服务计算
Service Computing

王旭
wangxu@buaa.edu.cn

三、面向服务的体系结构(SOA)

提纲

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

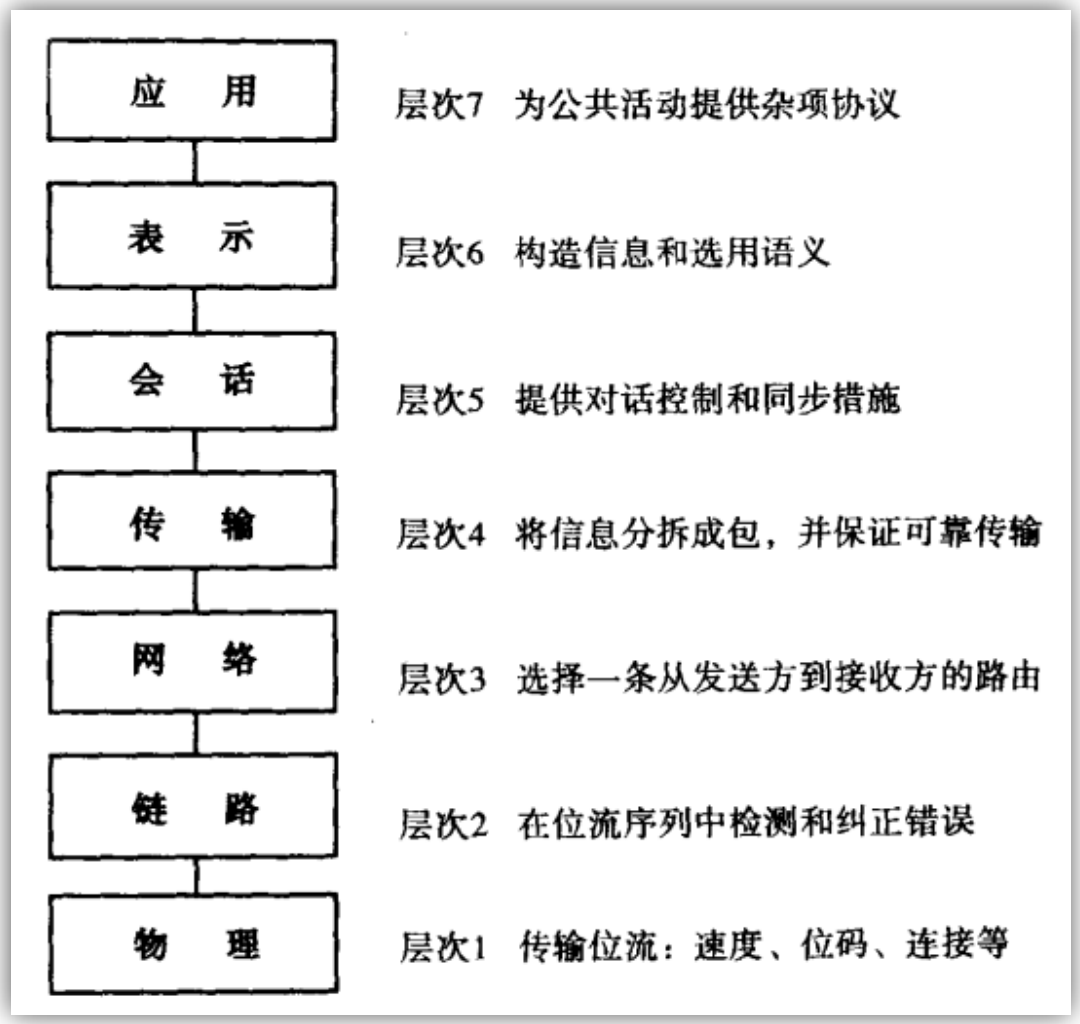
体系结构

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/互联来描述
  - 通过“契约(contracted)”接口，可标识组件并可指派和客户端组件交互的任务
  - 组件互联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

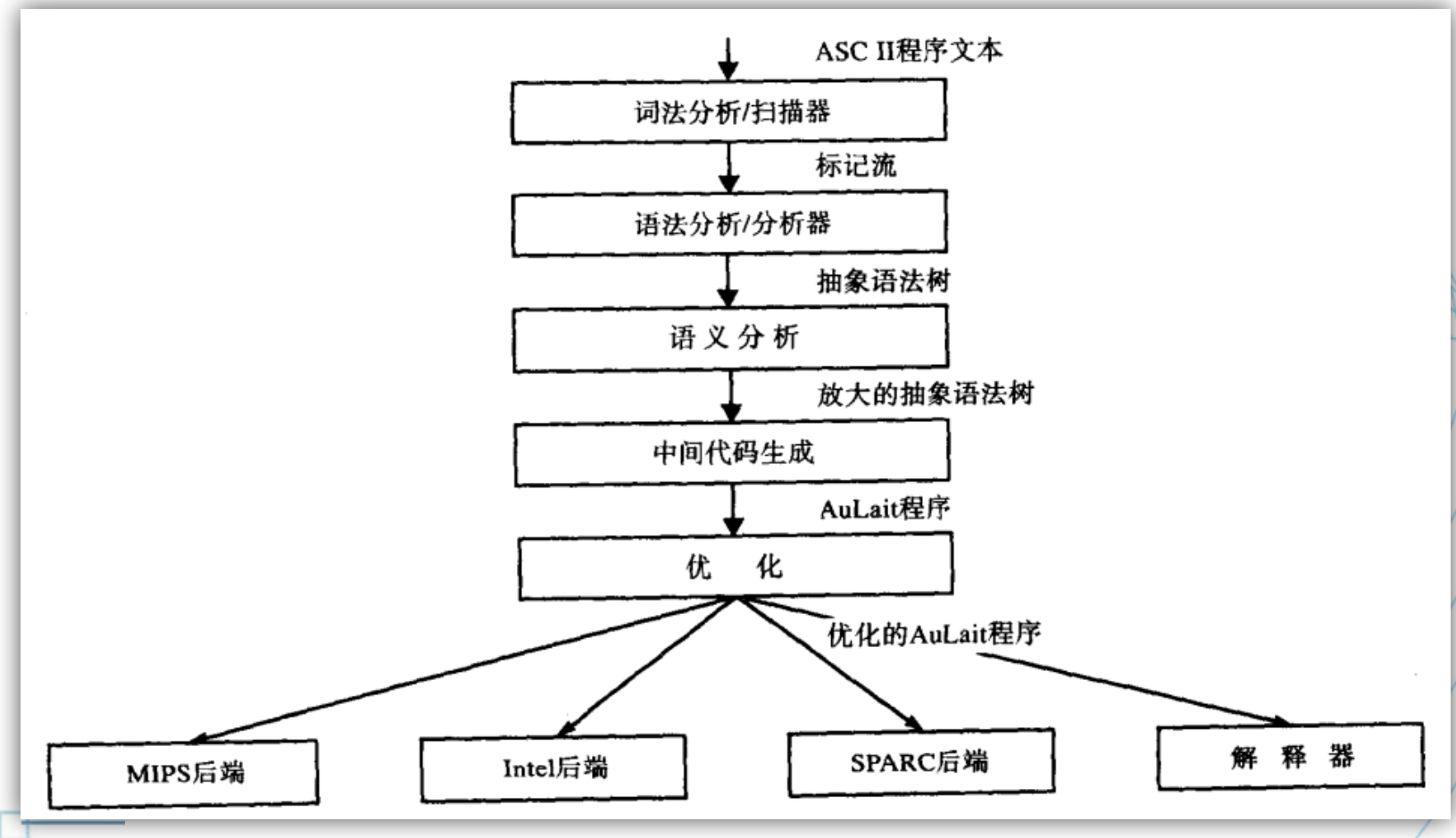
分层体系结构

- 分层体系结构 layers



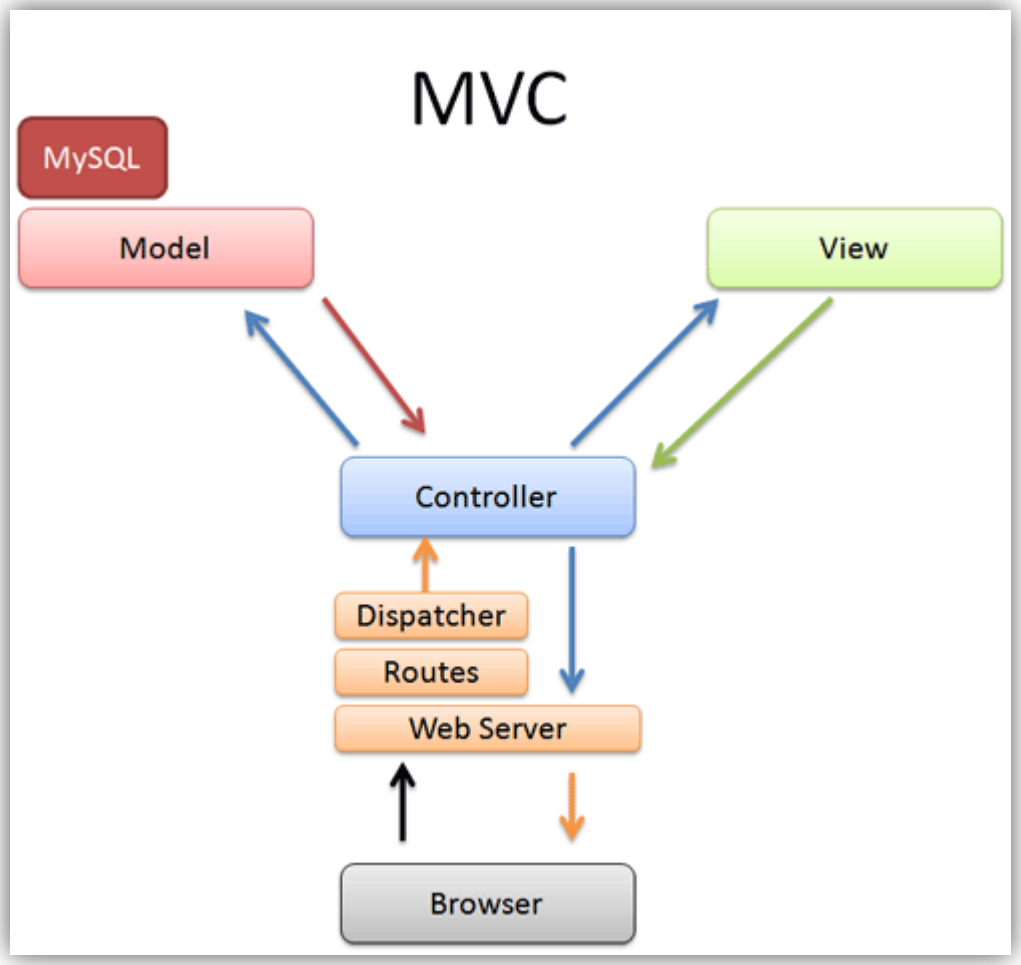
管道和过滤器体系结构

- 管道和过滤器体系结构 pipes and filters



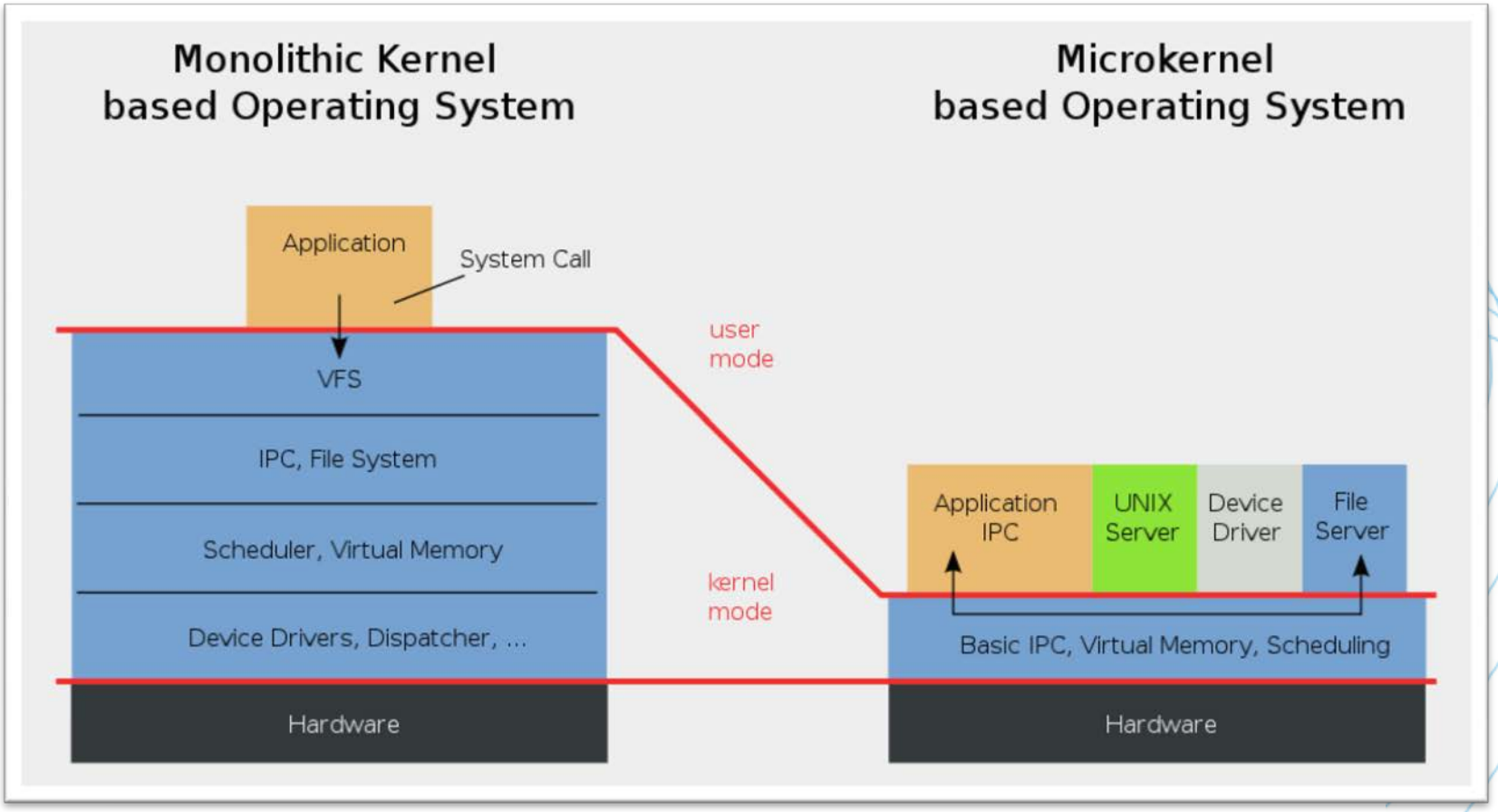
MVC体系结构

- 模型-视图-控制器（MVC）

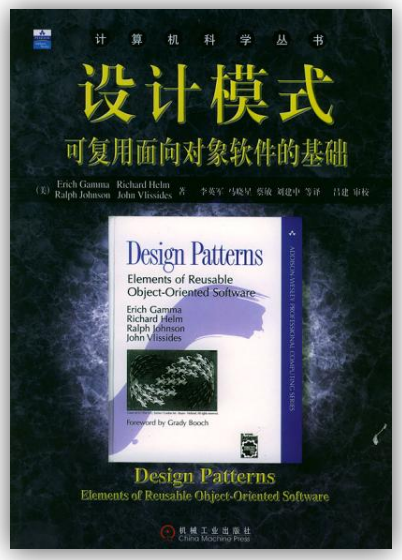


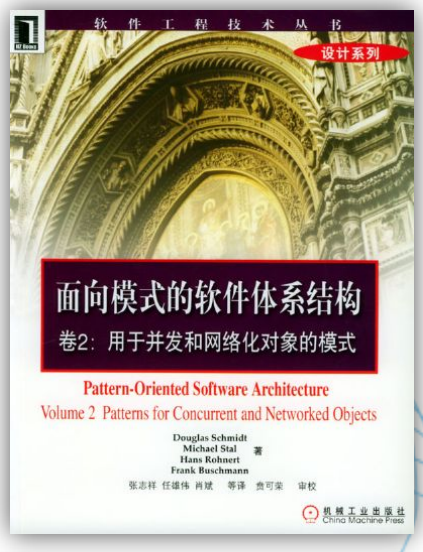
微内核体系结构

- 微内核体系结构 micro kernel



面向模式的体系结构





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

什么是“SOA”

- 从字面上看，SOA=Service(服务)+体系结构(Architecture)

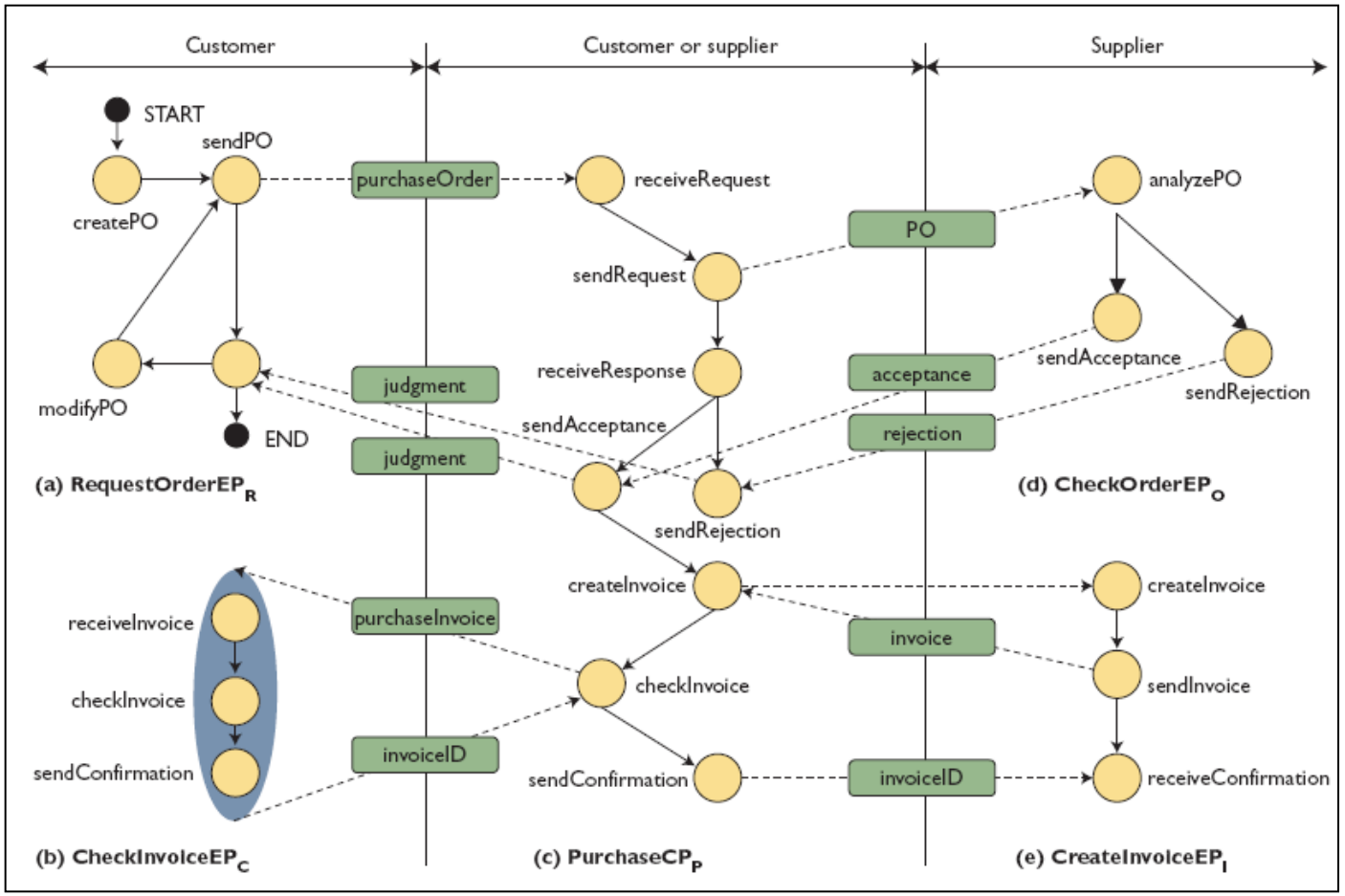
Roles

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 

什么是“SOA”

