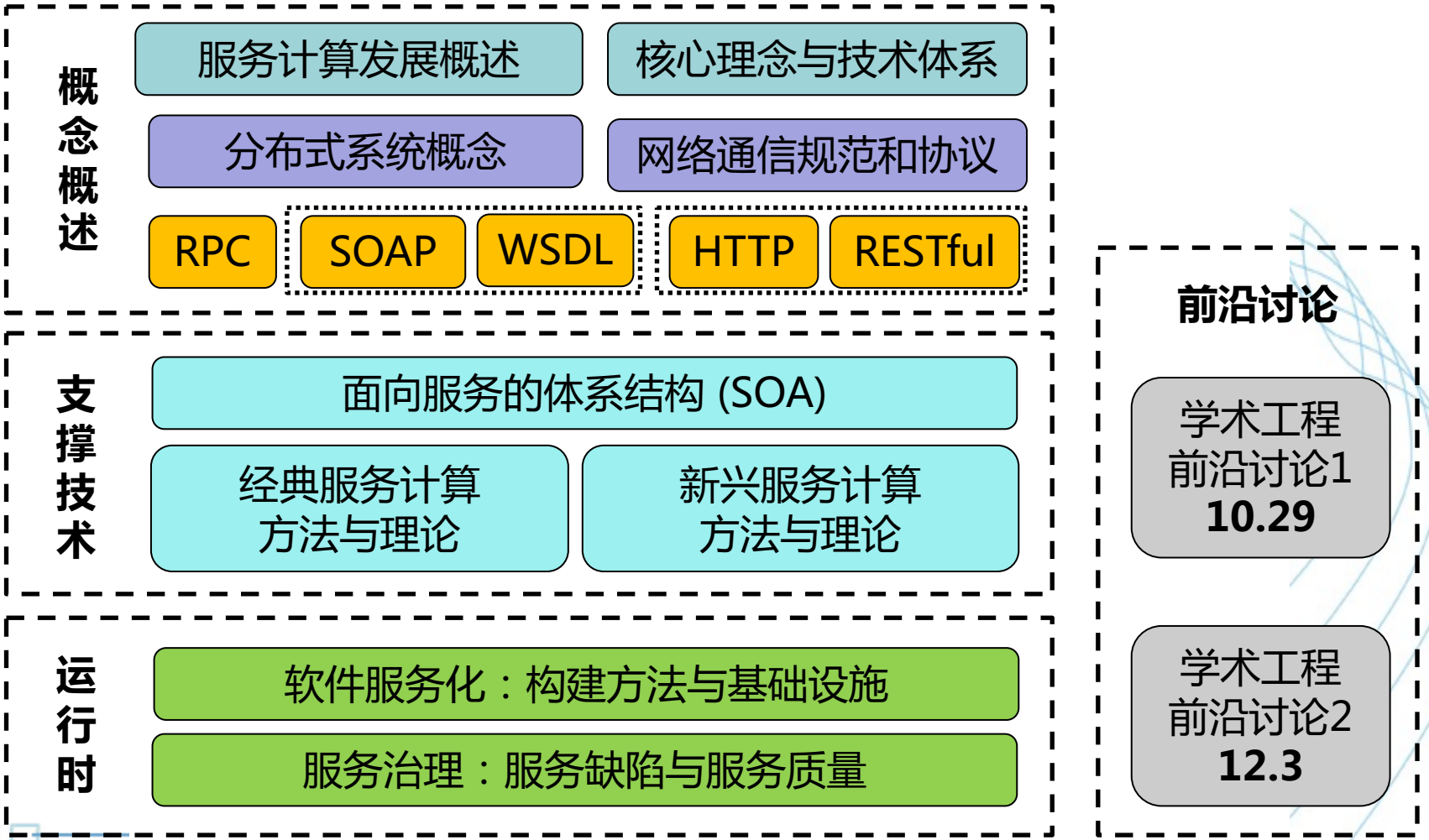
服务计算基础
Foundations of Services Computing

第三讲
面向服务的体系结构(SOA)

杨任宇 副教授
北京航空航天大学软件学院
北京市计算机新技术研究所
复杂关键软件环境全国重点实验室
renyuyang@buaa.edu.cn
yangrenyu.github.io

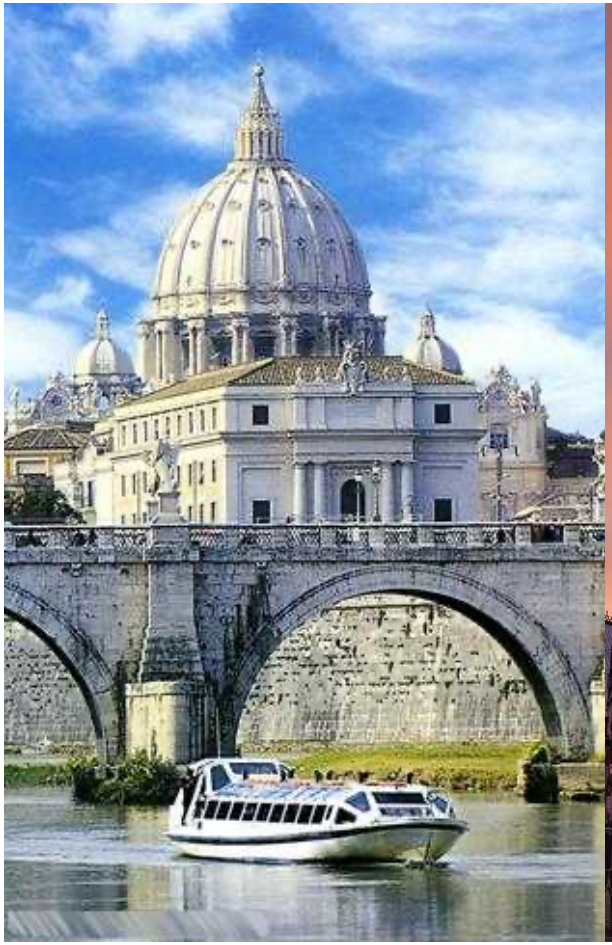
课程单元设置

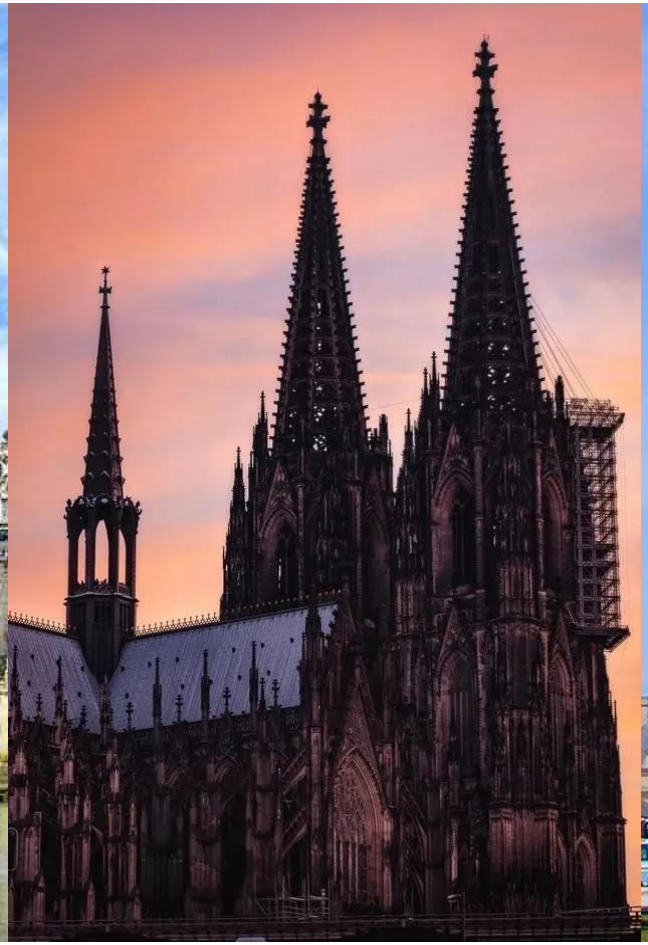


提纲

- 1. 软件体系结构简介
- 2. 什么是SOA
- 3. SOA参考架构
- 4. SOA描述语言
- 5. 云环境中SOA发展方向

建筑风格





什么是体系结构？

- 建筑风格：中国古代建筑风格、西方建筑风格
- 程序设计样式、设计模式、体系结构风格
- 软件体系结构(风格 style)：
  - 描述某一特定应用领域中系统组织方式的惯用模式
  - 反映领域中众多系统所共有的结构和语义特性
  - 指导如何将各个模块和子系统有效地组织成完整的系统

软件体系结构

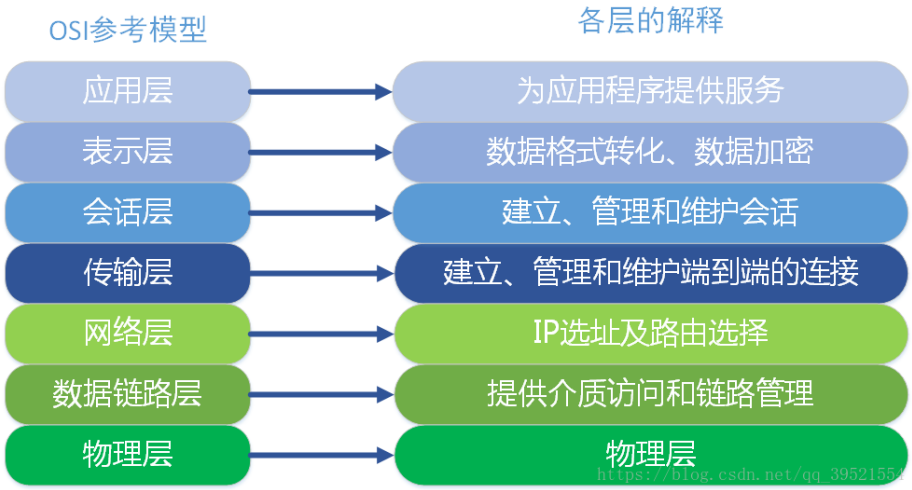
- IEEE 1471-2000
  - 软件体系结构涉及系统（软件组件）的结构描述、组件的外部可见特性、组件间的交互关系，以及管理、设计和演化的原理与准则
- 软件体系结构是软件系统的高层结构
  - 根据功能组件和组件间的交互进行结构描述
  - 通过“契约(contracted)”接口，可标识组件，可指派和客户端组件交互的任务
  - 组件交互规定了通信和控制机制，并支持实现系统行为所需的各种组件交互

软件体系结构（续）

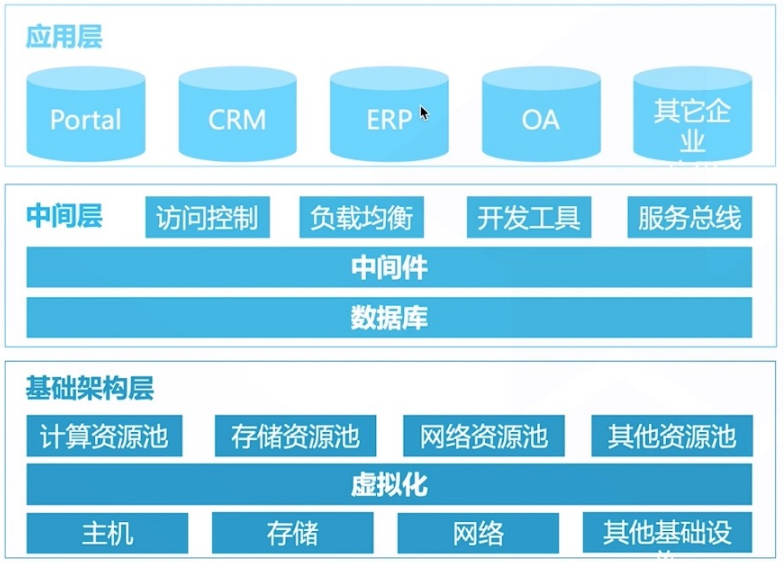
- 常见风格
  - 层次系统（Layered Systems）
  - 管道-过滤器（Pipes and Filters）
  - 客户/服务器（Client/Server）
  - 浏览器/服务器(B/S)
  - 微核（Micro Kernel）
  - 黑板系统（Blackboard）
  - Service-oriented architecture
  - Model-driven architecture

软件体系结构（续）

- **分层体系结构 Layered Architecture**



计算机网络 OSI 七层结构

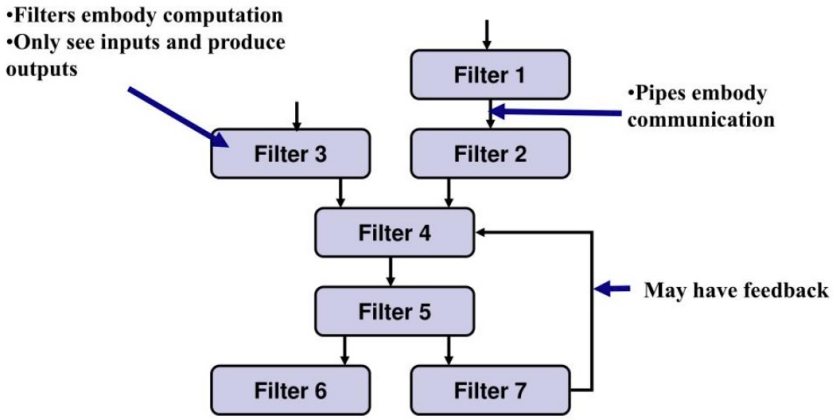


云计算经典三层架构

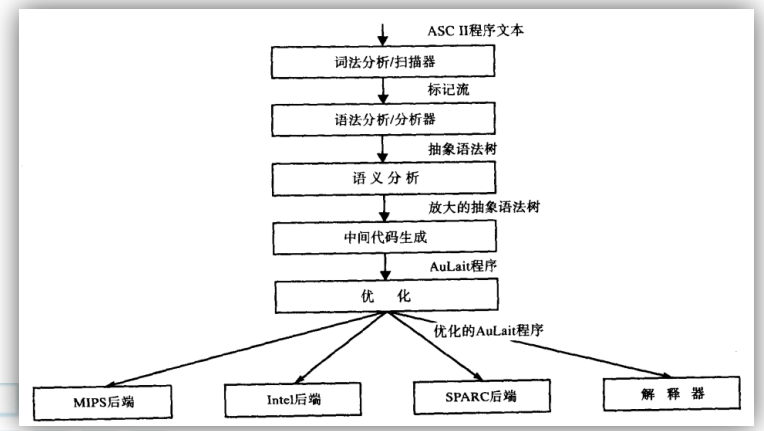
- 每一层实现各自的功能和协议，完成与相邻层的接口通信
- 通过接口提供给更高一层，每层所提供的服务与具体实现无关

软件体系结构（续）

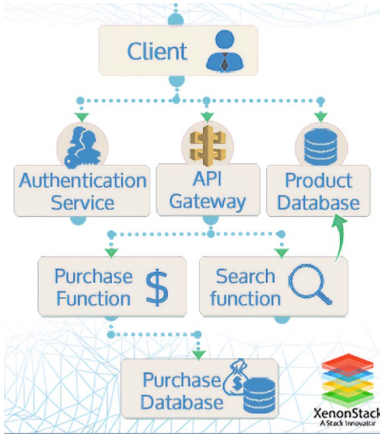
- **管道和过滤器体系结构 pipes and filters**



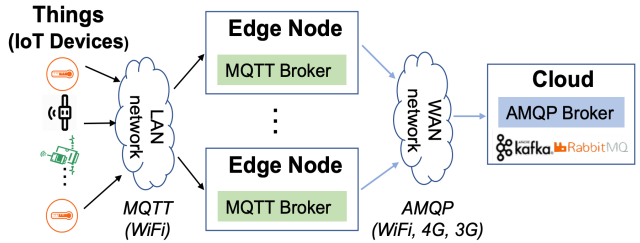
- 面向数据流的软件体系结构
- 管道是传输数据的组件，用于将数据从一个过滤器的输出接口传送到下一个过滤器的输入接口



程序编译



无服务器计算

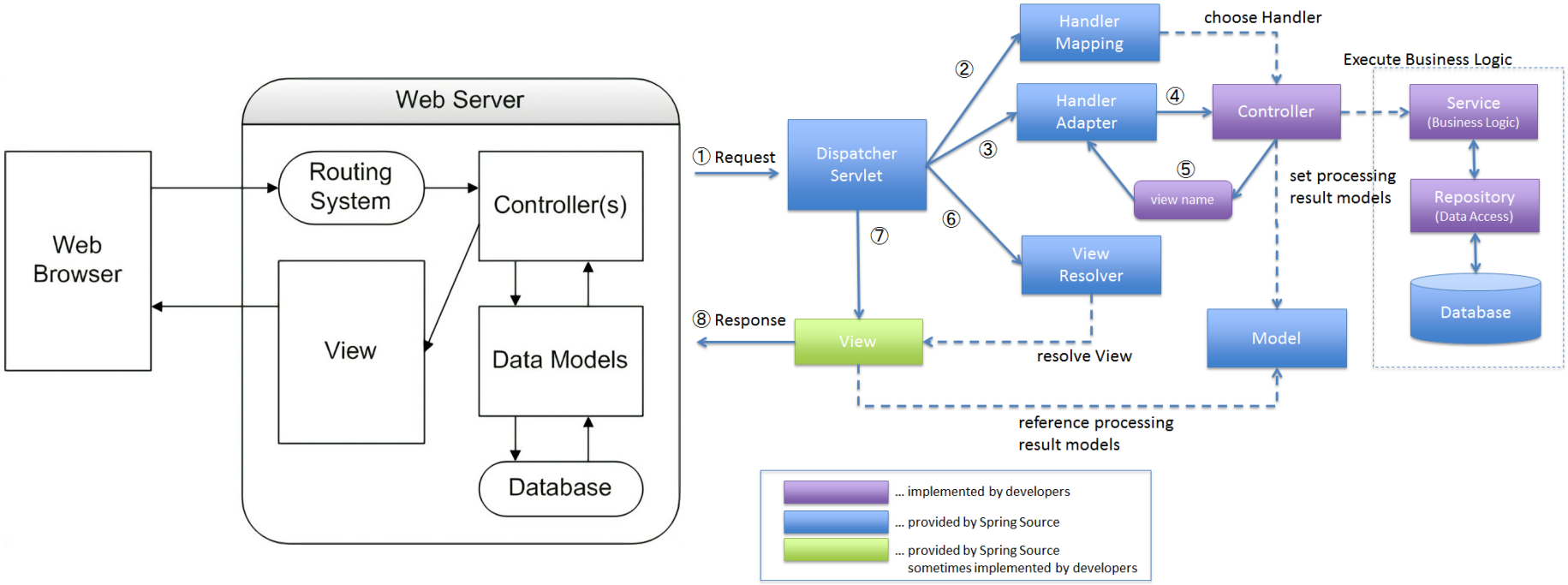


Streaming pipeline in Cloud-edge computing paradigm

IoT 物联网系统

软件体系结构（续）

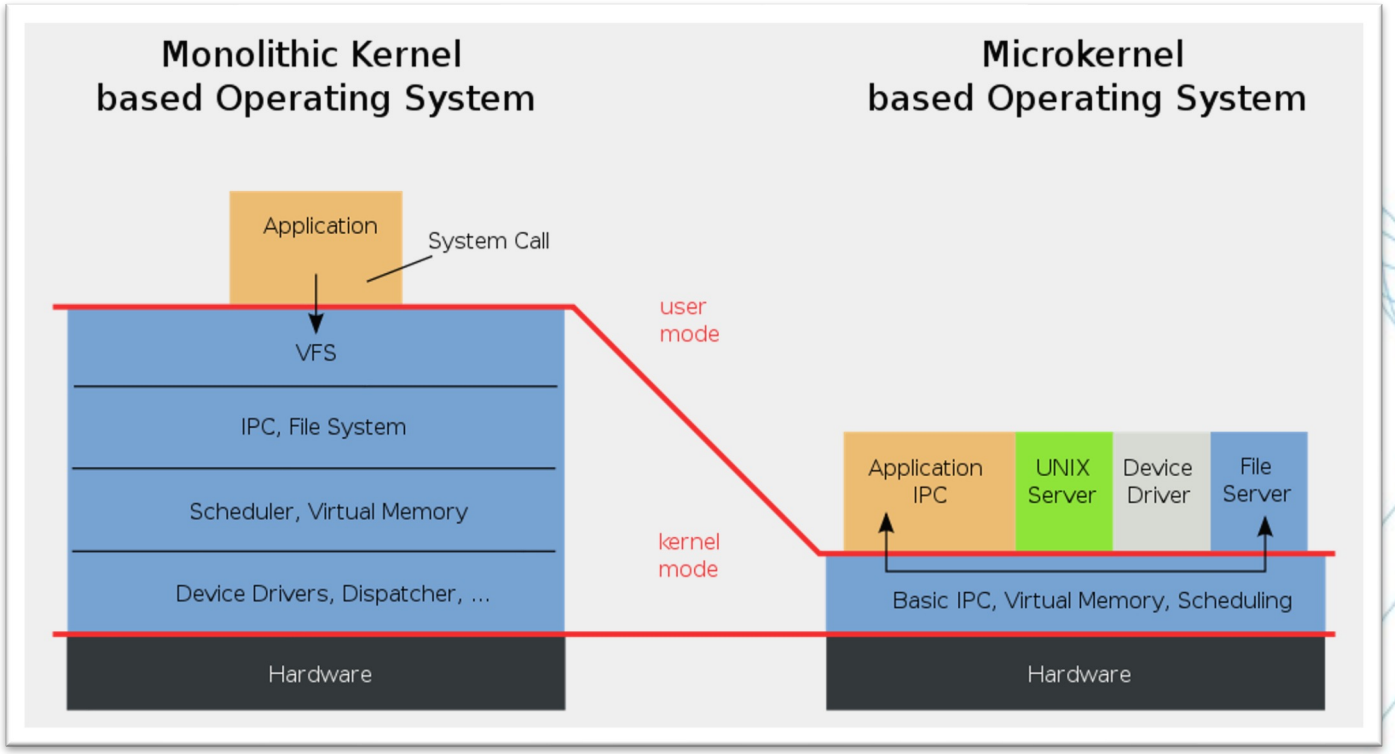
- **模型-视图-控制器（MVC）体系结构**



模型层（Model）：从现实世界中抽象出来的对象模型，封装数据和数据操作，反映应用逻辑
视图层（View）：应用和用户之间的接口，它负责将应用可视化给用户和显示模型状态
控制器（Controller）：控制器负责视图和模型之间的交互，控制对用户输入的响应方式和逻辑：1. 将用户的请求分发到相应的模型，2. 将模型更新及时反映到视图。

