2 显示了企业 SOA 的主要元素（请参见参考资料 [13]）。

图 2. 企业 SOA 概念



组织

拥有所有 SOA 相关的构件（模型、服务、流程、资源）并控制它们的创建、使用、访问和维护。这些构件的作用是支持组织及其业务目标。

业务模型

满足企业经营、战术和战略业务目标所需要的业务资源和流程的主要表示形式。

业务模型对于服务与业务目标和要求的成功一致性非常关键，从而对总体 SOA 实现的成功至关重要。

语义数据模型

定义给定企业的标准业务数据对象（例如，客户、协议，等等）。这些对象通过定义公共概念及其内容（它们描述企业的功能），从而实际创建企业数据的实体。

使用该数据模型来定义业务服务接口就导致创建了可互操作的语义服务接口定义——语义 SOA。

服务

实现特定的企业业务功能并访问其数据和资源。定义良好并与业务保持一致的服务是灵活、可扩展的企业 SOA 实现的关键要素。

服务的结构允许独立地开发和部署它们。正确地定义服务并使其与业务和语义模型保持一致可以产生即插即用的实现，从而允许有效地将它们组合到不同的企业范围的业务流程和解决方案中。

业务流程

编排业务服务的执行以实现业务模型中指定的企业功能（例如，订单处理或索赔处理）。

业务流程通常以关键绩效指标（Key performance indicator，KPI）的形式与经营目标和业务目标关联，例如保险索赔处理或工程开发处理。作为流程实现的一部分来收集的 KPI 通常用于评估企业功能的有效性。

信息

表示组织的数据资源。数据以各种不同的格式驻留在各种各样的不同存储区、应用程序中。不同级别的数据由不同级别的 SOA 构造所使用。语义数据模型定义用于业务流程和服务的数据。SOA 定义了用于将数据从其本来的操作格式转换为业务所需的语义数据的机制。

文档

表示法律实体，用于定义企业及其合作伙伴的义务（例如，财务文档、保险单和索赔，以及政府规章）。文档是现代企业至关重要的一部分，必须连同其他企业信息一起作为一级优先事项包括在 SOA 实现中。

什么使得 SOA 与众不同？

人们进行了若干尝试，以图将 SOA 表示为一种新形式的分布式系统体系结构，或表示为面向对象的扩展，或表示为下一代 EAI。让我们进一步了解这些类比。

SOA 与分布式系统

W3C Architecture 任务组（请参见参考资料 [9]）将 SOA 定义为某种形式的分布式系统体系结构，通常以属性为特征，如表 2 所示。

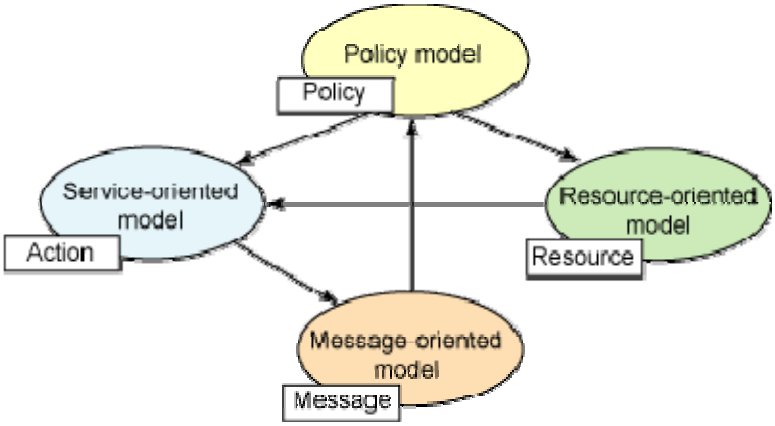
表 2. 分布式系统体系结构属性

属性	描述
逻辑视图	服务是实际程序、数据库、业务流程等的抽象逻辑视图，按照其功能（通常是执行某个业务级别的操作）来进行定义。服务被定义为一个具有业务意义的操作。
面向消息	按照提供者和使用者之间交换的消息来形式化地定义服务，而不是按照提供者和使用者本身的属性。实现的内部结构有意进行了抽象。服务接口独立于服务实现。