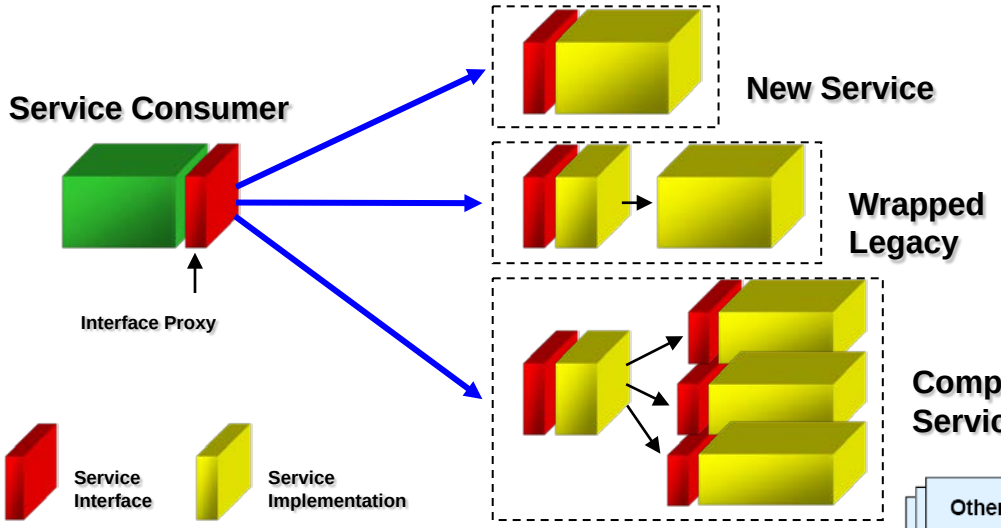
- 狭义的解释
  - SOA是一种支持面向服务思想的体系结构风格(architectural style)，是一种业务和IT的范式(paradigm)
- 广义的解释
  - 在软件工程中，SOA是设计和开发互操作服务形态的软件的一组原则和方法。这些服务是定义良好的业务功能，是为不同目的可重用的软件构件。
- OASIS的定义
  - 是一种组织和使用不同管理域控制下分布式能力的范式。它提供了一种统一的方法以提供、发现、交互和使用这些能力，从而产生预期的效果。
- Thomas Erl
  - SOA表示了一种开放、敏捷、可扩展、联邦、可组合的体系结构，由自治、具备QoS能力、松散、可互操作、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，使得不同管理域之间的松耦合。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

从业务的角度看SOA

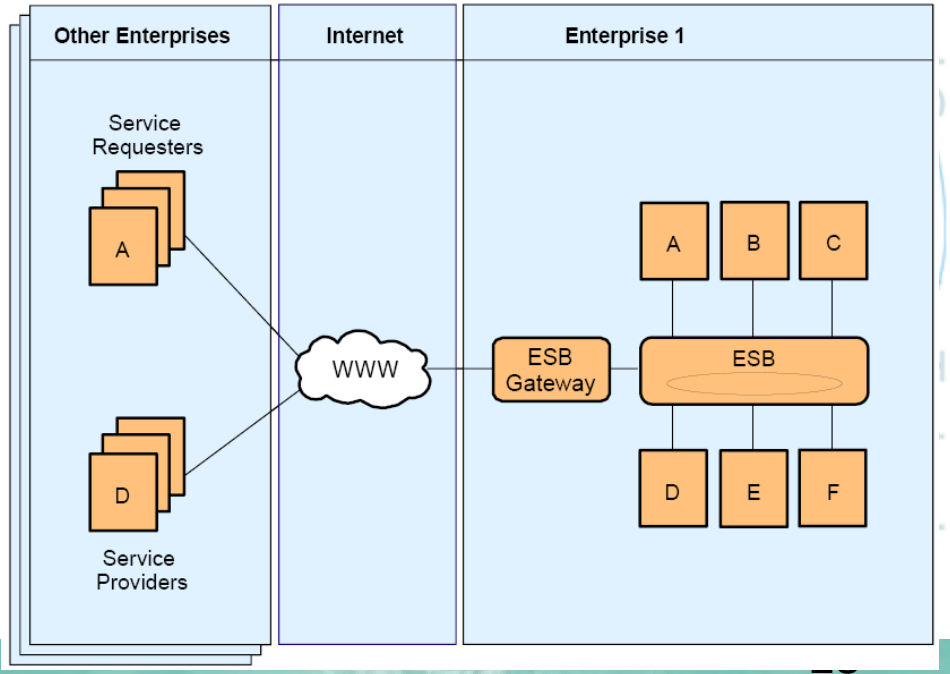


是业务服务的集合，用于表达企业向客户暴露的业务设计

从体系结构的角度看SOA



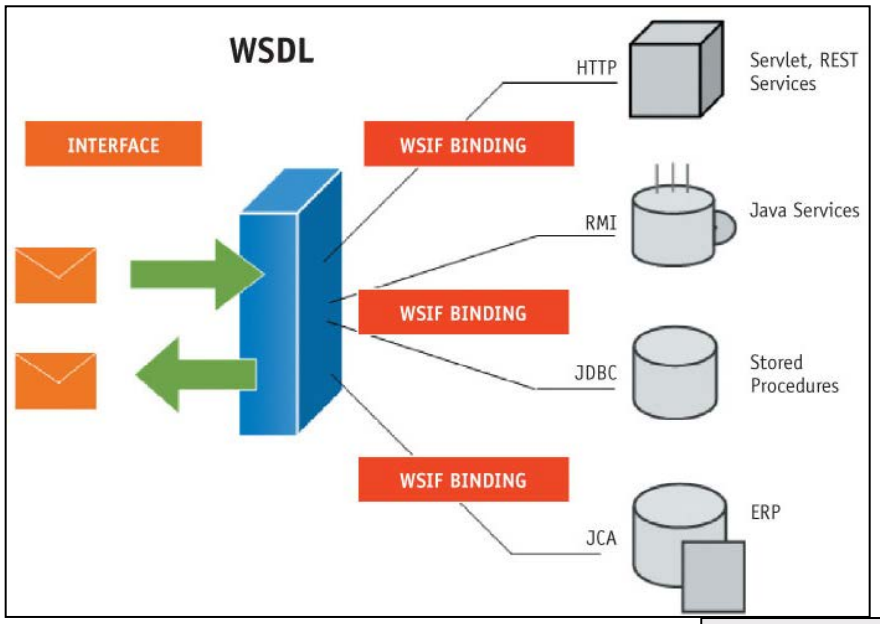
是一种体系结构风格(style)，由服务提供者、请求者和服务描述构成

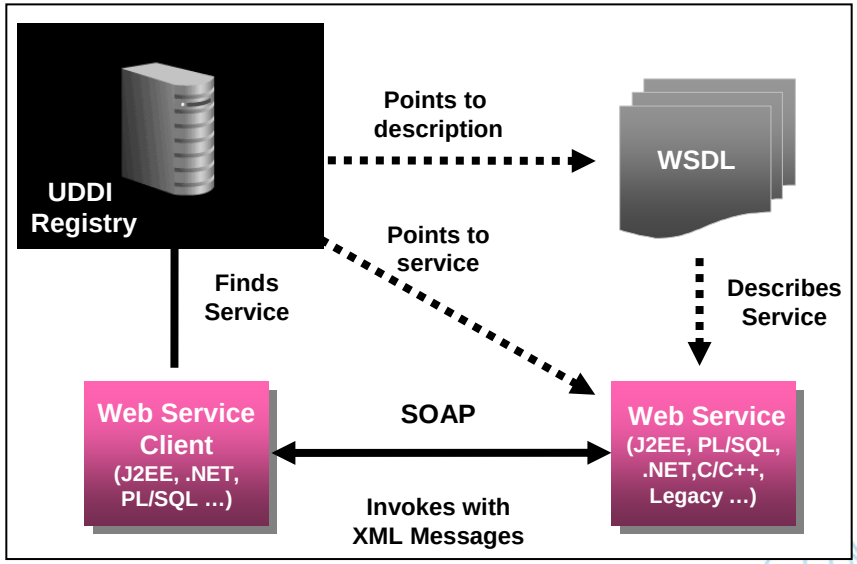


什么叫体系结构风格

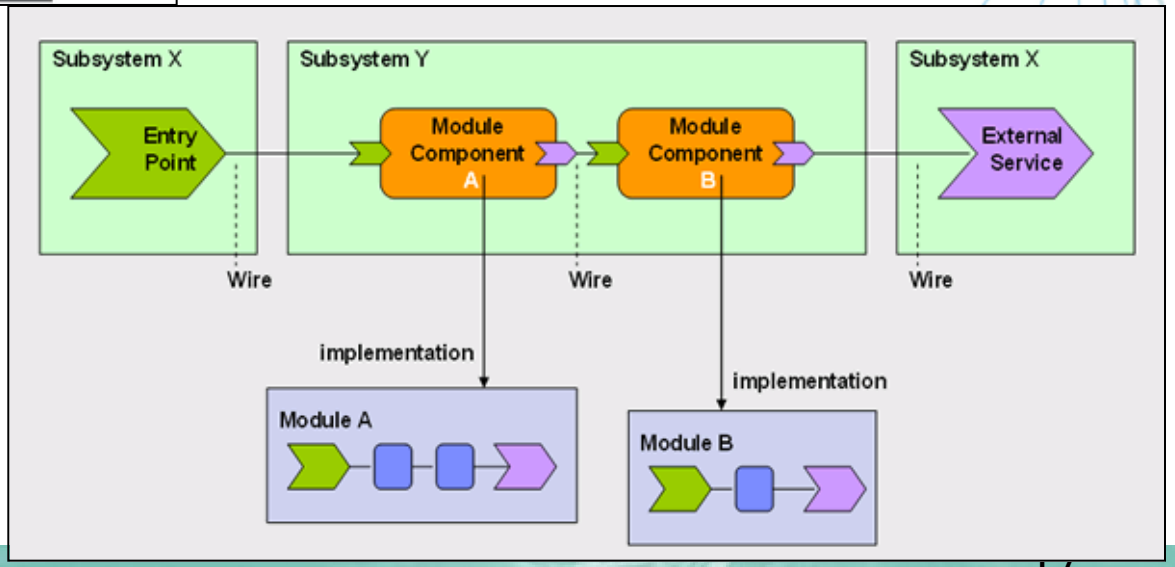
- 建筑风格：中国古代建筑风格、西方建筑风格
- 程序设计样式、设计模式、体系结构风格
- 软件体系结构风格是描述某一特定应用领域中系统组织方式的**惯用模式**
- 反映领域中众多系统所**共有的结构和语义特性**，并指导如何将各个模块和子系统有效地组织成一个完整的系统
- 常见风格
  - 管道-过滤器（Pipes and Filters）
  - 层次系统（Layered Systems）
  - 黑板系统（Blackboard）
  - 客户/服务器（Client/Server）
  - 浏览器/服务器(B/S)
  - 微核（MicroKernal）
  - Service-oriented architecture
  - Model-driven architecture

从编程模式的角度看SOA



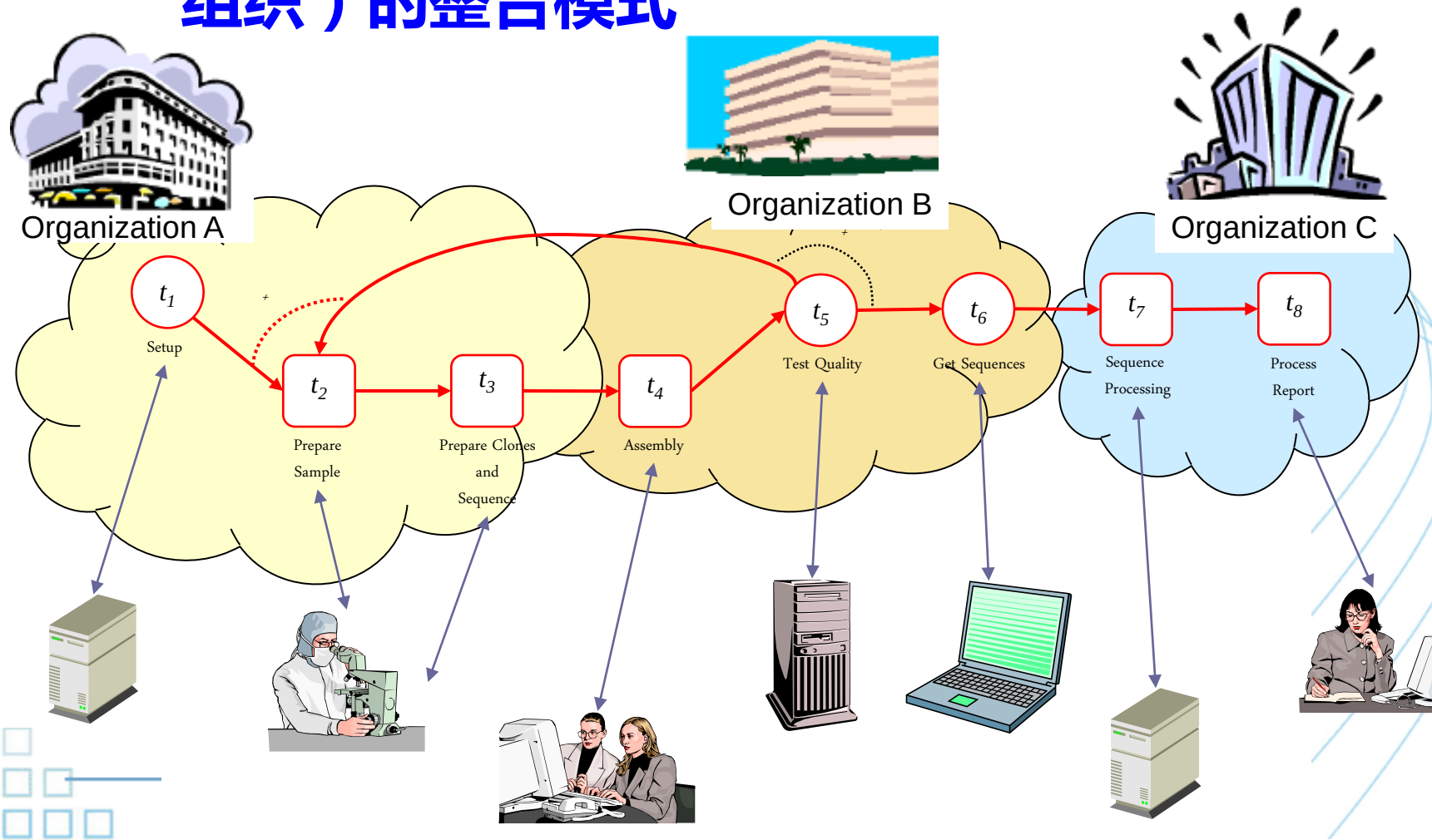


是一种编程模型，涉及标准、工具、方法、技术等



从应用的角度看SOA

跨功能（有时，跨组织）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

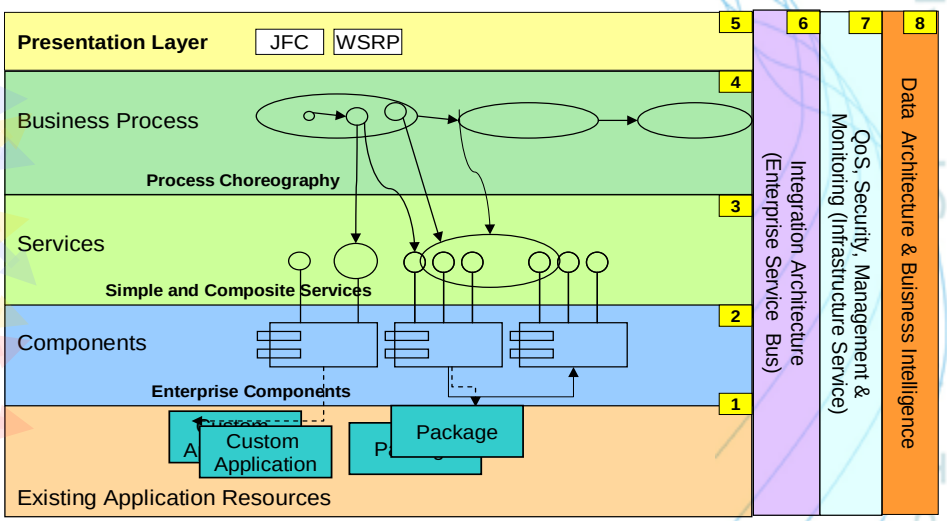
<< Input from: Business Componentization/Analysis >>