软件体系结构（续）

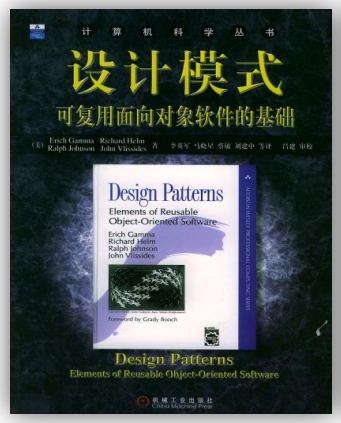
- **微内核体系结构 micro kernel**

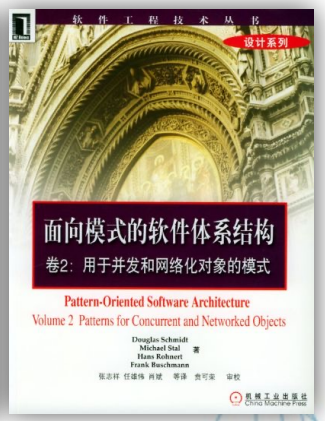


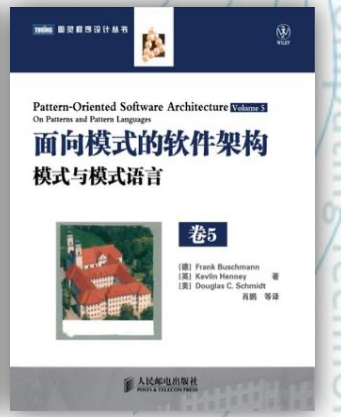
核心：微内核只完成最基本功能（中断、进程管理、进程调度、进程间通信）
优势：方便定制、小巧，适用于移动设备和 IoT 设备

面向模式的体系结构

- 软件设计模式（Software Design **Pattern**）描述了在软件设计过程中的**一些不断重复发生的问题**，以及该问题的**通用解决方案**
- 本质是面向对象设计原则的实际运用，是对类的封装性、继承性和多态性以及类的关联关系和组合关系的充分理解
- 敏捷软件开发实践提倡的“Refactoring to Patterns”是目前普遍公认的最好的使用设计模式的方法







面向服务的体系结构

- **Service Oriented Architecture - SOA**

… a service?

一个**可重用的独立功能组件**，如客户信用检查、开通新帐户

… service orientation?

将**业务集成**起来形成互联服务的方法，以及形成的**系统形态**

… service oriented architecture (SOA)?

一种支持“面向服务”的IT体系结构**风格**

… a composite service?

一组**集成**的服务，它们可支持SOA中的一个复杂**业务过程**

提纲

- 1. 软件体系结构简介
- 2. 什么是SOA
- 3. SOA参考架构
- 4. SOA描述语言
- 5. 微服务架构

什么是“SOA”

- **狭义的解释**
  - SOA是一种支持面向服务思想的体系结构风格(architectural style)，是一种**业务和IT的范式**(paradigm)
- **广义的解释**
  - 在软件工程中，SOA是设计和开发**互操作服务形态**的软件的一组原则和方法。这些服务是**定义良好的业务功能**，是为不同目的**可重用的软件构件**
- **OASIS (Online Academic Support and Information System)**
  - 是一种组织和使用**不同管理域**控制下**分布式能力**的范式。它提供了一种统一的方法以提供、发现、交互和使用这些能力，从而产生预期的效果
- **Thomas Erl（SOA领域的领军人物倡导者）**
  - SOA表示了一种开放、敏捷、可扩展、联邦、可组合的体系结构，由自治、具备QoS能力、松散、可互操作、可发现、可重用的服务构成。SOA可建立业务逻辑和技术的一种抽象，实现不同管理域间的松耦合。SOA是传统平台的进化，在保留传统体系结构优势的情况下，引入了面向服务的思想

什么是“SOA”

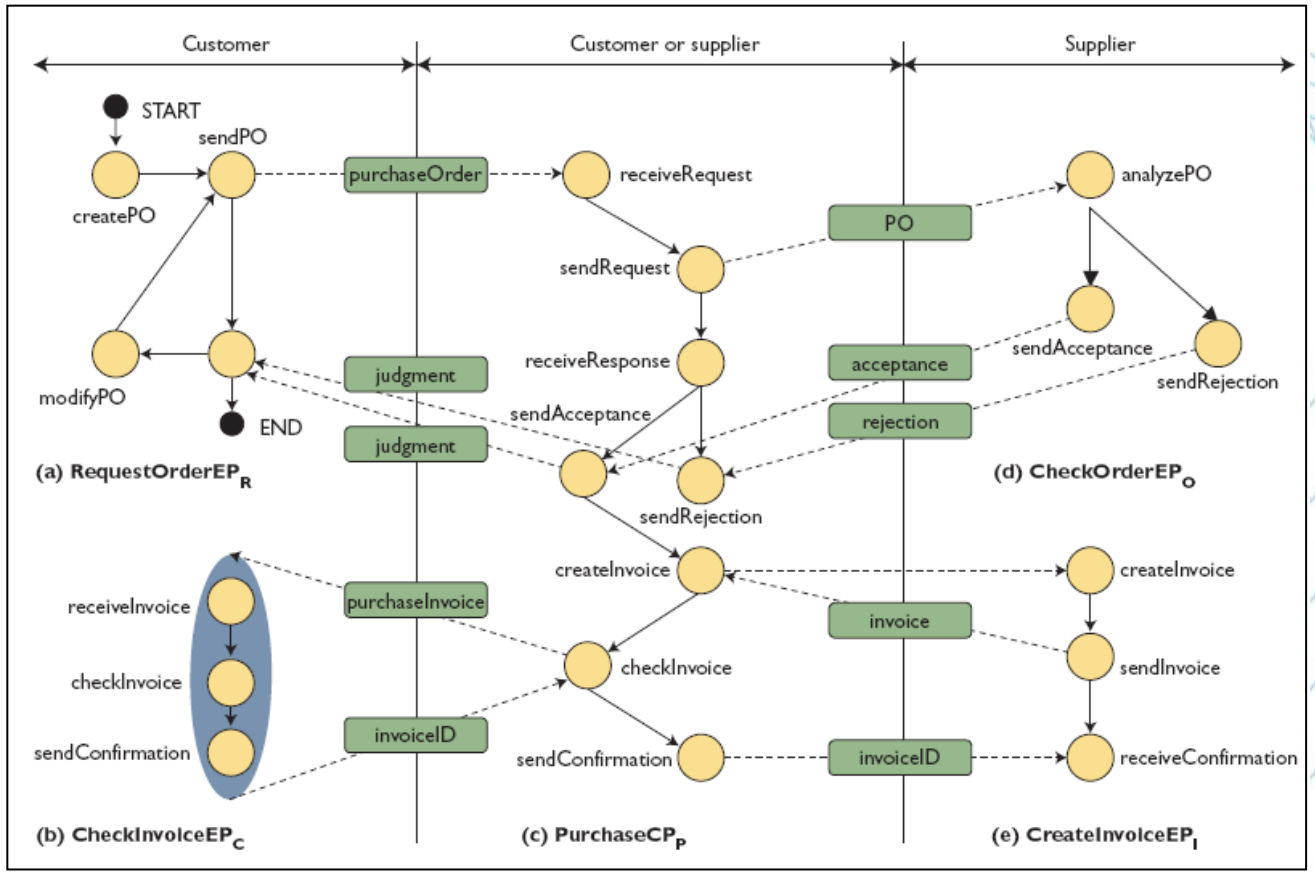
- **SOA = Service (服务) + 体系结构 (Architecture)**

	Roles	
企业向其客户暴露的一系列业务——”服务”	业务模型	
一种体系结构风格，包括服务的提供者、服务的使用者与服务中介。 一组体系结构设计原则与模式，强调模块化、封装、松散耦合、分离关注点、可复用、可组合性、接口与实现分离等特性。	体系结构	
一种编程模式，包括一系列的标准、开发工具、开发过程指南、运行时基础架构。	软件实现方式	

- 将应用程序的不同功能单元（服务）通过定义良好的接口和协议进行组合
- 服务独立可重用，可跨多个系统和组织进行交互

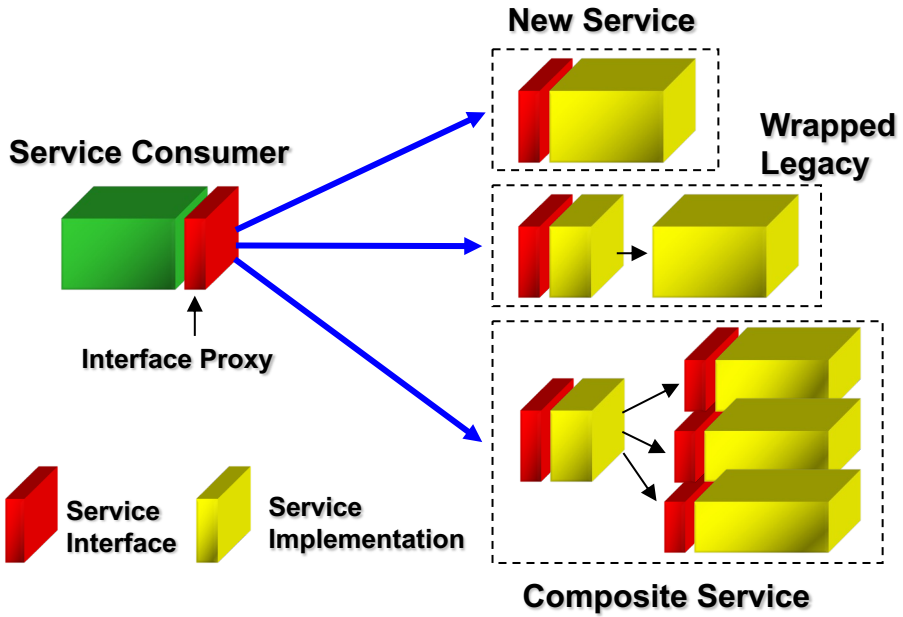
从业务的角度看SOA

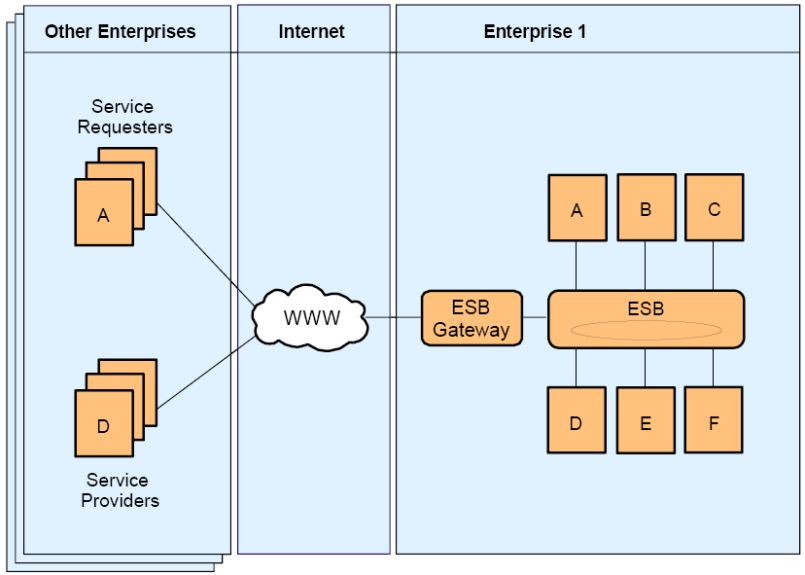
- 是**业务服务的集合**，用于表达企业向客户暴露的业务设计



从体系结构的角度看SOA

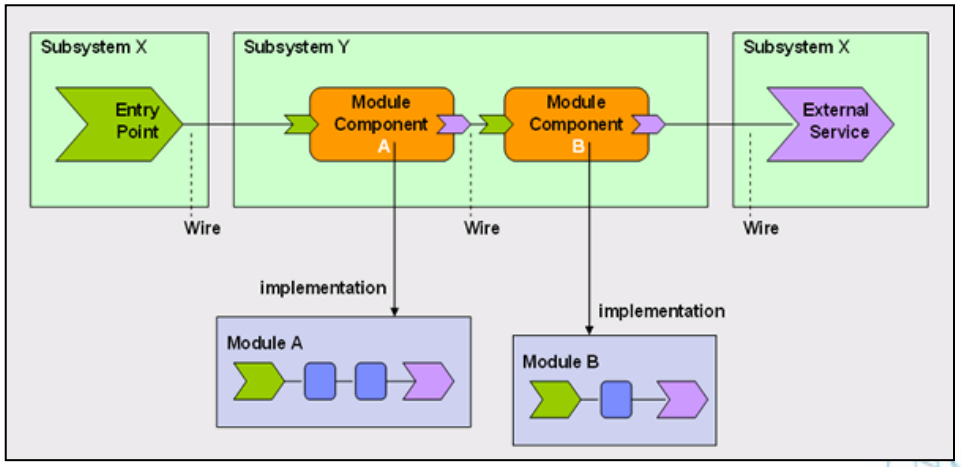
- 一种**体系结构风格(style)**，由服务提供者、请求者和服务描述构成

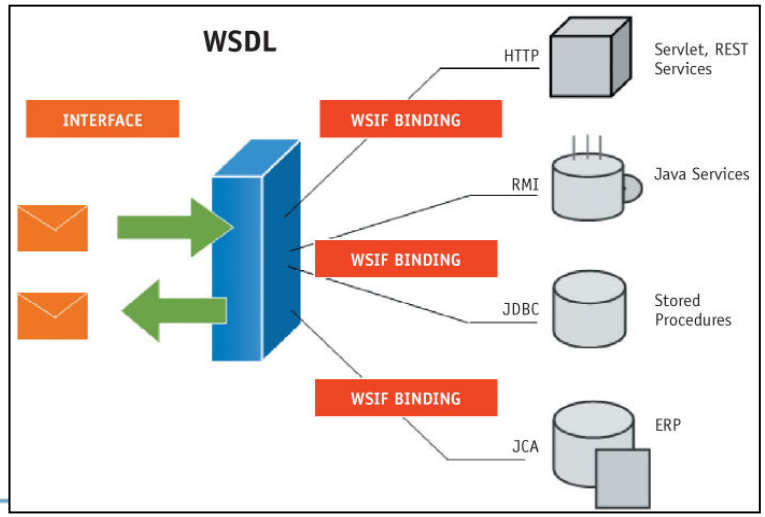


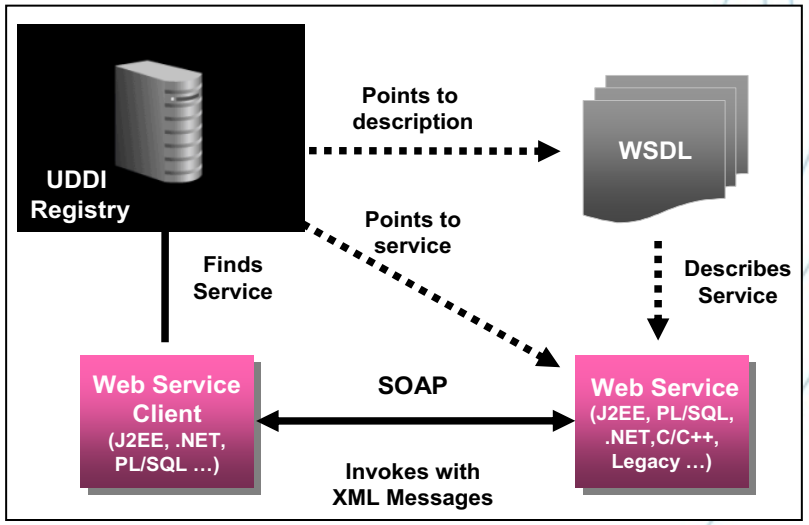


从编程模式的角度看SOA

- 是一种**编程模型**，涉及标准、工具、方法、技术等

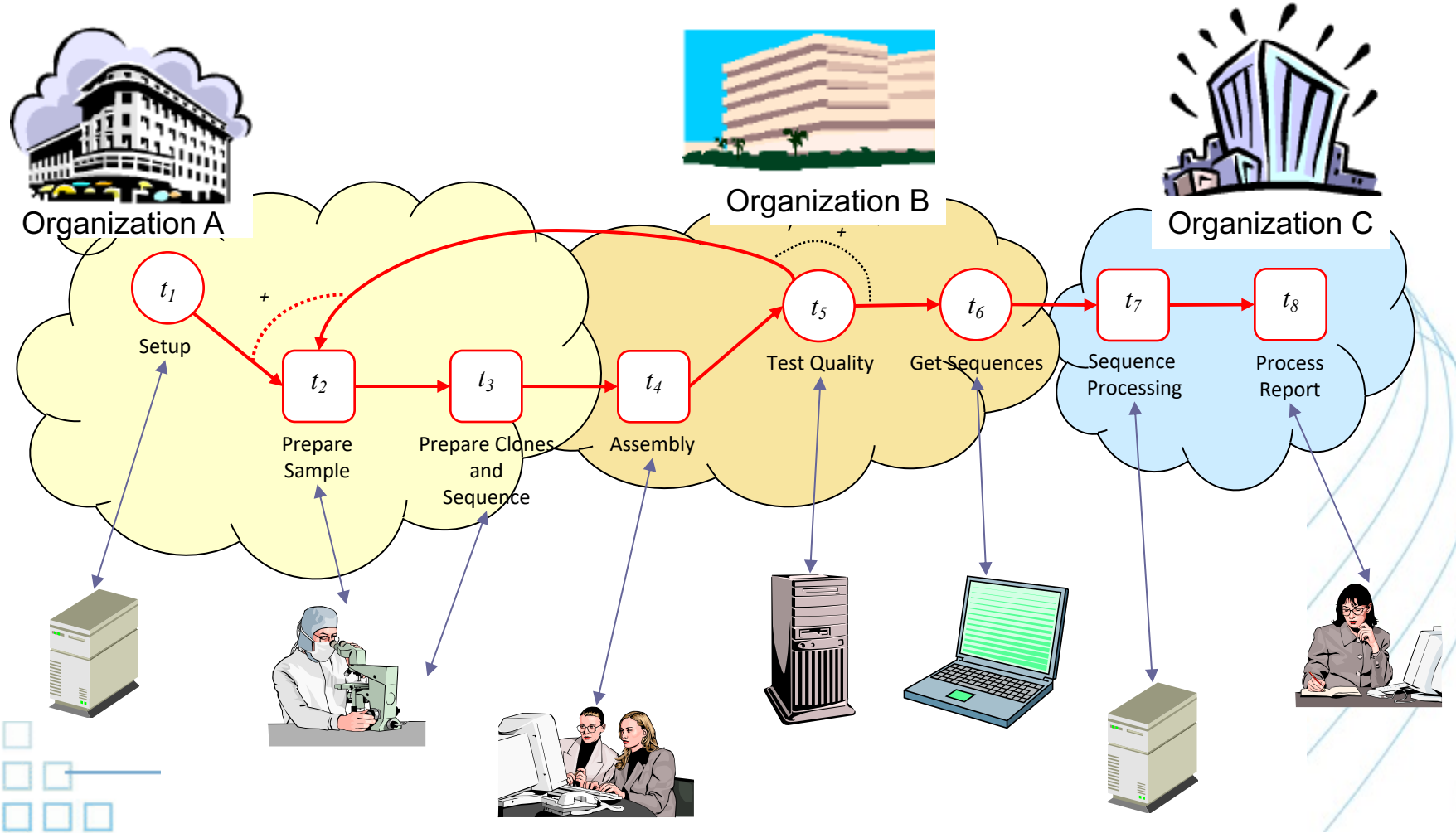






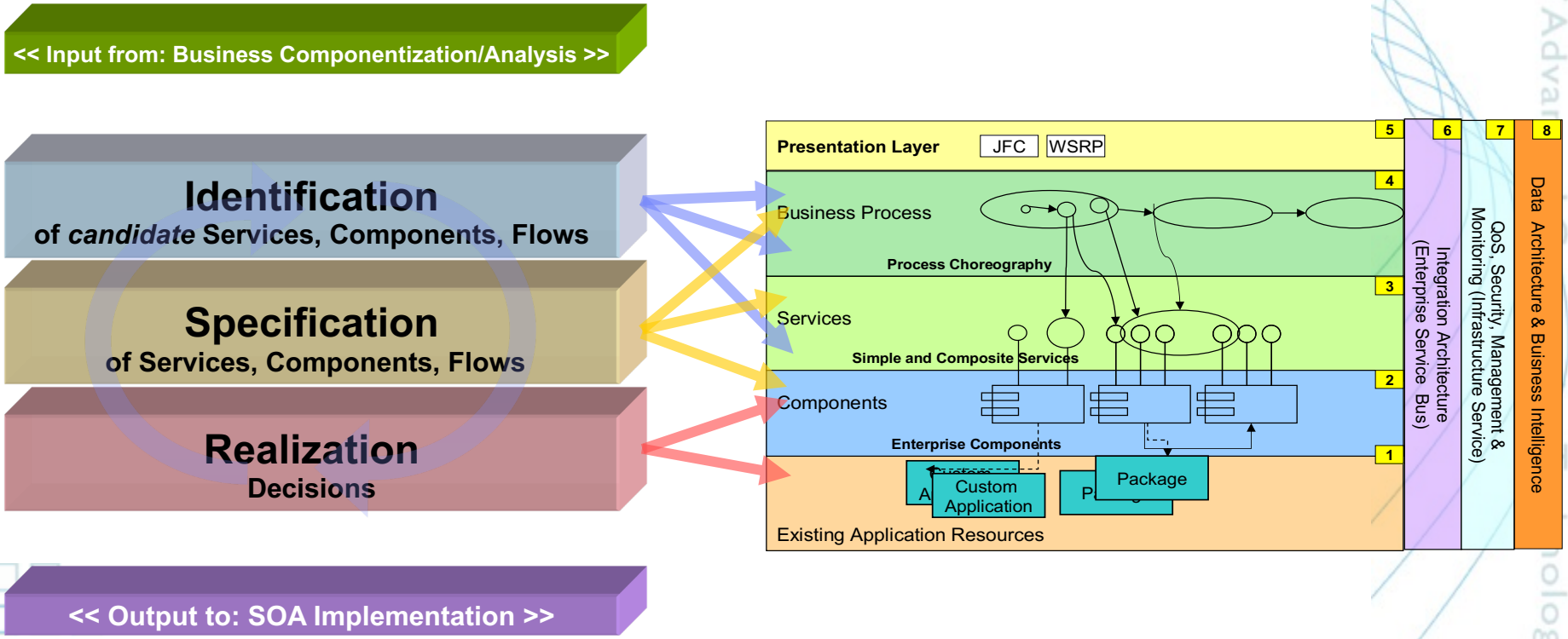
从应用的角度看SOA

- **跨功能**（有时**跨组织/跨管理域**）的整合模式



从开发过程的角度看SOA

- **面向服务的分析与设计**
  - 识别、设计和实现服务、构件、以及服务之间形成的协同。SOA是开发过程的核心对象，逐层精化



SOA的原则

- **开发，维护和使用SOA的基本原则**
  - 互操作性 interoperable
  - 可描述性 described
  - 可重用 reusable
  - 可发现性 discoverable
  - 可组合性 composable
  - 自包含 self-contained
  - 松耦合 loosely coupled
  - 可管理性 manageable