面向描述	通过计算机可处理的元数据/服务定义来描述服务。
粒度	服务往往使用少量操作，这些操作带有相对较大和复杂的消息（有效负载）。
面向网络	服务往往设计为通过网络进行使用，尽管这并不是一个绝对的要求。
平台无关	消息以平台无关的标准格式通过接口传递。XML 是满足此约束的最明显格式。

W3C 开发的 SOA 模型（如图 3 所示）可以按照调用消息、实现、拥有组织和描述服务的元数据来定义。

图 3. W3C 的 SOA 模型



- 面向消息的模型按照消息的内容（标题和正文）、传输方式和发起及执行代理来定义消息。
- 资源模型按照 URI、表示形式和拥有组织来定义资源或实现。
- 策略模型按照策略的主体（资源和操作）和组织来定义策略。策略是对允许的操作或状态的约束，例如某个安全策略。

尽管 SOA 是面向消息和网络的分布式系统体系结构，但它不仅限于此。将 SOA 与分布式系统等同起来，这仅强调了服务通信的技术和实现方面，而忽略了它的关键价值主张：业务-IT 一致性。

SOA 和面向对象

有些专业人员将 SOA 视为 OO 的直接发展，并将服务视为对象或组件（请参见参考资料 [10]）。这与事实差之甚远。它们之间的相似性并没有超出用于定义的系统分解和用于实现的封装。
诸如继承和多态性等其他对象特性并不适用于 SOA。两者之间的真正区别在于用法和编程模型（类似于基于实例的协作和基于服务的协作之间的区别，如参考资料 [11] 所述）。

- 在 OO 中，相同类型的多个对象实例（可能同时存在）基于表示特定执行上下文的内部状态来进行区分。因此，对象的生命周期由其使用者通过创建对象来显式控制。

  每个对象都公开多个方法，这些方法与某个特定的实例（执行上下文）绑定，并允许操作给定实例上的变量。

- 在 SOA 中，服务支持的不是特定使用者的执行上下文，而是支持与该特定服务关联的企业资源状态。典型的服务调用是无状态的；此规则的最明显例外是对话式组合服务，此类服务通常具有一个

执行上下文，并支持某个特定的对话。

因此，服务生命周期并不与任何特定使用者的生命周期关联——无论某个特定的使用者是否调用它，它都始终存在。由此而来的编程模型是服务的直接调用，而无需显式创建服务。

这个区别对于对象和服务的接口定义具有深远的影响。在 OO 中，接口上定义的多个方法从物理意义上讲始终属于该对象的同一实例，因为它们与同一执行上下文绑定。相反，由于服务没有执行上下文，因此服务接口中的方法关联纯粹是逻辑性的。

服务（也包括服务接口）实际上表示一个命名空间，此命名空间提供服务方法的逻辑分组，否则这些服务方法就是独立的实体，分别具有自己的服务质量要求、安全性、版本控制策略、端点地址等。让我们以一种编程语言作类比：服务的每个方法类似于 FORTRAN 子程序/函数，可彼此独立存在和执行。

执行状态和调用状态

这两个状态之间存在非常大的区别。*执行状态* 表示服务在其执行期间的状态。它始终存在，并包括在服务执行期间创建的内部变量。它用于跟踪服务执行的哪个部分已经完成、存储部分服务执行的结果，并在服务实现的多个组件之间传递参数。此状态通常封装在服务实现中，并对服务使用者不可见。

调用状态 是服务使用者和服务实现之间在某个特定会话中的共享上下文。使用者调用同一服务的不同方法，并假设在特定方法调用期间传递给服务的信息在所有后续调用中对该服务可用。在此情况下，一个服务参与多个具有不同使用者的对话，并单独地跟踪每个对话。例如，Java™ 2 Platform, Enterprise Edition (J2EE) 技术就在会话变量或有状态的会话 Bean 中使用调用状态。用于表示此类状态的更好术语是*对话状态 (conversation state)*。