Identification
of *candidate* Services, Components, Flows

Specification
of Services, Components, Flows

Realization
Decisions

<< Output to: SOA Implementation >>



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

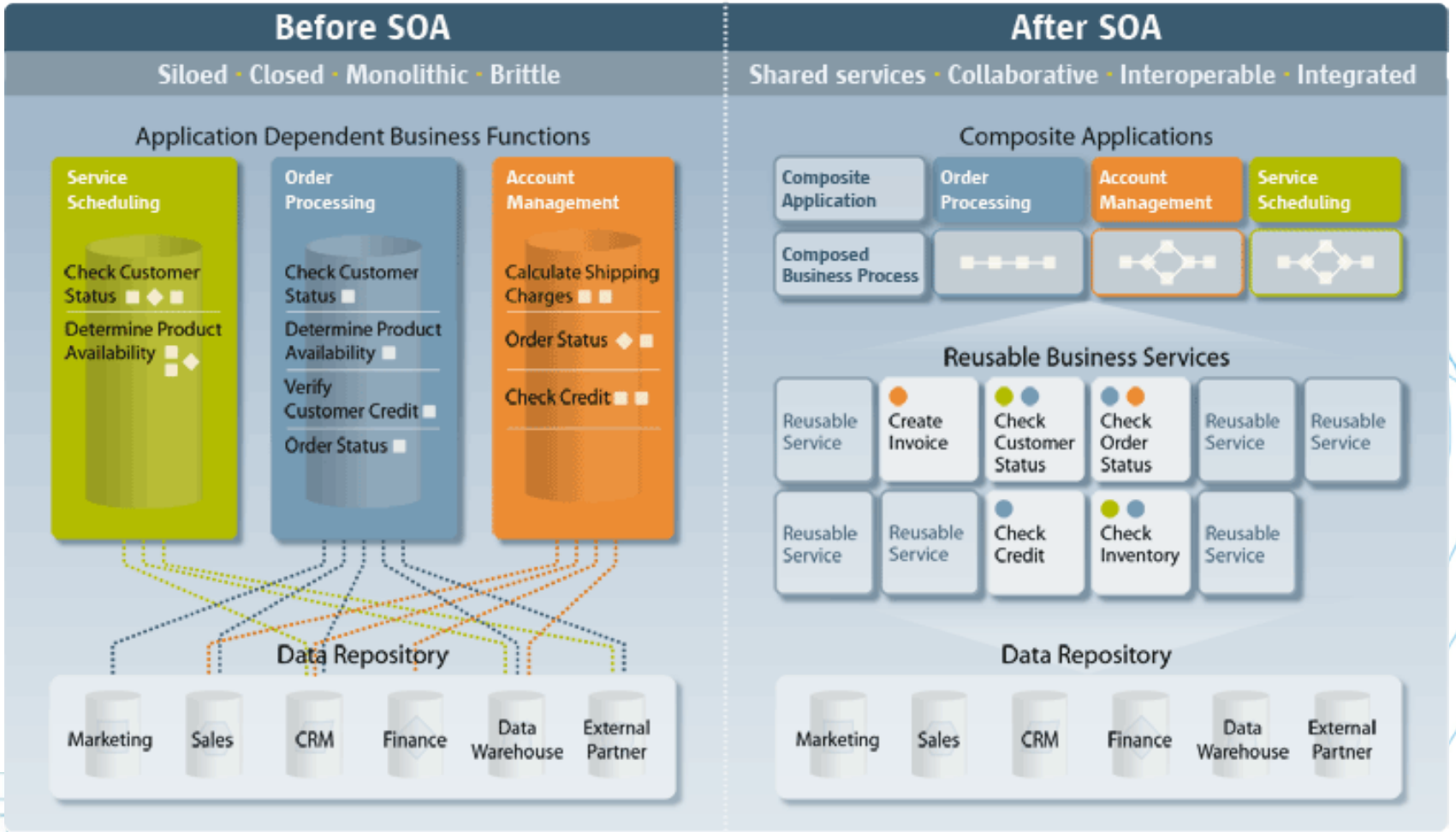
- 作为一种灵活、可扩展的体系结构框架，SOA具备以下优势
  - 低成本：有效利用遗留资源
  - 灵活性：业务过程与应用的快速构造与重配
  - 盈利：有效重用已有业务、松耦合的IT服务，拓展新市场
  - 敏捷性：快速推出业务应用
  - 整合能力：在孤立的应用和组织间集成IT系统

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。**
  - 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

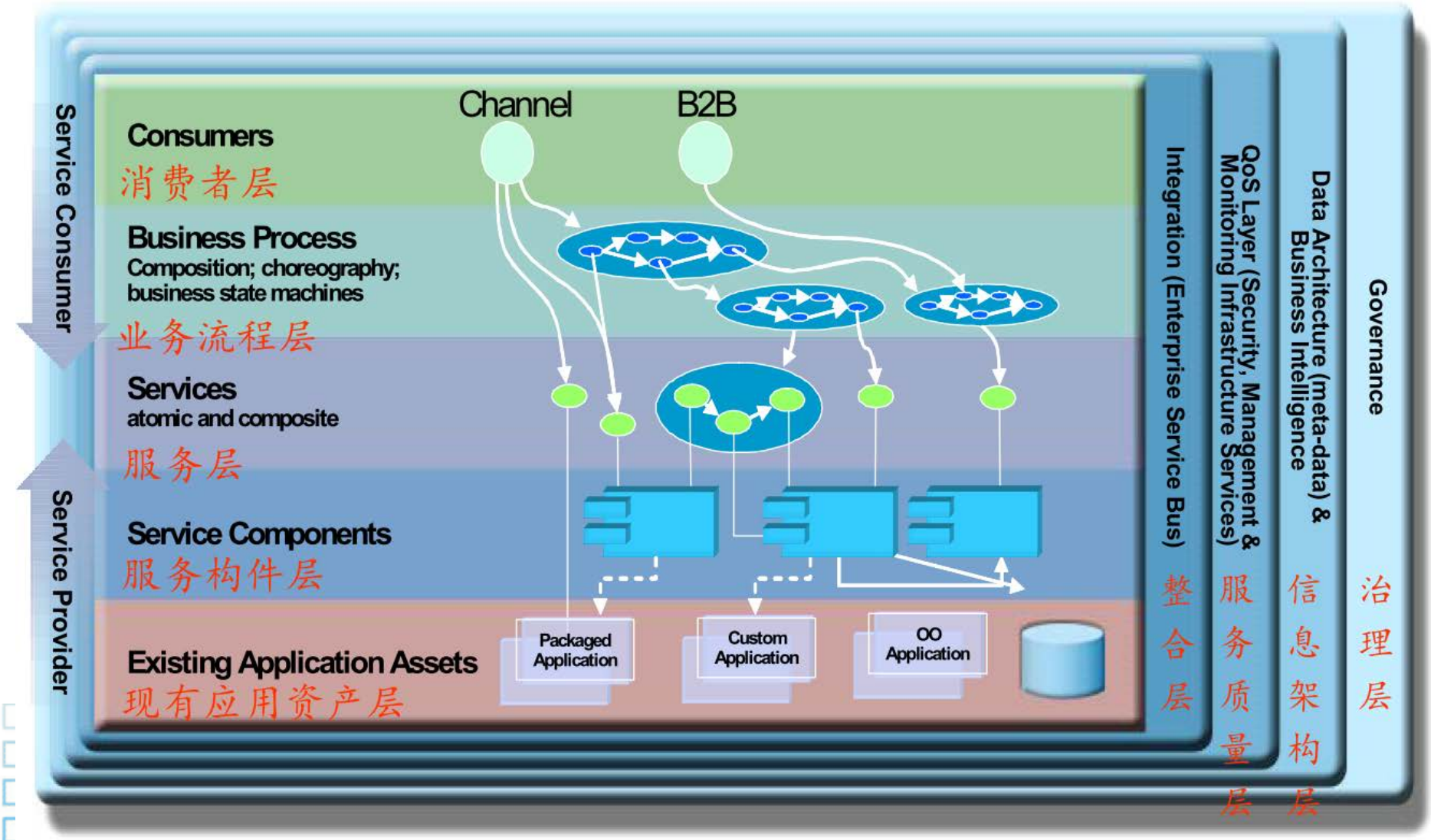
向SOA 转型

- 业务与IT，逐层抽象和映射，相互对齐



SOA参考架构层次图

- SOA solution stack



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
  - 企业服务总线

ESB(Optional)

- 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

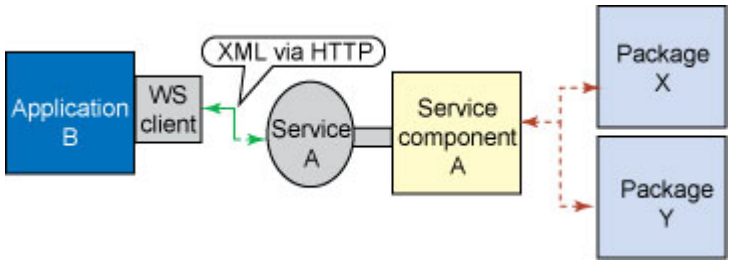
非功能面

层次1: 现有应用资产层

- 这一层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
- 主要由已有的应用软件系统构成，因此，它利用已有的IT资产来实现SOA方案，可大大减少SOA实现方案的成本
- 这些软件系统包括
  - 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
- 需要使用SOA技术，将他们集成起来，有效利用遗留资源

层次2: 服务构件层

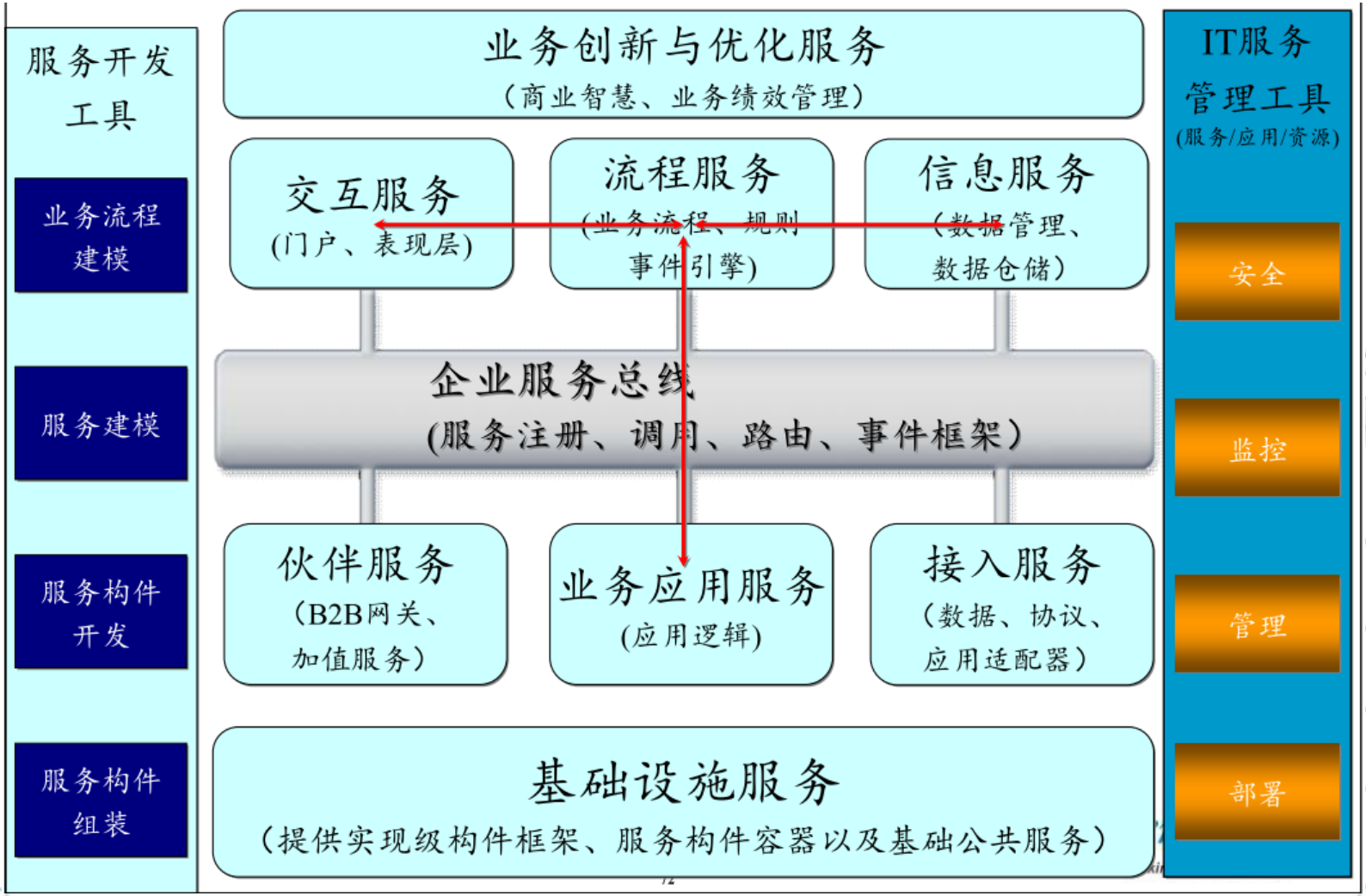
- 主要包含软件构件，提供服务的实现，因此称服务构件
- 服务构件反映了一个服务的定义，包括功能和QoS
- 服务构件符合服务层定义的服务契约，保证IT实现与服务描述的一致性
- 服务构件
  - 提供一个可信的服务实现的执行点，以确保服务质量和遵守服务层协议（SLA）
  - 通过支持IT服务的功能实现，提高了业务的灵活性
  - 通过对客户隐藏实现细节，增强解耦合能力，提高了IT的灵活性
- 是企业或业务单元层面上，受管理、治理的企业资产集合
- 使用基于容器的技术，如应用服务器，来实现构件、任务管理、高可用性、负载均衡等
- 服务构件的外观(façade)模式
  - 应用B仅与服务描述耦合
  - 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

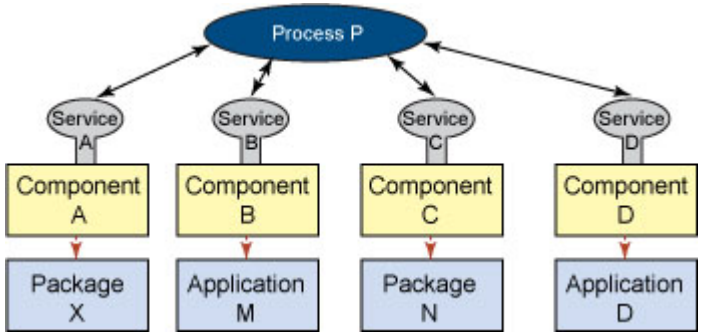
- 这一层由界定在SOA之内的所有服务组成
  - 服务被认为是功能独立、可网络访问的功能模块
- 提供了通过服务调用业务功能的详细信息，比如WSDL
- 可能包括:
  - 策略(Policy)文档
  - SOA管理描述
  - 分类，或显示服务的依赖性的附件
- 该层的服务可被发现、调用，或者编排组合服务

服务层的进一步细分



层次4: 业务流程层

- 该层定义服务层中这些服务的组合和编排，将一组服务组合或编排成一个流程



- 业务流程层涵盖流程的表示、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集成层来路由、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和Ajax）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

层次6: 整合层

- 整合层是一个关键的SOA使能器，因为它**提供调解，路由和传输服务要求的能力**，从服务请求者到正确的服务供应者。
- 通过一套可靠的机制实现服务整合
  - 端点整合、智能路由、协议调解、其他转换机制等（ESB）
  - WSDL规定了binding，隐含了服务位置；而ESB位置独立的整合能力
- 这里的整合主要是2到4 层的整合
- 实现了
  - 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

层次7: 服务质量层

- SOA固有特点，导致需加强现有的计算机系统的QoS
  - 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、需要聚积IT 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、警示每个层次中不符合非功能需求的发生
  - 实时监控其他层，当发生不符合QoS需求时，会发出信号、触发事件

层次8: 信息架构层

- 这一层包括信息架构，商业智能，元数据的考虑，并确保列入有关数据架构和信息架构重要的考虑因素
- 也可以通过数据集市和资料仓库，用来作为建立商业智慧的基础

- 包括储存在这一层的元数据内容

- 特别是，对于特定行业的SOA解决方案，这一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），及交换业务数据的商业协议
- 数据发现，数据挖掘，和数据解析的建模也包括在这一层

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
- 提供指导和政策以管理所有方面的在SOA解决方案的服务、SLA、容量和性能，安全性和监测。
- 它使SOA治理服务通过强调运行生命周期管理充分地被整合。
- 这一层可以应用到所有在SOA解决方案堆栈的其它层。从QoS和性能测量指针的角度来看，它是与第7层完好连接的。
- 在这一层，一个可扩展的和灵活的SOA的治理框架包括解决方案级别的基于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以设计和调SOA解决方案整，解决方案级别的从联邦应用角度的安全赋能指引 。
- 在这一层的架构决策被封装在咨询实践和框架，架构或SOA 容量规划的文件， SOA解决方案的SLA， SOA性能监测的政策，和SOA解决方案级别的安全赋能指引。

企业服务总线

- SOA的重要目标就是要在分布式环境下实现多组织之间业务的交互与协同；因此独立存在的服务是没有意义的；

- 即使采用上面的service integrator，一个组织中存在的和使用的服务数量仍然是巨大的，它们之间的关系也很复杂。

- 必须提供一种手段，能够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来管理它们。

企业服务总线

- ESB是一种基于标准的高速链路(backbone)，它利用MOM(面向消息中间件)功能连接异构系统
- 一种构建、部署企业SOA的新方法
- 综合不同类型中间件的特征
- 简化Web服务部署，解决集成问题
- 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-*，…

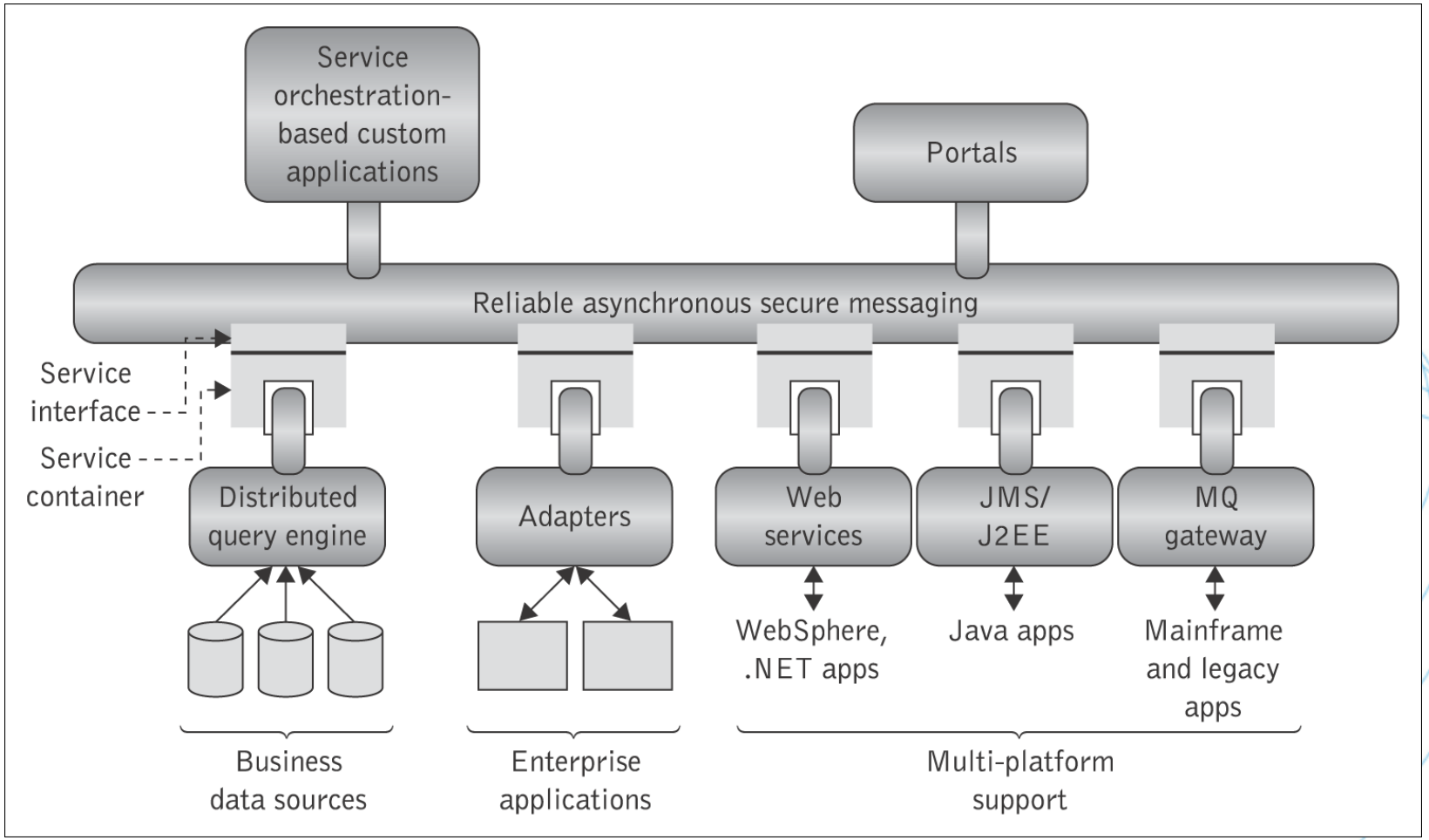
企业服务总线(ESB)