SOA的优势

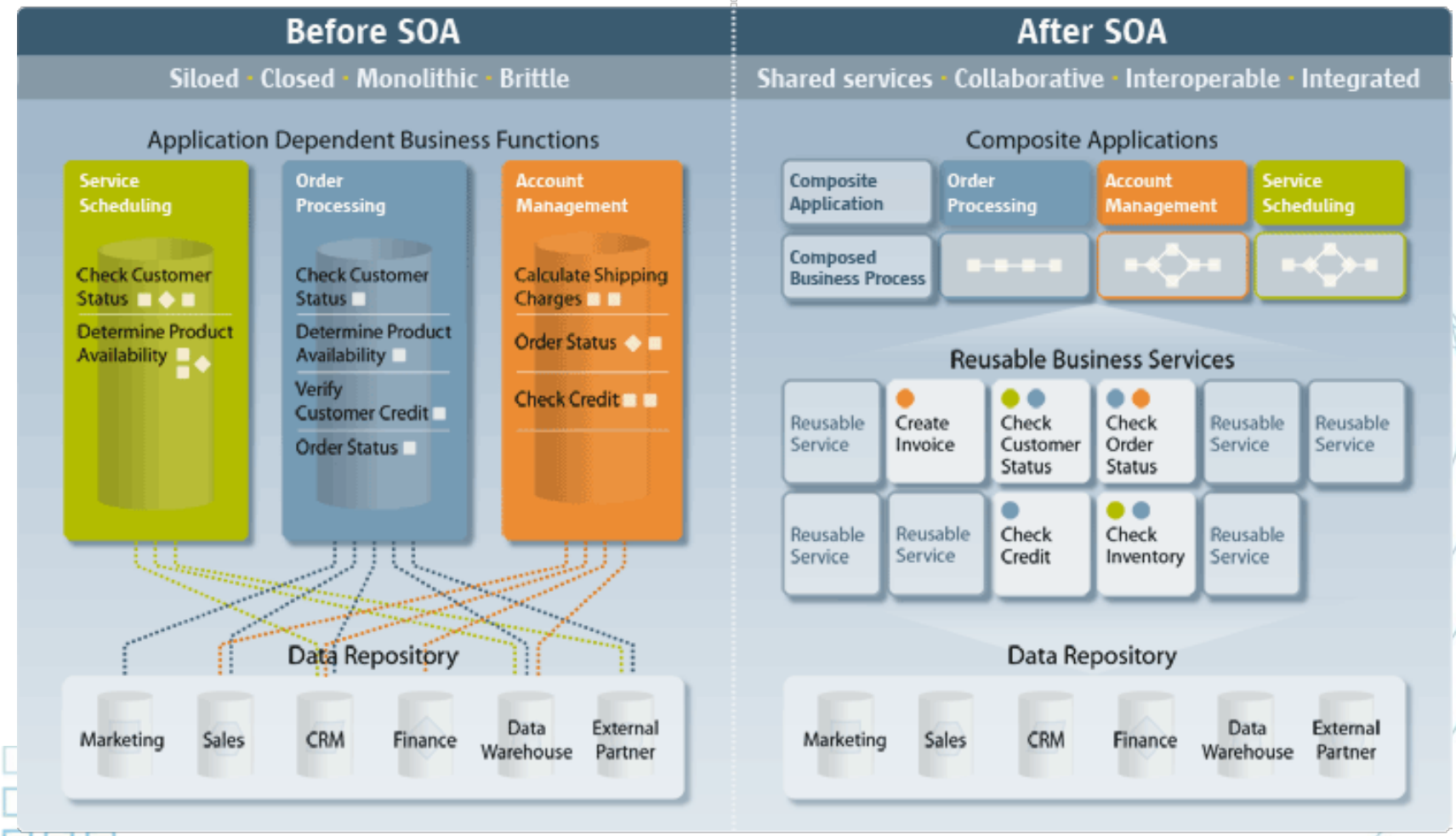
- **作为一种灵活、可扩展的体系结构框架，SOA具备以下优势**
  - 低成本：有效利用遗留资源
  - 灵活性：业务过程与应用的快速构造与重配
  - 盈利：有效重用已有业务、松耦合的IT服务，拓展新市场
  - 敏捷性：快速推出业务应用
  - 整合能力：在孤立的应用和组织间集成IT系统

SOA的革命性创造

- **强调划分可重用的服务模块，尽可能调用或组合已有的服务完成特定的业务并构造出软件系统**
  - “拿来主义”，“来之能战”
  - 服务的提供者与服务的使用者“松散耦合”
  - 每个服务可以并行开发，有助于开发效率提升
  - 每个服务可以单独运行和扩容，有助于保障软件质量
- **体现了软件开发方式的一种根本性的变化，可使业务环境变得更加灵活和强大：**
  - 以服务的形式提供独立的、可复用的、自动化的业务过程和功能
  - 通过快速组合与松散耦合来改善效率与生产率
  - 借助于开放的、强壮的、安全的基础平台，使企业能够快速向市场提供新的服务、快速的适应环境的变化

向SOA转型

- 业务与IT，逐层抽象和映射，相互对齐

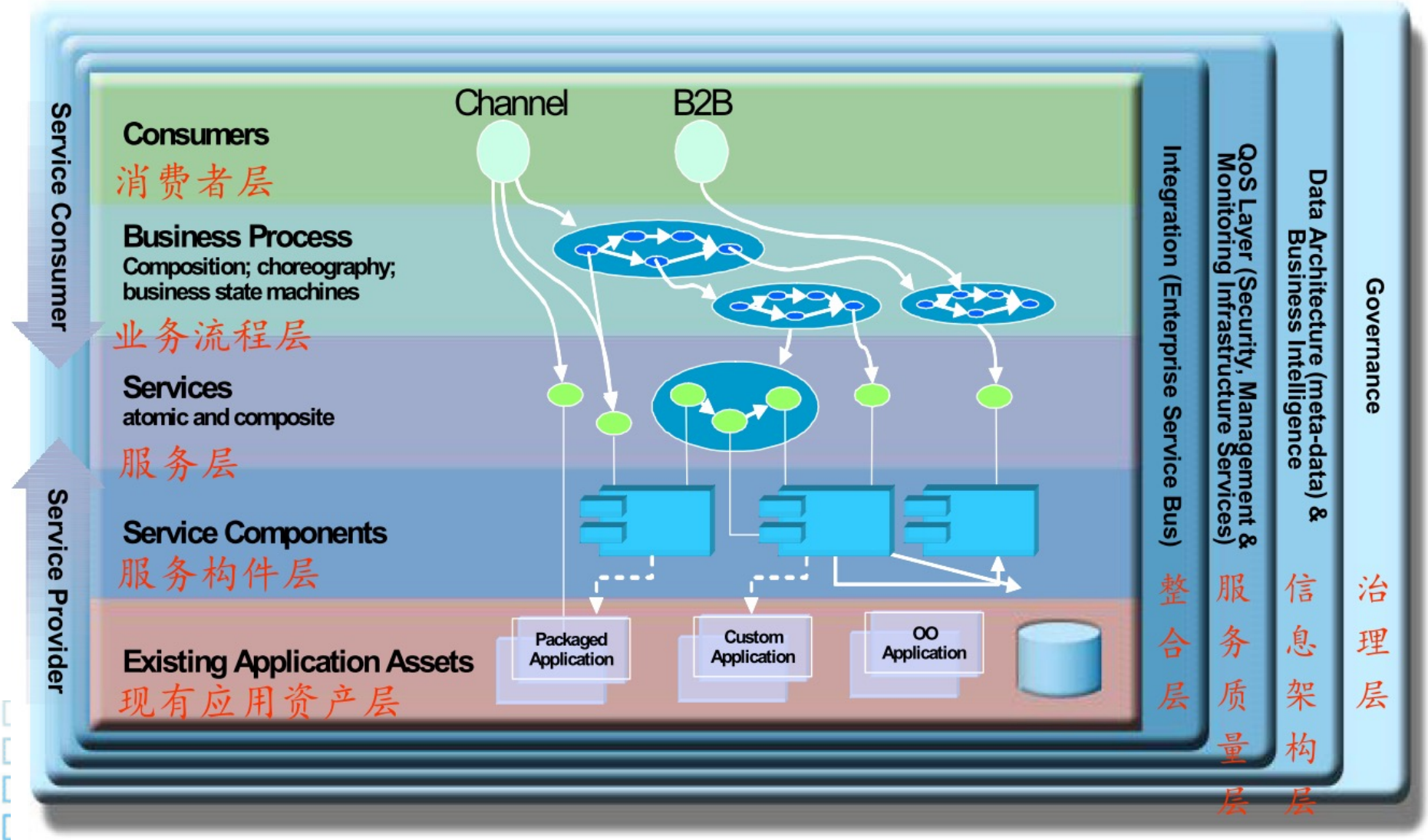


提纲

- 1. 软件体系结构简介
- 2. 什么是SOA
- 3. SOA参考架构
- 4. SOA描述语言
- 5. 微服务架构

SOA参考架构层次图

- SOA技术栈



SOA参考架构层次

- Layer 1: Existing Application Assets Layer 现有应用资产层
- Layer 2: The Service Component Layer 服务构件层
- Layer 3: Services Layer 服务层
- Layer 4: Business Process Layer 业务流程层
- Layer 5: Consumer Layer 消费者层

可弱化为手动实现

功能面

- Layer 6: Integration Layer 整合层 ESB(Optional)
  - 企业服务总线
- Layer 7: Quality of Service Layer 服务质量层
  - 安全、管理和监控的基础设施服务
- Layer 8: Information Architecture Layer 信息架构层
  - 数据架构（元数据）和商业智能
- Layer 9: Governance Layer 治理层

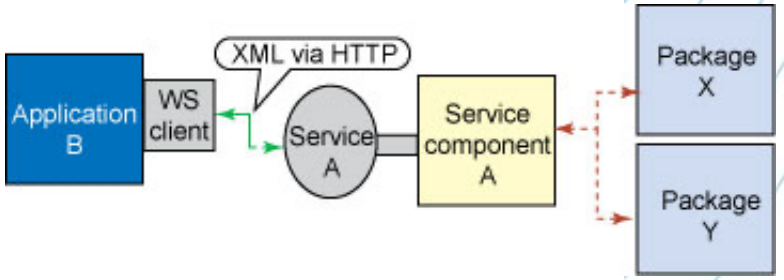
非功能面

层次1: 现有应用资产层

- **由已有的应用软件系统构成，包括所有定制或打包的应用资产，它们在IT环境下运行以支持业务活动**
- **有效利用已有的IT资产来实现SOA方案，可大大减少SOA实现方案的成本**
- **上述软件系统具体包括**
  - 已存在的自定义应用，包括JEE、.NET等应用
  - 遗留 (Legacy) 应用程序与系统
  - 已有的事务处理系统
  - 已有的应用和方案，如CRM、ERP等

层次2: 服务构件层

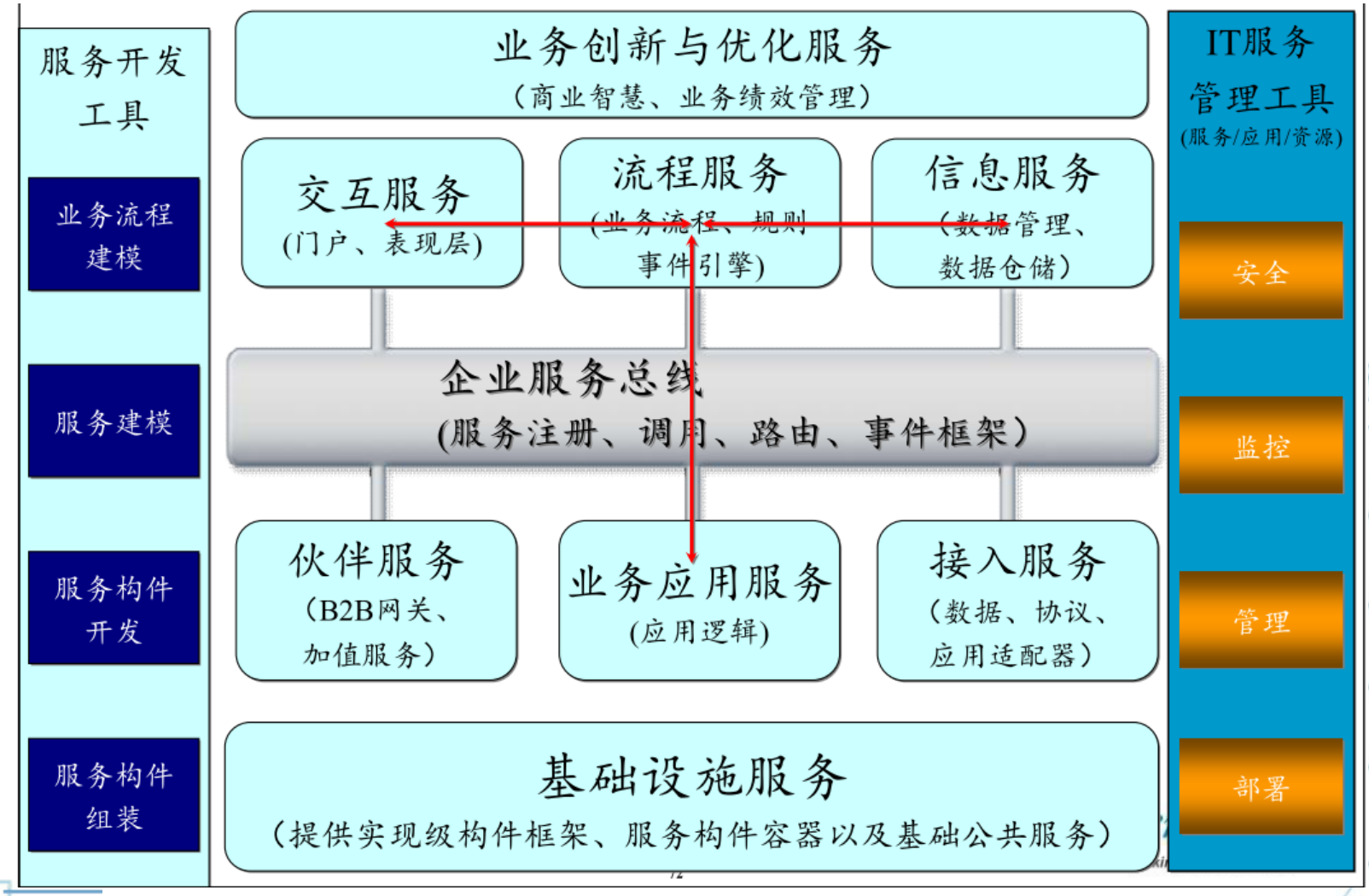
- 服务构件 (Service Component)：
  - 主要包含软件构件，反映了服务定义（功能和非功能）
  - 提供一个可信的服务实现的执行点
  - 符合服务层定义的服务契约，以确保服务质量和遵守服务层协议（SLA），保证IT实现与服务描述的一致性
  - 对客户隐藏实现细节，增强解耦能力，提高了IT的灵活性
- 是企业或业务单元层面上，受管理和治理的企业资产集合
- 使用基于容器的技术，如应用服务器，来实现构件、任务管理、高可用性、负载均衡等
- 服务构件的外观(façade)模式
  - 应用B仅与服务描述耦合
  - 服务提供者确保服务实现与服务描述的一致性
  - 服务实现与B独立
  - 服务构件A扮演了服务实现的façade，聚合可用的系统行为，为服务提供者给出了遵守服务的实现点



层次3: 服务层

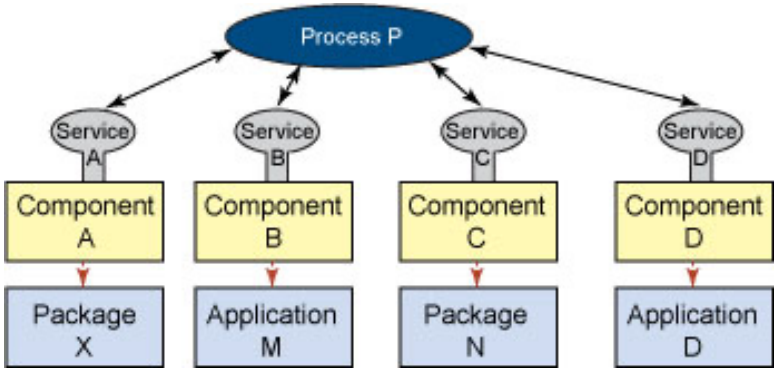
- **由界定在SOA之内的所有服务组成**
  - 服务被认为是功能独立、可网络访问的功能模块
- **提供了通过服务调用业务功能的详细信息，比如WSDL，可能包括:**
  - 策略(Policy)文档
  - SOA管理描述
  - 服务的依赖性(Dependency)的附件
- **该层服务可被发现、调用，或被编排为组合服务**