请注意，在关于有状态与无状态服务调用比较的讨论中，我仅提到了调用状态。

SOA 与 EAI

传统的 EAI 实现通常基于 EAI 供应商推出的专有解决方案，从而创建与该供应商平台的“锁定”关系。许多专业人员将 SOA 视为下一代 EAI 技术。例如，Gartner Group 提出了术语“面向服务的集成”（Service-Oriented Integration，SOI），EbizQ 后来在其应用程序集成路线图中包括了 SOA。

随着作为提供传输解决方案的技术的 Web 服务的最新发展，它们通常被视为基于标准的集成解决方案。这使得它们成为 EAI 实现的极具吸引力（并且独立于供应商）的替代选择。企业服务总线（Enterprise Service Bus，ESB）产品的引入使得基于 Web 服务、基于标准的集成解决方案更加流行。

尽管 EAI 和 SOA 的目标通常是相似的——使用现有的应用程序组合来支持企业业务流程——但它们以根本不同的方法来实现该目标。EAI 集中于通过集成服务来公开应用程序的功能（请参见参考资料 [12]），实际上是将现有的应用程序组合作为一个企业业务模型来公开。相反，SOA 集中于隐藏现有的应用程序，并改为公开一组与应用程序无关的业务服务，从而将企业业务模型投影到现有的应用程序组合上。

服务

SOA 引入的基本抽象之一是*服务*。SOA 实现通常包含三种类型的服务（请参见参考资料 [12]）：

- 业务服务（表示与业务一致的 IT 构件）
- 集成服务（带有通过 SOA 技术来完成的集成实现，通常是 Web 服务）
- 基础设施服务（表示针对基础设施支持的公共 IT 构件）

请注意，本文讨论的“服务”指的是业务服务。

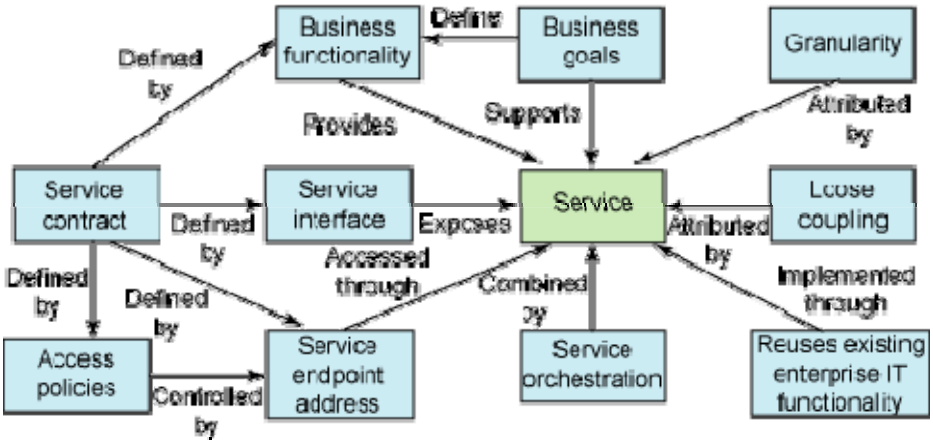
在最简单的情况下，服务可定义为独立地开发、部署、管理和维护的自包含软件实现，旨在支持与企业整体相关的特定功能，并且在设计上是“可集成的”。服务功能由服务接口定义，接口可由多种实现来支持。这意味着服务不是某种编程构造或一组 API，而是用于企业解决方案实现的体系结构构件（设计、实现和维护单元）。

服务实现关注事项

服务实现关注事项跨越从服务设计人员关注事项（如业务一致性和现有 IT 功能的重用）到公开给服务使用

者的关注事项（服务契约、服务接口和访问策略）的范围。图 4 大致描绘了这些元素以及它们之间的关系。

图 4. 服务实现关注事项



服务的业务功能

由企业业务模型定义，该模型又基于企业的业务目标。

服务契约

定义服务的业务功能、接口和一个或多个端点地址。

服务接口

以技术术语描述提供给潜在使用者的服务功能。接口被定义为一个服务名称和该服务支持的一组操作（方法）。每个操作的描述包括该操作调用（请求）所需的参数集定义，并在适用的情况下包括操作返回的结果（应答）。该描述还介绍操作的功能和前置及后置条件。