- 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。

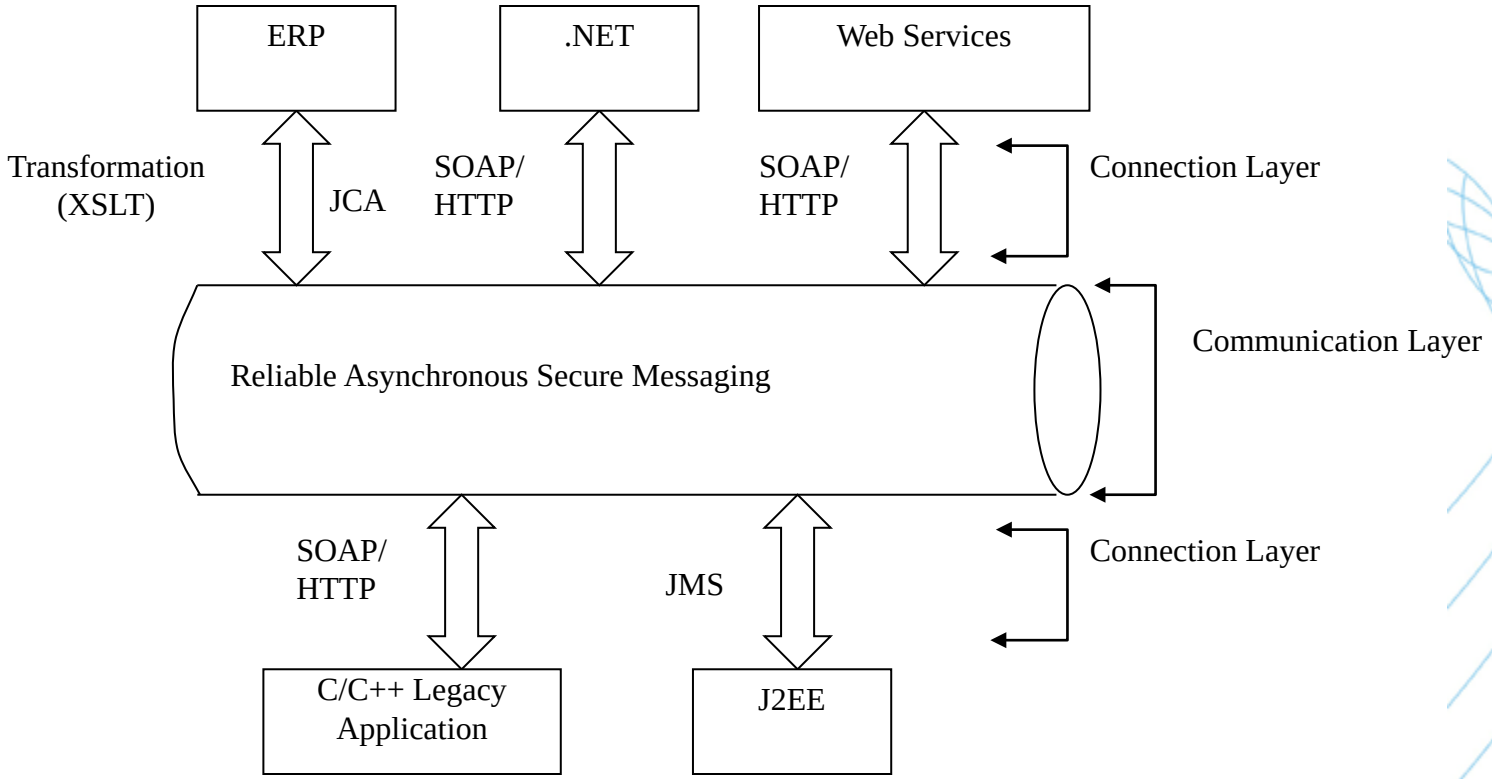
ESB在请求者和服务之间实现了：

- 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
- 处理分离资源间的业务事件

ESB体系结构



ESB体系结构



ESB的主要能力

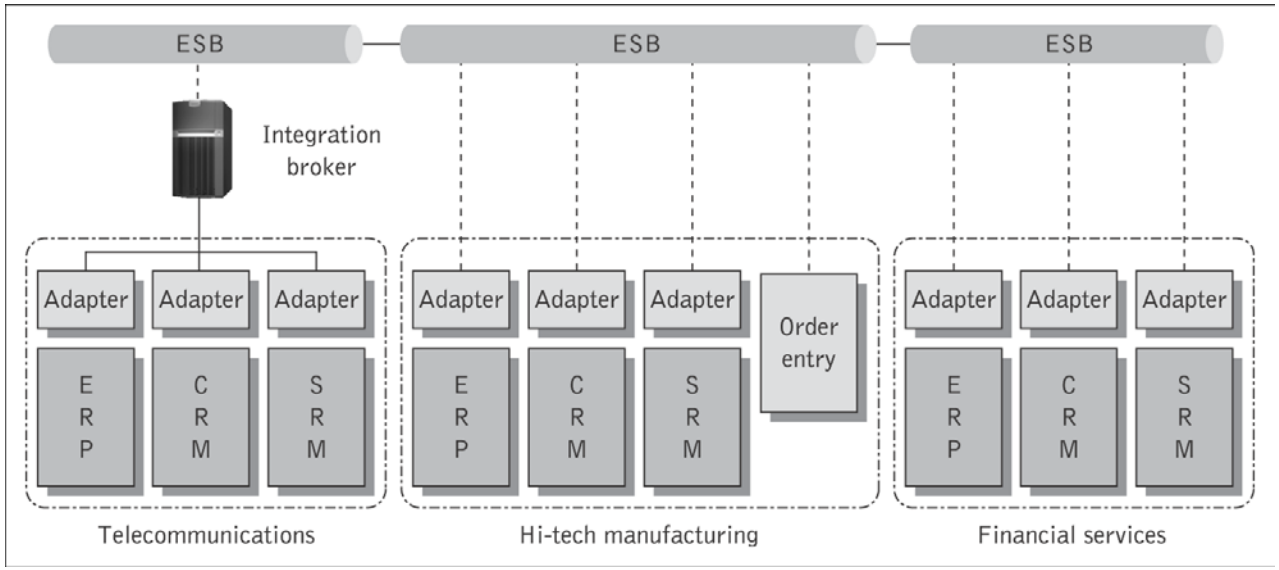
- 有效利用遗留资产
- 服务通信：在服务交互的不同协议间路由，并转换消息格式
- 动态连接：动态连接Web服务的能力，无需使用单独的静态API或代理
- 基于主题和内容的路由能力
- 支持多QoS的端点发现
- 集成和转换
- 安全性
- 长事务支持
- 管理和监控
- 可扩展性

ESB的功能及相关标准

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

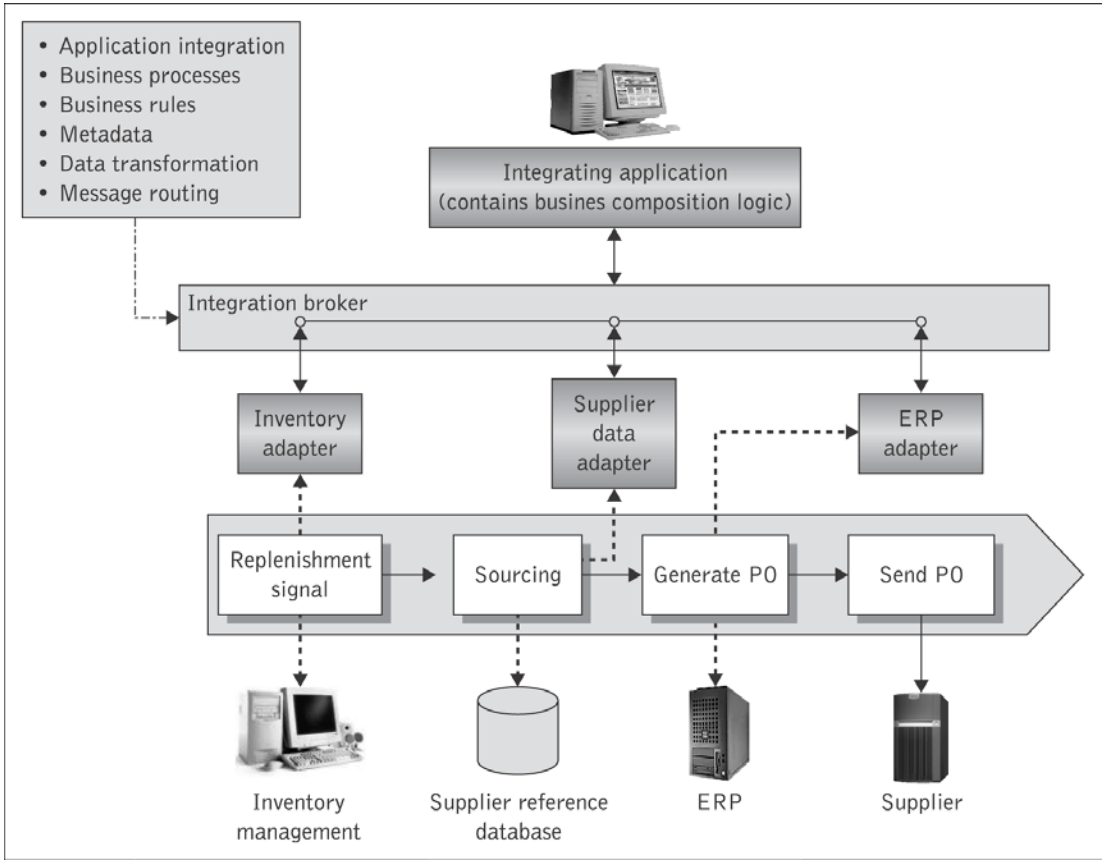
- ESB实现的不同方法：单中心的ESB、多个ESB的联邦



- ESB需要中间件设施的一个集成集合，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

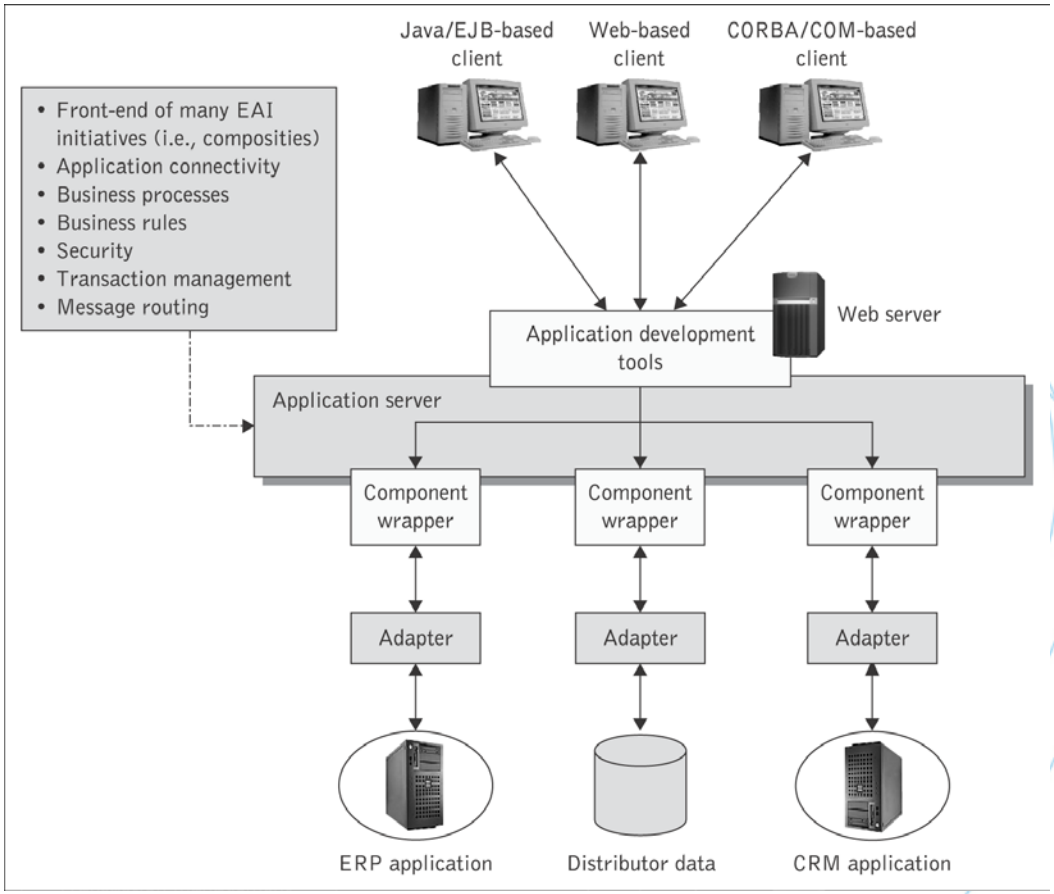
- 简化多个后端EIS间的信息移动，解决应用语义和异构平台的差异



ESB元素: 应用服务器

- 为开发和部署分布式Web/非Web应用，提供集成开发环境

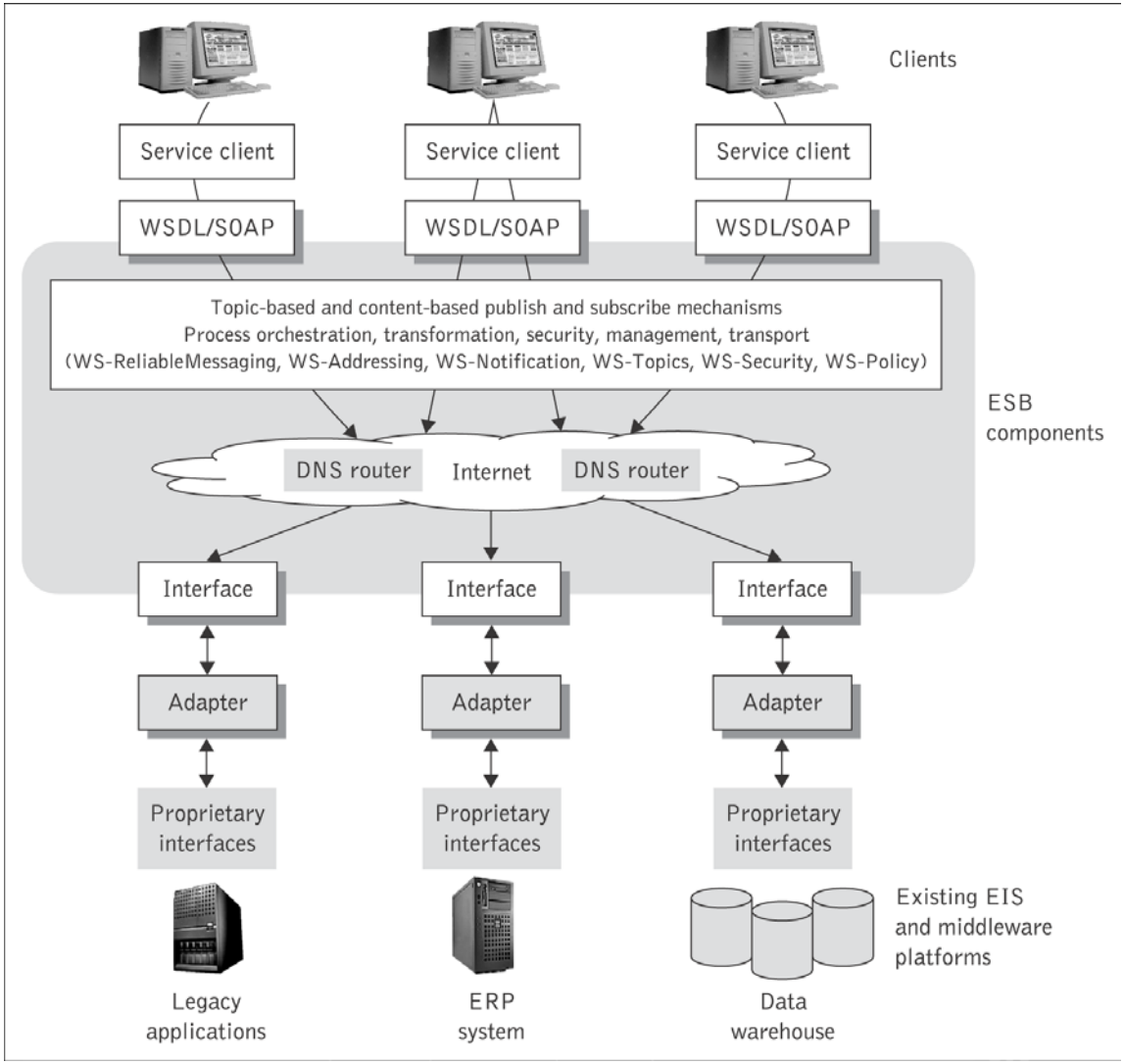
An application server typically provides Web connectivity for extending existing solutions and brings transaction processing mechanisms to the Web.