服务层的进一步细分



层次4: 业务流程层

- 定义服务层中服务的组合和编排，将一组服务组合或编排成一个流程



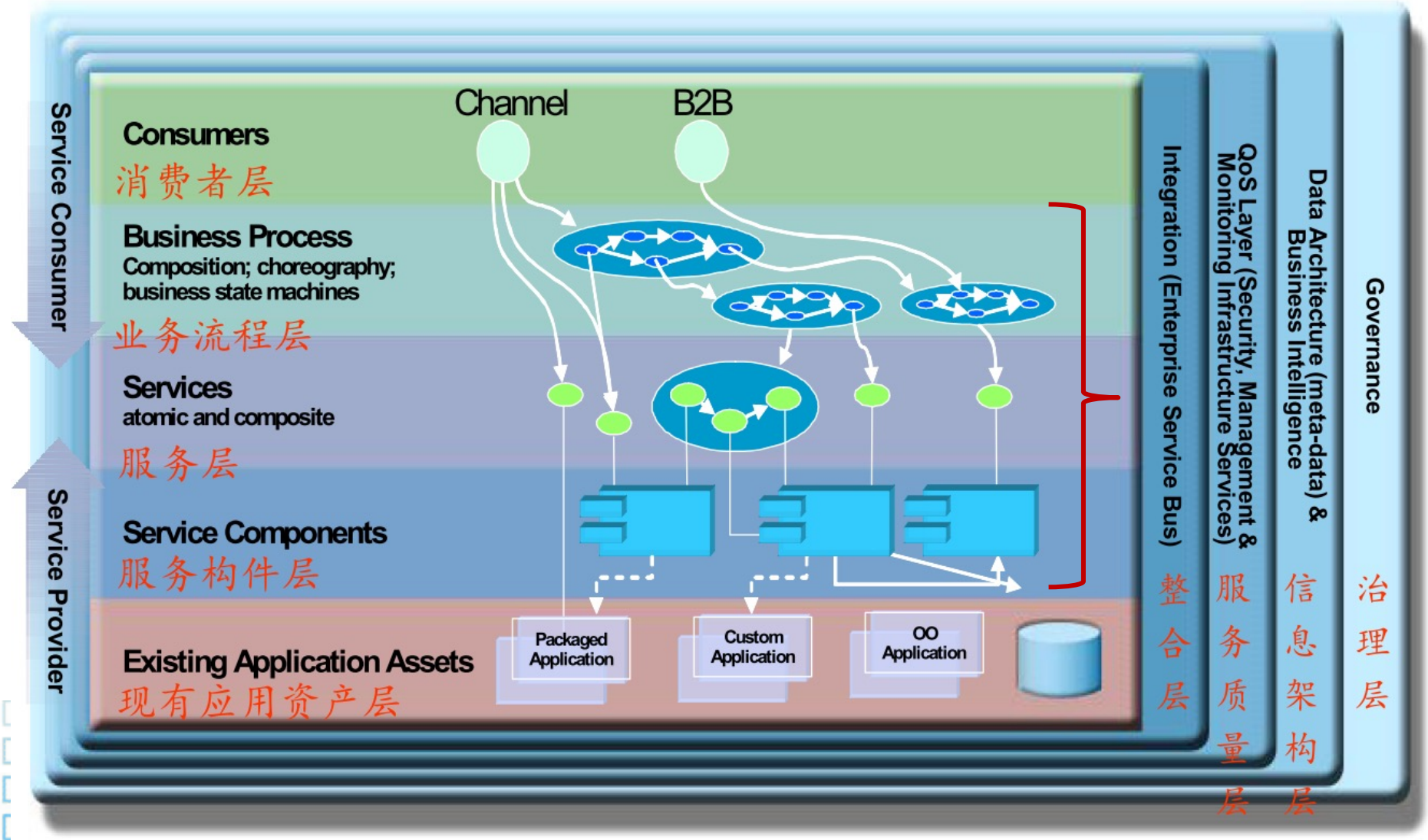
- 业务流程层涵盖：流程表示、组合方法、构建块(为了实现目标而聚合松耦合服务形成一个过程序列)
- 该层包括参与者（个人用户和商业实体）之间的信息交换，资源和流程的各种形式，以实现业务目标
- 负责业务流程编排的生命周期管理
- 与消费层进行输入和结果的通信，基于控制流或数据流的消息会通过整合层来路由、转换，而消息的结构通常由信息架构层定义
- 是SOA方案栈的中心角色，通过与整合层、QoS层、商业智能层、信息架构层、服务层协作，贯通了业务层需求和IT方案构件

层次5: 消费者层

- 也称表示层，提供交付IT功能和数据给最终用户所需的能力，提供应用与应用之间的接口
- 通过Channel、门户、富客户端（Mashups、Ajax、JavaScript）等技术，为业务流程、组合应用提供了快速**创建客户前端**的能力
  - Web Services for Remote Portlets (WSRP) Version 2.0
- 高效地开发前端应用、可维护性、可复用性

SOA参考架构层次图

- SOA技术栈



层次6: 整合层

- **原因**：传输协议、消息格式、服务描述多样化、差异大
- **需求**：需要一个关键SOA使能器，提供协调、路由和传输服务要求的能力，确保服务请求者能连接到正确的服务提供者
- **实现：整合层通过一套可靠的机制实现服务整合**
  - 服务消费者和提供者非直接交互，服务规范通过整合层间接发布
  - 消费者和提供者解耦合，方便地在方案中整合新的系统
  - 端点整合、智能路由、协议调解、其他转换机制等

层次7: 服务质量层

- **SOA的服务化特性需要实时管理计算系统的QoS**
  - 原因：增强的虚拟化、松耦合、广泛使用XML 、联邦服务的组合、异构计算基础设施、分布式的SLA
  - 需要 QoS测量指标来产生业务测量指标等
- **QoS层提供实现非功能性需求的机制**
  - 必须捕捉、监控、记录每个层次中不符合非功能需求的状态
  - 当发生QoS Violation，会发出告警信号、触发相关事件

层次8: 信息架构层

- 引入数据平面，管理**数据**、**元数据(Metadata)**、**服务质量**信息等, 构建数据仓库和信息架构

- 对于一些特定行业的SOA解决方案，信息架构层捕获所有跨行业和行业特定的数据结构，基于XML的元数据架构（例如XML概要）实现交换业务数据的商业协议

- 数据挖掘，大数据分析和建模 ，实现**商业智能**（Business Intelligence, BI)

层次9: 治理层

- 治理层涵盖所有方面的SOA业务运行生命周期管理，提供政策和指导以管理SOA解决方案的所有服务、SLA、容量、性能、安全性等
- 可应用到所有在SOA技术栈的其他层，尤其和服务质量层 天然连接
- 一个可扩展SOA治理框架通常包括：
  - 解决方案级基于QoS和KPI指标的服务水平协议(SLA)
  - 一套容量规划和性能管理政策
  - 解决方案级的面向联邦应用的安全指南
  - SOA性能监测的政策

重中之重：整合层

- **目标**：要在**分布式环境**下实现**跨域、跨组织**业务的交互与协同，独立存在的服务是没有意义的
- **挑战**：即使采用上面的service integrator，一个组织中已存在和正在使用的**服务数量巨大**，服务间**调用交互关系复杂**
- **需求**：将多方提供的服务集成在一起，并试图构造一种通用的**服务基础设施**来管理它们

企业服务总线

- **ESB（Enterprise Service Bus):** 一种构建、部署企业SOA的新方法，简化Web服务部署，解决集成问题
- 基于标准的高速链路(Backbone)，利用MOM (面向消息中间件, Message Oriented Middleware)连接异构系统, 综合不同类型中间件的特征
- **主要特征**
  - **Web服务**：SOAP, WSDL, UDDI
  - **消息**：异步store-and-forward方式的分发
  - **路由**：发布/订阅，基于内容，路线itinerary
  - **平台中立：** 连接企业中的任何资源：Java, .NET, 遗留应用，WS-*，…

企业服务总线(ESB)

- 企业服务总线(Enterprise Service Bus)是一个整合应用和服务的灵活的连接基础设施

ESB在请求者和服务之间实现了：

- 路由服务间的消息
- 转化请求者和服务之间的传输协议
- 转换请求者和服务之间的消息格式
- 处理分布式服务资源之间的业务事件

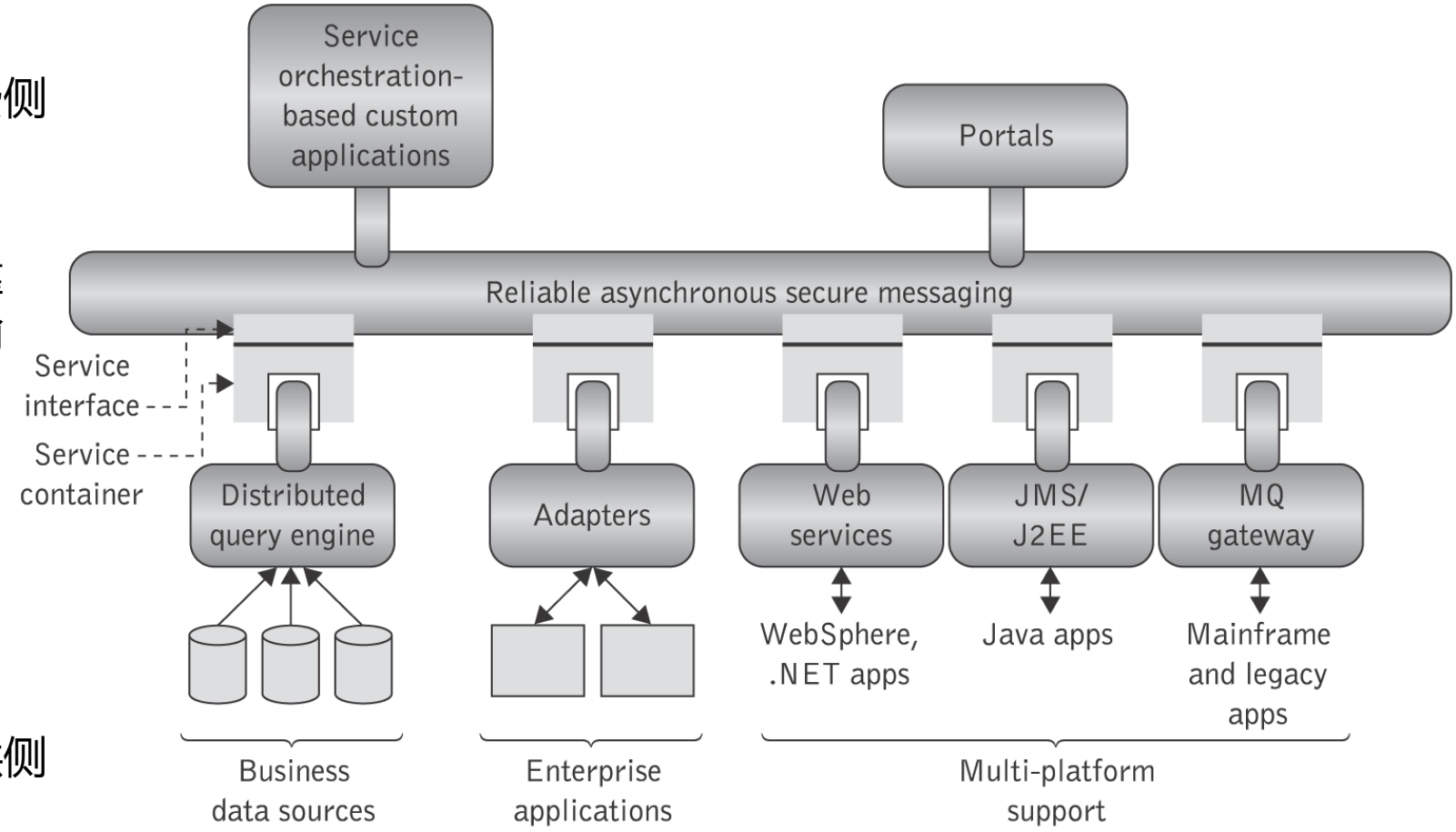
形状 = 协议;　颜色 = 数据类型

ESB体系结构

服务消费侧

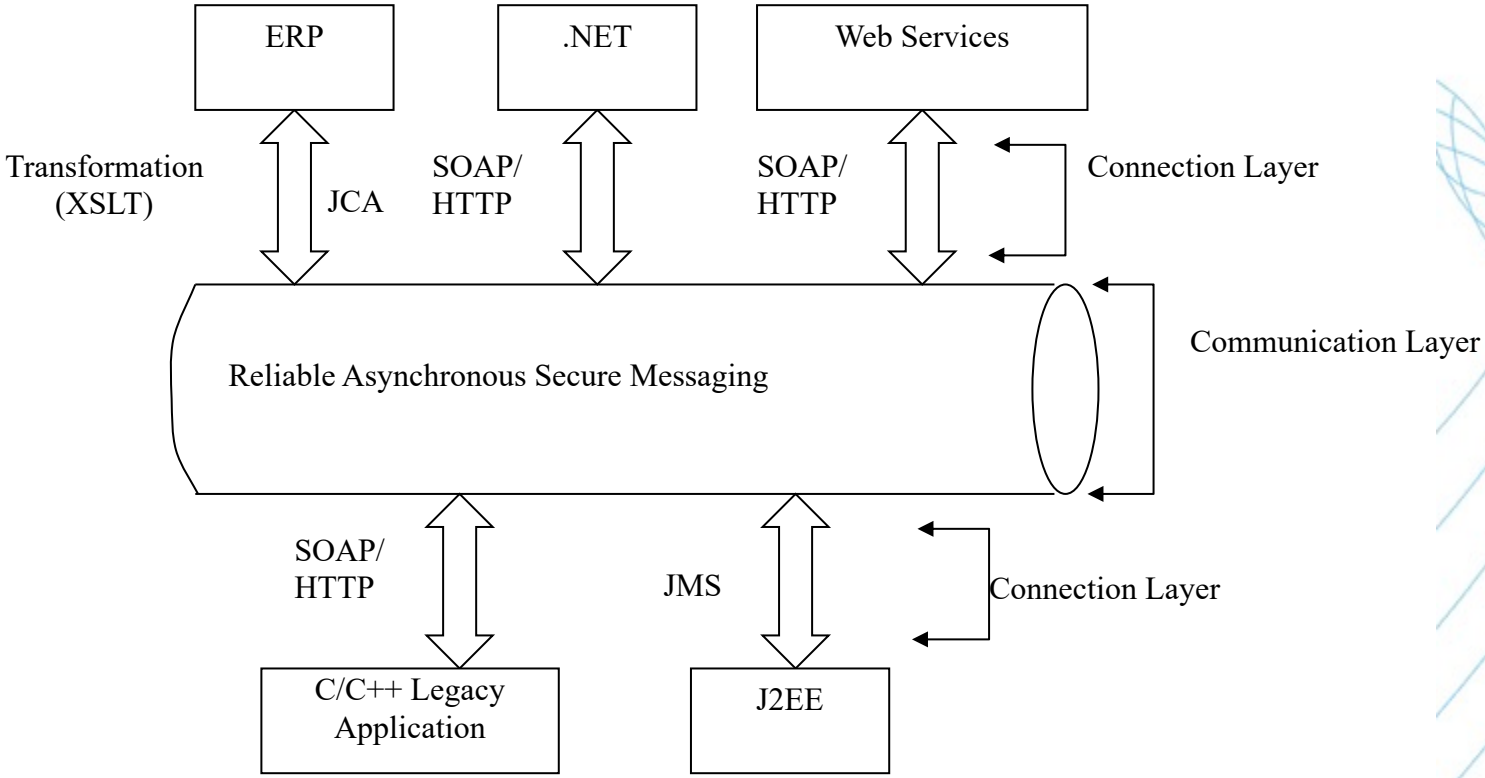
安全可靠
消息传输

服务提供侧



ESB体系结构

支持了不同消息协议转化、不同数据格式的转换



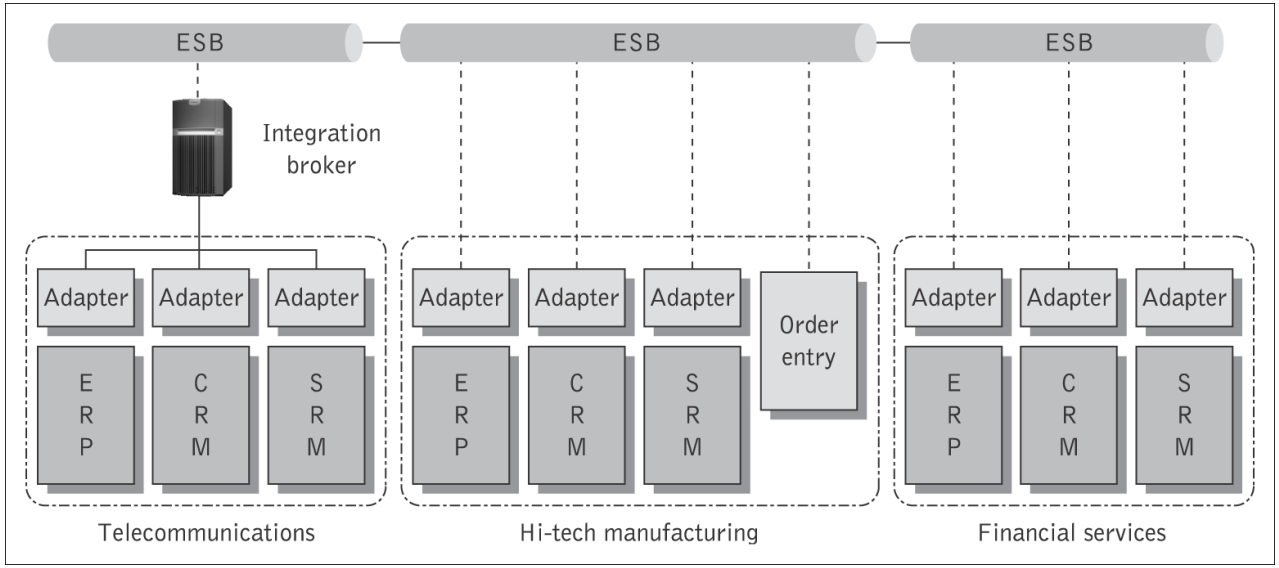
ESB的功能及相关标准

- 有效利用遗留资产
- 服务通信：在服务交互的不同协议间路由，并转换消息格式
- 动态连接：动态连接Web服务的能力，无需使用单独的静态API或代理
- 基于主题和内容的路由
- 支持多QoS的端点发现
- 集成和转换
- 安全性
- 长事务支持
- 管理和监控
- 可扩展性

Functional area	*Capabilities*	*Relevant standards*
Connectivity	Transport Guaranteed delivery Routing	SOAP WS-ReliableMessaging WS-Addressing
Content-based routing and event notification	Content-based routing Topic-based routing Business event notification	XPath WS-Topic WS-Notification
Transformation	Protocol transformation Message transformation Data transformation	XSLT WS-Addressing WS-ReourceFramework
Service enablement	Wrapping Transformation Access to legacy resources	WSDL BPEL
Service orchestration	Process description Process execution Long-running processes	BPEL
Transaction management	Transactional services Coordination	WS-Transaction WS-Coordination
Security	Authentication Authorization Access rights Encryption	WS-Security WS-SecurePolicy
Discovery with multiple QoS	Run-time service discovery using multiple QoS capabilities and policies	WS-PolicyFramework
Management and monitoring	Monitoring QoSs Enforcing SLAs Controlling tasks Managing resource lifecycles	WS-DistributedManagement WS-Policy

ESB拓扑结构

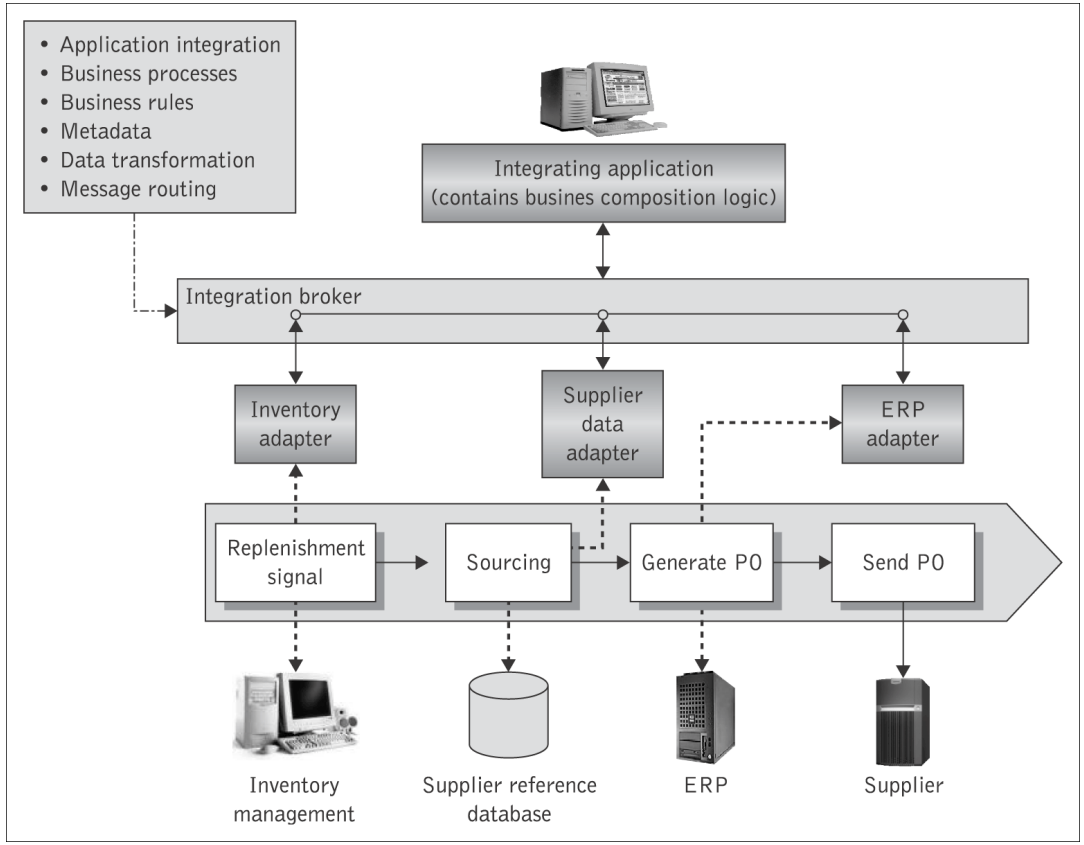
- ESB实现方法：单中心的ESB、多个ESB的联邦



- ESB需要中间件设施集合，来支持以下体系结构风格
  - 面向服务的体系结构
  - 消息驱动的体系结构
  - 事件驱动的体系结构

ESB元素: Integration brokers

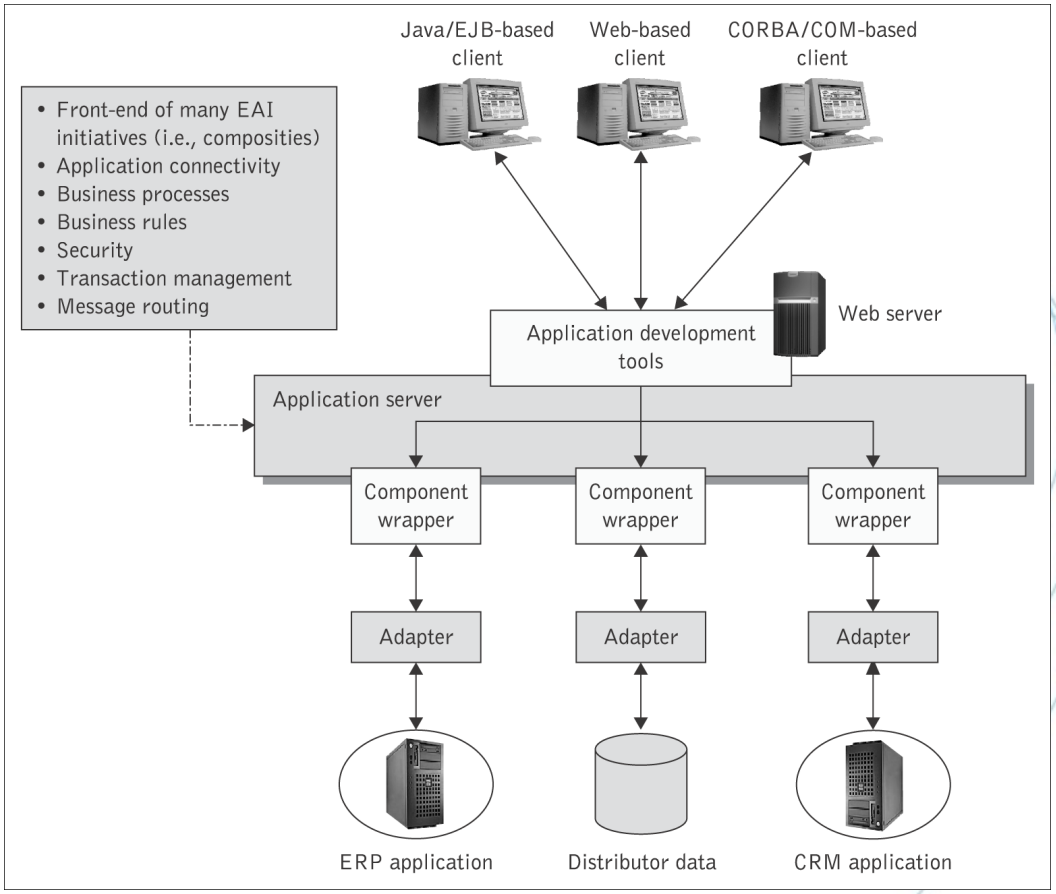
- 通过 broker 代理应用或者系统之间点对点的交互与程序通信
- 简化多个后端信息系统间的信息移动，解决应用语义和异构平台的差异