服务方法

可通过端点地址来访问，通常被定义为某个地址的网络位置。每个端点地址由一组访问策略控制。这些策略定义用于数据传输的通信协议、实际服务调用和服务质量（Quality of Service，QOS）（例如，某个给定端点地址所提供或要求的安全性和保密性）。

粒度和松散耦合

表示重要的服务设计特性。

服务实现

争取最大限度地重用现有的企业 IT 功能。

服务编排

表示将服务组合为更大的服务和在服务基础上构建企业解决方案的普遍机制。

以下各部分将进一步研究主要的服务实现关注事项。

业务一致性

SOA 的主要承诺之一是业务-IT 一致性。仅当服务与企业的业务模型保持一致时，才能实现此承诺。这意味着企业业务模型是成功的 SOA 实现的先决条件。必须准备一个高级模型来设定服务的方向、分区和分类。模型的细节可以随时间而与服务的开发并行发展变化。（请参见参考资料 [12]。）

根据聚合体系结构的规定，这种定义企业业务服务的方法可以实现更好的业务、组织和应用程序（服务）体系结构一致性。（请参见参考资料 [8] 和 [14]。）然后将软件实现（服务、流程）追溯到业务体系结构就更加容易了，并使得软件分析人员更容易理解软件应用程序，同时还可以简化业务功能变更的实现。

服务粒度

传统的分布式系统技术，如 CORBA 和 DCOM，都提供了本地和远程透明性。无论目标位置如何，它们都提供相同的编程模型。这样的一致性不仅简化了编程模型，而且还可以使系统设计人员和开发人员不必花大量时间去考虑和定义边界。这种开发简单性与执行速度之间的权衡，是许多使用分布式对象构建的应用程序

性能低下的主要幕后原因之一。这决不是分布式对象技术的问题，而是跨系统边界的细粒度对象交互的一个症状。（请参见参考资料 [15]）。

在 SOA 中，所有服务调用都是远程的。 这使得服务内（在服务内部）和服务间（跨服务边界）调用的不同特征变得非常明显。 这个区别强调了服务调用的开销非常大。结果，粒度就变成了服务设计的重要特征之一。

由于服务调用涉及到网络，因此将它们设计为粗粒度的。服务方法的执行所交付的价值必须能够弥补网络请求延迟的代价。因此，公开的服务接口必须是粗粒度的。与公开许多提供有限功能的接口不同，服务应该公开少量接口，这些接口允许各个请求执行完整业务功能。

> 在现实中，某些服务内调用也可以是远程的，例如在将服务实现为多个 Enterprise JavaBeans (EJB) 组件时。

恰当的服务粒度不仅允许创建性能更好的系统，而且还可以减少耦合。粗粒度的服务往往是自包含的，从而对其他服务的依赖性更少。

耦合

如果您希望使用服务来构建、维护和修改灵活的企业解决方案，以及支持整个企业中的联合开发方法，则那些服务的松散耦合是先决条件。下面的列表包括了一些耦合的后果（请参见参考资料 [16]）。

- 更紧密的耦合往往随时间推移而增加开销：
  - 同步多个组织的变更
  - 调整和重新部署已更新的组件而不影响其他组件
  - 难于做出更改，并且开销非常高或无法实现
  - 难于单独管理解决方案的不同部分
  - 难以移动、难以扩展、难以分发、难以替换
  - 更多的耦合意味着更高的测试开销
- 更松散的耦合需要更大的前期投资：
  - 更多的设计工作
  - 更多的实现工作

SOA 实现中的耦合的最重要方面包括：

功能耦合

尽管基于接口的设计并不新鲜，但是 SOA 则更进一步，它使（服务）接口基于企业范围的语义。早期的 SOA 采用者低估了使用企业范围的语义来实现服务互操作性的重要性。Web 服务社区希望，定义良好的 SOAP 消息内容和结构与有效负载的 XML 表示形式以及传输的标准化相结合，将确保 Web 服务通信的互操作性。在现实中，这仅提供了技术上的互操作性，并确保来自一个系统的 SOA 消息可由另一个系统接收和成功处理。然而，它并没有以任何方式帮助实现语义互操作性——两个系统相互“理解”的能力。