ESB元素: 业务过程管理

- BPM提供端到端的可见性，控制跨企业、跨应用的长期、多步信息请求或事务过程
- 监控单个过程实例和所有过程实例的状态
- 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



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
- 开发工具Modelio
  - http://www.modeliosoft.com/

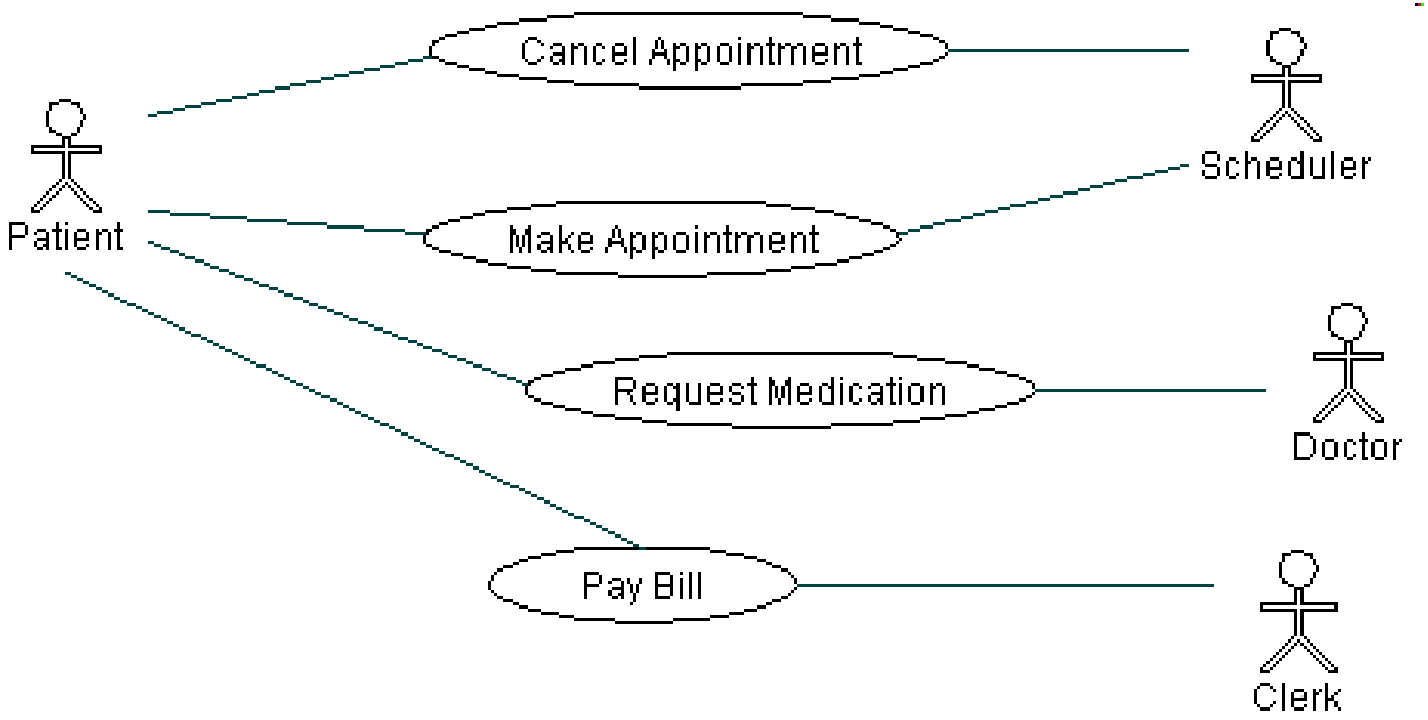
SoaML的目标

- 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
- 方便地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

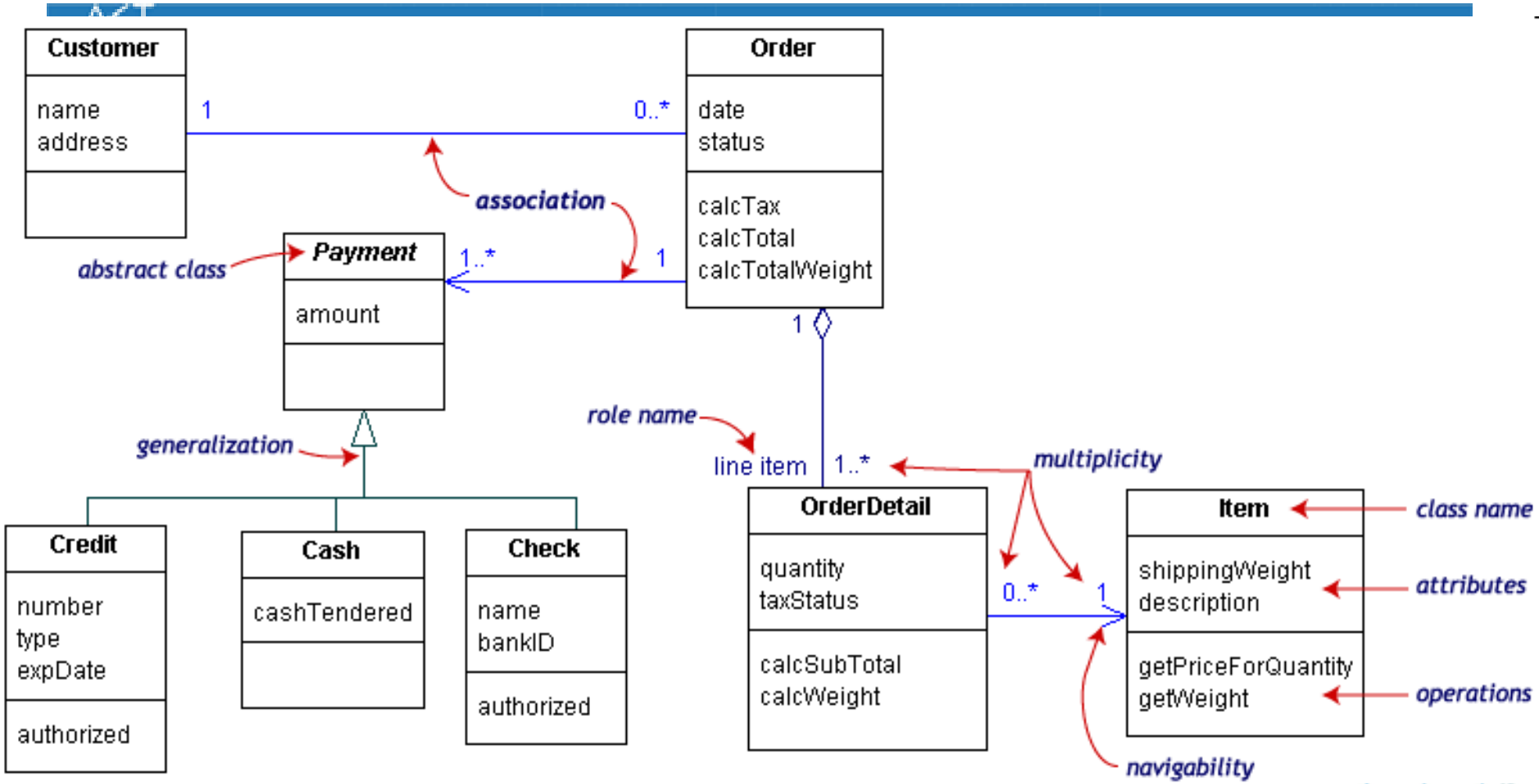
- **结构建模：**
  - 类图
  - 对象图
- **行为建模**
  - 用例图
  - 交互图（顺序图、协作图）
  - 活动图
  - 状态图
- **体系结构建模**
  - 构件图
  - 实施图

Use Case Diagrams



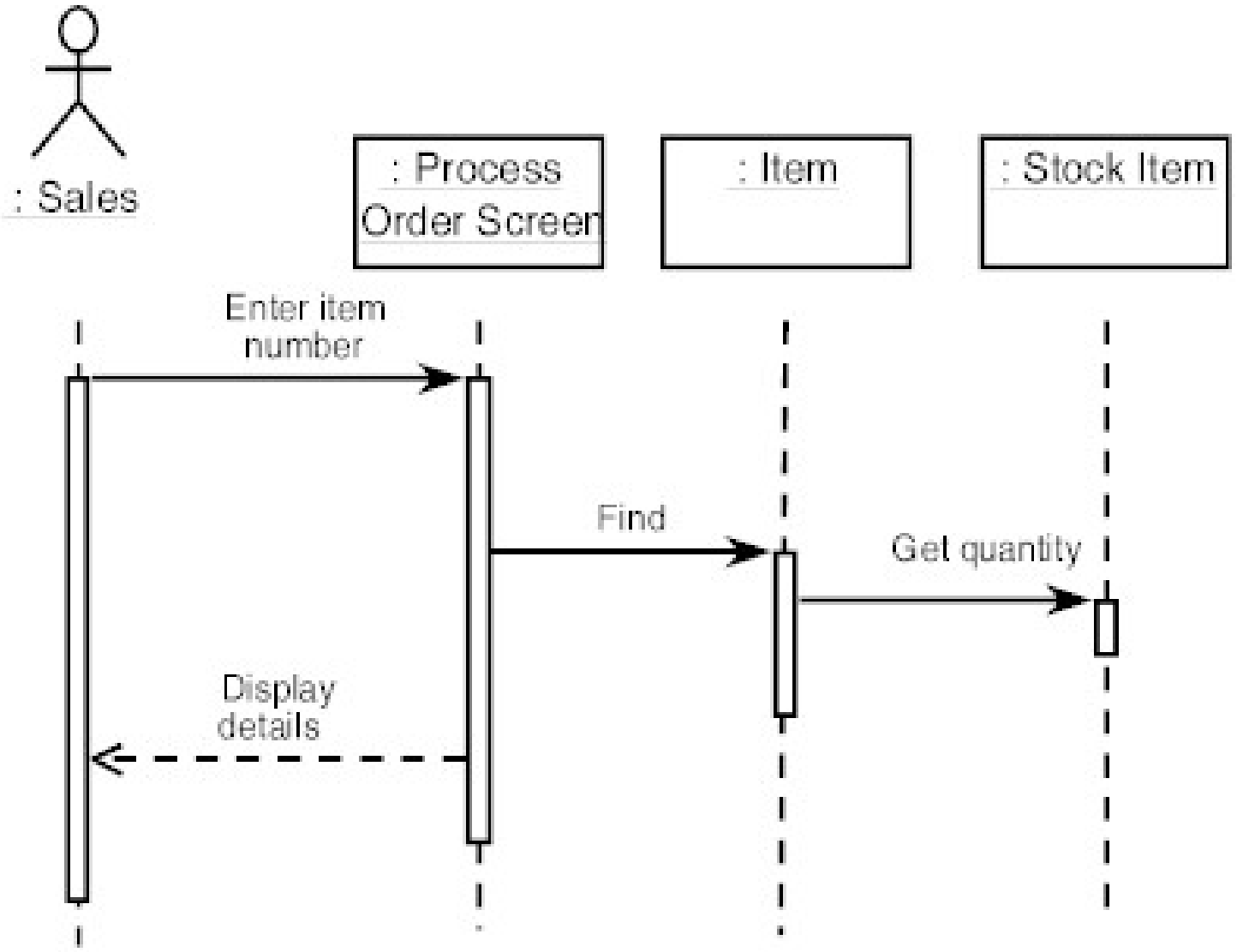
Use Case diagrams describe what a system does. HEASSISTANT simply had 1 actor, but generally, we have multiple actors.

Class Diagrams

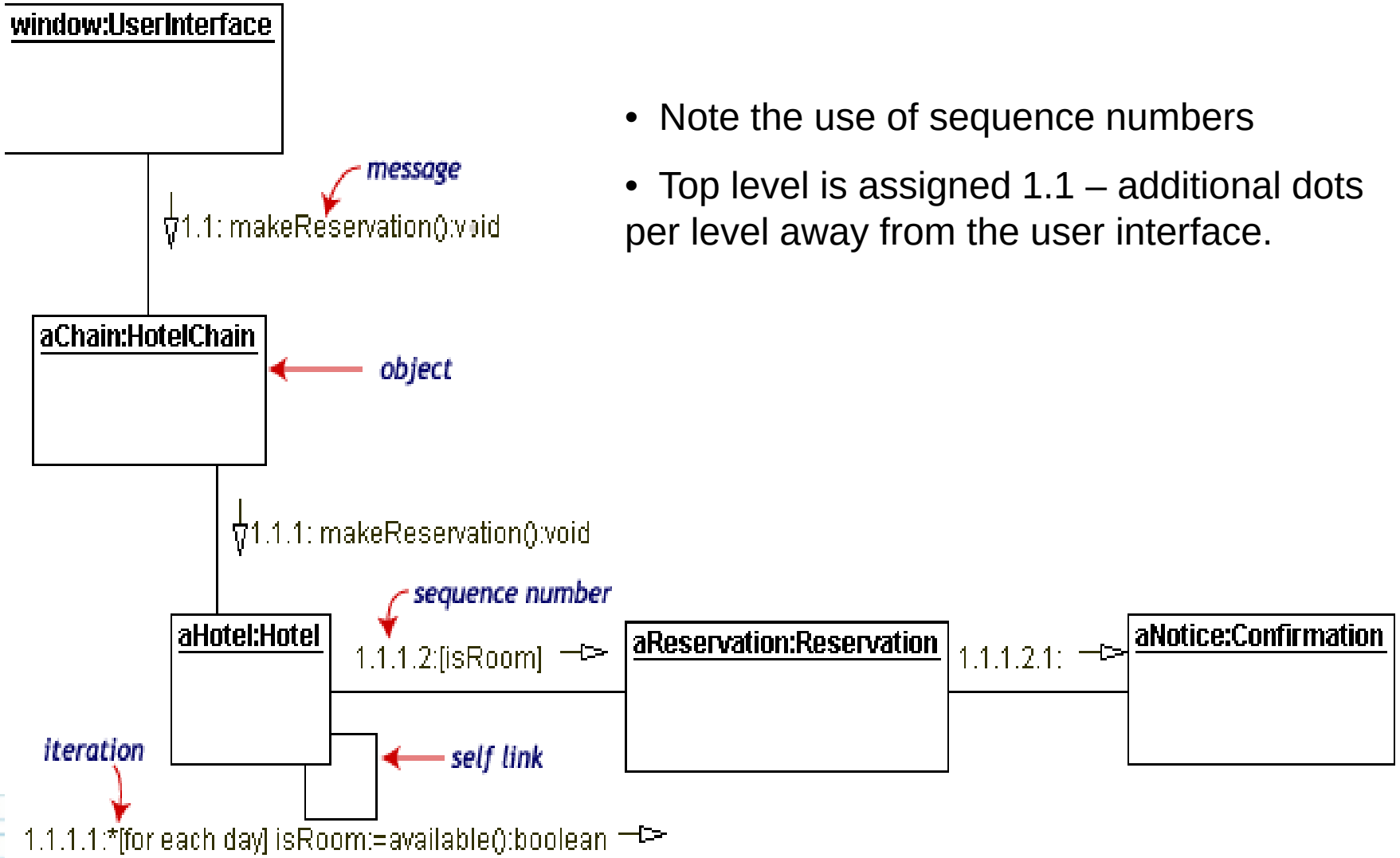


Note advanced additional features such as abstract classes, generalization (inheritance), aggregation (orderdetails make up order)

Sequence Diagrams

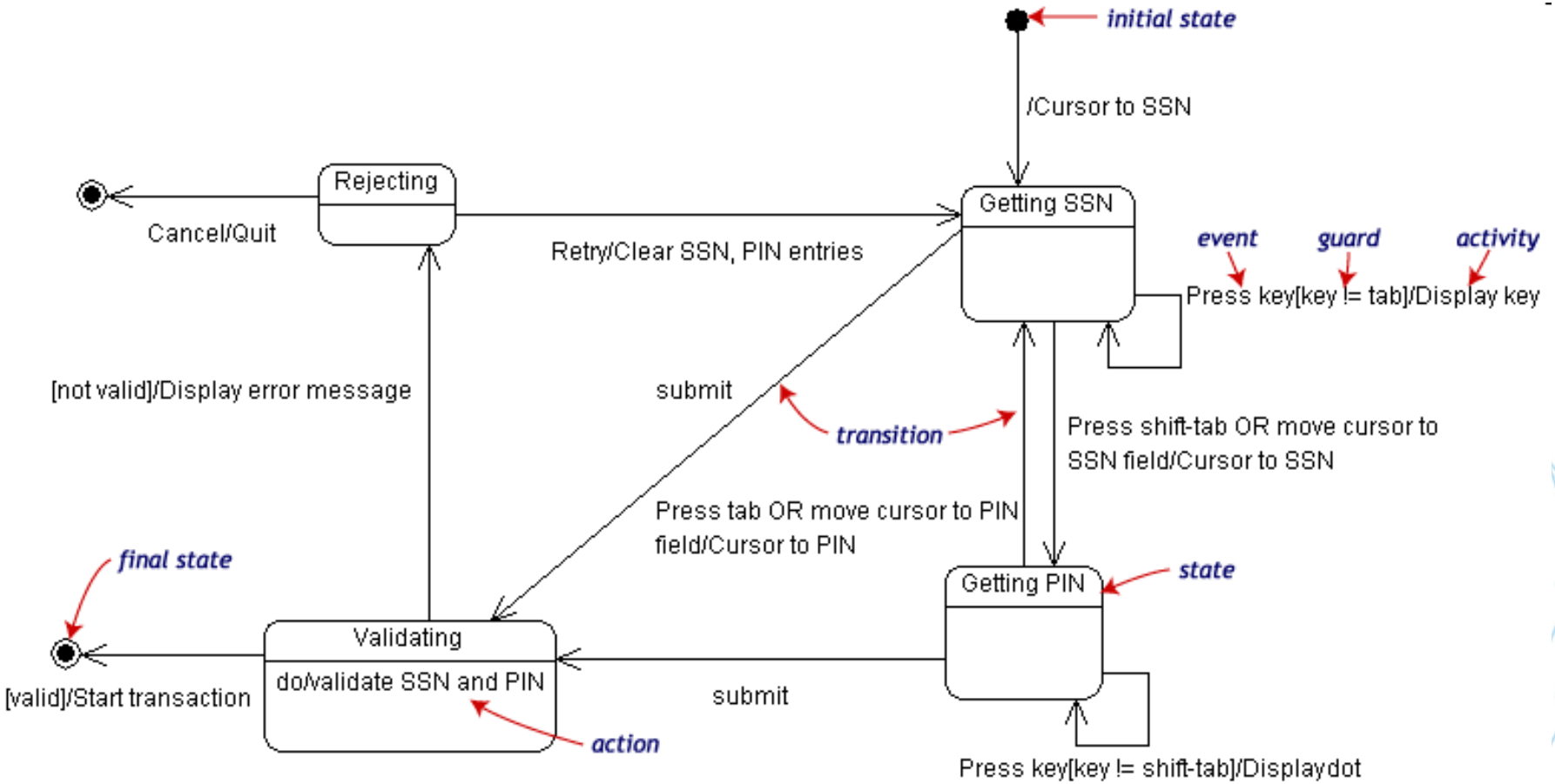


Collaboration Diagrams



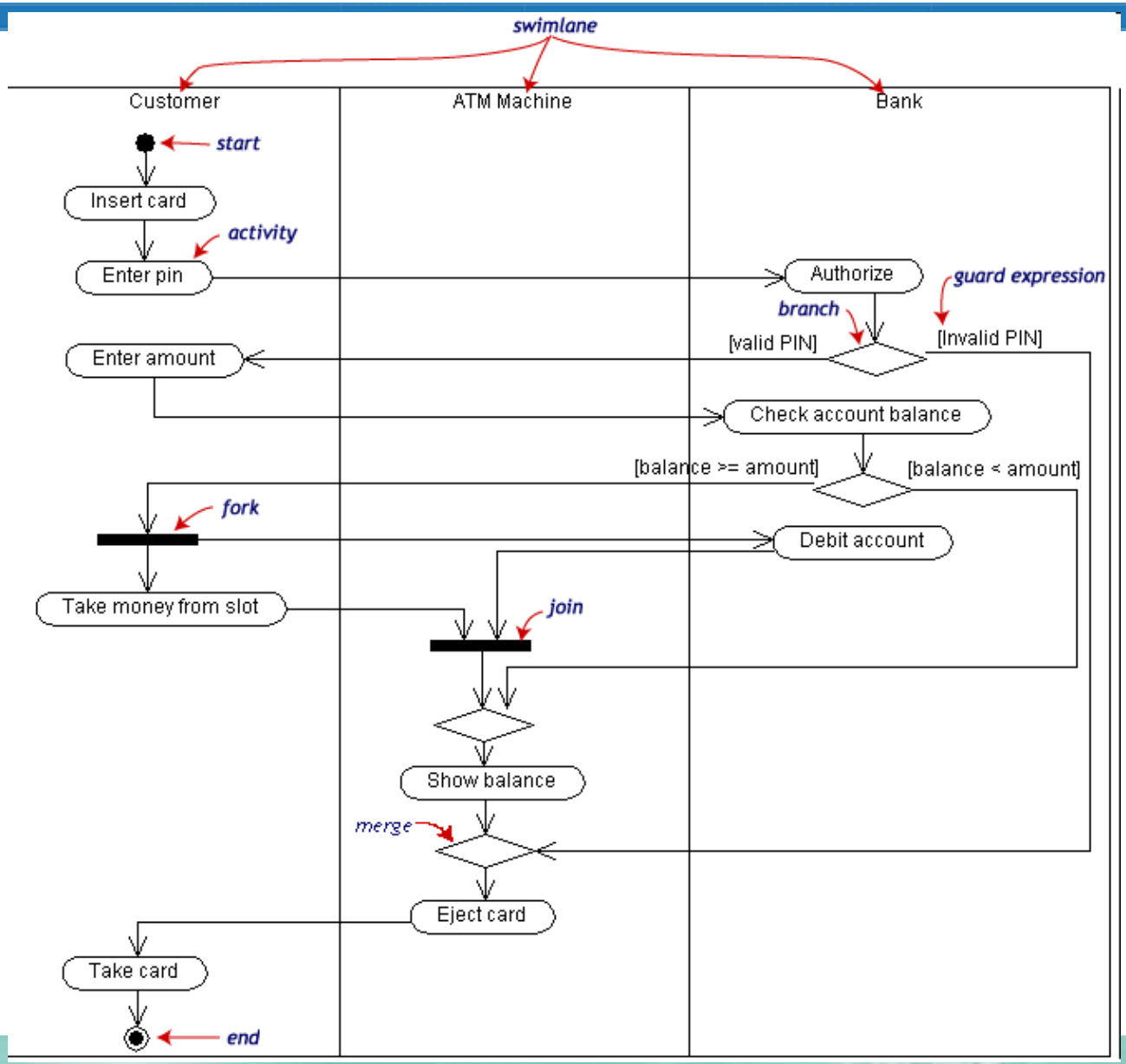
- Note the use of sequence numbers
- Top level is assigned 1.1 – additional dots per level away from the user interface.