ESB元素: 应用服务器

- **Application Server为开发和部署分布式Web/非Web应用，提供集成开发环境**

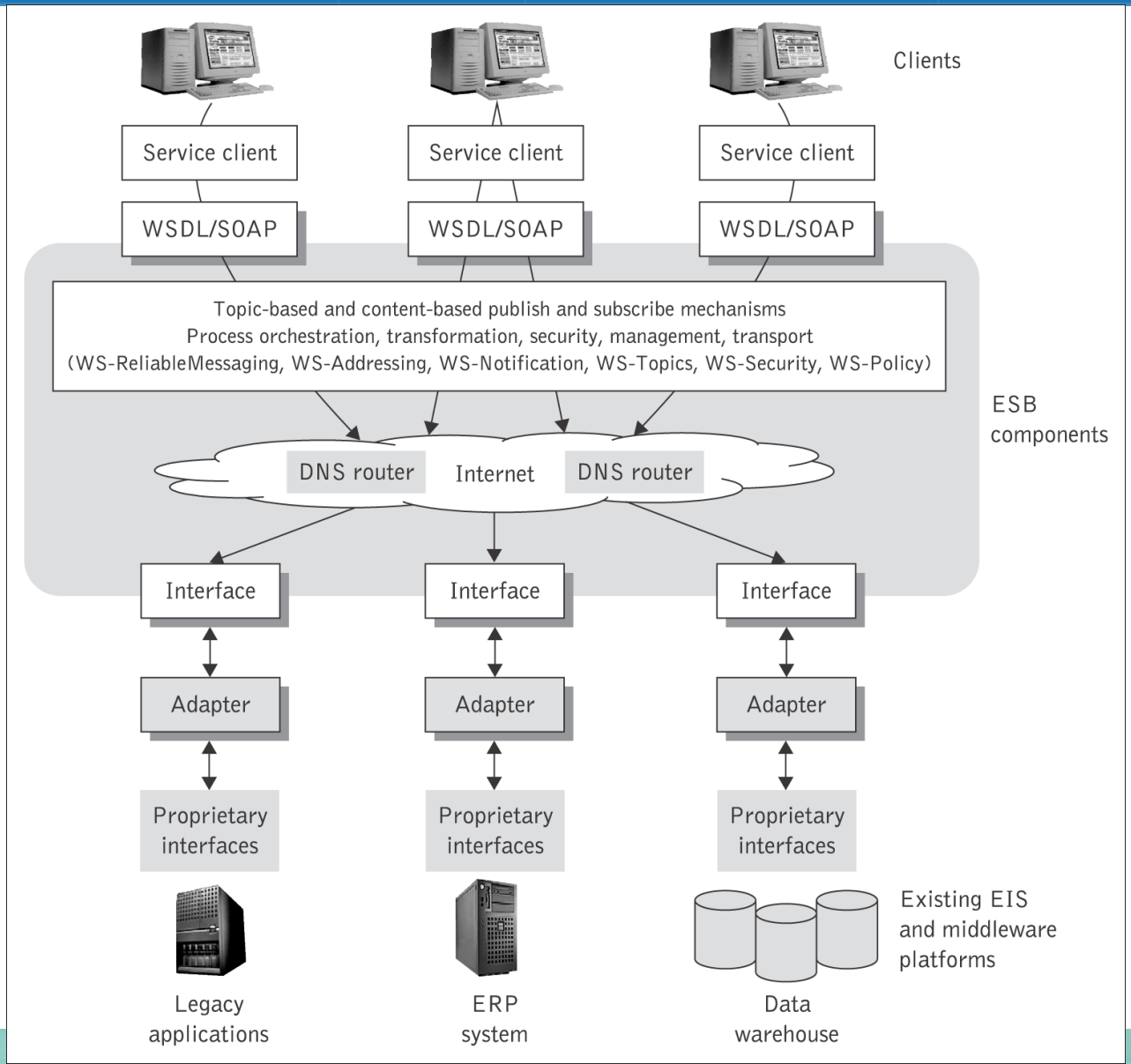
An application server typically provides Web connectivity for extending existing solutions and brings transaction processing mechanisms to the Web.



ESB元素: 业务过程管理

- **Business Process Manager (BPM)**
- 控制跨企业、跨应用的长期、多步信息请求或事务过程, 监控过程实例的状态
- BPM软件方案提供ESB环境下基于工作流的业务过程、过程分析和可视化
  - 允许业务过程与底层集成代码的分离
  - 允许过程编制引擎(BPEL)架构在ESB之上

利用遗留资产

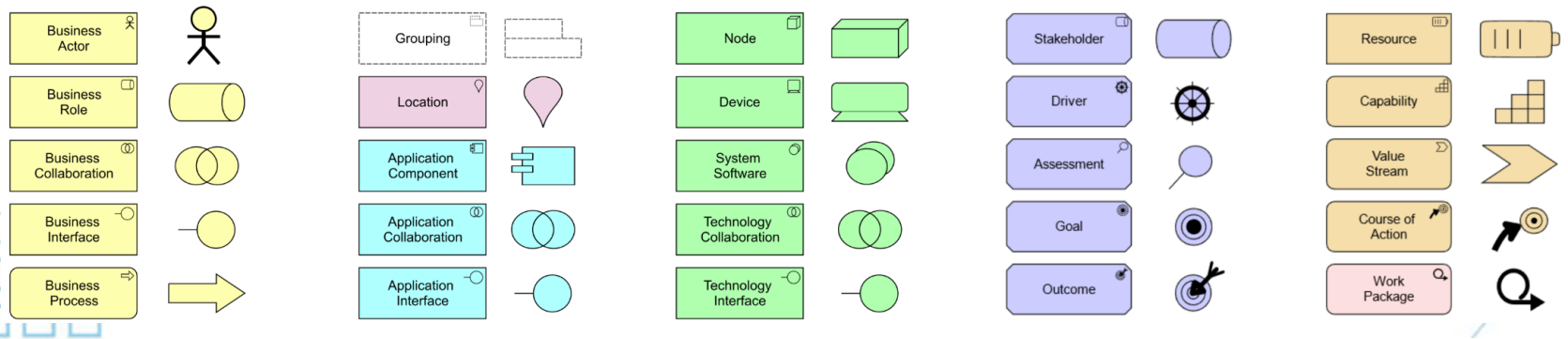


提纲

- 1. 软件体系结构简介
- 2. 什么是SOA
- 3. SOA参考架构
- 4. SOA描述语言
- 5. 微服务架构

SOA描述语言：SoaML

- **OMG的标准规范**
- **OMG SoaML**
  - Service oriented architecture Modeling Language
  - UML Profile and Metamodel for Services
  - https://www.omg.org/spec/SoaML/1.0.1/PDF
- **开发工具Modelio**
  - http://www.modeliosoft.com/



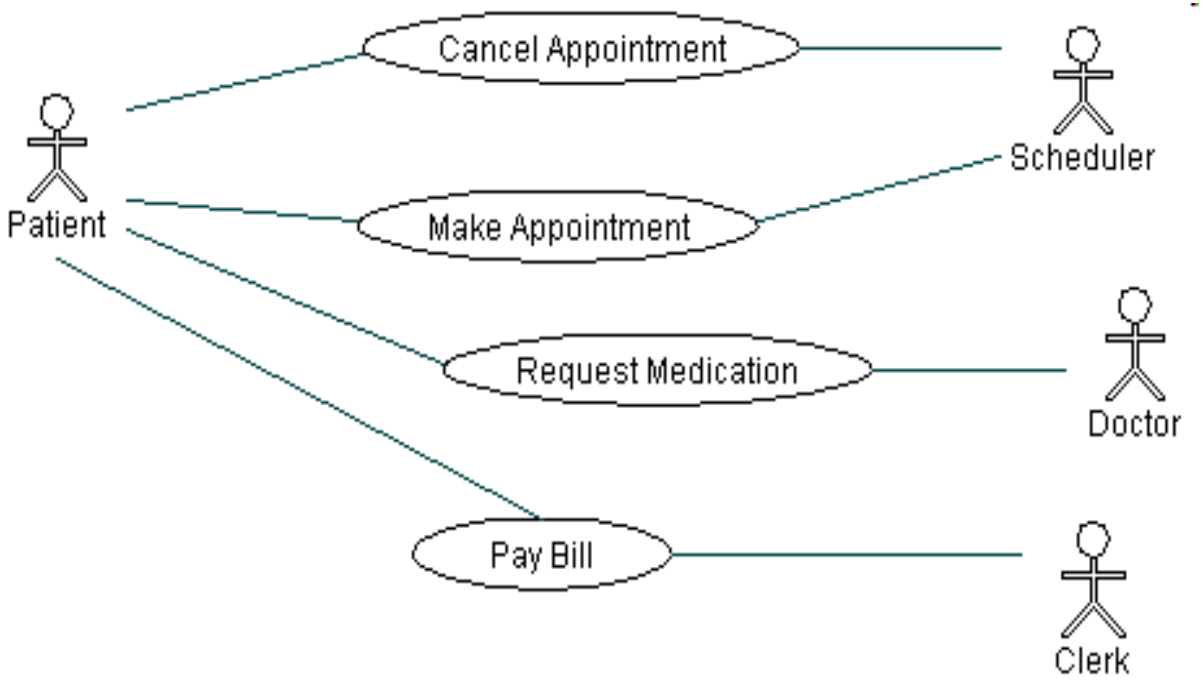
SoaML的目标

- 直观、完整地支持用UML进行服务建模
- 支持多个参与者之间双向异步服务
- 支持SOA，参与者提供、使用服务
- 支持服务的组合
- 方便将建模内容映射到业务过程规范
- 与UML、BPMN等规范兼容
- 直接映射到Web服务
- 支持自顶向下、自底向上和中间汇合的建模方式
- 基于Contract契约的设计、服务动态适应
- 规定服务能力和Contract
- 不对UML做修改

UML

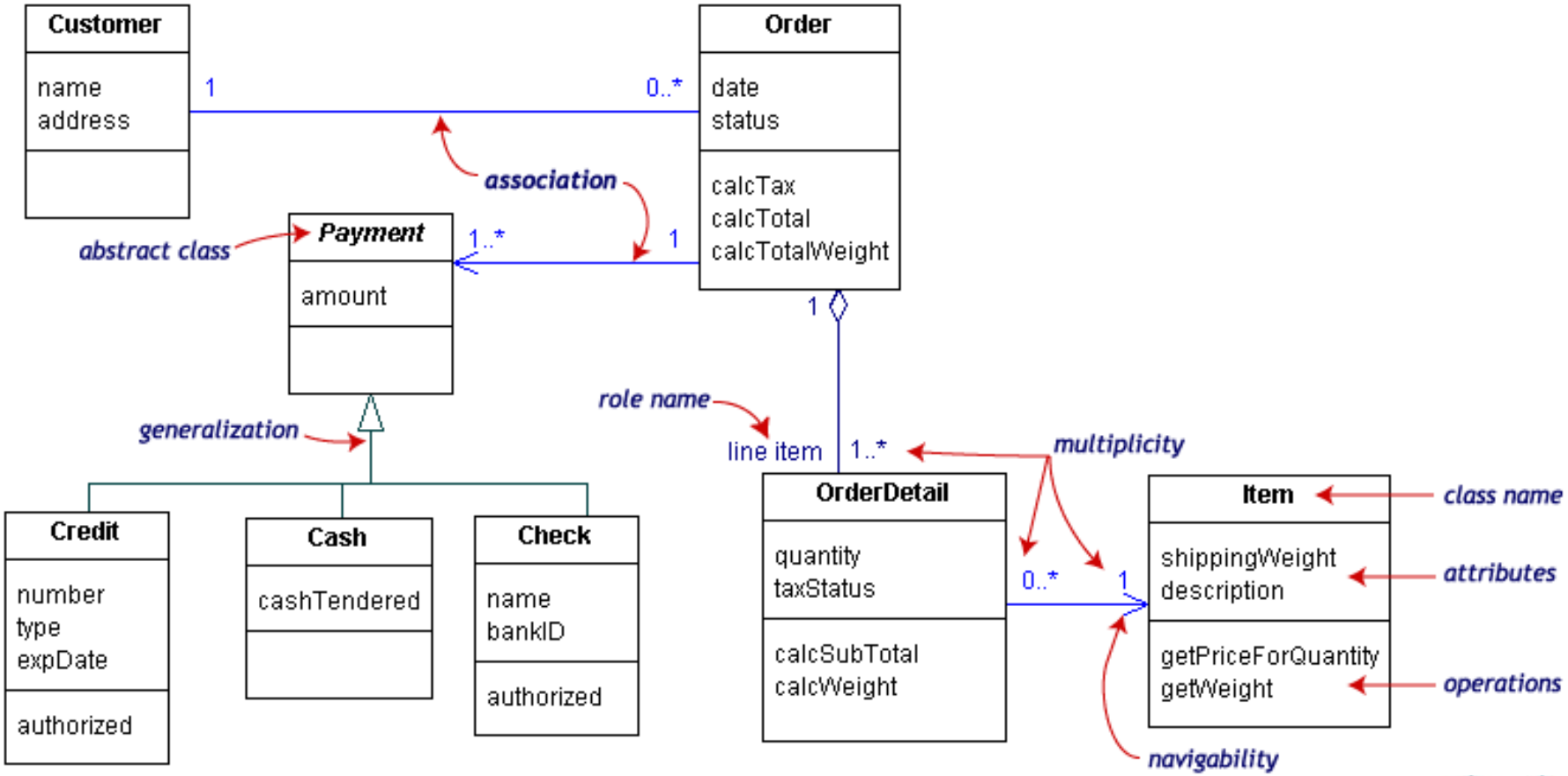
- 结构建模：
  - 类图
  - 对象图
- 行为建模
  - 用例图
  - 交互图（顺序图、协作图）
  - 活动图
  - 状态图
- 体系结构建模
  - 构件图
  - 实施图

Use Case Diagrams



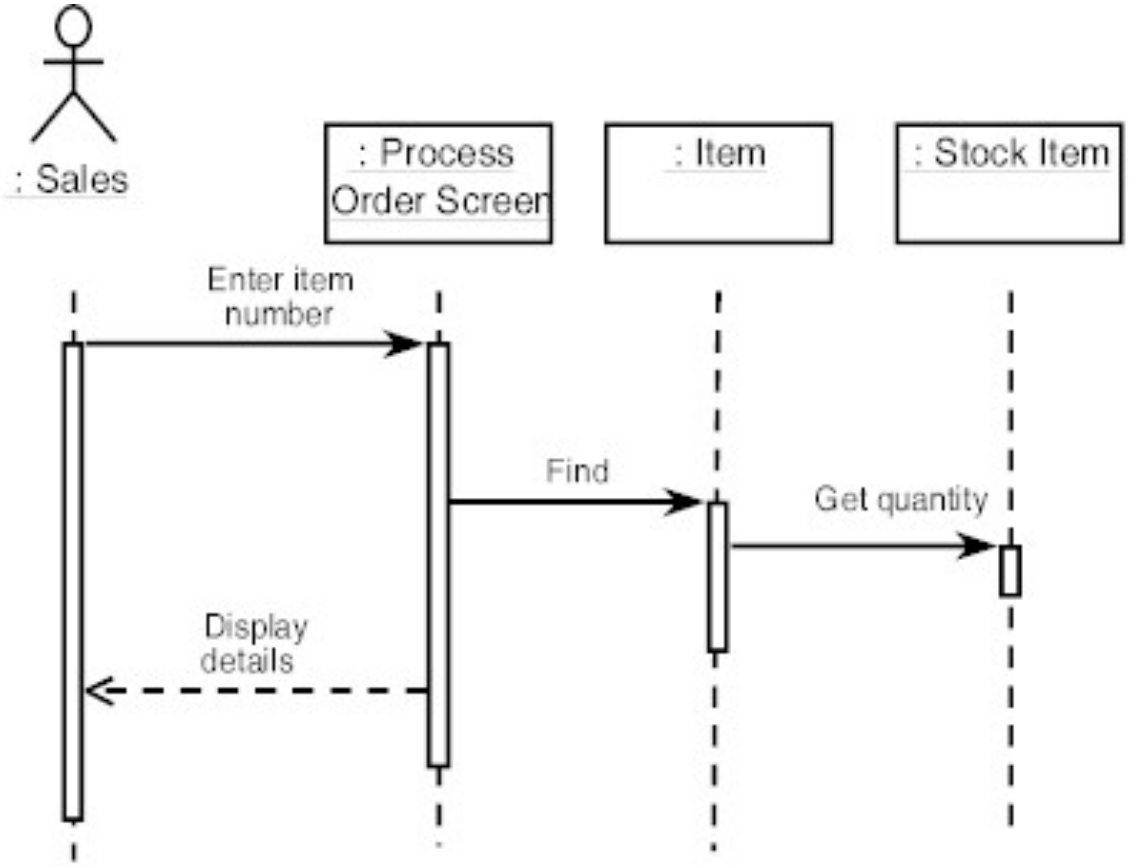
Use Case diagrams describe what a system does. HE ASSISTANT simply had 1 actor, but generally, we have multiple actors.