此类区别的例子随处可见。例如，仅仅是因为许多语言都使用罗马字母，并不意味着使用这些语言（它们使用相同的字母）的人就能够相互了解。纯粹的字母知识不足以理解语言。

使用企业语义数据模型来定义服务接口能够创建可互操作的语义接口定义。（请参见参考资料 [13]）。这些语义 SOA 显著增强了服务之间的互操作性：它们全都在接口级别使用同一模型。

语义消息的引入要求服务实现支持两个完全不同但同等重要的数据模型。（请参见参考资料 [17]）。

- 内部数据模型：由服务实现所使用。该模型与服务的内部实现相关，从而特定于基础的服务和组件。该内部模型不对服务使用者公开。

- 外部数据模型：用于服务间交换，并且是企业语义数据模型。

每个服务负责企业和内部数据模型之间的语义代理和转换。

时间耦合

服务（尤其是 Web 服务发布）往往集中于同步服务调用。当前实现的缺点有时被视为 SOA 本身的缺点。因此，有若干出版物主张将事件驱动的体系结构（Events-Driven Architecture，EDA）和 SOA-2 作为修复 SOA 的方法。依我看，需要修复的不是 SOA，而是其当前的用法。

使用同步通信来实现服务调用创建了服务使用者和提供者之间的时间耦合。

- 服务提供者必须正常运行，服务使用者才能访问它。
- 由于同步调用是阻塞调用，其中一个服务执行或请求/响应传递中的性能变化都会对服务使用者的执行产生重要影响。

使用单独的应答调用来实现异步服务调用允许使用者在提供者等待响应时继续执行。这导致了时间上分离的系统。在这种情况下，暂时的服务提供者中断和网络延迟只具有很小的影响。只要服务提供者返回响应，总体系统就能正常工作。

服务代表总体系统中的粗粒度独立实体，它们使得服务的可伸缩性和高可用性成为极其重要的问题。异步调用可以提供更好的可伸缩性和可用性特征，更适合用于 SOA 实现。

事务耦合

事务性，尤其是原子性、一致性、隔离性和持久性 (ACID)，是解决分布式计算中可靠性问题的流行方法的代表。金融应用程序采用它来进行资金转帐，电子商务系统使用它开进行支付处理，制造应用程序使用它来进行库存控制，电信记帐系统使用它来进行呼叫计费，等等。

ACID 事务通常是使用事务监视器（例如，Tuxedo、CICS、Encina）或组件平台（例如， J2EE 应用程序服务器或 MTS）来实现的。对 ACID 事务的支持需要整个事务环境的耦合，从而限制了互操作性和灵活性。

ACID 事务实现的另一个要求是事务持续过程期间的资源锁定，这需要有保证的较短服务执行时间。较长的事务时间通常导致总体事务资源吞吐量的恶化。

ACID 事务虽然非常适合于对象和组件，但是对服务来说则通常太受限制了。SOA 通过传统的两阶段提交协议来支持业务事务。（请参见参考资料 [18] 和 [19]）。

业务流程

业务服务本身很少构成完整的业务解决方案，它们通常充当构件。完整解决方案的创建通常是通过组合或编排各个参与服务来实现的。此类组合通常使用业务流程来完成。

业务流程和以流程为中心的计算并不新鲜。从办公自动化和工作流系统开始，它们已在 IT 中成功使用了 20 多年。希望提高效力和竞争力的公司必须作出转变，使流程成为基于计算机的自动化和支持的基本单元。它们必须将重点从记录系统转向流程系统。“流程处理”需要取代“数据处理”。