State Diagrams

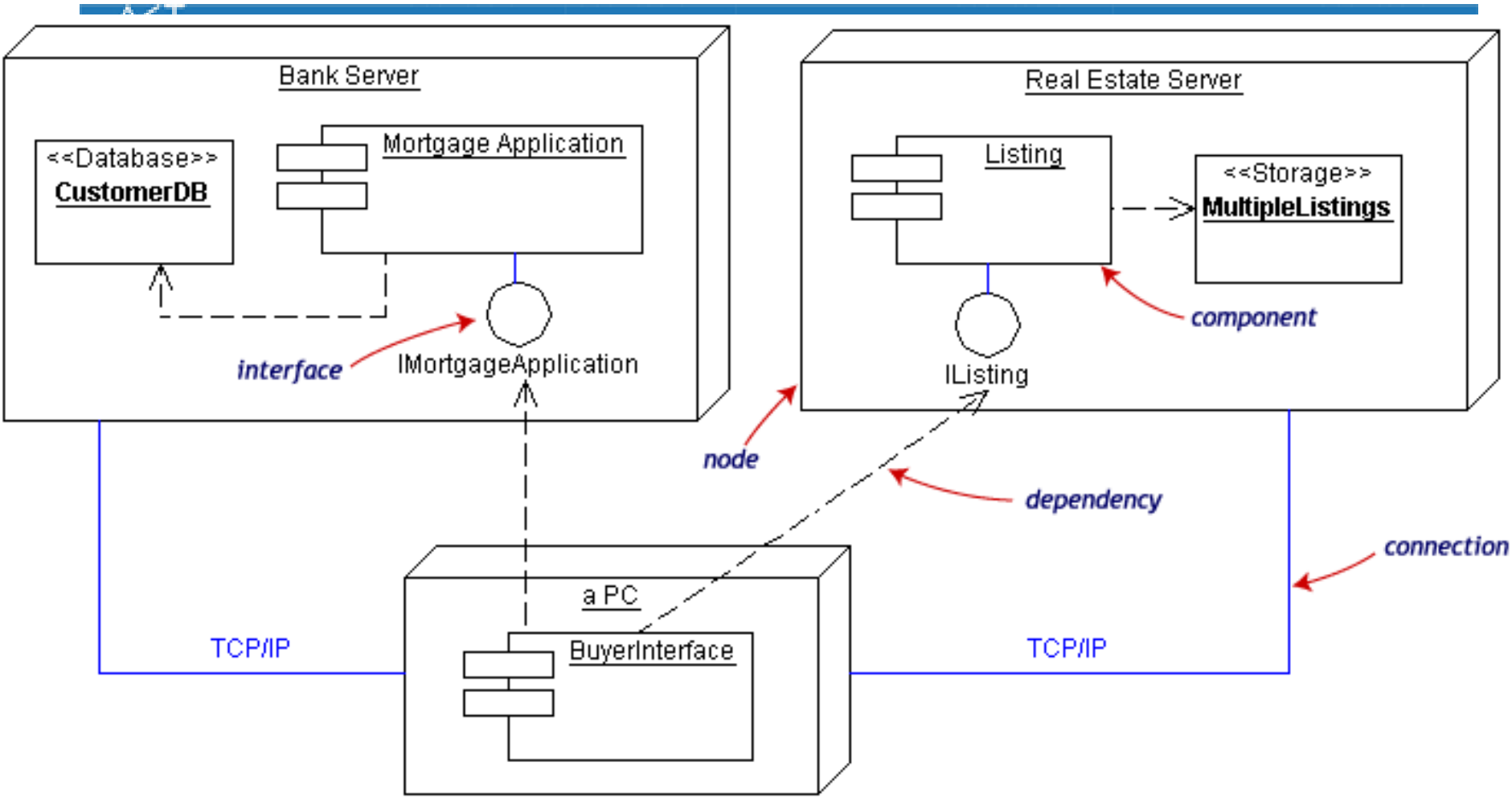


-
- Not necessary for all objects – just the critical ones.
- Super states – can be used to nest states to make diagram easier to read.

ACTIVITY DIAGRAMS



COMPONENT AND DEPLOYMENT DIAGRAMS



High level component communication and dependencies.

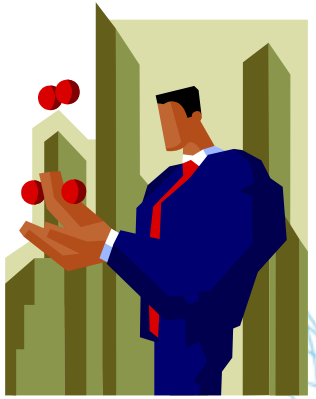
SOA Marketplace Example

Mechanics Are Us
Dealer

Order

Conformation

Shipped



Acme Industries
Manufacturer

Status

Physical
Delivery

Ship Req

Shipped

Delivered

GetItThere Freight Shipper

Marketplace Services

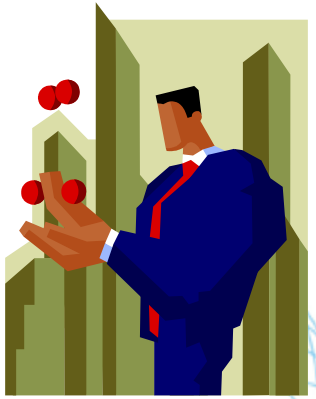
Consumer

Order

Conformation

Shipped

Provider

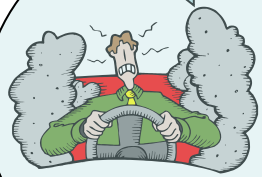


Mechanics Are Us
Dealer

Acme Industries
Manufacturer

Consumer

Status



Physical
Delivery

Provider

Provider

Ship Req

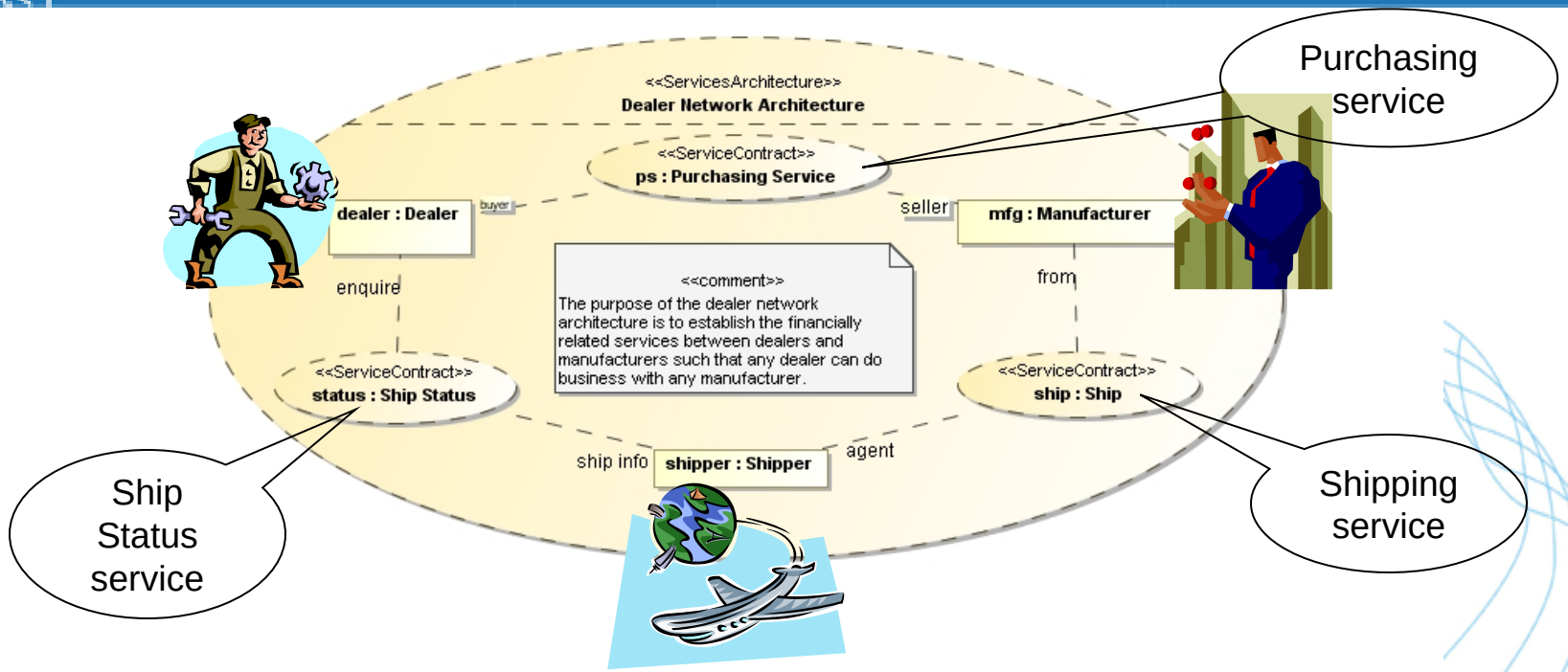
Shipped

Delivered

Consumer

GetItThere Freight Shipper

Services Architecture



Services Architecture (or SOA) 是参与者角色的网络，它们提供/消费服务来达到目标。SOA定义了参与者类型和完成角色的服务实现

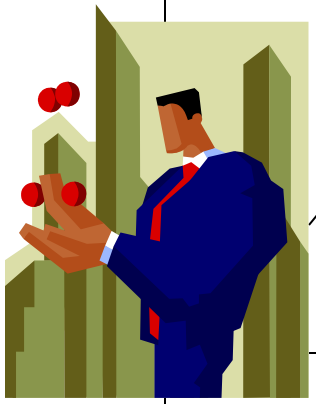
SOA显示参与者如何共同工作；通过UML协作图定义一个组织的Services Architecture

Inside the Manufacturer

Order

Conformation

Shipped



Order Processing

Service

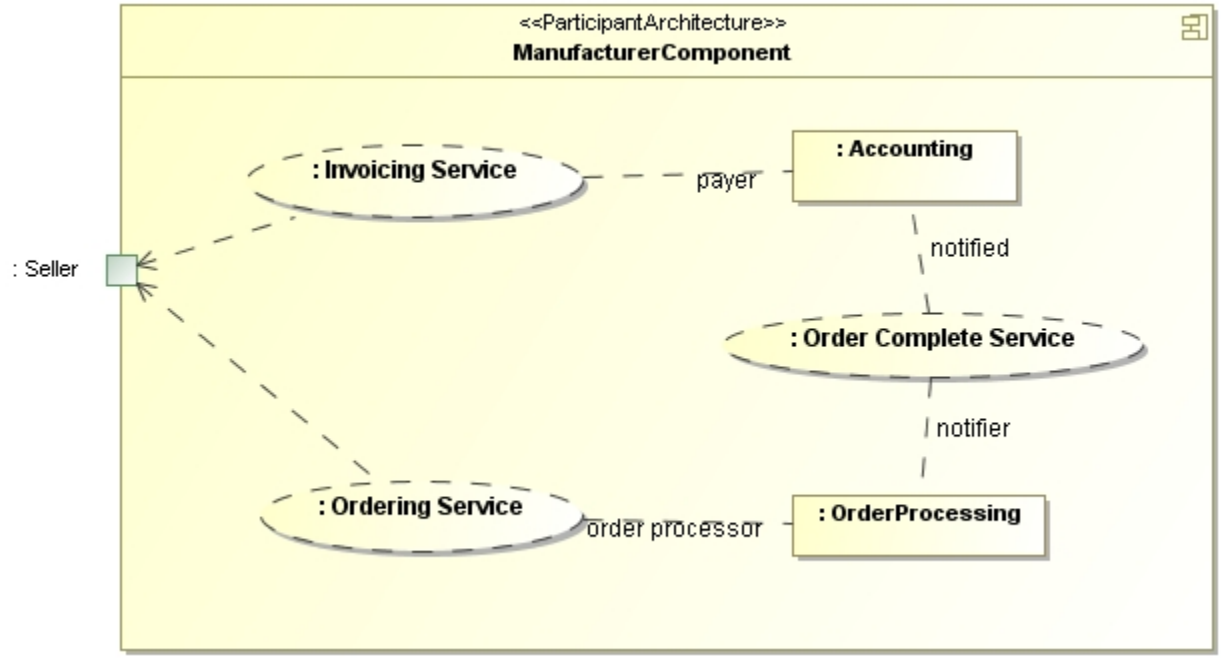
Accounting

Ship Req

Shipped

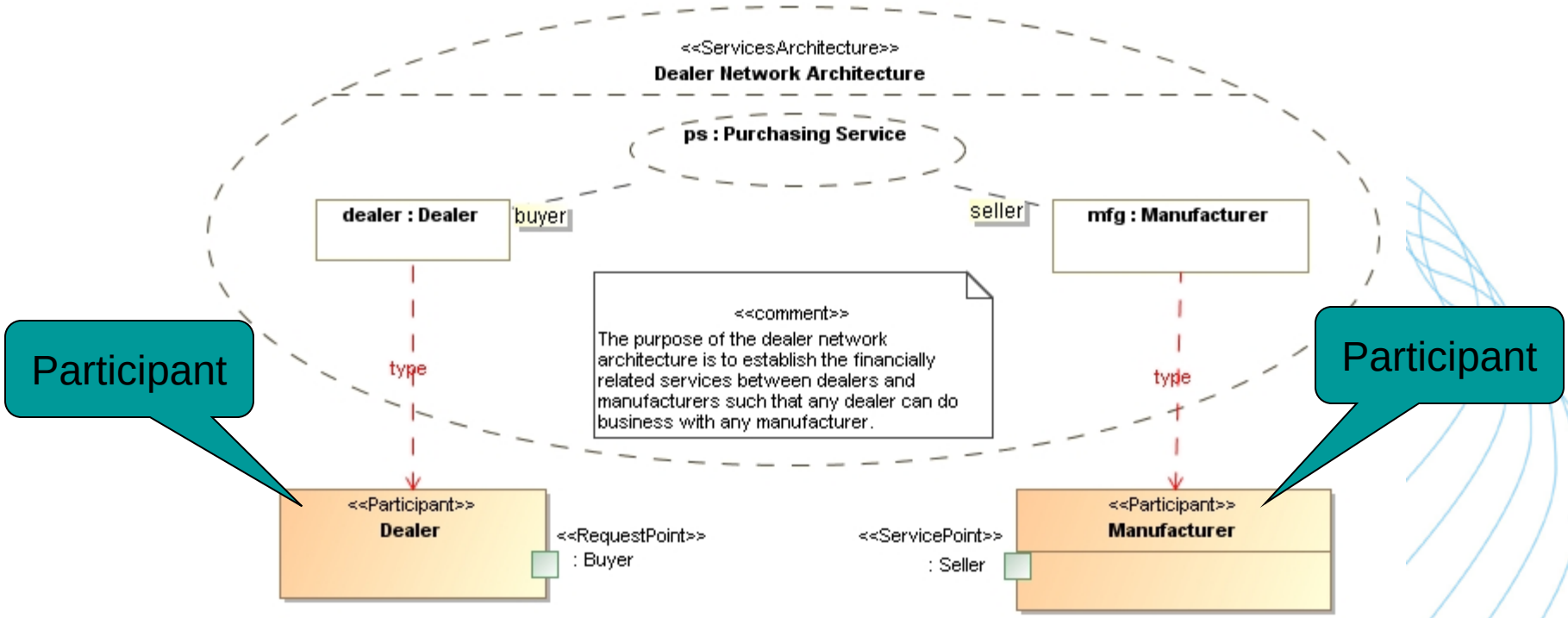
Delivered

Services architecture for a participant



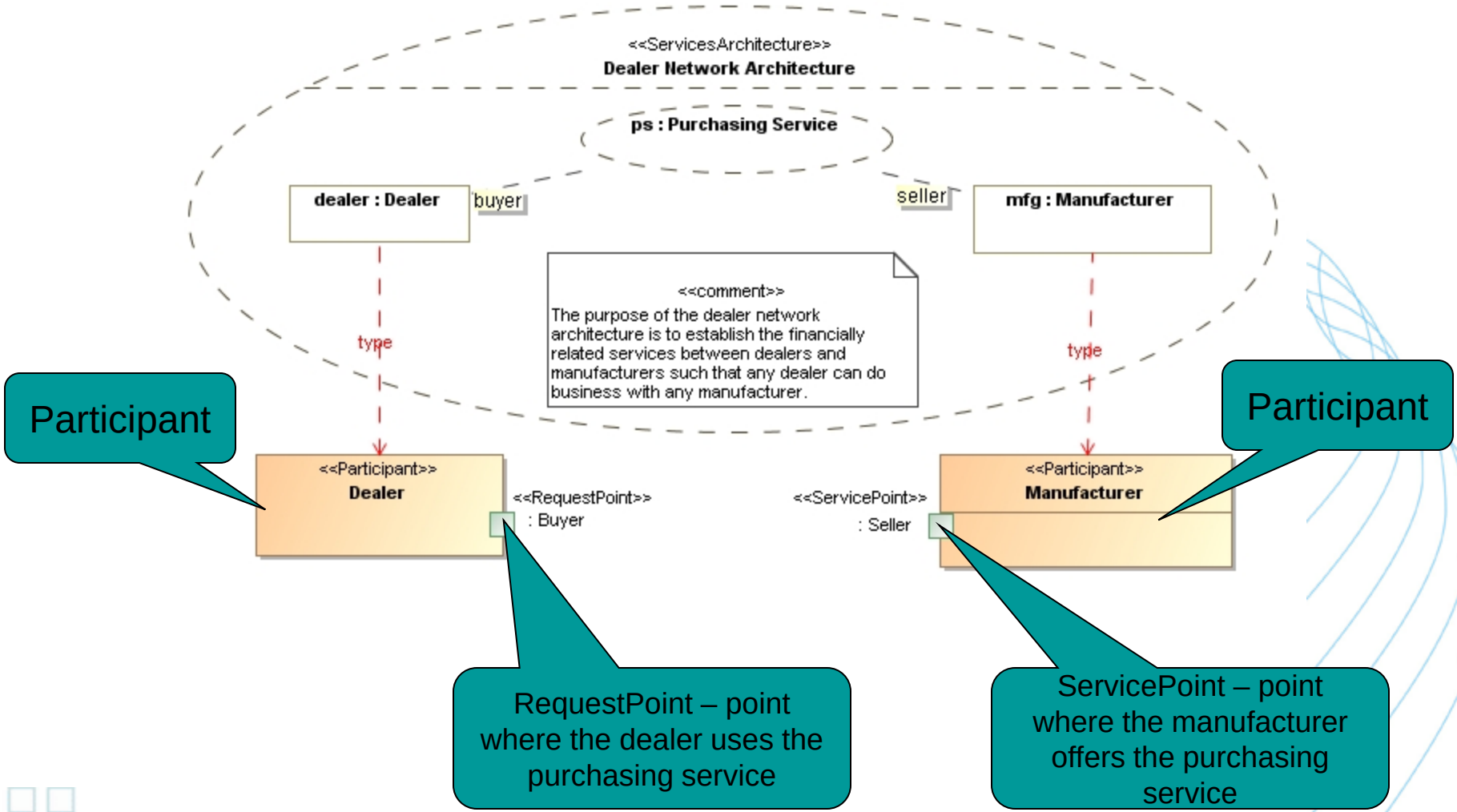
Participant Architecture 是一种高层的services architecture，用于定义一组内/外部参与者如何使用服务来实现参与者的职责，通常一个参与者也包括业务过程