Class Diagrams

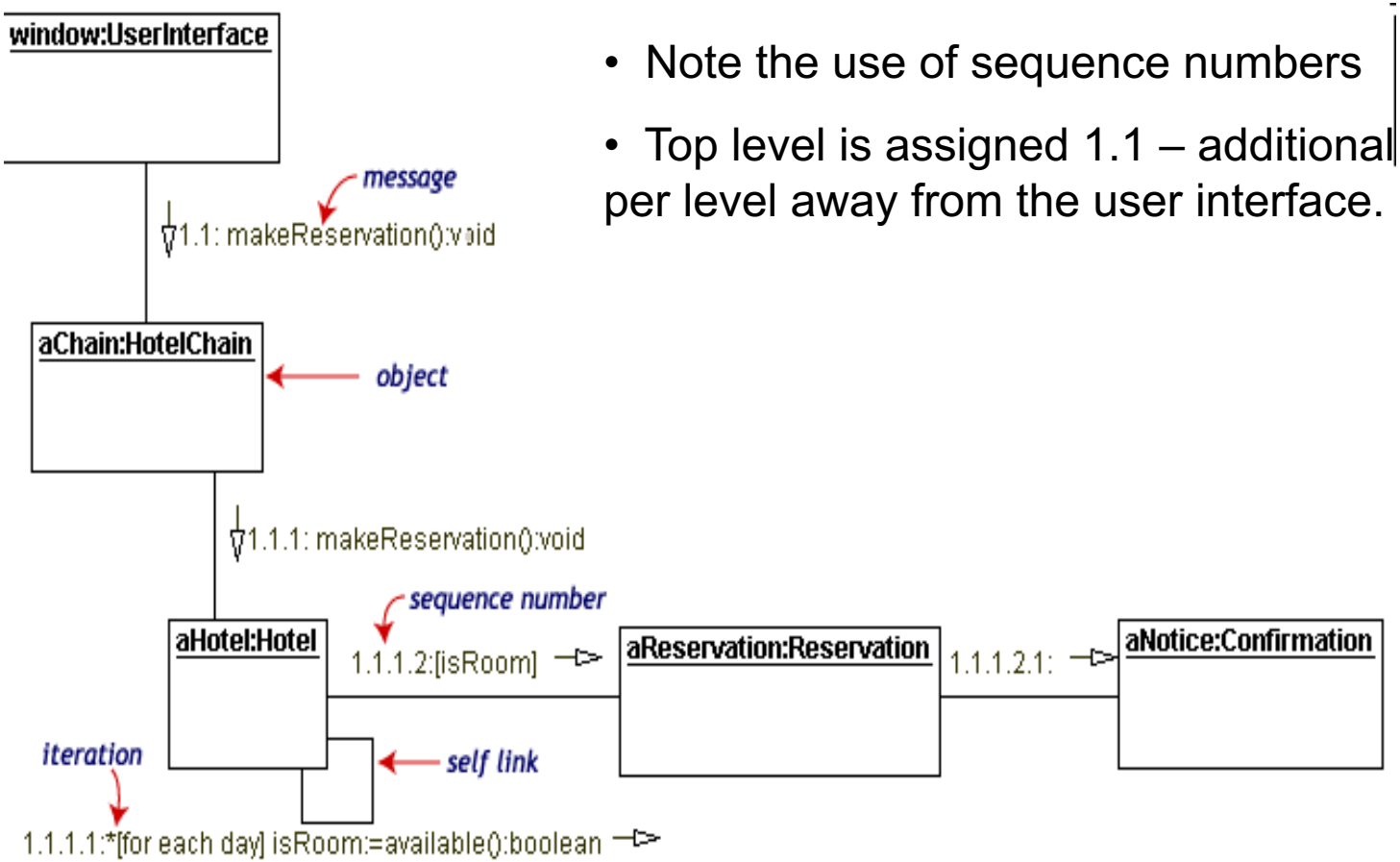


Abstract classes, interfaces, association (关联关系) generalization (泛化、继承关系 inheritance), aggregation (聚合关系 e.g., orderDetails make up order)

Sequence Diagrams



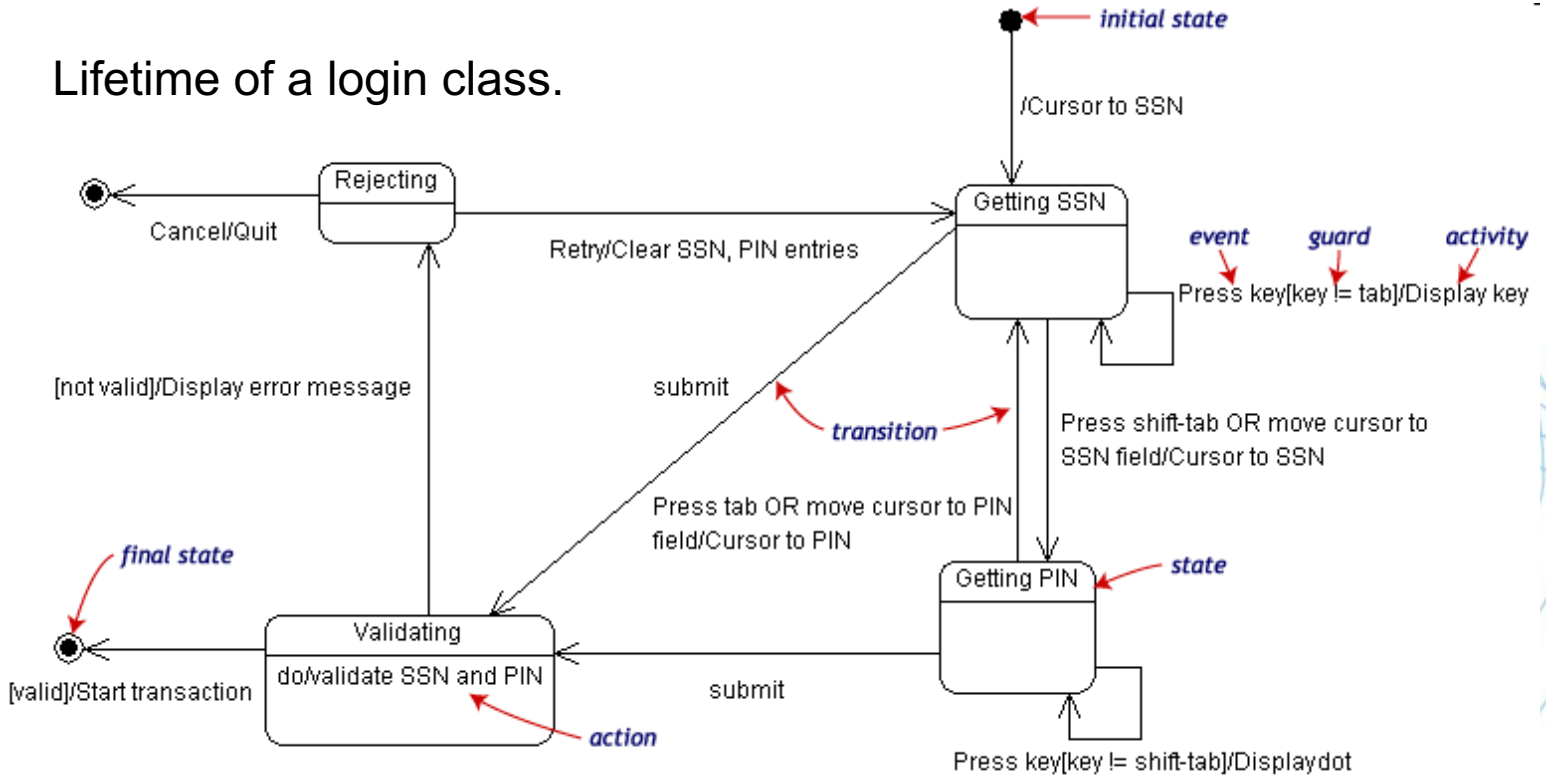
Collaboration Diagrams



- Note the use of sequence numbers
- Top level is assigned 1.1 – additional dots per level away from the user interface.

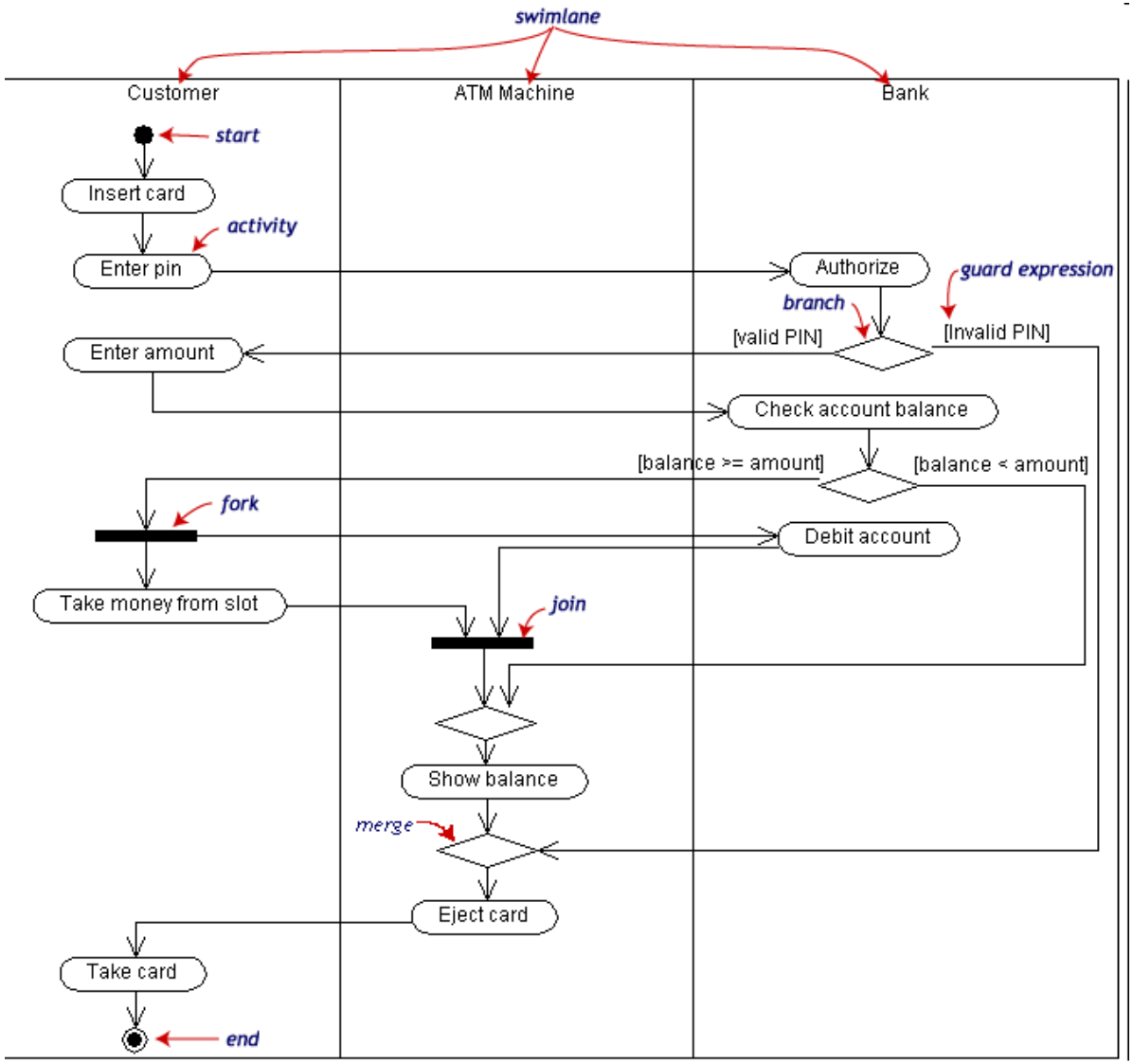
State Diagrams

Lifetime of a login class.

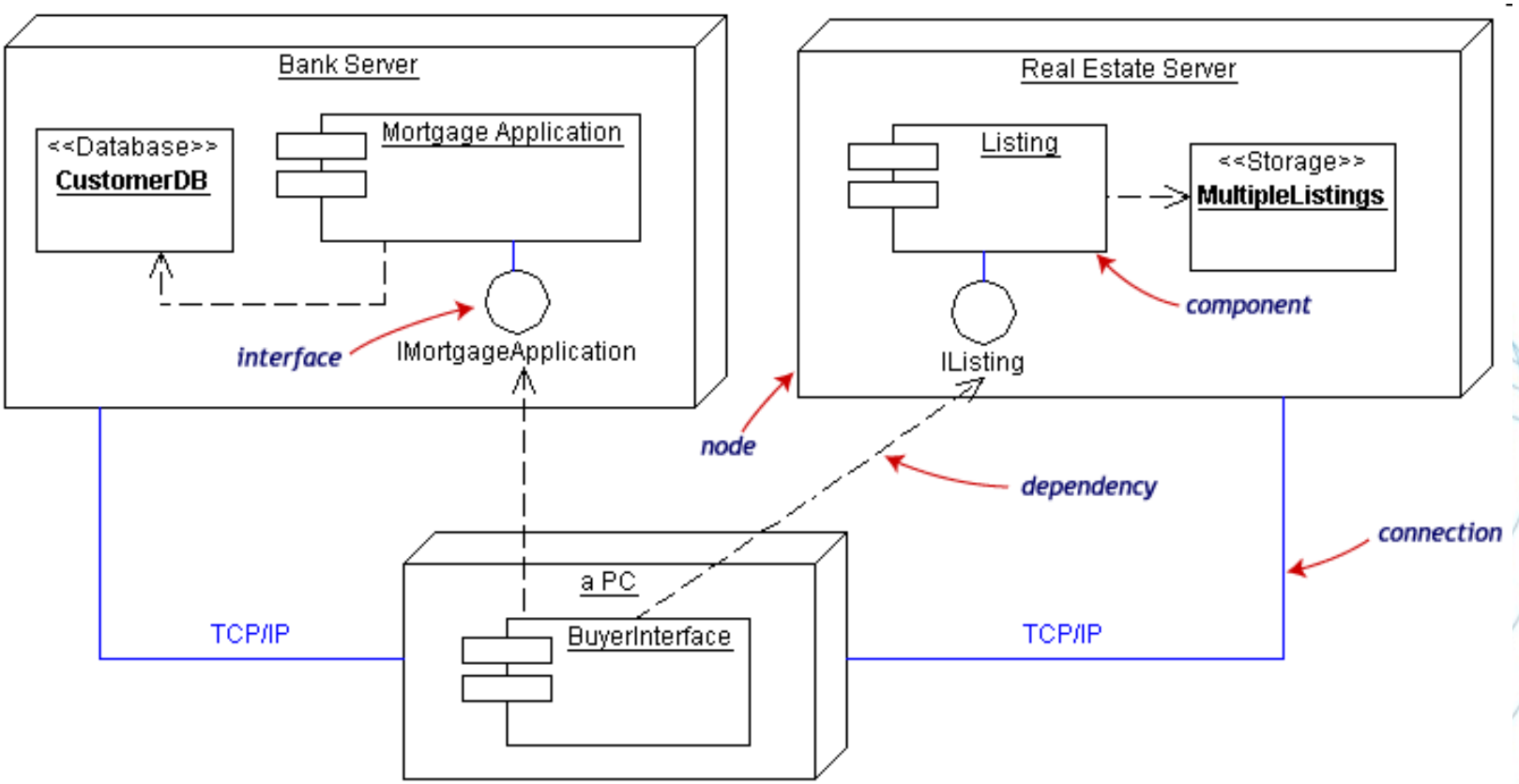


- Show the state of a particular object throughout the system lifetime.
- Not necessary for all objects – just the critical ones.
- Super states – can be used to nest states to make diagram easier to read.

ACTIVITY DIAGRAMS



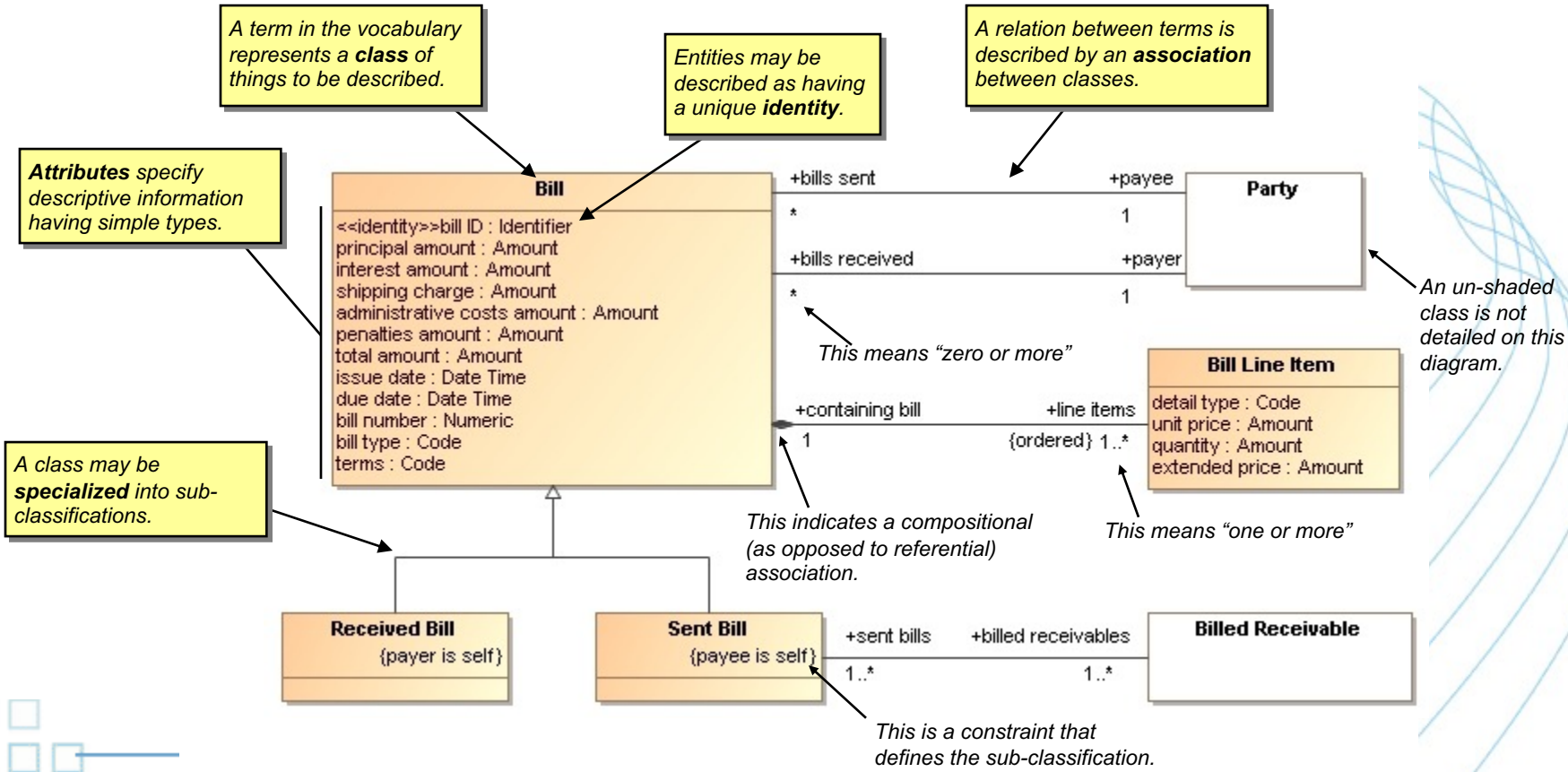
COMPONENT AND DEPLOYMENT DIAGRAMS



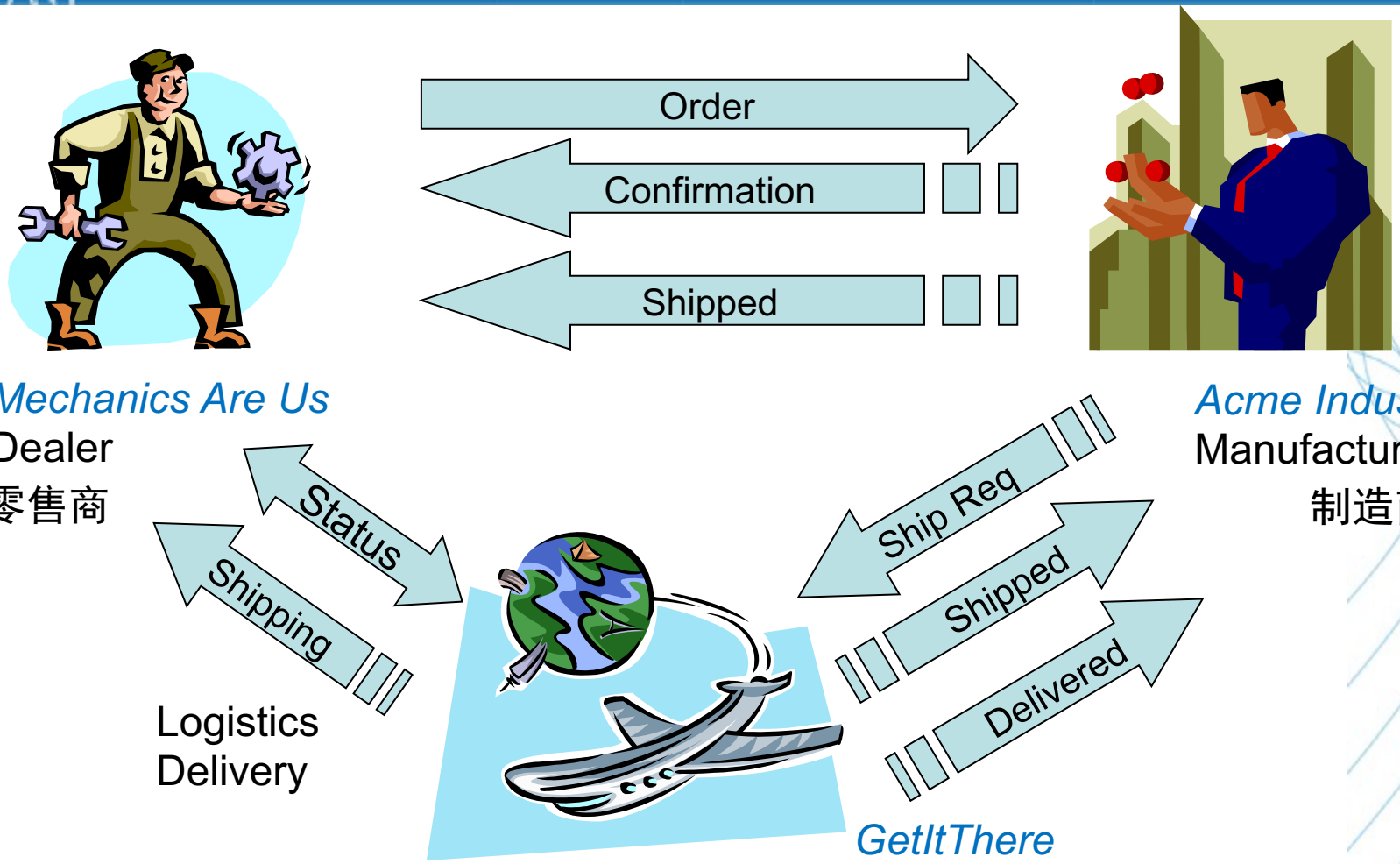
High level component communication and dependencies.