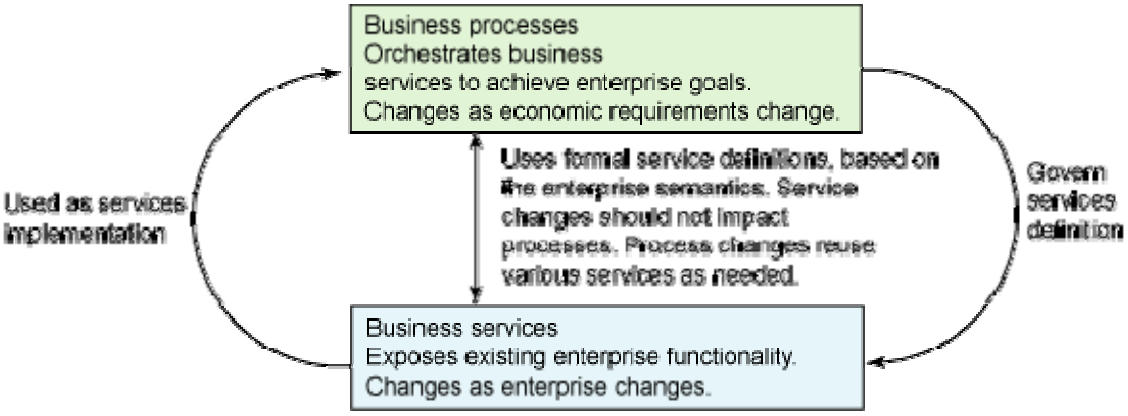
BPM 的核心是编排概念，其中某个流程引擎在操作输入/输出数据时对若干个手动和自动流程步骤进行编排。SOA 使得 BPM 的实现变得更切实可行。（请参见参考资料 [20]）。SOA 中的业务流程促进了业务流程从纯粹的自动化到*受到管理的灵活性* 的下一个发展阶段。

SOA 的目标是将组织的计算资产公开为可重用的业务服务，并实现基本的业务功能，这些功能更容易通过业务流程来使用（重用）和集成。这种使用现有服务来快速组装和重新组装解决方案的能力是 SOA 的主要优势之一。如图 5 所示，业务服务和业务流程之间的关系为实现真正灵活的企业铺平了道路。

- 业务服务支持稳定的业务构件，并整合了其接口很少更改的处理和规则。（然而，服务实现可以并且通常随时间而更改。）

- 业务流程支持相当易变的业务过程和规则，这些过程和规则可能每隔几个月甚至每隔几周就会更改。
- 业务流程和业务服务之间的交互基于企业语义，从而最小化了服务更改对业务流程的影响，并简化了从业务服务构建流程的过程。

图 5. SOA 中的服务和流程之间的关系



这种职责分离允许业务分析人员和 IT 架构师重用 IT 功能，这些 IT 功能通过编排服务的业务流程封装在业务服务中。这简化了新流程的创建和现有流程的优化。更重要的是，这些流程一经设计，即可快速对其进行修改以响应市场条件。所有这一切都转换为业务灵活性和竞争力的提高，而不会显著增加进行频繁流程更改所导致的边际成本。

SOA 和模式

通过模式来解释和实现某种体系结构风格是已得到证实的方法之一。

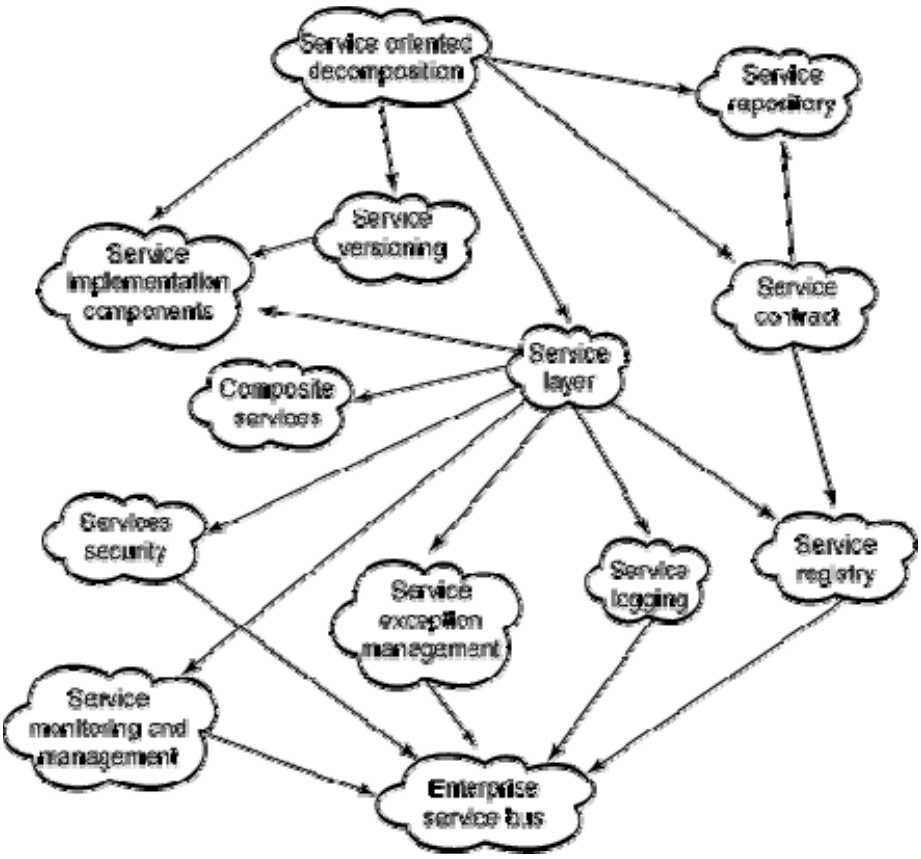
“模式和模式语言是描述最佳实践、经证实的设计和获取过去经验的方法，以便其他人能够学习这些经验。模式是一种已得到证实的方法，用于快速理解设计指导和可在其中应用这些指导的各种上下文”。——John Evdemon（请参见参考资料 [23]）