Participants



Participant表示逻辑或真实的人/组织单位，他们参与到services architectures and/or业务过程中. SoaML中，参与者通过定义外部contract，提供并使用服务

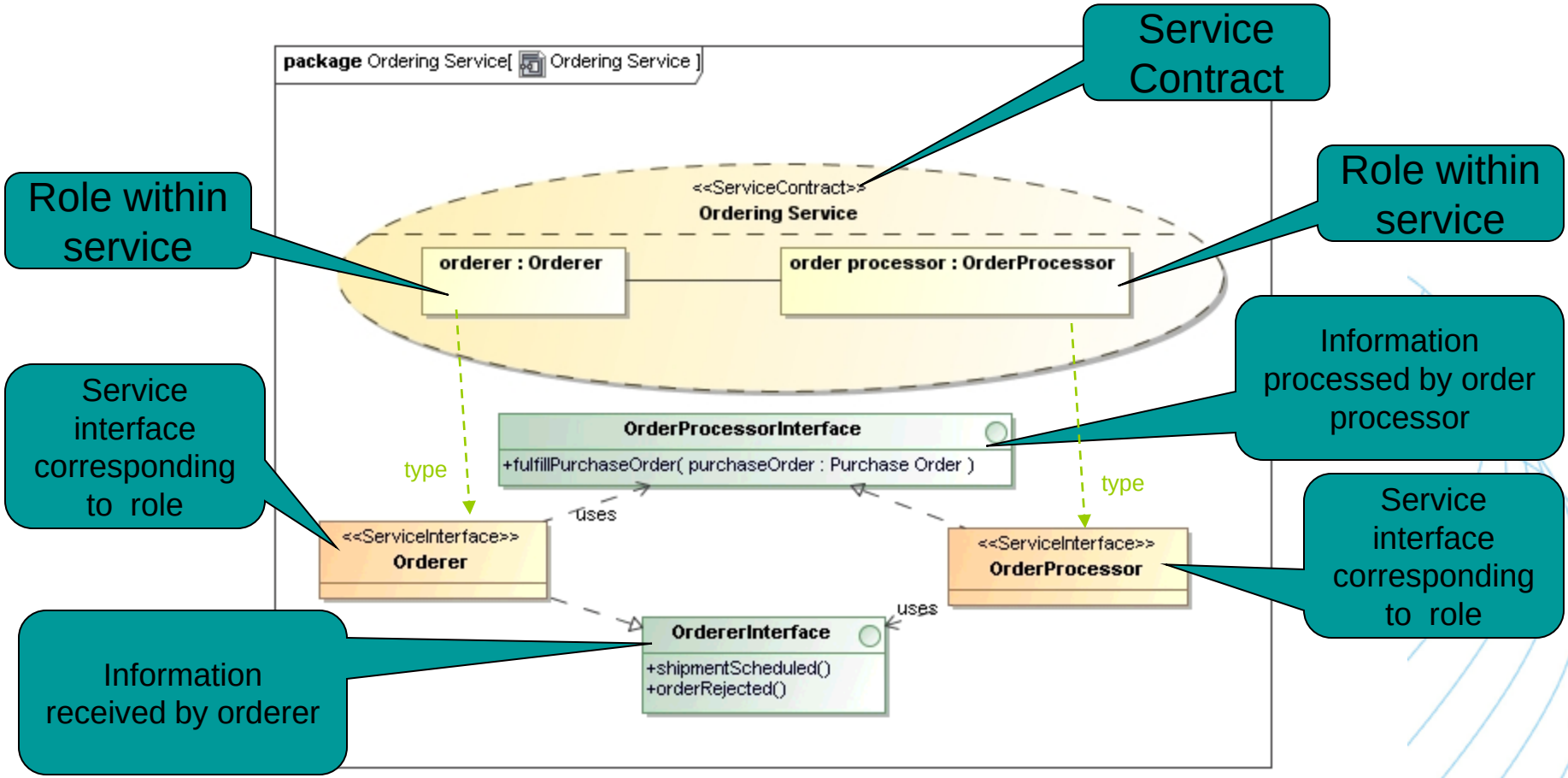
Participant的ServicePoint and RequestPoint



ServiceContract

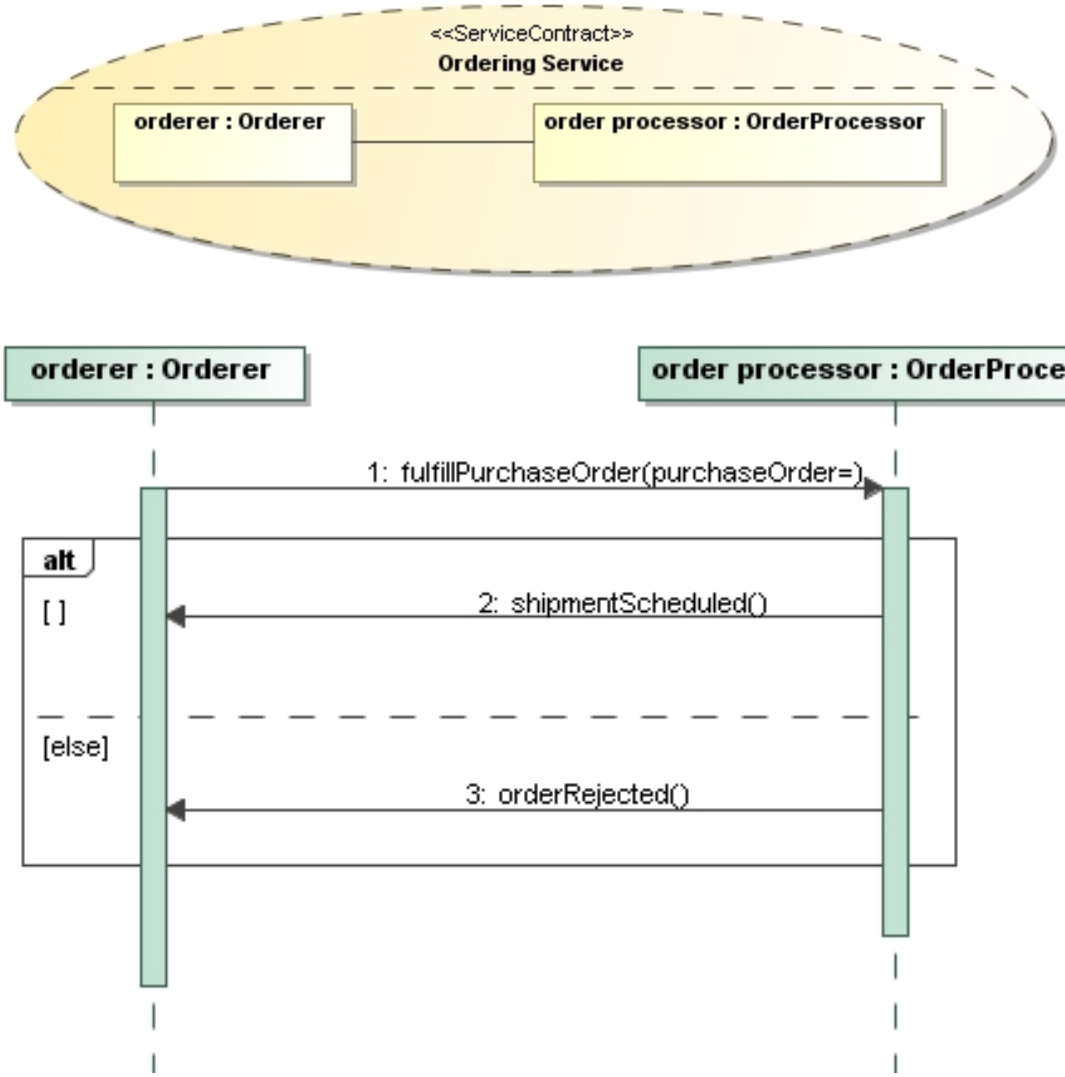
ServiceContract定义了参与者必须遵守的术语、条件、接口和编排
服务的完整规范包括信息、编排和任何术语与条件
ServiceContract被同时绑定到服务的提供者和消费者上，其基础也是UML的协作图（关注服务之间的交互）

Service Contract



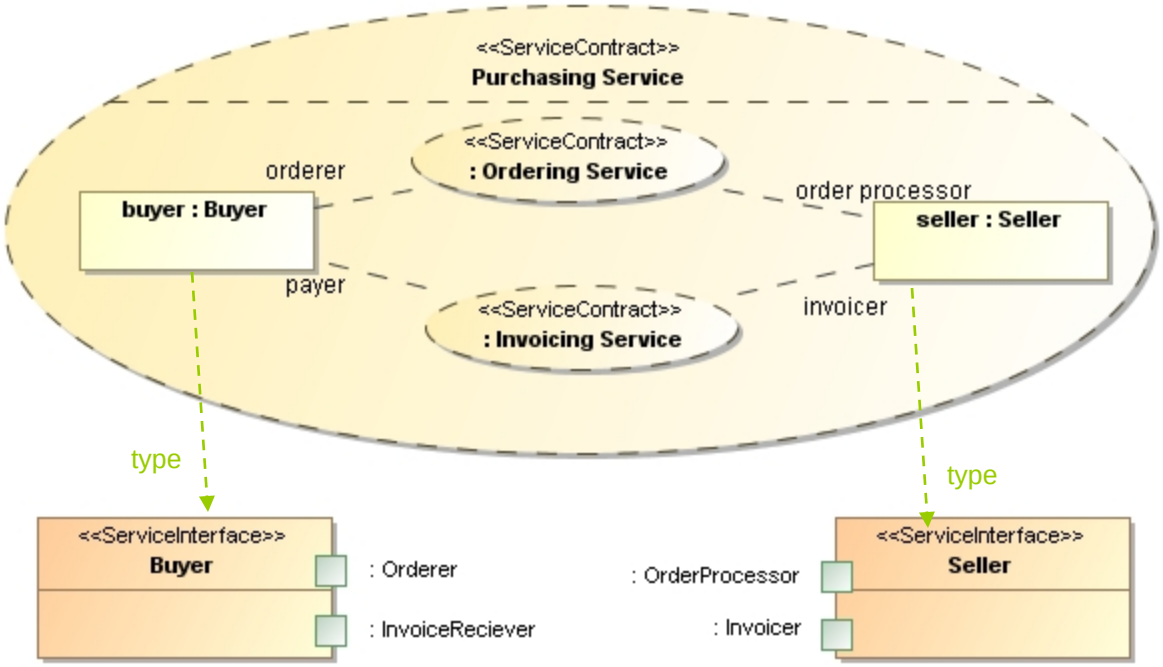
The service contract specifies the details of the service – what information, assets and responsibilities are exchanged and under what rules

Simple Protocol Choreography for Ordering Service Contract



Compound services

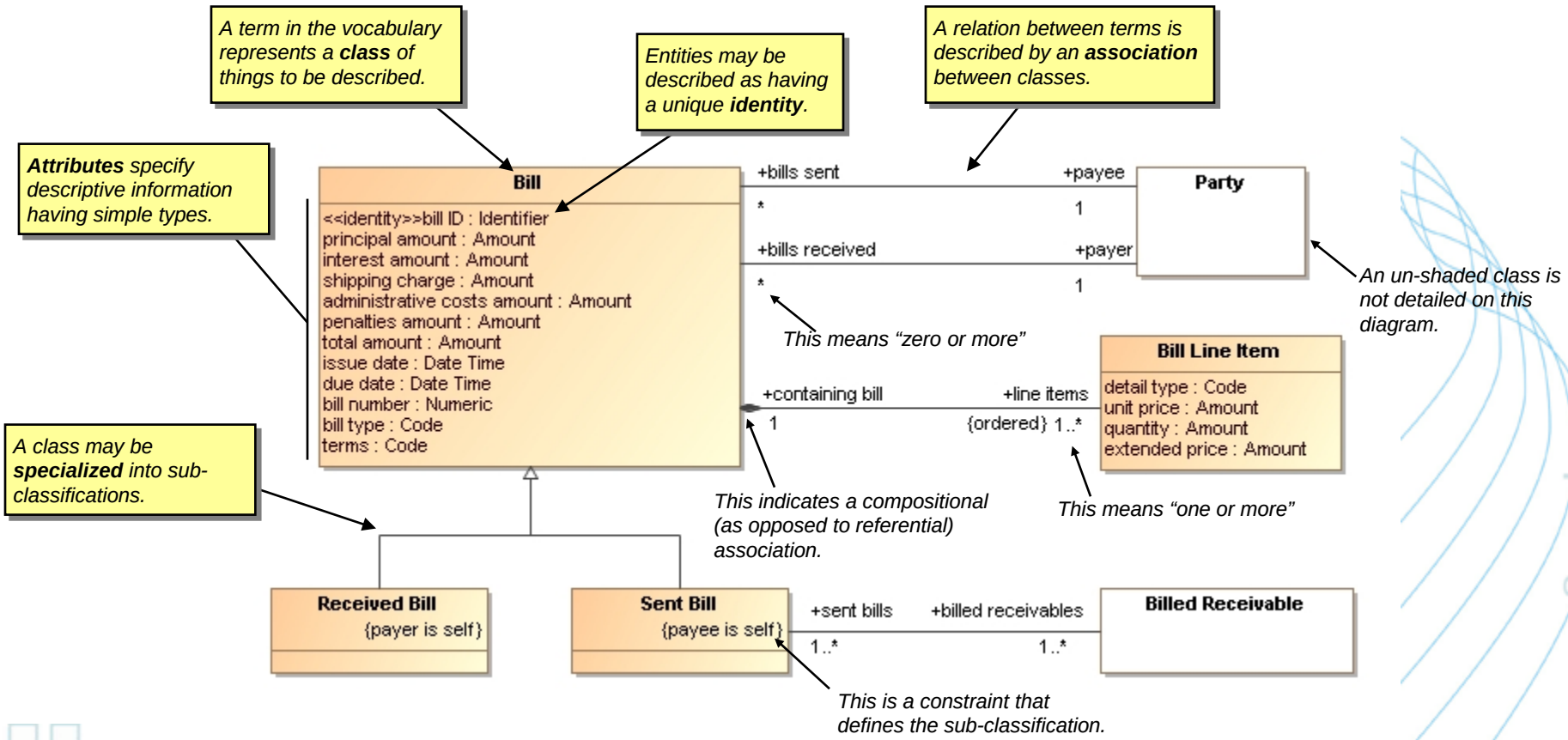
Compound Services are defined by using more granular services to "build up" an enterprise scale service



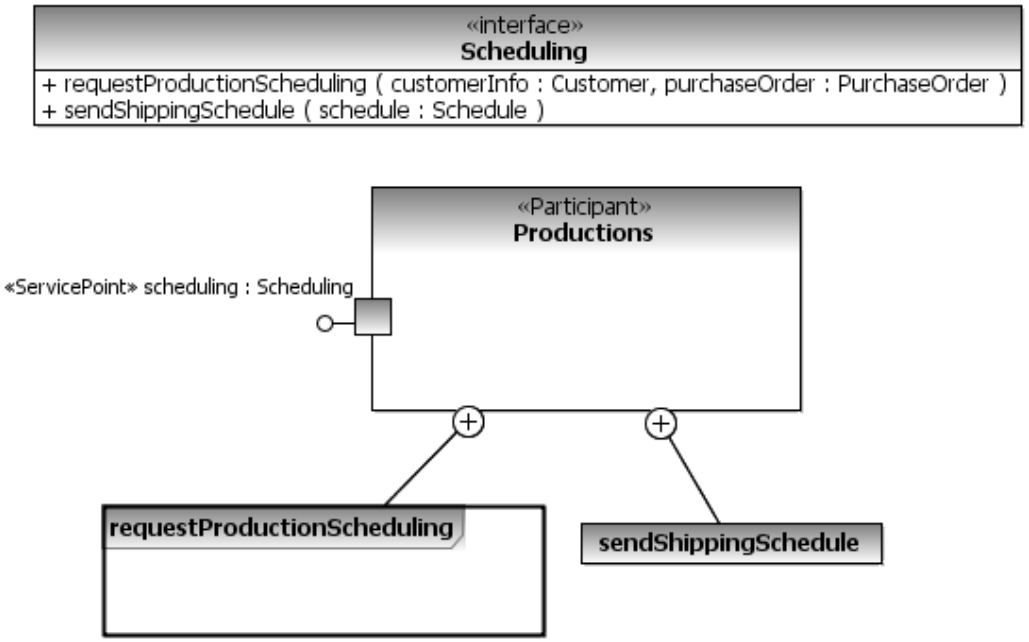
Each composed service becomes a port on the service interface