Information Model

For Messages and Entities



SOA例子：Supply Chain供应链



- 每个角色都是参与者，提供服务或者消费服务

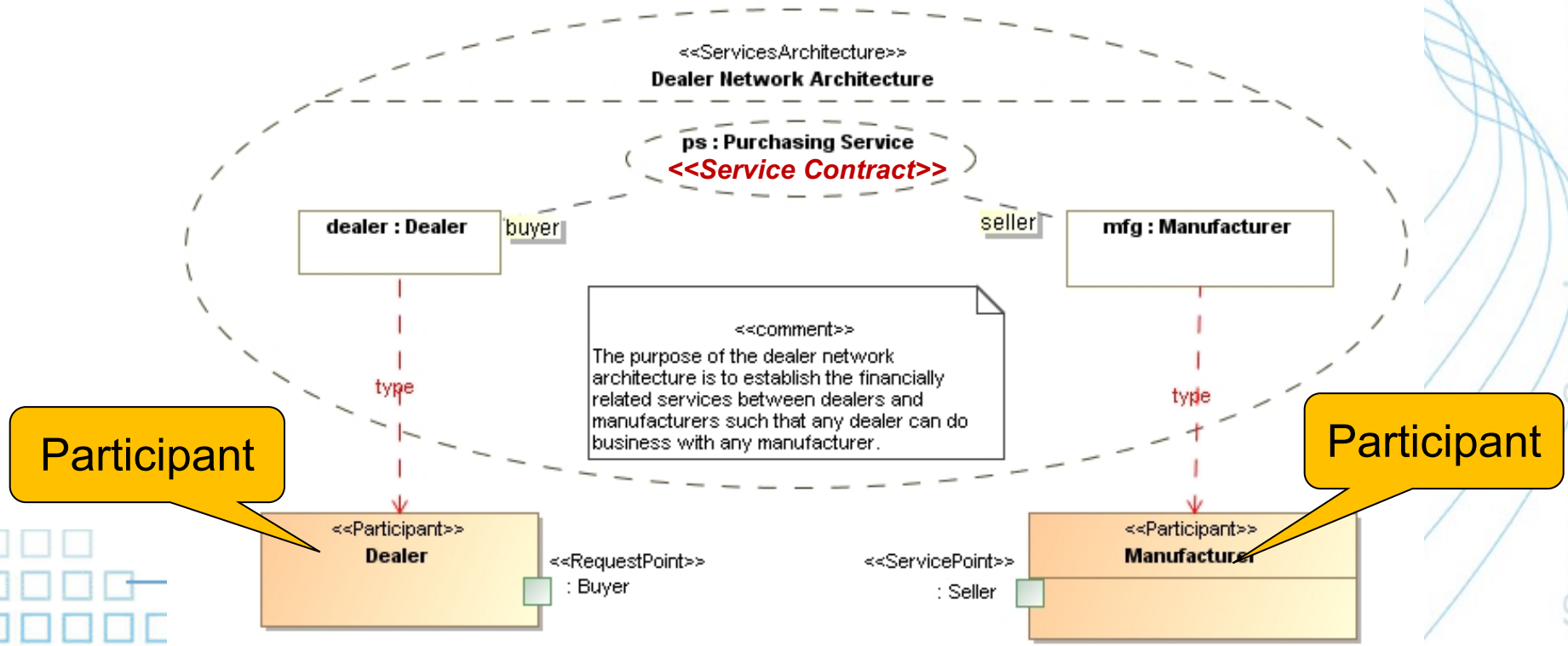
SoaML 重要概念

- **Participants**
  - Specific entities or kinds of entities that provide or use services
- **Ports**
  - Participants provide or consume services via ports.
- **Service Description**
  - The description of *how the participant interacts to provide or use a service* is encapsulated in a specification for the service.
  - Three ways to specify a service
    - **UML Interface:** [one-way] The participant receives operations on this port and may provide results to the caller
    - **ServiceInterface**: [bi-directional] specifies the interface that the provider offers and the interface, if any, it expects from the consumer. The ServiceInterface is the type of a "Service" port on a provider and the type of a "Request" port on the consumer.
    - **ServiceContract**: A service contract approach defines service specifications (the service contract) that specify how providers, consumers and (potentially) other roles work together to exchange value.

https://www.omg.org/spec/SoaML/1.0.1/PDF

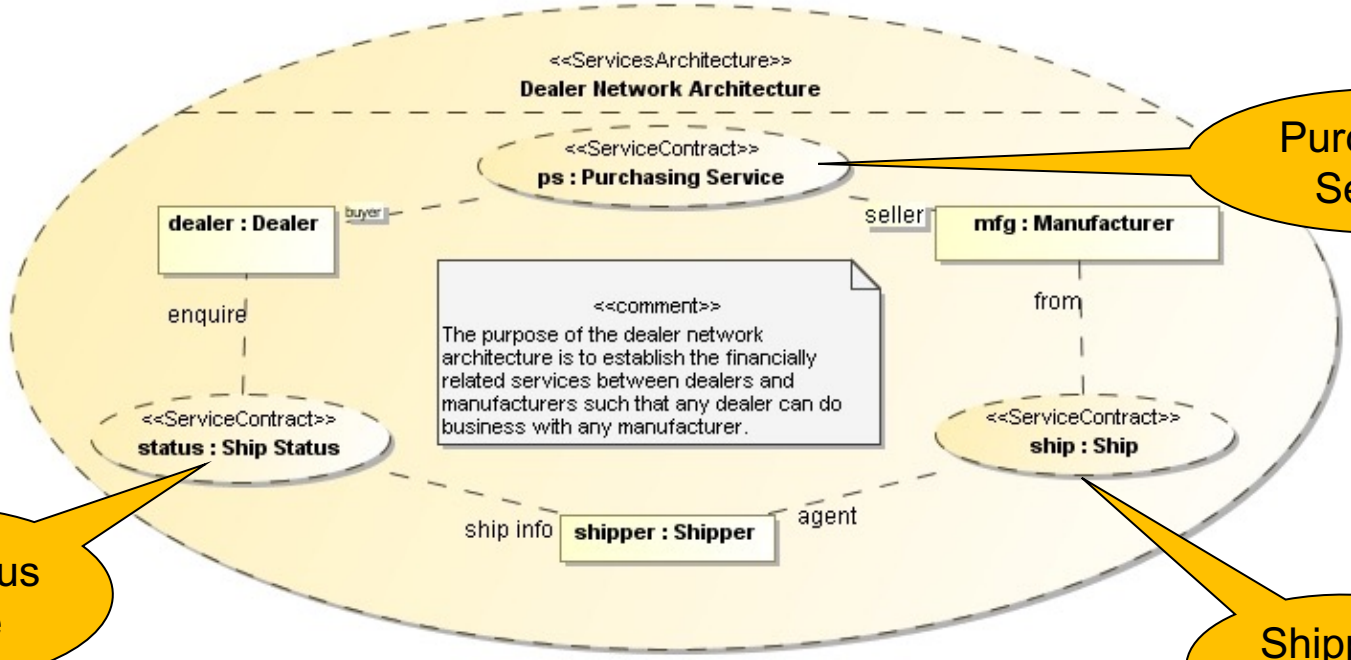
Participant (参与者)

- Participant表示逻辑或真实的人/组织单位，他们参与到services architectures 或业务过程中
- SoaML中，参与者通过定义外部contract （契约/合约)，提供 and/or 消费服务



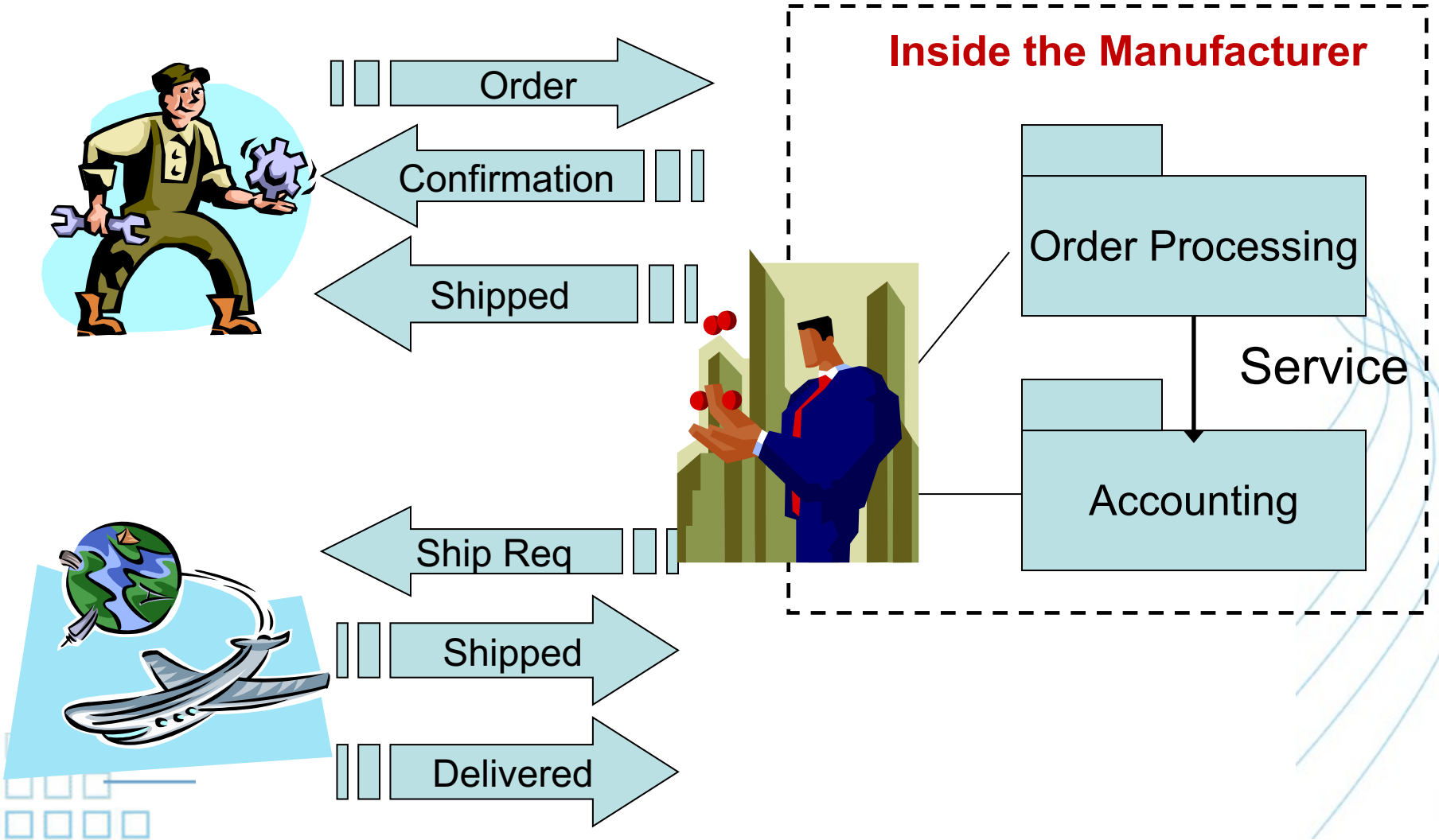
Services Architecture 服务体系结构

- Services Architecture是一组参与者形成的参与者角色网络
- 定义多个服务契约来规定参与者之间如何交互，参与者如何提供/消费服务
- SOA定义了参与者类型和完成角色的服务实现, 描述参与者如何共同工作
- 可通过UML协作图定义一个组织的Services Architecture



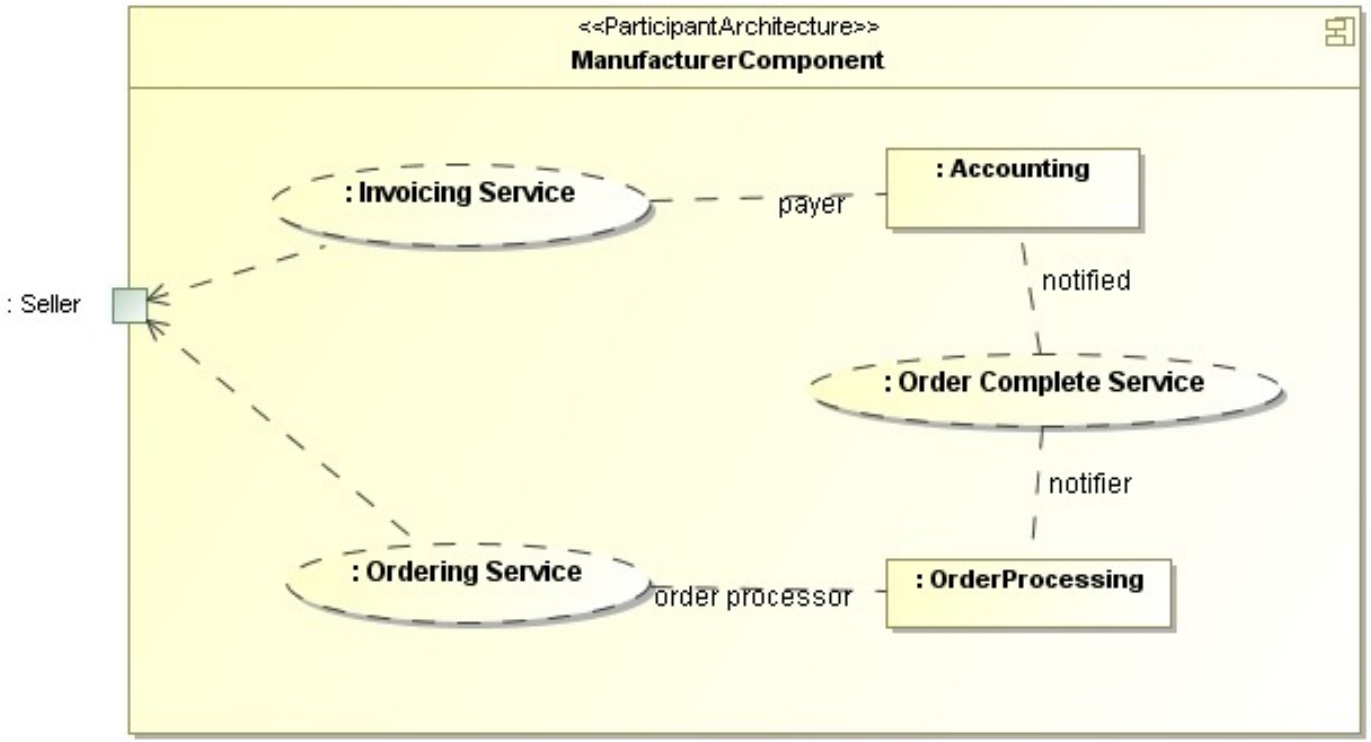
- 3个参与角色，3个服务契约

具体某个Participant业务流程



Participant Architecture 参与者体系结构

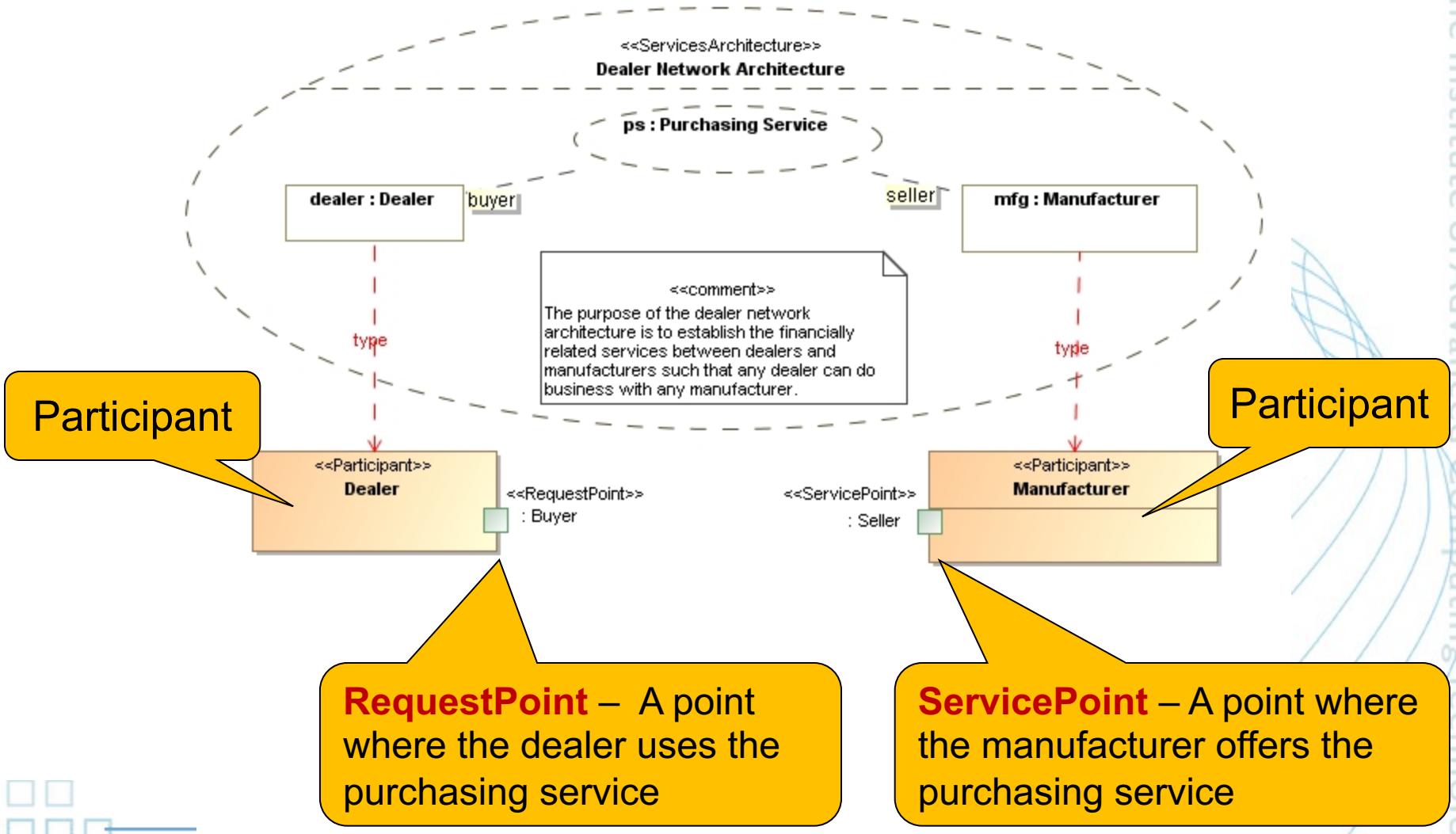
- Participant Architecture 是一种高层的services architecture
- 通常定义某个参与者的具体业务过程
- 定义一组内/外部参与者如何使用服务来实现参与者职责



SoaML技术细节

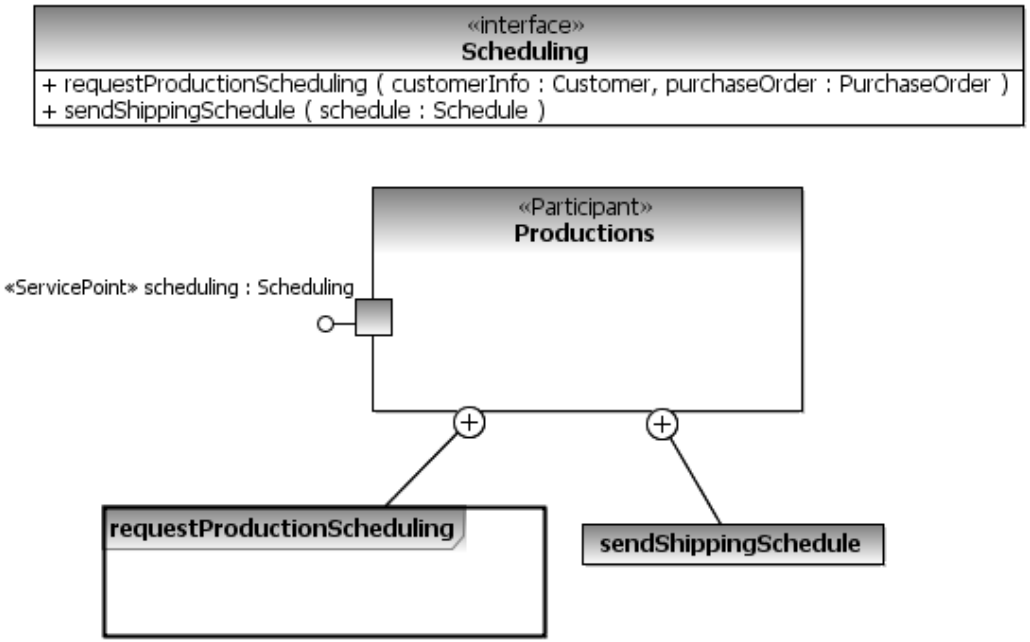
- ServicePoint/RequestPoint
- ServiceContract
- ServiceInterface 实现方法
- 使用 SoaML 流程

ServicePoint and RequestPoint



一个URL，暴露出来可调用的接口（ip：port)