最近有许多关于 SOA 模式和模式语言的出版物（请参见参考资料 [21-26]）。然而，每种出版物仅集中于 SOA 实现的某个特定方面，包括：

- 服务访问和基础设施（参考资料 [21]）
- 服务组合（参考资料 [22]）
- 服务定义和实现（参考资料 [23]）
- 服务版本控制（参考资料 [24]）
- 服务安全性（参考资料 [25]）
- 企业数据访问（参考资料 [26]）

真正的 SOA 模式语言应该定义全面的服务设计、创建、利用和执行——完整的 SOA 生命周期——方法。图 6 显示了此类语言的一个示例。

图 6. SOA 的模式语言



该模式语言包括三个主要方面（有些模式可能属于多个方面）：

服务设计

这一组模式包括一些最重要的服务设计活动。面向服务的分解模式提供起点，描述如何基于手边的问题来定义服务，包括企业业务模型和所需的长期体系结构质量。

服务契约模式定义正式的服务定义的创建，并允许系统设计人员为他们的实现选择最适当的服务，允许架构师和开发人员正确地调用那些服务。

服务存储库模式定义用于组织企业中的服务相关构件的解决方案，从而简化服务定义、创建和使用中涉及到的涉众之间的协作。

服务版本控制模式提供指导方针，用于处理服务契约和实现中不可避免的更改，从而使您可以减轻这些更改对服务使用者的影响。

服务实现

这些模式包括一些与服务实现活动相关的问题。服务实现层显示了如何利用现有企业应用程序组合来实现与企业业务模型保持一致的服务服务实现组件模式通过定义典型服务实现中涉及的组件以及它们的关系，从而对前述模型形成补充。

组合服务模式探索了一种方法，用于对多个现有服务的功能进行组合以支持服务使用者所需的功能。

服务基础设施

这组模式涵盖服务运行时基础设施的一些重要问题。服务注册中心模式集中于晚期绑定的实现，以便通过集中的注册中心实现服务调用，从而允许在单个企业范围的点控制服务的部署和运行时功能。

服务安全性是一种专门的模式语言（请参见参考资料 [25]），它定义一组模式并控制服务安全性的设计和实现。

服务日志记录和服务异常管理模式定义用于在高度分布式环境中进行集中日志记录和异常管理的

方法。

服务监视和管理模式集中于对服务执行进行监视和管理的方法。最后，企业服务总线模式提供对服务通信虚拟化的认识，从而简化前面提到的基础设施模式的实现。

总结

在本文中，您了解了 SOA 体系结构风格的构成。您探索了此风格的主要元素以及它们的交互。您研究了此风格背后的基本原理，以及它与其他用于构建企业体系结构的方法的区别。我们还简单谈到了一种模式语言，您可以使用它来理解和简化此风格的实现。

敬请关注即将推出的文章，它们将更详细地讨论模式。