Information Model

For Messages and Entities



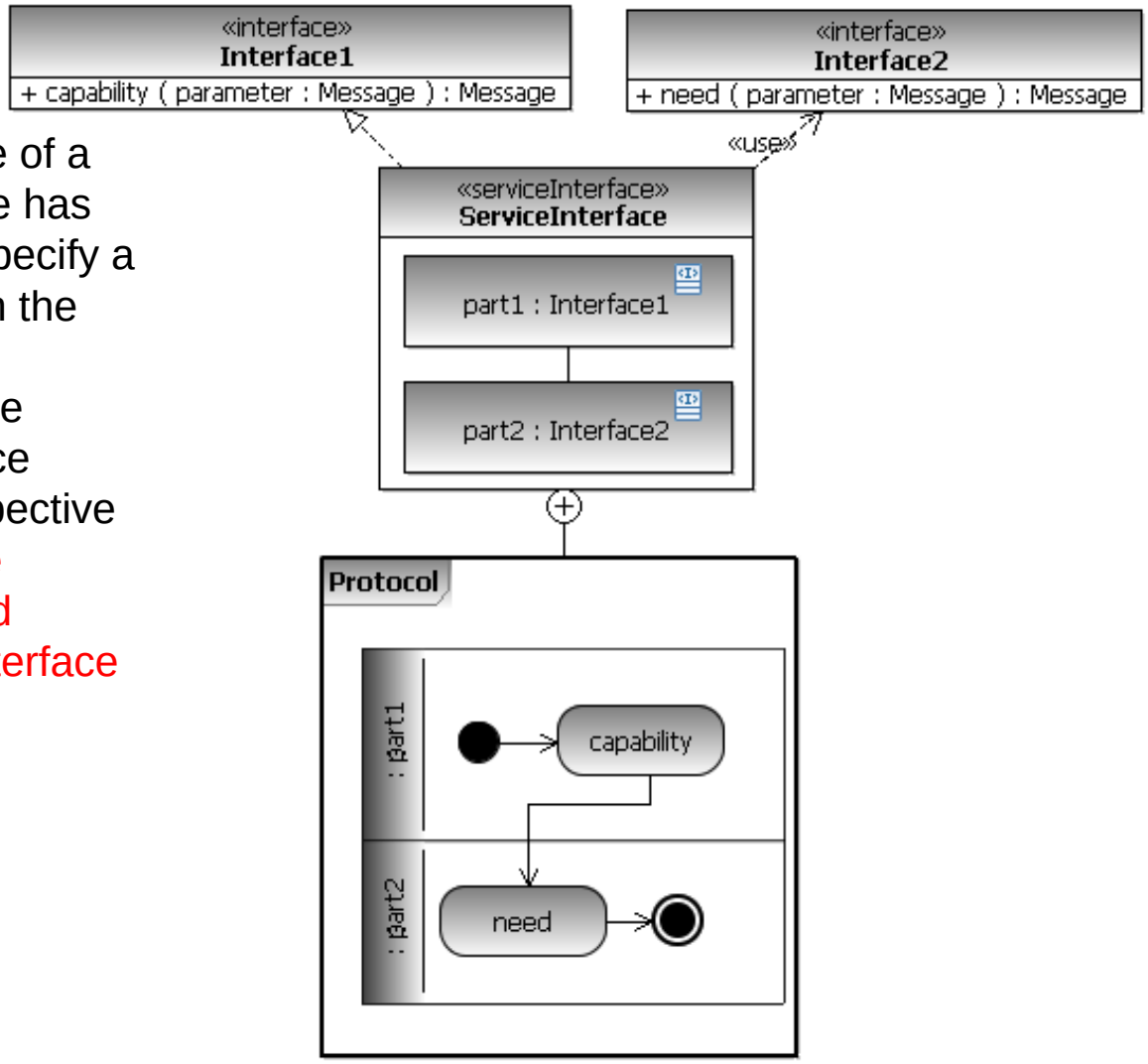
ServicePoints and Service Participants



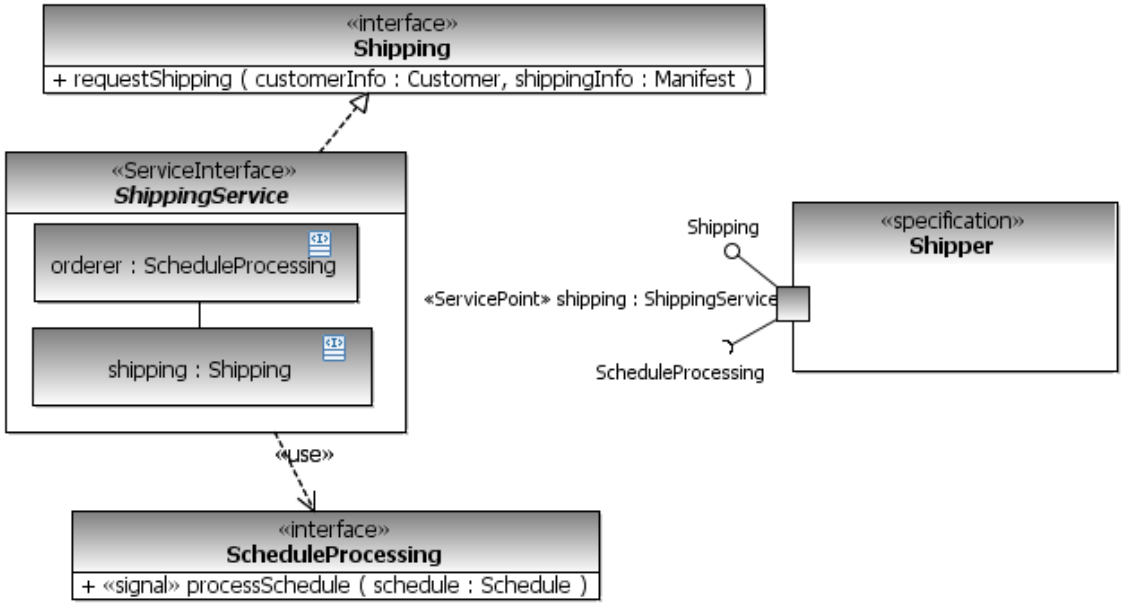
A ServicePoint is the offer of a service by one participant to others using well defined terms, conditions and interfaces. A ServicePoint defines the connection point through which a Participant offers its capabilities and provides a service to clients.
A ServicePoint is a mechanism by which a provider 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», .

ServiceInterface

a ServiceInterface can be the type of a service port. The service interface has the additional feature that it can specify a bi-directional service – where both the provider and consumer have responsibilities to send and receive messages and events. The service interface is defined from the perspective of the service provider using three primary sections: the provided and required Interfaces, the ServiceInterface class and the protocol Behavior.



Participant with ServicePoint

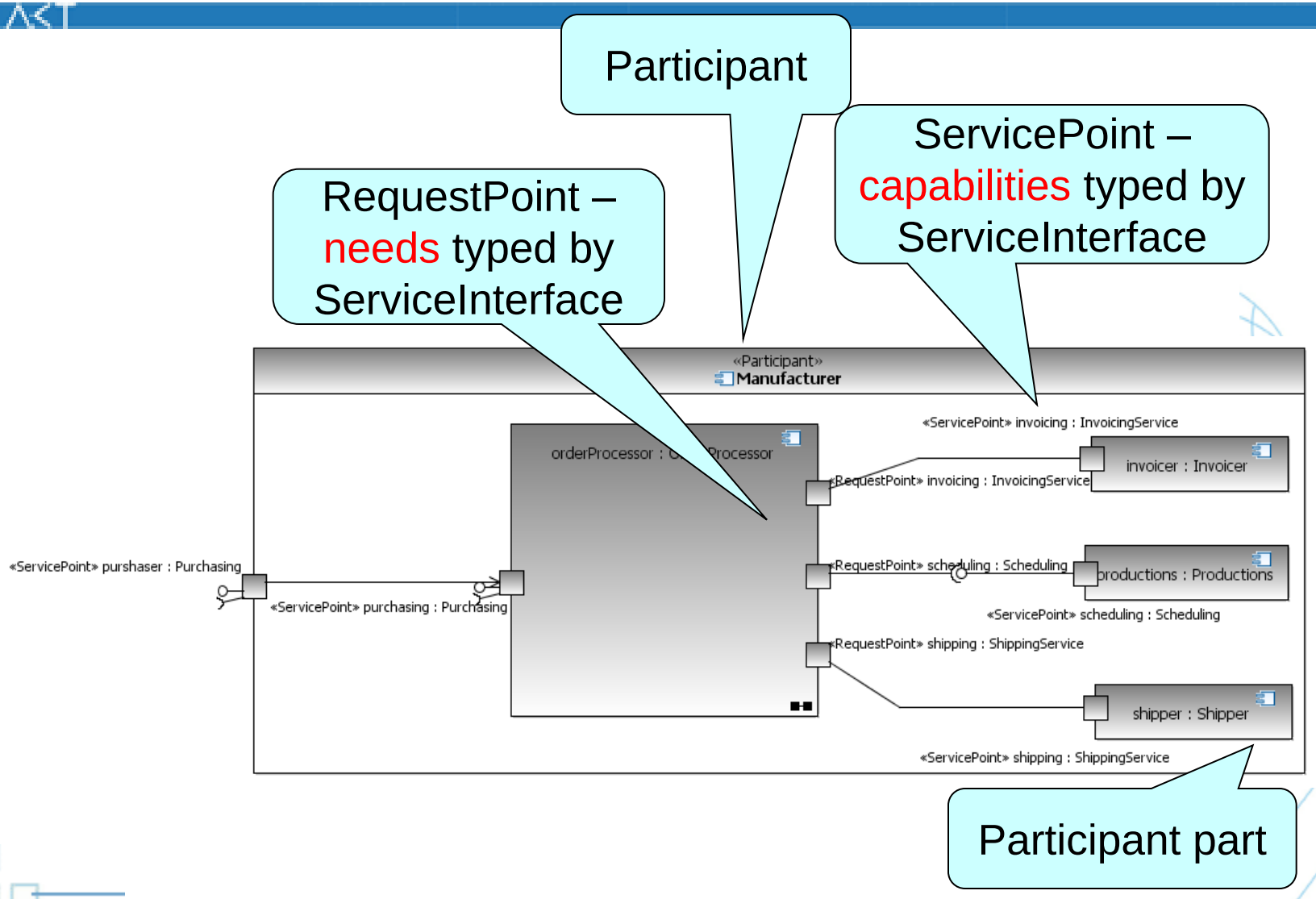


A ServicePoint is the point of interaction on a Participant where a service. On a service provider this can be thought of as the “offer” of the service (based on the service interface). The ServicePoint is the point of interaction for engaging participants in a service via its service interfaces.

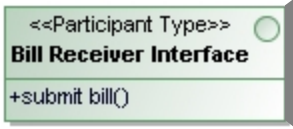
Participant with ServicePoints and RequestPoints

The type of a RequestPoint is also a ServiceInterface, or UML Interface, as it is with a Service point. The RequestPoint is the conjugate of a ServicePoint in that it defines the use of a service rather than its provision. This will allow us to connect service providers and consumers in a Participant.

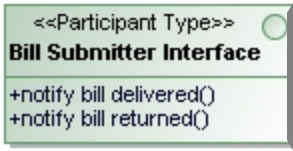
Participants may be assemblies of other Participants



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

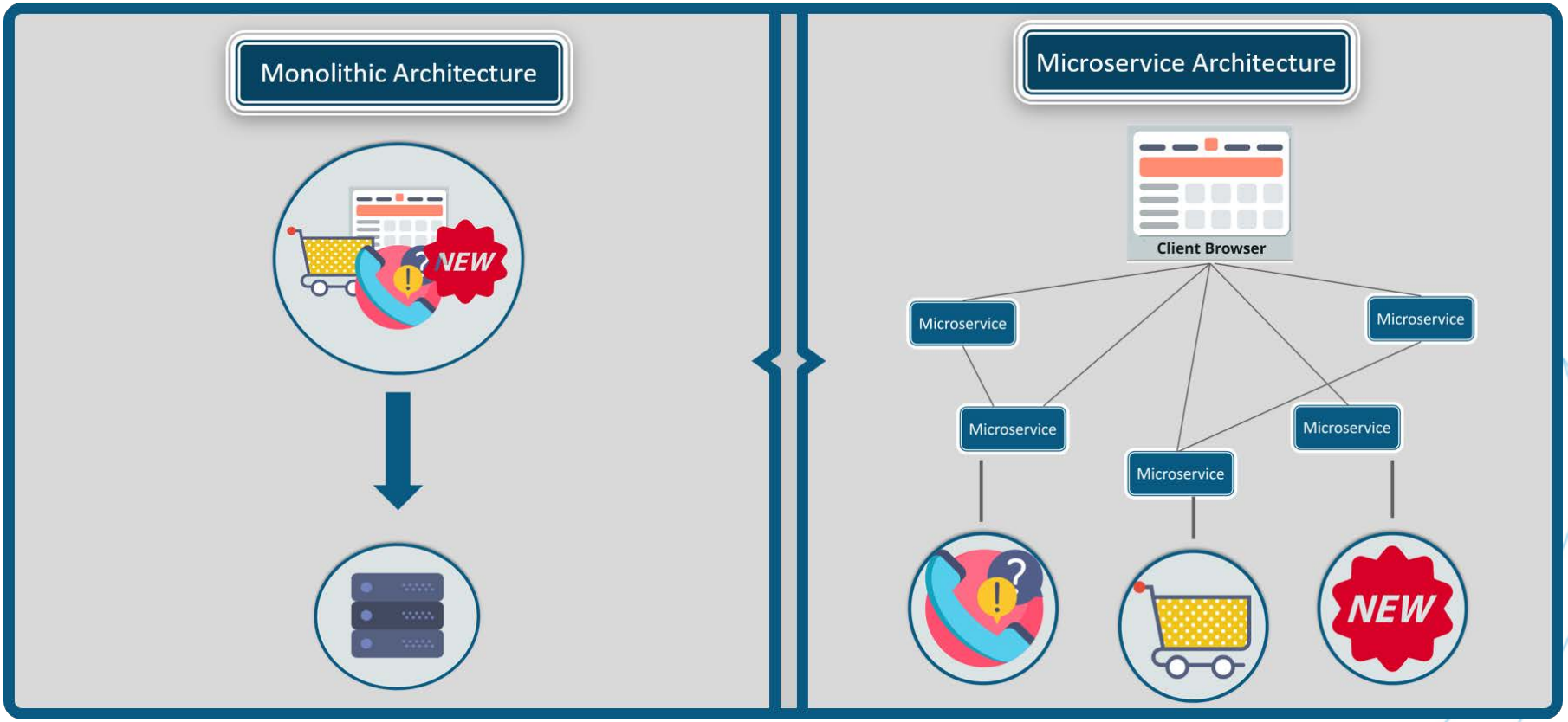
- 传统SOA架构
  - 服务之间是对等的，提倡服务的公开和共享
  - 以业务流程和企业服务总线提供灵活的服务组合能力，以适应敏捷的复杂业务变化
- 云服务
  - 大部分云服务一般只在云平台内部提供，即使提供远程服务，也只支持云平台的管理功能
  - 即使用户开发和托管的服务，绝大部分不对外共享
  - 面向大规模公众用户，更强调服务的性能和可靠性
- 影响
  - 业务流程层和整合层（企业服务总线）开始弱化
  - 服务质量层开始加强

RESTful服务、Web APIs和微服务

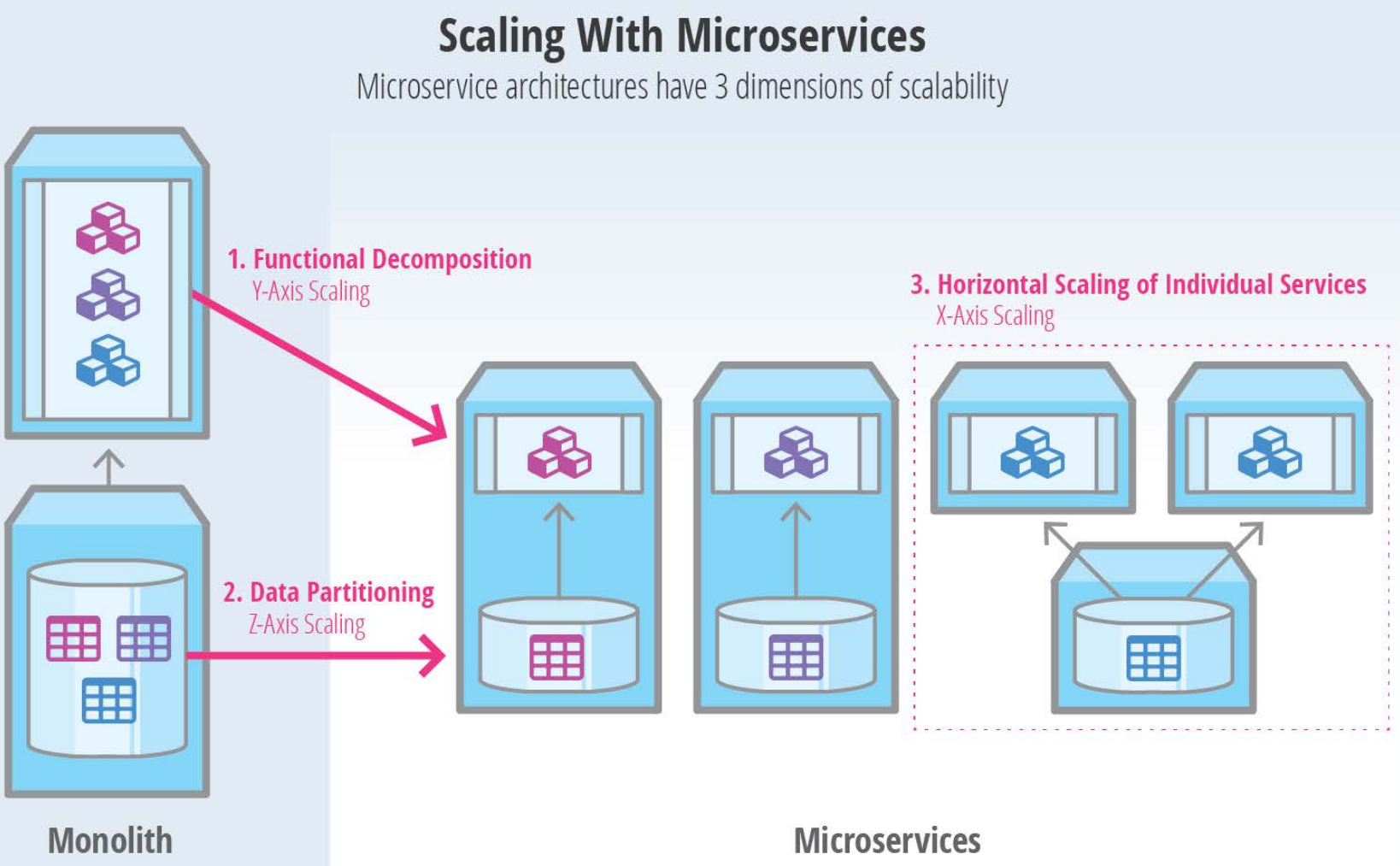
- 传统的SOAP Web服务
  - 一个目标是为了业务流程和自动化的服务组合
  - 服务的自描述、自动发现，所以需要XML, WSDL以及业务流程模型描述，包括动态代理动用等
- 云计算背景下： RESTful服务和微服务
  - 重性能和可靠性，轻量化
  - 弱化自描述、自动发现和自动组合
  - 可利用Web中已有的大量Web APIs
- SOA的新发展：微服务架构
  - 泛化的服务：SOAP、RESTful、Web APIs
  - 服务组合和业务流程可以概念化，用于设计和分析，并不一定是实体功能
  - 服务性能和可靠性强化

微服务架构

- 混合架构→微服务架构：手动进行服务的组合



服务分割和横向扩容



结 束!