ServicePoints 定义

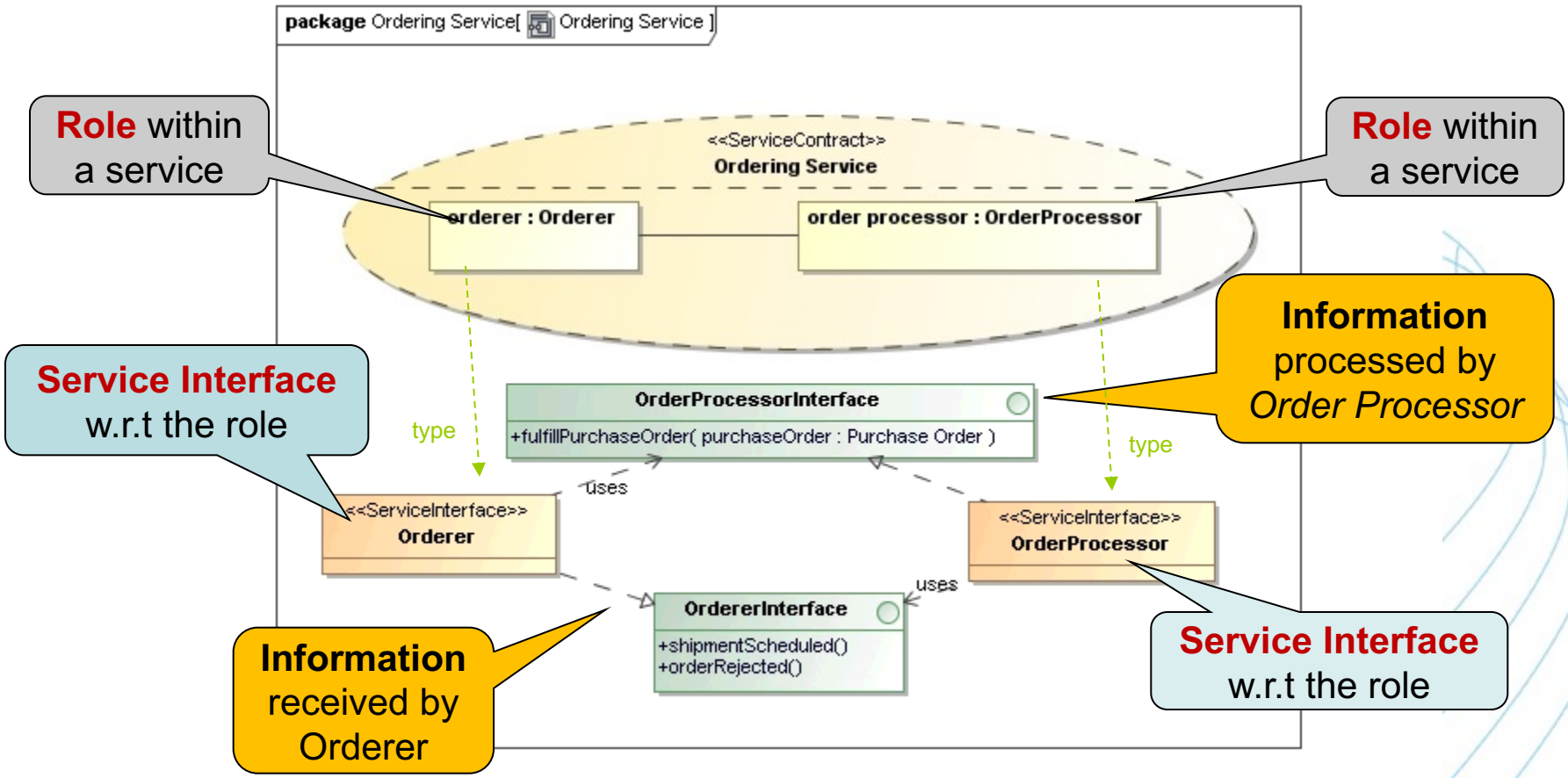


- A ServicePoint is the offer of a service by one participant to others using well defined terms, conditions and interfaces. A ServicePoint defines the connection point through which a Participant offers its capabilities and provides a service to clients.
- A ServicePoint is a mechanism by which a provider Participant makes available services that meet the needs of consumer requests as defined by ServiceInterfaces, Interfaces and ServiceContracts.
- A ServicePoint is represented by a UML Port on a Participant stereotyped as a «ServicePoint», .

ServiceContract

- **ServiceContract 服务契约**
  - 定义了参与者必须遵守的术语、条件、接口
  - 被同时绑定到服务的提供者和消费者上，其基础也是UML的协作图（关注服务之间的交互）
- **服务的完整规范**：包括信息、编排和任何术语与条件

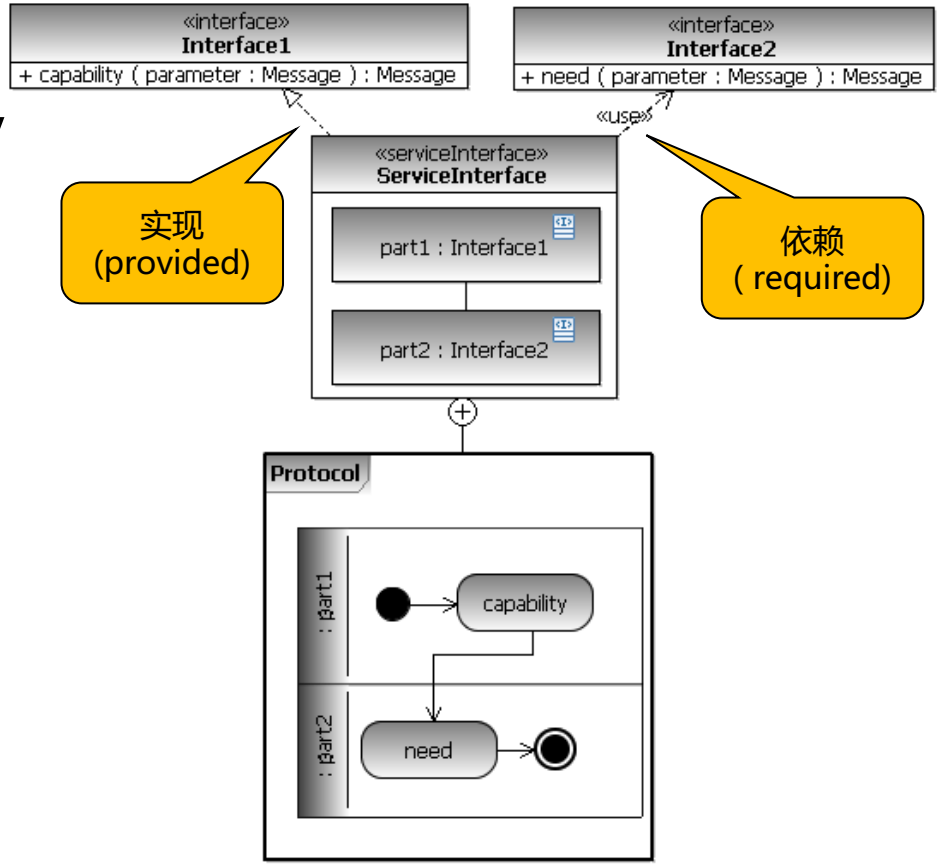
Service Contract



The service contract specifies the details of the service – what information, assets and responsibilities are exchanged and under what rules

ServiceInterface 实现方法

- The service interface has the additional feature that it can specify a bi-directional service.
- The service interface is defined from the perspective of the service provider using three primary sections:
  - the provided and required Interfaces
  - the ServiceInterface class
  - the protocol Behavior (send and receive messages and events)



ServiceInterface 实现方法 (续)

- A ServiceInterface is a UML Class and defines specific roles each participant plays in the service interaction. These roles have a name and an interface type.
- The interface of the provider (which must be the type of one of the parts in the class) is realized (provided) by the ServiceInterface class.
- The interface of the consumer (if any) must be used by the class.

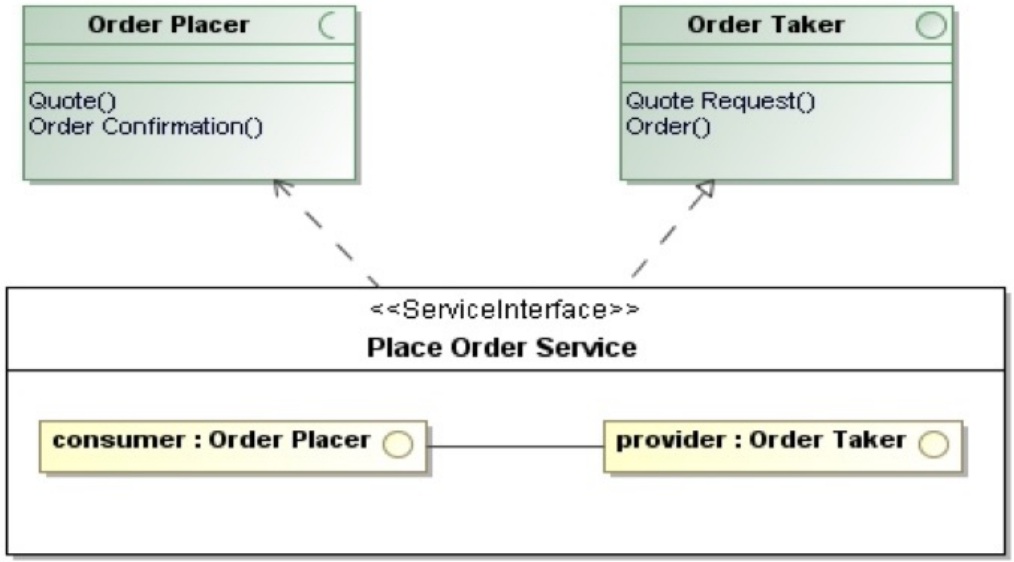
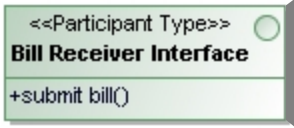


Figure 6.4 - Service Interface Definition

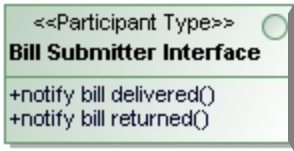
使用 SoaML 流程

1. 创建一个"participant"元素： 在模型中创建一个新的"participant"元素，通常通过从工具的工具箱中选择相关元素来实现。"participant"元素通常表示为一个矩形框，包含参与者的名称、属性和其他信息。
2. 填写"participant"的属性： 在"participant"元素上填写参与者的属性，包括名称、标识符、描述等。这些属性应该清晰地描述参与者的特性和角色。
3. 关联参与者与服务： 可以使用关联或关系来将参与者与相关的服务建立联系。这表示参与者与哪些服务进行交互。你可以在参与者和服务之间绘制关系线，以表示它们之间的关联。
4. 创建服务契约： 对于每个参与者参与的服务，你可以创建相应的服务契约。服务契约描述了参与者如何与服务进行交互，包括输入、输出、约束等信息。服务契约通常可以作为"contract"或"service contract"元素来表示。

Web Services Generation




```
<wsdl:portType name="BillSubmission.BillSubmissionReceiverInterface">
   <wsdl:operation name="submitBill">
     <wsdl:input message="tns:BillSubmissionCluster"
                 name="billSubmission">
     </wsdl:input>
   </wsdl:operation>
 </wsdl:portType>
```




```
<wsdl:portType name="BillSubmission.BillSubmissionSubmitterInterface">
   <wsdl:operation name="notifyBillDelivered">
     <wsdl:input message="tns:BillDeliveredCluster"
                 name="billDelivered">
     </wsdl:input>
   </wsdl:operation>
   <wsdl:operation name="notifyBillReturned">
     <wsdl:input message="tns:BillReturnedCluster"
                 name="billReturned">
     </wsdl:input>
   </wsdl:operation>
 </wsdl:portType>
```

Transaction Message XML Document

```
<BillSubmissionCluster>
      <BusinessTransaction>
            <transactionID> … </transactionID>
      </BusinessTransaction>
      <BillSubmission>
            <bill>
                  <Bill>
                        <billID> … </billID>
                        <principleAmount> … </principleAmount>
                        …
                        <payer>
                              <Party>
                                    <partyID> … </customerID>
                              </Party>
                        </payer>
                        …
                        <lineItems>
                              …
                        </lineItems>
                  </Bill>
            </bill>
            <billingAddress>
                  <BillingAddressCluster>
                        <Address> … </Address>
                        <BillingAddress> … </BillingAddress>
                  </BillingAddressCluster>
            <billingAddress>
      </BillSubmission>
</BillSubmissionCluster>
```

云计算背景下的新发展

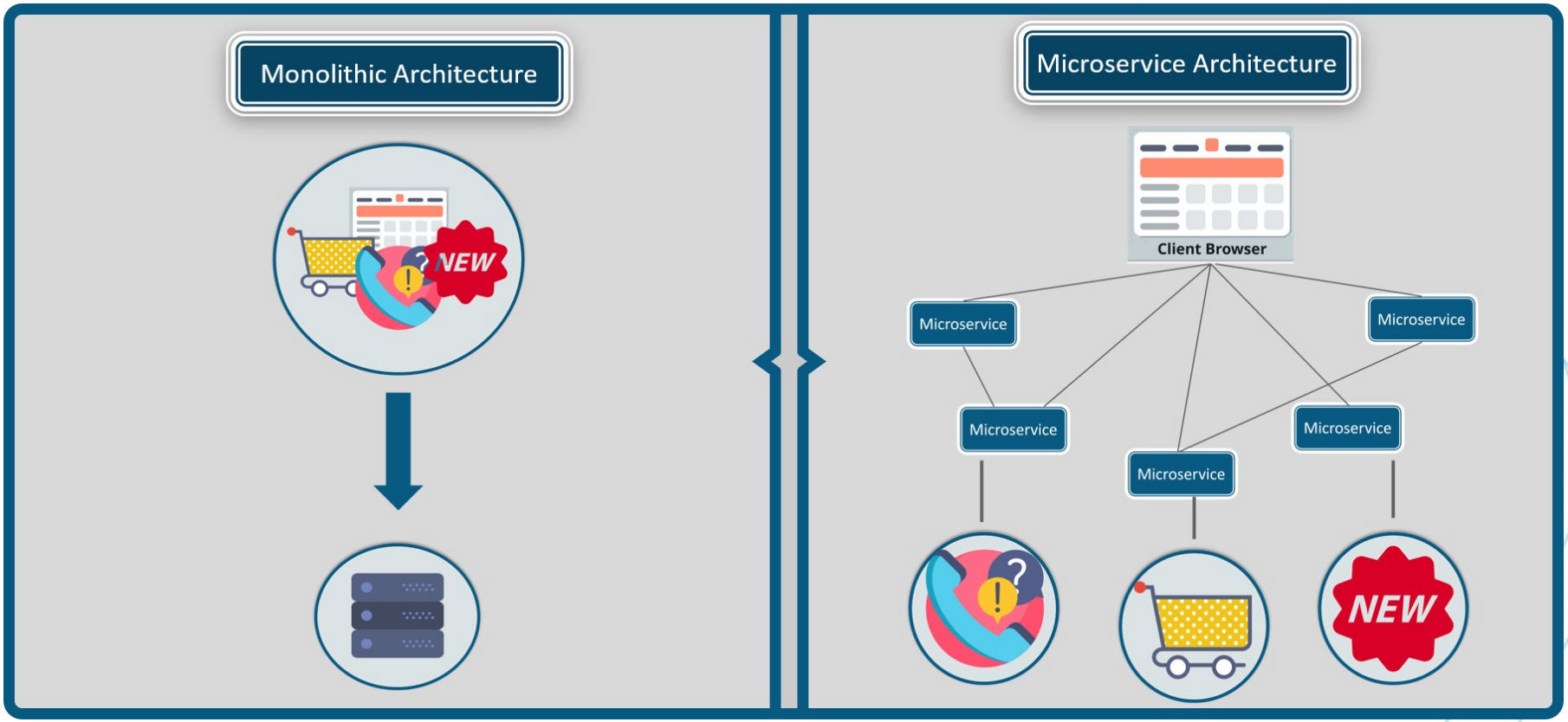
- **传统SOA架构**
  - 服务之间是对等的，提倡服务的公开和共享
  - 以业务流程和企业服务总线提供灵活的服务组合能力，以适应敏捷的复杂业务变化
  - 相对成本高，笨重，开销大
- **云服务**
  - 大部分云服务一般只在云平台内部（管理域内）提供，即使提供远程服务，也只支持云平台的管理功能
  - 即使用户开发和托管的服务，绝大部分不对外共享
  - 面向大规模公众用户，更强调服务的性能和可靠性
- **影响**
  - 业务流程层和整合层（企业服务总线）开始弱化
  - 服务质量层开始加强

RESTful服务、Web APIs和微服务

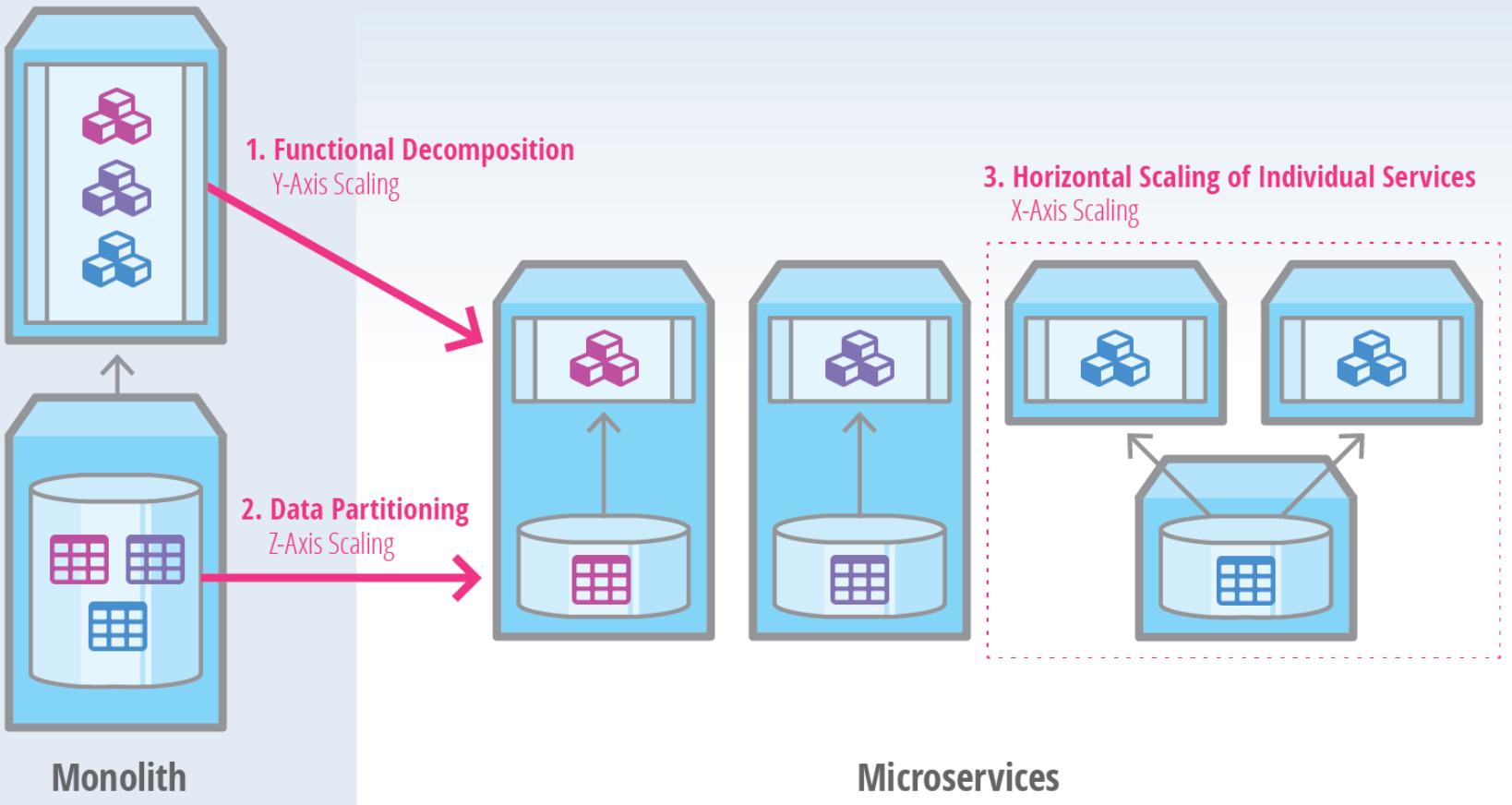
- **传统的SOAP Web服务**
  - 一个目标是为了业务流程和自动化的服务组合
  - 服务的自描述、自动发现，所以需要XML, WSDL以及业务流程模型描述，包括动态代理动用等
- **云计算背景下： RESTful服务**
  - 重性能和可靠性，轻量化
  - 弱化自描述、自动发现和自动组合
  - 可利用Web中已有的大量Web APIs
- **SOA的新发展：微服务架构**
  - 泛化的服务：SOAP、RESTful、RPC
  - 服务组合和业务流程可以概念化，用于设计和分析，并不一定是实体功能
  - 进一步解耦， 细粒度划分(break-down)：微服务化、函数化
  - 服务性能和可靠性强化，服务描述和发现弱化

微服务架构

- 混合架构→微服务架构：手动进行服务的组合



服务划分和横向扩容



The End...

谢谢观赏

- thanks for watching -

欢迎联系讨论，请提前邮件或微信预约

杨任宇
办公室： 曾宪梓科教楼631-b/新主楼G513
微信 ：buaayangry
邮箱 ：renyuyang@buaa.edu.cn
主页 ：http://yangrenyu.github.io